

Contents

1	General	3
1.1	Introduction	3
1.2	Conventions	3
2	Language Description.....	4
2.1	Statements.....	4
2.2	Operators and precedence	4
2.3	Remarks.....	5
2.4	Keywords, Identifiers (Names) and Labels	5
2.5	Assignments and parameter passing	5
2.6	Scopes	6
2.7	Data Types.....	6
2.8	Variables and Constants.....	10
2.9	Constants	12
2.10	Functions and Subroutines	12
2.11	Classes	14
2.12	Control Statements	14
2.13	Error Handling.....	17
2.14	Folder Path	19
3	Runtime Context.....	20
3.1	Object hierarchy	20
3.2	Cell context	21
3.3	Call Flow	22
3.4	Object Hierachy	23
4	Variables as Objects	24
4.1	String.....	24
4.2	Date.....	27
4.3	Id	31
4.4	Binary	31
4.5	Error	33
4.6	Array.....	33
4.7	Object (Variables)	34
5	Application Objects.....	36
5.1	cTubes	36
5.2	Net.....	37
5.3	License.....	39
5.4	Tubes	41
5.5	Views.....	42
5.6	Tube	43
5.7	Structures.....	48
5.8	View	49
5.9	Catbrowser.....	50
5.10	Data.....	51
5.11	Selections.....	54
5.12	DataAreas	55
5.13	DataArea	55
5.14	Cells	56
5.15	Layout	56
5.16	LayoutAreas.....	57
5.17	LAYOUTAREA.....	58

5.18	CELLGROUPS	58
5.19	CELLGROUP	58
5.20	CELL	59
5.21	FONT (CELL)	62
5.22	DRAG/DROP	63
5.23	DIALOG BOX	64
5.24	TIMER	65
5.25	ImportCsv	66
5.26	FileIndex	69
5.27	File	71
6	Database Objects	74
6.1	FOLDER	74
6.2	RECORD	77
6.3	FIELDS	78
6.4	FIELD	78
6.5	GROUPS	79
6.6	GROUP	81
6.7	USER	83
7	Events	85
7.1	Built in Events	85
8	Global Methods	86
8.1	Mathematics	86
8.2	File Handling	88
8.3	Miscellaneous	89
8.4	Alternative Global Functions	92
9	System Names	93
10	Example of COM use	94
11	Formatting / Format Strings	96
11.1	Numbers / Currency	96
11.2	Date / Time	96
11.3	Extensions for Numbers and Currency - Measurement module	97
11.4	Special format for value lists	98
12	Elements	99
12.1	Element ValueBar	99
13	Other	100

1 General

1.1 Introduction

cTubes has a built-in scripting language allowing to automate processes, customize functionality and build complex computations - **cTubes Basic**.

Since *cTubes Basic* is derived from the simple BASIC programming language and shares some concepts with the well known Microsoft Visual Basic for Applications™ as well as with other languages, it should be very easy for the beginner to become familiar with it very quickly while providing the advanced user a means to efficiently create powerful applications.

cTubes Basic contains powerful string handling capabilities like associative arrays and regular expressions, all combined with the ease and simplicity of the BASIC syntax.

Starting with **cTubes Network PRO** you can use global formulas (see 8.1 and 8.4) in Layout Cells or in computed fields.

cTubes Developer gives you full access to all the scripting objects and functionality described in here.

Note: This manual is beta, items still needing work are marked as TBD.

1.2 Conventions

Code Snippets, Declarations, etc.

Function, property and syntax declarations:

[]	... Array arguments
< >	... Optional arguments
R/W	... Read/Write property
R/O	... Read only property
<i>italic</i>	... Identifier
bold	... default if none of the possible alternatives is specified
	... separator for alternative mutual exclusive elements

2 Language Description

2.1 Statements

Like with any other programming language a *cTubes Basic* program consists of a sequence of statements. A statement can be considered an execution entity which is executed as a whole and must, therefore, be semantically complete. A statement may modify values, diverge the program flow to any other statement or define variables or functions being used with other statements in the code.

One statement usually starts at the beginning of one text line and ends with the end of the text line. While it is possible to have multiple statements in one line by use of the semicolon ';' as separator between the statements, it is a good practice to have just one statement per line.

Note: The semicolon ';' as statement separator is an intentional deviation from the standard BASIC syntax (where the colon ':' is used for the same purpose) and may be more familiar to users having experience with the programming languages C or C++.

Example of valid statements:

```
a = 1
b = 2
c = 3 ; d= 4
```

Invalid:

```
a =
1
c = 3 d= 4
```

2.2 Operators and precedence

- | | | | |
|-----|------------------|--------------------------|-----------------------------|
| 1. | () | DOT | |
| 2. | | post increment/decrement | x++ x-- |
| 3. | ^ | power | x= v ^ 2 |
| 4. | | pre increment/decrement | ++x --x |
| 5. | / * | division multiply | |
| 6. | % | modulo | x = v % 16 |
| 7. | + - | unary sign | |
| 8. | & | string concatenation | |
| 9. | << >> | bit shift left / right | x= v << 4 |
| 10. | < > <= >= = <> | comparison | |
| 11. | not ! | bitwise NOT | |
| 12. | and | bitwise AND | |
| 13. | or | bitwise OR | |
| 14. | xor | bitwise XOR | |
| 15. | && | logical AND | |
| 16. | | logical OR | |
| 17. | ? | alternative value | x = condition ? val1 : val2 |
| 18. | = += -= *= /= ^= | &= assignment | x +=5 adds 5 to x |

2.3 Remarks

cTubes Basic supports only single line remarks. A remark begins at the remark identifier '#' and ends at the end of the current text line.

All characters within a remark section are ignored by the script.

Examples:

```
c = 3 # this is a comment
d = 4 # this: d = c + 1 is not interpreted
```

2.4 Keywords, Identifiers (Names) and Labels

Keywords are predefined names for language elements defining any specific kind of operation when the statement gets executed.

Identifiers (in cTubes terms most often simply called *Names*) are names of user defined language elements such as variables, functions, subroutines, objects etc.

In general keywords and identifiers **are case insensitive**.

Identifiers can contain any number of alphanumeric characters being in the set of 'a' – 'z', 'A' – 'Z', '0' – '9' and '_'. An identifier has to start with an alpha character and can not start with a digit.

Keywords are "reserved identifiers", i.e. it is not possible to have a user defined object like e.g. a variable with the name of a keyword.

cTubes script gives quick access to identifiers defined outside the script. Such system identifiers (aka *SysNames*) can be names of fields or layout names. System identifiers are always prefixed by a dollar '\$' character and may optionally consist of a path specification in case the name is not unique enough (refer to section System Identifiers).

Labels are identifiers for jump-targets for 'goto' statements. Label names follow the same rules as all other identifiers.

2.5 Assignments and parameter passing

Normally the assignment operator '=' assigns a base-type value (string, number, date, etc.) to the left. If the right expression returns an object the default base-type value is assigned to the left.

To assign an object reference use the 'Set' statement. If the right value is a simple base type such as a string, the value is assigned just as with normal assignment.

In function calls simple types are passed as values, objects passed as reference by default. In addition parameters can be declared as byref so that also simple types can be passed as reference.

To return an object from a function use Set Result = object.

To clear a variable pointing to an object reference SET it to a variable that is still Null. TBD.

```
c = 3 # value assignment
set f = Tube.folder # this assigns the folder object as object reference
fname = tube.folder # assigns the default value of the folder (in this
case the folder name)
```

2.6 Scopes

Scopes define the visibility and most often the lifetime of variables and objects. A scope is an entity to which an e.g. object belongs. Scopes types are hierarchical.

Following scopes exist in cTubes Basic:

Function Scope

Everything declared or defined within a sub or function exists within the function scope. Objects defined within a function like e.g. variables are local to this function and are not visible to any other function or anything else outside this function. Variables defined in different functions are different variables, they may even have the same name. This is the innermost scope level.

Module Scope

Modules in cTubes can exist at the Data Area, Layout (Data), and Tube level.

Everything declared in a lower module as public can be accessed from higher up.

A Public variable or function defined at the Tube level can be access from any cell, data area or layout in the Tube. Please see chapter 3, runtime context for details.

2.7 Data Types

cTubes Basic supports various built-in data types for script variables, object properties and fields in the database. Data can be converted manually or automatically from one type to another when appropriate. Except for objects, all data types are also database field types and can be stored as-is within a record.

Long Integer

Represents a signed numerical 32-bit value which is always integer and can be in the range of 2147483647 to -2147483648.

Keywords for declaring this data type: *long*, *integer*, *int*

Constant representation: +/- integer number

Examples: 1, 2, -1, -1243, +1243

Double Precision Float

Represents a 64-bit floating point value in the range of +/-1.7E308 with at least 15 digits of precision.

Keywords for declaring this data type: *double*, *float*

Constant representation: +/- integer portion dot ('.') fraction, optionally thousands can be grouped by a comma (',')

Examples: 3.1415, -1,234,567.89

String

Represents a sequence of characters. Valid characters are all of the UTF-16 character set. The length of the string is limited practically only by available hardware resources.

Keywords for declaring this data type: *string*

Constant representation: character sequence enclosed in either double quotation marks (") or in single quotation marks (').

Strings delimited with **double quotes** can not contain any escape sequences. Every character between the enclosing double quotes will be found unchanged in the resulting string, this includes return and newline.

A string delimited by **single quotation marks** may contain escape characters and sequences. Escape sequences start with a backslash character '\' and can be followed by one or more escape characters. Supported sequences are:

\n	... newline (LF), decimal code 10
\r	... carriage return (CR), decimal code 13
\t	... horizontal tab, decimal code 9
\\	... single backslash
\'	... single quote
\"	... double quote
\b	... backspace
\v	...vertical tab

Examples: "Hello World", 'Hello World\nNext Line...'

Future option, not yet implemented, TBD:

Additionally any character can occur in the string by using numerical escape codes:

\xhh	... a 2-digit hexadecimal character code in the range from 01 to FF
\uhhhh	... a 4-digit (UTF-16) hexadecimal character code in the range from 0001 to FFFF
\ddd	... a 1 to 3 digit decimal character code in the range from 1 to 255

Boolean

Represents the logical values *True* or *False*. When a boolean value is converted into a numerical data type *False* is converted to 0 and *True* to 1. When a numerical value is converted to a boolean value a value of 0 results in *False* and all other non-zero values result in *True*.

Note: The numerical value of *True* is different from standard BASIC where logical *True* results in a value of -1. This might lead to unexpected results when mixing boolean und other data types with binary operators like *And*, *Or* and *Not*.

Keywords for declaring this data type: *boolean*, *bool*

Constant representation: literal *True* or *False*

Date, Time

Represents a date/time value. The value has range of +/- 5.000.000 years with a resolution of 1 millisecond.

Keywords for declaring this data type: *date*

Constant representation: Date time value in the format *dd.MM.yyyy hh:mm:ss.xxx* enclosed in curly brackets '{' and '}'. The date time values need not be specified with full precision. It is possible to specify only the date or the time value. The Time value components can be omitted in the order from higher to lower precision.

Alternative representation: In most cases it is also possible to specify date/time values as strings (within quotation marks). The same rules apply, but in this case the constant type is a string and being converted to a date type value as appropriate.

FORMAT : dd.MM.yyyy

Examples: {13.09.2006 17:01}, {3.Sep.1968}

Binary

Represents a stream of data bytes. The length of the stream is limited practically only by hardware resources. In addition to the data bytes the binary value also can store the corresponding mime type.

Keywords for declaring this data type: *binary*

Constant representation: none

Unique ID

Represents the unique id of a database record or object.

Keywords for declaring this data type: *id*

Constant representation: Hexadecimal numerical value from 1 to 32 digits, enclosed in curly brackets '{' and '}'.

Object

Represents a reference to a scriptable object.

Keywords for declaring this data type: *object*

Constant representation: none

Any Type

Represents any data type i.e. the type of the value is unspecified and can be any. This type is used when declaring an untyped variable or parameter (see below).

Keywords for declaring this data type: *variant*, *any* or no type declaration at all

Constant representation: any of the above

Undefined

The type of variables, fields or values returned by functions and properties may be undefined. I.e. it has no specific value type assigned and is empty.

Keywords for declaring this data type: a variable defined with data type variant will have its type undefined until a value is assigned to it.

Constant representation: none

Variable states

Empty is a special kind of value. Variables are empty before a value is assigned to them. With undefined and empty three cases are possible:

Variable	IsUndef (type)	IsNull	IsEmpty (content of var)
Undefined Dim x	true	true	true
Defined base types	false	false	value depending
Defined Dim x as Tube	false	true	true
Defined Dim x as new Tubefalse depending	false	false	DefaultValue().IsEdefault value

When checking if a variable contains an object reference use IsNull.

2.8 Variables and Constants

A variable is a storage for values identified by a unique name within the script code. Generally variables can be read from or written to.

Variable Types

Variables can be typed or untyped. Data stored in a typed variable always has the type of the variable. If data with any specific type is assigned to a variable with different type the data is automatically converted to the type of the target variable. Untyped variables adopt the type of data being assigned to.

```
# Typed:
Dim x as long
Dim dt as date = "15.04.1994" # automatically converts string into date

# Untyped:
Dim var # empty/undefined value
Dim var2 as any

var = "15.04.1994" # var becomes a string
```

Variable Declaration

Variables are declared by using the keyword *DIM*. Declaring a variable allows to define a specific type for that variable. Additionally the variable can be initialized with a value. It is not mandatory to declare variables.

Not declared variables are initialized on their first use within the code. In this case the variable is always untyped and local to the current scope (see below).

Syntax for variable declaration:

```
<Public|Private> <Dim|Static> <Persistent> <WithEvents>
Name[<[<Dimension1><,<Dimension2><,<...>]>] <As Type = Initializer>
```

Dim	...	Default keyword to declare a non-static variable, this keyword must be specified if none of <i>Public</i> , <i>Private</i> or <i>Static</i> keywords is specified.
Static	...	Function scope: variable retains value among calls to the function where this variable is declared, otherwise the variable gets re-initialized on every function entry. Module scope: same as <i>Private</i>
Public	...	Only for module scope: variable is accessible also from within other objects and modules
Private	...	Only for module scope: variable is visible only within the current module.
Persistent	...	The value of the variable is retained also when cTubes gets restarted.
Withevents	...	Only for module scope: Used for object type variables to define that the object may generate events.
Name	...	Any valid identifier
Type	...	Refer to section <i>Data Types</i>
Initializer	...	Any expression used to define the value of the variable when it gets initialized. The expression does not need to be constant and may use function parameters or other variables in scope and defined previously.
Dimension	...	Size of dimension of array variable

Arrays

Arrays are variables storing more than one value at a time. To access values within an array indices are used. An index can be an integer value specifying the position of the value within the array. Alternatively the array can operate like an associative array allowing to access elements using a key string. Both ways of accessing the elements are possible with the same array variable in every dimension. Numerical indices and key-strings can be used interchangeably.

Numerical indices are always one-based i.e. the first index of an array is 1 and the last index value is the size of the dimension.

Arrays may have any number of dimensions, but the total number of elements must not exceed 100000. During declaration the specification of a dimension size may be omitted. In this case the maximum size of the specific dimensions undefined is increased on demand.

Arrays can be initialized using array initializer expressions. Such initializers are any expression resulting in an array value including a list of array elements enclosed in square brackets '[' and ']'.
Examples:

```
Dim myArray [10] as long           # one dimension with 10 elements
Dim myOther [3,5] as string        # two dimensions sized 3 and 5 elements
Dim AnotherArray [, , ] as long   # three dimensions with undefined size

myArray = [1,2,3,4,5]             # initialize it with a value list
myOther["myKey1","myKey2"] = "anyvalue" # use as assoc array
```

2.9 Constants

Constants are very similar to variables. In contrast to variables, constants are initialized only once with a constant value and can not be changed anywhere else in code. Therefore, constants always have to be declared and always have to have an initializer. The initializer has to be constant and must not include variables.

Examples:

```
Const PI as double = 3.1415
Const CRLF as string = Chr(13) & Chr(10)
```

Syntax for constant declaration:

```
<Public|Private> Const Name[<Dimension1><,Dimension2><,...>] <As Type> =  
Const-Initializer
```

2.10 Functions and Subroutines

A function is a block of statements declared somewhere in the script having a unique name by which it can be invoked from somewhere else. A defined number of parameters can be passed to a function when it is invoked. Functions can also return a resulting value. Subroutines can not return a value.

Declaring Functions and Subroutines

Syntax declaration of a function:

```
<Public|Private> <Static> Function(<ByVal|ByRef> <Argument1 <as Type> < =  
DefaultValue> ><, ...>) <As Type>  
...  
<Statements>  
...  
Exit Function  
...  
End Function
```

Syntax declaration of a subroutine:

```

<Public|Private> <Static> Sub(<ByVal|ByRef> <Argument1 <as Type> < =
DefaultValue> ><, ...>)
...
<Statements>
...
Exit Sub
...
End Sub

```

Static	...	Implicitly declares all variables at function scope as <i>Static</i> .
Public	...	This function may also be called from other objects and modules.
Private	...	This function is local to the current module.
ByVal	...	The argument is passed by value i.e. the original variable can not be modified.
ByRef	...	A reference to the caller's variable is passed i.e. the variable can be modified and a different value can be returned back.
Argument	...	Name of the argument passed into the function. The argument can be used like a normal local variable.
Type	...	Refer to section <i>Data Types</i>
Defaultvalue	...	Default value if the caller does not pass an argument when calling this function.

The function or subroutine can be exited at any time using the statements *Exit Function* or *Exit Sub* respectively.

The function declaration ends with *End Function* or *End Sub*.

Returning a value

Functions can return a value to the caller. To return a value, simply assign the value to a virtual local variable with the name of the function.

Example:

```

Function Get5() as Long
  Get5 = 5
  = 5          # abbreviated form, also returns the value
  result = 5  # alternative form, also returns the value
End Function

```

Only functions can return a value.

2.11 Classes

Classes allow creation of user defined objects. The order of class declarations is free. TBD.

```
Class cMyClass
Dim n as long
Dim u as long
Public Function GetNplusU() as Long
    = n +
End Function
Public Function GetNbyU() as Long
    = n * u
End Function
End Class

...
Dim mc as new cMyClass
mc.n = 3
mc.u = 4
v = mc.GetNplusU() # returns 7
```

2.12 Control Statements

If ... Then ... Elseif ... Else ... End If

The *If* construct executes different blocks of code depending on the result of conditional expressions.

Syntax declaration:

```
If conditional-expression Then
...
<statements>
...
ElseIf conditional-expression Then
...
<statements>
...
Else
...
<statements>
...
End If
```

Conditional ... Any expression resulting in a value. The resulting value is implicitly converted to

Expression a boolean type and checked for *True* or *False*.

The code after the *If* branch is executed only if the conditional expression after the *If* keyword results *True*. If it results in *False* the condition specified for the *Elseif* branch is evaluated. If that condition results *True* the block of statements following *Elseif* is executed. If none of the conditions are *True* the code in the *Else* branch gets executed.

There can be any number of *Elseif* branches or none at all in an *If* statement. If there are *Elseif* branches they have to appear before any *Else* branch. The *Else* branch is optional and can be omitted. If neither *Elseif* nor *Else* branches are specified, no code gets executed if the condition specified for *If* is *False*.

The *If*-construct is always terminated with *End If* (as an alternative *EndIf* is allowed as well).

For ... Next

This construct is a loop initializing a variable to a start value and increments the variable on every loop cycle until the end value is exceeded.

Syntax declaration:

```
For variable = startvalue to endvalue <step increment>
...
<statements>
...
Exit For
...
<statements>
...
Next variable
```

variable	...	Name of a variable which receives the current value
startvalue	...	Expression resulting in an integer value being used as start value for <i>variable</i>
endvalue	...	Expression resulting in an integer value being used as end value for <i>variable</i>
increment	...	Expression resulting in an integer value being used to increment the <i>variable</i> . If the start value is less than the end value, the increment should be negative. If omitted the default is 1.

Do Until, While ... Loop

Loops *Until* the conditional expression becomes *True*, or loops *While* the conditional expression is *True*.

Syntax declaration:

```
Do Until|While condition
...
<statements>
...
Exit Do
...
<statements>
...
Loop
```

condition ... Expression interpreted as *True* or *False*.

While ... Wend

Loops while the conditional expression is *True*.

Syntax declaration:

```
While condition
...
<statements>
...
Exit While
...
<statements>
...
Wend
```

condition ... Expression interpreted as *True* or *False*.

Goto ... Label

Continues the program execution at a specific position in code.

Syntax declaration:

```
...
Goto label
...
<statements>
...
label:
...
<statements>
```

label ... Any unique label identifier with the current function scope.

The target position is identified by a label. A label is a statement consisting of an identifier (name) immediately followed by a colon ‘:’.

The goto statement and the corresponding label have to be within the same function. It is not possible to jump outside a function.

2.13 Error Handling

By using the *OnError* construct, runtime errors can be caught and lead to execution of a specific statement.

Syntax declaration:

```
On Error Goto 0 | Resume Next | {<Exit | Terminate> <statement>}
...
errorlabel:
Resume Next
...
```

The statement contained in the *On Error* construct is not executed immediately when the *On Error* statement is executed. Instead, the construct acts as guard for specific sections of statements and executes the statement contained in *On Error* asynchronously whenever a runtime error occurs. The construct is effective from the position the *On Error* construct is encountered until the end of the function or until another *On Error* construct takes effect.

If execution diverges to a label using an *On Error Goto Label* statement, execution can be resumed after the position which caused the error by use of the *Resume Next* statement.

Usage:

On Error Goto 0	...	Disable the <i>On Error</i> guard.
On Error Resume Next	...	Ignore any runtime error.
On Error Exit <statement>	...	Exits the current function. Executes the optional statement before exiting.
On Error Terminate <statement>	...	Terminates further script processing and exits all functions up to the initial caller. Executes the optional statement before terminating.
Resume Next	...	Returns execution to the statement after the error.

Examples:

```
OnError Goto labelname      # jumps to label "labelname"

OnError Method()           # calls "Method" and continues
OnError f = 5*3            # assigns 15 to f and continues

OnError Resume Next       # ignore errors

OnError Exit               # exit the current function or sub
OnError Terminate         # terminate the script
OnError Exit Method()     # calls "Method" and exits the current script
```

To determine what kind of error occurred during runtime, a global function *LastError()* can be used to return the script global error object (refer to object *Error*).

2.14 Folder Path

A Folder Path is a string that defines a specified folder in the database structure.

The Path can contain one or more elements of folder names or id's, separated with a "/".

You can specify an absolute or a relative path.

An *absolute* path always begins with a slash "/", and begins at the tube data root folder.

Examples:

/Articles

/Articles

/Types/My

// database root

{id}/ starts any path

A *relative* path begins at the object at which the method or property is used.

Examples :

Articles

Articles/Types/My

Parent Access (supported only for single parent):

../ one level up (the parent folder).

../../Articles 2 levels up and folder path from there

3 Runtime Context

Depending on the context a special script function is running in, different objects are available as 'global objects' within this function.

When e.g. a cell event is called the event function is running within the cell context and therefore also has access to its context objects like the cell's field reference, Backcolor or Font.

Since a cell exists within a data area, which lives in a Layout (Data) which in turn lives in a Tube all these objects belong to the cell's runtime context.

Context objects have predefined names. Their public properties and functions can be accessed from higher levels to the lower.

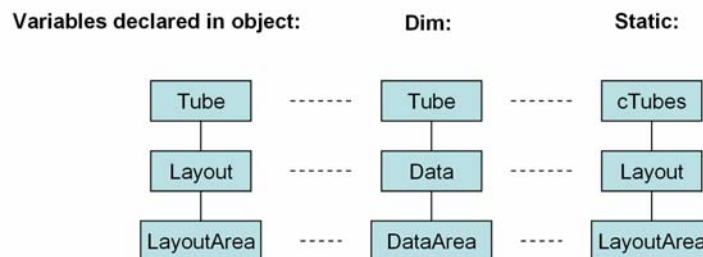
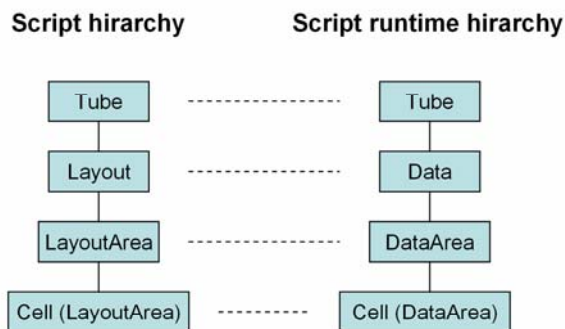
3.1 Object hierarchy

1. Cell
2. LayoutArea, Record, Field At runtime the resolved LayoutArea is called *DataArea*.
3. Layout (Data) At runtime the resolved Layout is called *Data*.
4. Tube
5. cTubes

Scripts are always running within the context of one of these objects.

The script can access context objects at the same and at lower levels.

cTubes script model



3.2 Cell context

TBD: section to be revised, picture of event flow to be added

The cell has 3 different value properties:

cell.Value Display value (String)

cell.FieldValue assign value to, and read value from referenced field

assign triggers OnInput, read comes from OnOutput

cell.UserValue value (as String) from and to Field via formatting

Assign leads to FormatIn to OnInput, read comes from OnOutput via FormatOut

You can use these to set values to other cells.

Note: FieldValue and UserValue only work in resolved state.

Note: if FALSE is returned in OnInput then the record is not updated. TBD

OnInput

Within the OnInput event the values are stored into the record with all update handling.

See examples below.

Examples:

```
function OnMouseClickedLeft
    FieldValue = 35
end function

function OnInput( Input )
    grp::OnInput( Input by ref )
    Field = Input * 4
end function
```

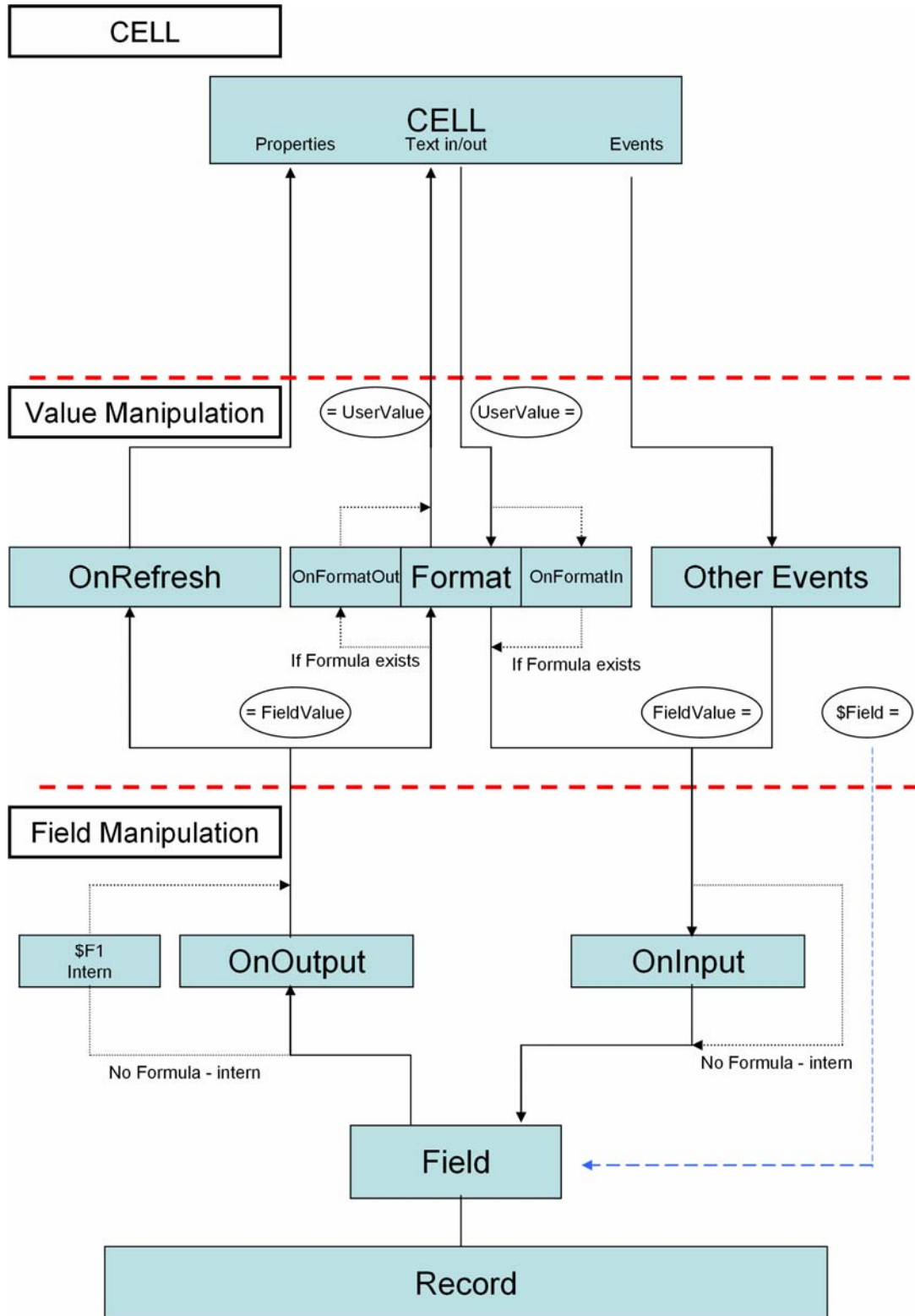
Assigning a value to a number of fields:

Important: this only works if the Cell does not have a Field reference! When there is a field reference in the Cell the other fields are shown as changed in a detail layout but not stored to the record!

```
function OnInput( Input )
    ra = tube.ExtractKML (input)
    $Latitude = ra[1]
    $Longitude = ra[2]
end function
```

3.3 Call Flow

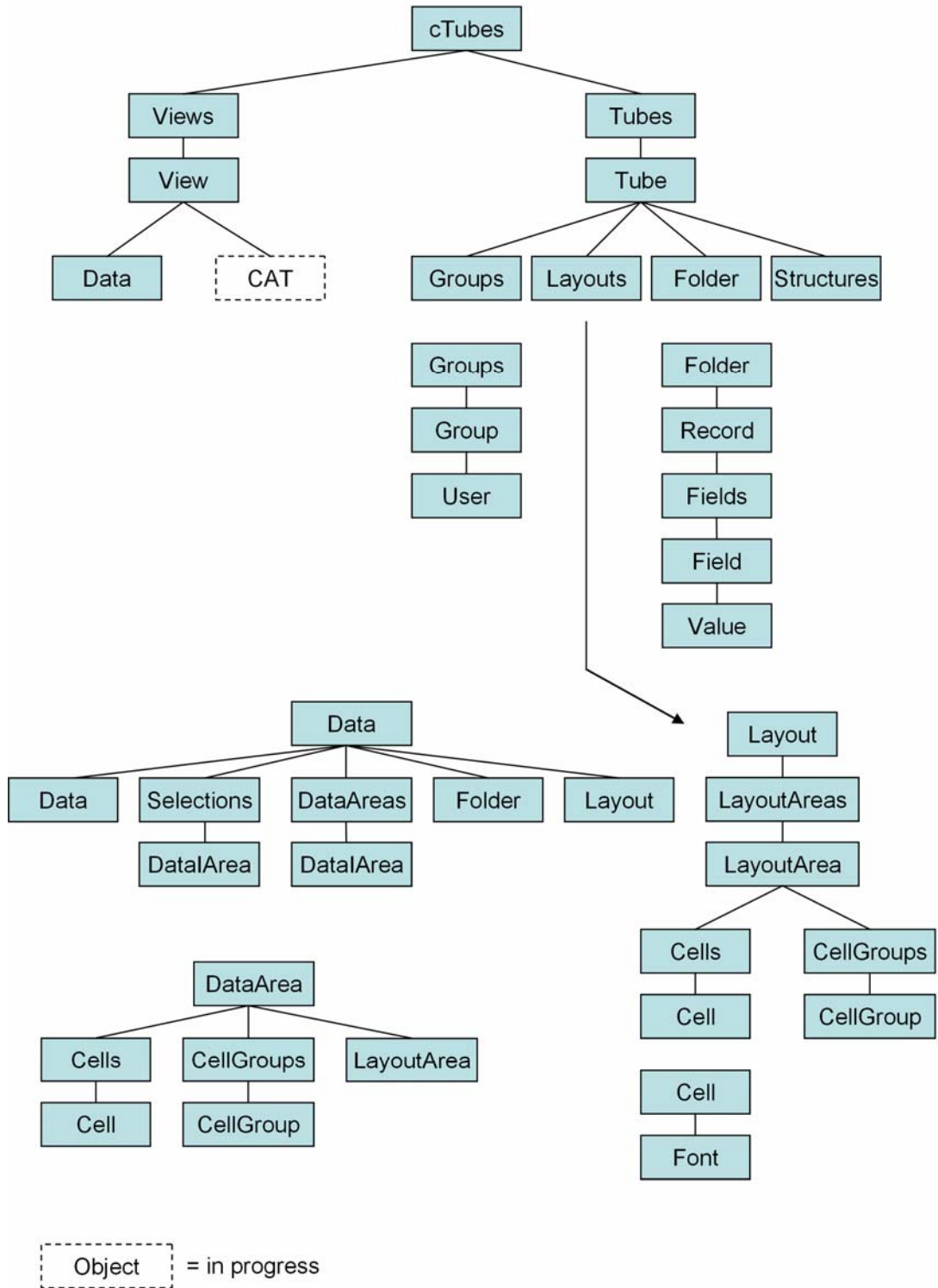
There are 2 Layers between the Cell on the User Interface and the Record: the Value and the Field Layers.



3.4 Object Hierachy

This is the overall object hierarchy within cTubes:

cTubes object model



4 Variables as Objects

Depending on their type, variables provide additional methods and properties.

4.1 String

IsEmpty Property

`String.IsEmpty` as `bool` R/O

Returns *true* when the string is empty.

Len Property

`String.Len` as `Integer` R/W

Returns the number of characters stored in the string. If a value is assigned to this property the string is set to a length as specified by the assigned value. When the string gets enlarged blank characters are inserted.

asUpper Property

`String.asUpper` as `String` R/O

Returns the string as uppercase characters.

asLower Property

`String.asLower` as `String` R/O

Returns the string as lowercase characters.

AsMultibyte Property

`String.AsMultibyte` as `binary` R/O

Returns the string as multibyte binary block.

AsUTF8 Property

`String.AsUTF8` as `binary` R/O

Returns the string as UTF8 binary block.

Empty Method

```
String.Empty()
```

Clears the string and makes it empty.

Mid Method

```
String.Mid( Startposition as Integer , <NumberOfCharacters as Integer > )  
as String
```

Returns a substring of the original string where the substring starts from the one-based index *Startposition* and has a maximum of *NumberOfCharacters*. When *NumberOfCharacters* is not specified, the returned substring includes all characters from the specified start to the end of the string.

InStr Method

```
String.InStr( SubString as String, <StartPosition as Integer> ) as Integer
```

Returns the zero-based position of *SubString* within the original string. If the original string does not contain *SubString* a value of 0 is returned.

InStrRev Method

```
String.InStrRev( SubString as String, <StartPosition as Integer> ) as  
Integer
```

Returns the zero-based position of *SubString* within the original string from behind. If the original string does not contain *SubString* a value of 0 is returned.

InStrCharSet Method

```
String.InStrCharSet( Chars as String, <StartPosition as Integer> ) as  
Integer
```

Returns the zero-based position of *the first found character* within the original string. If the original string does not contain *SubString* a value of 0 is returned.

InStrCharSetRev Method

```
String.InStrCharSetRev( Chars as String, <StartPosition as Integer> ) as  
Integer
```

Returns the zero-based position of *the first found character* within the original string from behind. If the original string does not contain *SubString* a value of 0 is returned.

CompareNoCase Method

```
String.CompareNoCase( String as String, <Length as Integer> ) as Bool
```

Returns true if the strings are equal – not case sensitive.

Replace Method

```
String.Replace( Search as String, Replace as String, <StartPosition as Integer>, <NumberOfReplaces as Integer> ) as Integer
```

Replaces all instances of *Search* by *Replace* within the given String.

If specified the search starts at *StartPosition* and *NumberOfReplaces* limits the number of replacements made.

Returns the count of replaces.

Left Method

```
String.Left( NumberOfCharacters as Integer ) as String
```

Returns a substring of the original string containing at maximum *NumberOfCharacters* characters starting at the first character of the string.

Right Method

```
String.Right( NumberOfCharacters as Integer ) as String
```

Returns a substring from the end of the original string containing at maximum *NumberOfCharacters*.

RegMatch Method

```
String.RegMatch( <RegularExpression as String>, <CaseLess as bool> ) as Array
```

Returns an array of expressions found in string.

To return a match the *RegularExpression* must contain at least one capturing block (parentheses).

If an error occurs or no matches are left over the result is *undef*.

If you have already set a Regular expression to the string variable before, you can keep the argument empty.

NOTE:

If you have named your capturing blocks, you can access your array with position AND named index.

Example of Regular Expressions:

```
b=s.regmatch( "(?P<R3>NNNx) | (?P<L3>xNNN) | (?P<R2>NNx) | (?P<L2>xNN)" )
```

This expression uses named matches. The resulting array stores the matches, they can be accessed with their names: R3, L3, etc.

RegMatchNext Method

```
String.RegMatchNext( <RegularExpression as String>, <CaseLess as bool> ) as  
Array
```

Same as RegMatch() method, but continues at the last found match.

RegMatchAll Method

```
String.RegMatchAll( <RegularExpression as String>, <CaseLess as bool> ) as  
Array [loopindex, captureindex]
```

Returns a two dimensional array of all matches found.

Internally it calls the RegMatchNext() method until there is no match left.

The first array index is the loop count, the second index the number of the capturegroup within the expression.

Load Method

```
String.Load( FilePath as String ) as bool
```

Loads a file into the string buffer.

The method returns *false* if the file can not be loaded.

Save Method

```
String.Save( FilePath as String, <Multibyte as Integer> ) as bool
```

Saves the string into the file specified by *FilePath*. If the file specified exists already it gets overwritten.

Set Multibyte to true if you want to save string in one byte format instead Unicode.

The method returns *false* if the file can not be saved.

Multibyte:

0 = Unicode

1 = Multibyte OEM

2 = Multibyte UTF8

4.2 Date

IsValid Property

```
Date.IsValid as bool R/O
```

Returns *true* if the date variable contains valid date information.

Day Property

Date.Day as Integer R/W

Returns the current day of the month in the range from 1 to 31. If the date is not valid this property returns an *undefined* value instead of a value with type *long*.

When a value is written to this property, the current day of the month of this date value is modified.

Errorhandling TBD

DayOfWeek Property

Date.DayOfWeek as Integer R/O

Returns the current day of the week in the range from 1 to 7. If the date is not valid this property returns an *undefined* value instead of a value with type *long*.

A week starts with *Monday*, i.e. a value with 1 represents a *Monday* whereas a value of 7 represents a *Sunday*.

Month Property

Date.Month as Integer R/W

Returns the current month of the year in the range from 1 to 12. If the date is not valid this property returns an *undefined* value instead of a value with type *long*.

When a value is written to this property, the current month of this date value is modified.

Errorhandling TBD

Year Property

Date.Year as Integer R/W

Returns the current year. If the date is not valid this property returns an *undefined* value instead of a value with type *long*.

When a value is written to this property, the current year of this date value is modified.

The year value can be in a range of -5.000.000 to 5.000.000 allowing to span a total range of 10 Million years.

Errorhandling TBD

MSecond Property

Date.MSecond as Integer R/W

Returns the current millisecond fraction of the date value. If the date is not valid this property returns an *undefined* value instead of a value with type *long*.

When a value is written to this property, the current date value is modified.

Errorhandling TBD

Sec Property

Date.Sec as Integer R/W

Returns the current second fraction of the date value. If the date is not valid this property returns and *undefined* value instead of a value with type *long*.

When a value is written to this property, the current date value is modified.

Errorhandling TBD

Min Property

Date.Min as Integer R/W

Returns the current minute fraction of the date value. If the date is not valid this property returns and *undefined* value instead of a value with type *long*.

When a value is written to this property, the current date value is modified.

Errorhandling TBD

Hour Property

Date.Hour as Integer R/W

Returns the current second fraction of the date value. If the date is not valid this property returns and *undefined* value instead of a value with type *long*.

When a value is written to this property, the current date value is modified.

Errorhandling TBD

WeekNumber Property

Date.WeekNumber as Integer R/O

Returns the number of the week within the current year. If the date is not valid this property returns and *undefined* value instead of a value with type *long*.

DaysInMonth Property

Date.DaysInMonth as Integer R/O

Returns the number of days for the entire month. If the date is not valid this property returns and *undefined* value instead of a value with type *long*.

DaysInYear Property

Date.DaysInYear as Integer R/O

Returns the number of days for the entire year. If the date is not valid this property returns and *undefined* value instead of a value with type *long*.

IsLeapYear Property

```
Date.IsLeapYear as bool R/O
```

Returns *true* if the current year is a leap year. If the date is not valid this property returns an *undefined* value instead of a value with type *long*.

IsUTC Property

```
Date.IsUTC as bool R/O
```

Returns *true* if the current date value is stored as UTC. In this case the date/time value is stored as universal coordinated time and converted to local time whenever the value is displayed. Such values may show different time values (i.e. the specific local time) on different systems in different time zones while maintaining the underlying stored value. This behavior is useful for timestamps when the exact time of an event considering the time zone is important.

In contrast, when the value is stored as local time (*IsUTC* returns *false*), the value is stored and displayed as-is and no automatic time-zone correction is done.

By default a date/time value is stored as local time and not UTC.

Format Function

```
Date.Format( Format as String ) as String
```

Returns a formatted String from Date declared in Format String.

Example: `str = Date.Format("dd.MM.yy")`

ToLocalTime Function

```
Date.ToLocalTime() as Date
```

Returns a new date object converted to local time.

ToUtcTime Function

```
Date.ToUtcTime() as Date
```

Returns a new date object converted to utc time.

DaysTo Function

```
Date.DaysTo( date as date ) as Integer
```

Returns the difference to the "date" in days.

SecsTo Function

```
Date.SecsTo( date as date ) as Integer
```

Returns the difference to the "date" in seconds. (same as minus operator)

4.3 Id

IsSystemId Property

```
Id.IsSystemId as bool R/O
```

Returns *true* if the id value is a predefined system id and not an automatically generated unique id.

IsEmpty Property

```
Id.IsEmpty as bool R/O
```

Returns *true* if the id value is empty and not set.

SysValue Property (Intern)

```
Id.SysValue as Integer R/W
```

Allows access to the base value of the system id. If the id is no system id a value of 0 is returned. When setting the value the id is automatically converted to a system id.

Empty Method

```
Id.Empty()
```

Resets the id value.

Hash Method

```
Id.Hash( NumberOfCharacters as Integer ) as String
```

Returns a string with a length of *NumberOfCharacters* consisting of a numerical hash value of the id value. The length must be in a range from 1 to 32.

Unique Method

```
Id.Unique()
```

Create and set a unique id to the object.

4.4 Binary

Byte Property

```
Binary.Byte[ Index as Integer ] as Integer R/W (default)
```

Allows accessing a specific byte in the buffer.

This property is the default property if the object is accessed with no property specified.

Errorhandling Index out of range TBD

IsEmpty Property

```
Binary.IsEmpty as bool R/O
```

Returns *true* if the buffer is empty.

MimeType Property

```
Binary.MimeType as string R/W
```

Allows accessing the mime type information of a data buffer. This property is used for special purposes like opening or displaying a data block from within a layout. When data is loaded from a file or dropped into the application, the mime information usually is automatically filled.

Name Property

```
Binary.Name as string R/W
```

Allows accessing a name associated with a data block. This property is automatically set to the filename when the data block is loaded from a file or dropped into the application.

Extension Property

```
Binary.Extension as string R/O
```

Returns only the extension of name.

Size Property

```
Binary.Size as Integer R/W
```

Returns the number of bytes stored in the binary data buffer. The data buffer can be resized by assigning a new value to this property.

Empty Method

```
Binary.Empty()
```

Clears the data buffer and sets the size to 0.

Load Method

```
Binary.Load( FilePath as String ) as bool
```

Loads a file into the binary buffer. The size of the buffer is set according to the file size. The maximum size of the file is limited to 4 MB. This function automatically tries to determine the mime type. If this is not possible the mime type is empty.

The method returns *false* if the file can not be loaded.

Save Method

```
Binary.Save( FilePath as String, <AddExtension as bool> ) as bool
```

Saves the binary data into the file specified by *FilePath*. If the file specified exists already it gets overwritten. The method returns *false* if the file can not be saved.

Set *AddExtension* to true and the binary adds the stored extension, if available.

4.5 Error

Text Property

```
Error.Text as String R/W (default)
```

Short description of error.

Code Property

```
Error.Code as Integer R/W
```

The error codes if type is a script error :

uqs_include.h	-	script engine codes
uqs_script_errorcodes.h	-	cTubes errorcodes

Type Property

```
Error.Type as Integer R/W
```

Usually 13, for script errors.

4.6 Array

Count Property

```
Array.Count as Integer R/W
```

Set or gets the size of array.

Name Property

```
Array.Name[Index as Integer] as String R/O
```

Get the name of an index (if any) otherwise returns undef.

Position Property

```
Array.Position[Name as String] as Integer R/O
```

Get the position of name index (if available) otherwise returns undef.
Also throws and Error! TBD

Find Method

```
Array.Find( Variable as Value ) as Integer
```

Get the position of found variable in array, returns undef if not found.

Add Method

```
Array.Add( Variable as Value ) as bool
```

Adds an entry in the array.

Insert Method

```
Array.Insert( Position as Integer, Variable as Value ) as bool
```

Insert an entry in the array.

Remove Method

```
Array.Remove( Index as Value ) as bool
```

Removes an entry from array.
Index can be a position or name.

Empty Method

```
Array.Empty() as bool
```

Clears the array.

4.7 Object (Variables)

IsUndef Property

```
Object.IsUndef as bool R/O
```

Returns *true* if the variable is not defined.

IsEmpty Property

`Object.IsEmpty` as bool R/O

Returns *true* if the variable is empty.

String: empty String.

Long, Float: 0

IsNull Property

`Object.IsNull` as bool R/O

Returns *true* if the variable is undefined or the variable does contain a valid object.

IsString, IsInteger, IsBool, IsFloat, IsID, IsDate, IsObject, IsNumber , IsArray, IsBinary Properties

`Object.IsString` as bool R/O

Returns *true* if the variable is the specified type.

asString, asInteger, asBool, asFloat, asNumber, asDate, asID, asBinary Property

<code>Object.asString</code> as String	R/O
<code>Object.asInteger</code> as Integer	R/O
<code>Object.asBool</code> as Bool	R/O
<code>Object.asFloat</code> as Float	R/O
<code>Object.asNumber</code> as Float or Integer	R/O
<code>Object.asDate</code> as Date	R/O
<code>Object.asID</code> as ID	R/O
<code>Object.asBinary</code> as Binary	R/O

Returns *the converted type of variable*.

5 Application Objects

5.1 cTubes

The *cTubes* object is the application's root object allowing access to all user interface objects used within cTubes.

Tubes Property

`cTubes.Tubes` as Tubes R/O

Returns a reference to the application's tubes collection.

Views Property

`cTubes.Views` as Views R/O

Returns a reference to the application's views collection.

Language Property

`cTubes.Language` as String R/W

Returns or set the current language of cTubes in a 2 character format.

Available Language constants:

EN	- English
DE	- German
FR	- French
ES	- Spanish
CS	- Czech
HU	- Hungarian
RU	- Russian
SK	- Slovakian
PL	- Polish

TraceLevel Property

`cTubes.TraceLevel` as Integer R/W

Set or get the current trace level. The trace level is a combination of bits:

Net Property

`cTubes.Net` as Net R/O

Returns the net object for network settings.
Only available if license is “network” or above.

License Property

`cTubes.License` as License R/O

Returns the license object for license settings.

ExePath Property

`cTubes.ExePath` as String R/O

Gets the file path of the running cTubes exe.

ExeName Property

`cTubes.ExeName` as String R/O

Gets the name of the running cTubes exe.

MyTubesPath Property

`cTubes.MyTubesPath` as String R/W

Gets the folder path of the stored user tubes.
Setting a MyTubesPath will be stored into settings database, set empty string for standard path.

TempPath Property

`cTubes.TempPath` as String R/O

Gets the standard windows temp path.

5.2 Net

Network in cTubes.

PublicIP Property

`Net.PublicIP` as String R/W

Get or set the server internet ip. (optional)
Clients will connect to this ip.

PublicPort Property

Net.PublicPort as String R/W

Get or set the server internet port. (optional)

Clients will connect to this port.

ServerPort Property

Net.PublicPort as String R/W

Local port of server on the computer.

Status Property

Net.Status as Integer R/O

Stopped = 0,
Idle,
GetShare,
PushRecord,
OpenPullQuery,
PullRecord,
Cleanup,
ShareQuery,
Sync,
OpenSession,
Next,
Connect,
Connecting,
Exit,
Check,
Done,
SyncNodes,
CheckPull,
CheckPush,
Init,
Error,
Success,
Break,
CheckRights,

Start Method

Net.Start()

Start the network server.

Stop Method

Net.Stop()

Stop the network server.

CheckUserPassword Method

```
Net.CheckUserPassword( User as String, Password as String ) as bool
```

Checks the password of given user.

SetUserPassword Method

```
Net.SetUserPassword( User as String, OldPassword as String, NewPassword as String ) as bool
```

Sets a new password to given user.

5.3 License

Handle the license of application with this object.

Type Property

```
License.Type as Integer R/O
```

- 1 – Test Version
- 2 – Single User
- 3 – Multi User

Module Property

```
License.Module as Integer R/O
```

- 1 – Reader
- 2 – Personal
- 3 – Network
- 4 – Network Pro
- 5 – Designer
- 8 – Server

DaysExpire Property

```
License.DaysExpire as Integer R/O
```

Says the number of days the application will expire.
If the app not expires, the return value will be smaller than null.

User Property

```
License.User as User R/W
```

Get or sets the local user.
Does not work with .Groups! TBD

UserName Property

```
License.UserName as String R/W
```

The Nick Name of current user.
It's recommended to use the User property.

UserID Property

```
License.UserID as ID R/O
```

The User ID of current user.
It's recommended to use the User property.

KeyUser Property

```
License.KeyUser as String R/O
```

Returns the user's current key.
Password of license key.

SetKey Method

```
License.SetKey( Key as String, <User as String> ) as Bool
```

Sets a license key.
If the key needs a user name, set the second parameter too.

SetKeyOnce Method

```
License.SetKeyOnce( Key as String, <User as String> ) as Bool
```

Same as SetKey() Method, but if key already set, the call will be ignored.

CheckKey Method

```
License.CheckKey( Key as String, <User as String> ) as Bool
```

Returns true, if given key is valid.

IsActualKey Method

```
License.IsActualKey( Key as String ) as Bool
```

Returns true, if given key is the actual key of application.

Check if current key.

CheckUserPassword Method

```
Net.CheckUserPassword( User as String, Password as String ) as bool
```

Checks the password of given user.

SetUserPassword Method

```
Net.SetUserPassword( User as String, OldPassword as String, NewPassword as String ) as bool
```

Sets a new password to given user.

EncodePassword Method

```
Net.EncodePassword( Password as String, <ReferenceEncode as String> ) as String
```

Encode the given password to a string, containing 13 ascii signs.

5.4 Tubes

The tubes object is a collection containing all tubes currently being open.

Item Property

```
Tubes.Item[ Index as Value ] as Tube R/O (default)
```

Returns a reference to the tube specified by *Index*. The index argument can be a zero-based numerical index into the list of Tubes, a string specifying the name of a tube, or the unique id of the tube.

If the index is not valid an *undefined* value is returned instead of a tube object.

If no index is specified the property returns the tube of the view currently being active.

Count Property

```
Tubes.Count as Integer R/O
```

Returns the total number of tubes within the collection.

Settings Property

```
Tubes.Settings as Tube R/O
```

Returns the settings tube. The settings tube is a special tube containing configuration data and application settings which are local to the current workstation.

Open Method

```
Tubes.Open( FilePath as String ) as Tube
```

Opens the specified tube and shows a view for this tube if enabled.

The Method returns the opened tube or undef.

OpenDirectory Method

```
Tubes.OpenDirectory( FolderPath as String ) as bool
```

Opens all cTubes in the directory.

The Method returns true if ok.

Note: Only available for cTServer

Remove Method

```
Tubes.Remove( Index as Value ) as bool
```

Closes the specified tube and removes all open views for this tube. If the *Index* argument is not value the method returns *false* and performs no action.

Clear Method

```
Tubes.Clear()
```

Closes all open views and tubes.

5.5 Views

The views object is a collection containing all views currently being open.

Item Property

```
Views.Item[ Index as Value ] as View R/O (default)
```

Returns a reference to the view specified by *Index*. The index argument can be a zero-based numerical index into the list of Views or a string specifying the name of a view.

If the index is not valid an *undefined* value is returned instead of a tube object.

If no index is specified the property returns the view currently being active.

Count Property

```
Views.Count as Integer R/O
```

Returns the total number of views within the collection.

OpenAutoView Method

```
Views.OpenAutoView( Folder as Folder, <Layout as Layout>, <Tube as Tube> )  
as bool
```

Opens an auto view.

Remove Method

```
Views.Remove( Index as Value ) as bool
```

Closes the specified view. If the *Index* argument is not value the method returns *false* and performs no action.

Clear Method

```
Views.Clear
```

Closes all open views.

5.6 Tube

Id Property

```
Tube.Id as Id R/O (default)
```

Returns the unique tube id.

ShareId Property

```
Tube.ShareId as Id R/O
```

Returns the share id of the tube.

FilePath Property

```
Tube.FilePath as String R/O
```

Returns the full path of the current tube file.

Name Property

```
Tube.Name as String R/O
```

Returns the name of the current tube.

Folder Property

```
Tube.Folder as Folder R/O
```

Returns the user data root folder.

SysFolder Property

```
Tube.SysFolder[ Index as Value ] as Folder R/O
```

Returns the specified system folder.

Keywords are:

Tubes, tube, temp, language

Structures Property

```
Tube.Structures as Structures R/O
```

Returns the structures list object.

SyncActive Property

```
Tube.SyncActive as bool R/W
```

Returns or sets the synchronization active status of the tube.

Layouts Property

```
Tube.Layouts as Layouts R/O
```

Returns the layouts collection object.

DefaultLayout Property

```
Tube.DefaultLayout as Id R/W
```

Returns or sets the tube's DefaultLayout.

Note: The default layout opens when there has not been a previous view in the settings file or there is no settings file yet, at first start.

Groups Property

```
Tube.Groups as Groups R/O
```

Returns the groups collection object.

DefaultRight Property

```
Tube.DefaultRight as Integer R/O
```

Return the actual default rights setting of the tube.

- 0 – Standard Rights
- 1 – Minimum Rights
- 2 – Maximum Rights

Folder Method

```
Tube.Folder( FolderPath as String, <CreateFolders as bool> ) as Folder
```

Returns the folder of given path, starting at user data root.

Set CreateFolder to true, to create missing folders at path.

Keywords:

- \$tube – tube folder
- \$tubes – tubes folder
- \$temp – temp folder

Rename Method

```
Tube.Rename( Name as String ) as boolean
```

Renames the current tube. The tube name and the file name are not necessarily exactly the same. When a tube gets renamed, the filename is changed to a name close to *Name* but adhering to the file system's naming convention.

On success, the method returns *true*.

CheckRights Method

```
Tube.CheckRights( Object as Object, Rights as Integer ) as bool
```

Returns true if the specified rights set on the object.

This function supports an array for the Rights argument.

Note and Example:

Normally input cells in Layouts are automatically set to ReadOnly if the user has no change rights to the record. You can overrule this behavior and set ReadOnly to FALSE to enable writing into the cell to take input by script (writing to the record is inhibited nevertheless).

This describes the standard ReadOnly behaviour if it were implemented in Script:

```
ReadOnly = !tube.CheckRights( record, 1 )
```

```

RIGHT_NONE          = -1,
  RIGHT_UNDEF       = -1,

  // record rights
RIGHT_READ          = 0,  // read current record
RIGHT_UPDATE        = 1,  // write/update current record
RIGHT_REMOVE        = 2,  // remove current record
RIGHT_RIGHTS        = 3,  // set rights of current record
RIGHT_ADD           = 4,  // add current record to any other folder

  // folder rights
RIGHT_QUERY         = 5,  // browse/query childs
RIGHT_ADD_CHILDS    = 6,  // add/insert childs
RIGHT_REMOVE_CHILDS = 7,  // remove childs from current parent
RIGHT_ADD_FOLDERS   = 8,  // add/insert childs folders

RIGHT_EXPORT        = 10,
RIGHT_PRINT         = 11,

Application Rights # (object = 0b folder)
  Layout Selector = 56      (2 = OFF, 3 = *)
  Layout Editor   = 57      (2 = OFF, 3 = *)
  Import          = 62      (2 = OFF, 3 = *)
  Export          = 63      (2 = OFF, 3 = *)

Tree      *      ON      OFF
  58      2      3      2
  59      3      3      2

Change App Rights = 1
Create Groups     = 6

```

SetRights Method

```

Tube.SetRights( Object as Object, Group as ID, Rights as Integer, SetType
as Value, <ToObject as bool>, <ToChilds as bool> ) as bool

```

Set the right on the object with specified group.

SetType can be true, false or undef. (this is the status to be set)

Returns true if no failure occurs.

This function supports an array for the Rights argument.

Records need to be saved before setting the rights and to be updated afterwards.

Example, setting the rights on a new record:

```
dim ro as record
  dup r = $Structure:sMyStruct
  data.folder.insert(r, ro) # save the record and get it back in ro
  SetRights( ro, TUGid , 1, FALSE, TRUE, TRUE )
  data.folder.update(ro)
```

GetRights Method

```
Tube.GetRights( Object as Object, Group as ID, Rights as Integer,
<FromChilds as bool> ) as Integer
```

Returns a number defining the status of Right.

If FromChilds is undefined or false you get the object rights.

This function supports an array for the Rights argument.

undef – more than one rights were taken for query and no unique match was possible.

0 – Unset Inherited

1 – Set Inherited

2 – Unset

3 – Set

4 – Unset Locked from application

5 – Set Locked from application

GetScriptRights Method

```
Tube.GetScriptRights( Number as Integer, <AllMask as Integer> ) as Variable
```

Returns true, false or undef state of given script right number.

True – right is defined granted

False – right is defined not granted

Undef – not defined, will ask user what to do

2 – Only in Multiple mode: results are different.

For multiple results, set Number to -1 and set AllMask the bits you would like to get.

ResetScriptRights Method

```
Tube.ResetScriptRights( Number as Integer ) as Bool
```

Set the actual number to undef. (for set states see GetScriptRights())

Take number -1 to set all numbers.

GetSyncState Method

```
Tube.GetSyncState( <ByRef Date as Date>, <User/Group as Value>, <Share as Value> ) as bool
```

Returns TRUE if tube was replicated.

SetStory Method

```
Tube.SetStory( Object as Object, <MaxDepth as Integer>, <MaxTime as Integer> ) as bool
```

Set the story settings in the object.

Optionally set the Max Depth (or undef) and the time in seconds (or undef).

RemoveStory Method

```
Tube.RemoveStory( Object as Object ) as bool
```

Remove the story settings from object.

The settings will be inherited from parents now. (standard)

GetStory Method

```
Tube.GetStory( Object as Object ) as Array
```

Array[1] = Depth as Integer - undef means infinite

Array[2] = Time as Integer - undef means infinite

Array[3] = Inherit as Bool

InvalidateIndex Sub

```
Tube.InvalidateIndex( field as value )
```

See also Data.InvalidateIndex TBD

Close Method

```
Tube.Close()
```

Closes the current tube and all corresponding views.

After calling this method, the current object reference becomes invalid.

5.7 Structures

The structures object is a collection containing all structures available.

Item Property

```
Structures.Item[ Index as Value ] as Record R/O (default)
```

Returns a copy of the structure.

Count Property

```
Structures.Count as Integer R/W
```

Returns the total number of structures available.

DefaultStructure Property

```
Structures.DefaultStructure as Record R/W
```

Get or set the actual default structure.

Add Method

```
Structures.Add( Value as Value, <Name> ) as Id[ ]
```

Adds n numbers of empty structures if an integer is given,
Or the structure if value is a record object.
Returns the added structures in an id array. (call Item property to get record)

Remove Method

```
Structures.Remove( Index as Value ) as bool
```

Removes the structure from tube.

Update Method

```
Structures.Update( Structure as Record, <Name> ) as bool
```

Updates the given structure in the tube.

5.8 View

Tube Property

```
View.Tube as Tube R/O
```

Returns the tube object the current view belongs to.

Name Property

```
View.Name as String R/W
```

Returns the name of the current view. When a new view is opened the default name of the view is set to the current name of the tube.

The default name can be changed by assigning a different name to this property.

Id Property

```
View.Id as ID R/O
```

Returns the id of the current view.

Active Property

```
View.Active as boolean R/W
```

Returns the current active state of a view. Only one view can be active at a time. When one view becomes active all other views become inactive.

If the value *true* is assigned to this property the current view becomes the active one. If *false* is assigned the current view becomes inactive and previously active view becomes active again. If there is no other view the current view stays active.

Data Property

```
View.Data as Data R/O
```

Returns the data object of the view.

Close Method

```
View.Close()
```

Closes the current view.

After calling this method, the current object reference becomes invalid.

5.9 Catbrowser

Note: (Version 2.2.6) The catbrowser may not be accessed in Print Preview. Workaround: check the name of the current view not being "PRINT" before accessing the catbrowser object.

IsOpen Property

```
Catbrowser.IsOpen as bool R/O (default)
```

Returns the open/closed status of catbrowser.

Close Method

```
Catbrowser.Close()
```

Closes the view's catbrowser.

5.10 Data

Id Property

```
Data.Id as Id R/O (default)
```

Returns the unique id of the resolver object.

Height Property

```
Data.Height as Value R/W
```

Get or sets the height of data.

Width Property

```
Data.Width as Value R/W
```

Get or sets the height of data.

Child Property

```
Data.Child as Data R/O
```

Returns the child data if available.

Layout Property

```
Data.Layout as Layout R/W
```

Returns the currently loaded layout object. A new layout can be loaded by assigning a layout object, a layout id or the name of the layout to load.

If the new layout has a structure specified or a folder specified, cTubes sets the current folder or structure according to the values specified within the layout. Otherwise the current folder and structure are kept. The layout is loaded immediately (synchronously).

If any error occurs a run-time error is generated and the global error variable contains the corresponding error code.

Effect on other modes (Page view) TBD

Folder Property

Data.Folder as Folder R/W

Returns the currently displayed folder object. A new folder can be displayed by assigning a folder object, a folder id or a name (or path string).

Changing the folder has no effect to the layout or structure currently being used for display.

If any error occurs a run-time error is generated and the global error variable contains the corresponding error code.

Structure Property

Data.Structure as Value R/W

Returns the the structure(record) currently being used. A new structure can be used by assigning an unique id, a structure name or an record. The property can be cleared by assigning an undefined value, an empty id or empty string.

The structure is used for filtering the correct records within the current folder and displaying the proper fields. By default if no structure is specified no structure filtering will be performed.

Changing the folder has no effect to the layout or structure currently being used for display.

If any error occurs a run-time error is generated and the global error variable contains the corresponding error code.

Type Property

Data.Type as Integer R/W

Returns a value indicating how the data is displayed. Usually this value is defined in the layout. Assigning a value to this property overrides the type defined by the layout.

Possible values are:

- 0 ... List of child records contained in the folder
- 1 ... List of story records of the current folder
- 2 ... One page form showing detail data of the current folder
- 3 ... Multiple one-page forms showing detail data of child records contained in the folder

Selections Property

Data.Selections as Selections R/O

Returns the selections object.

InitialFolder Property

Data.InitialFolder as Folder R/O

Returns the first folder the resolver was initially started with.

Concepts TBD

Areas Property

```
Data.Areas as DataAreas R/O
```

Returns the collection of items currently displayed.

Cell Property

```
Data.Cell as Cell R/O
```

Returns the main cell of data.
(inside of scroll frame)

View Property

```
Data.View as View R/O
```

Returns the parent view of data, if available.

LoadLayout Method

```
Resolver.LoadLayout( Layout as Layout ) as boolean
```

Loads a new layout and allowing to additionally specify the folder, structure and load options. The method returns *true* on success.

Print Method

```
Data.Print( Setup/Pages as String, <Layout as value>, <Printer as String> )  
as boolean
```

Refresh Sub

```
Data.Refresh( <record> )
```

Refresh the complete data or a single item.

InvalidateIndex Sub

```
Data.InvalidateIndex( field as value )
```

Invalidates the Index for the specified field in the current view.
This is required when a computed field takes changing input that is not coming from its own database records. An invalidated index is automatically recreated again at next use, i.e. a sort is correct according to the changed inputs.

Examples: computations involving a changing time value such as Now(), calculations for a distance to externally defined values, if those input values change the Index has to be invalidated

OnInitListbox ()

```
Data.OnInitListbox( ParentData as Data, ParentArea as Area, ParentCell as Cell )
```

Parent Properties for Listbox.

5.11 Selections

General description TBD

Item Property

```
Selections.Item[ Index as Value ] as DataArea R/O (default)
```

Allows accessing the selection item.

Count Property

```
Selections.Count as Integer R/O
```

Returns the number of selections.

SelectItem Method

```
Selections.SelectItem( Object as Value )
```

Selects an area. Integer argument steps to position(+/-).

Remove Method

```
Selections.Remove( Index as Value )
```

Clear Method

```
Selections.Clear()
```

5.12 DataAreas

Item Property

```
DataAreas.Item[ Index as Integer ] as DataArea R/O (default)
```

Returns the data item specified by the zero-based *Index*.

Count Property

```
DataAreas.Count as Integer R/O
```

Returns the number of items currently displayed in the view.

5.13 DataArea

Note: As the other User Interface Objects, *DataArea* can accessed from the On_Refresh event, but not from the On_Output Event.

Cells Property

```
DataArea.Cells as Cells R/O (default)
```

Returns the collection of cells used to display the data items.

LayoutArea Property

```
DataArea.LayoutArea as LayoutArea R/O
```

Returns the layout area object as defined in the layout. The layout area is used as a template to display the data items.

CellGroups Property

```
DataArea.CellGroups as CellGroups R/O
```

Returns the layout CellGroups object.

Record Property

```
DataArea.Record as Record R/O
```

Returns the record of area.

RecordChanged Method

```
DataArea.RecordChanged() as void
```

Mark area as changed and refresh it.

Call this function after the record of the area has changed directly by script.
It should be done automatically in the future.

Reload Method

```
DataArea.Reload() as void
```

Reload record data of area.

5.14 Cells

Cells Property

```
Cells.Item[ Index as Value ] as Cell R/O (default)  
Cells.Item[ Row as Integer, Column as Integer ] as Cell R/O (default)
```

Note: The Order with Version 2.2.2. is: Column , Row!

Returns the cell object as specified by the *Index*, or *Row* and *Column* arguments. The index can be a numerical one-based index, a unique id or a string in the format "A1" or "R1C1". Additionally the cell can be specified using dedicated zero-based *Row* and *Column* arguments.

If no or an undefined index argument is specified the property returns the root cell of the data item.

The root cell is the container for all other cells for this data item and covers the entire display area of the item.

Count Property

```
Cells.Count as Integer R/O
```

Returns the number of cells in the collection.

CreateGroup Method

```
Cells.CreateGroup( <CellFrom as cell>, <CellTo as cell> ) as CellGroup
```

Returns a new CellGroup of the given cells.

To get all cells in a group, call the method with no arguments.

5.15 Layout

Id Property

```
Layout.Id as Id R/O (default)
```

Returns the unique id of the layout object.

Name Property

```
Layout.Name as String R/W
```

Allows access to the name of the layout. The name can be read or set. object.

ReadOnly Property

```
Layout.ReadOnly as bool R/O
```

Returns true if layout is write protected.

LayoutAreas Property

```
Layout.LayoutAreas as LayoutAreas R/O
```

Returns the *LayoutAreas* collection object.

5.16 LayoutAreas

Item Property

```
LayoutAreas.Item[ Index as Value ] as LayoutArea R/O (default)
```

Allows accessing a *LayoutArea* by a zero-based numerical index or by unique id..

Count Property

```
LayoutAreas.Count as Integer R/O
```

Returns the number of areas in the layout.

Remove Method

```
LayoutAreas.Remove( Index as Value )
```

Removes the layout area specified by *Index*. The *Index* argument can be a numerical index or a unique id.

Clear Method

```
LayoutAreas.Clear
```

Removes all areas from the layout.

5.17 LAYOUTAREA

Item Property

```
LayoutArea.ID as ID R/O (default)
```

Returns the unique id of the area.

CellGroups Property

```
LayoutArea.CellGroups as CellGroups R/O
```

Returns the collection of CellGroups.

5.18 CELLGROUPS

Item Property

```
CellGroups.Item[ Index as Value ] as Value R/O (default)
```

Allows accessing a LayoutArea by a zero-based numerical index or by unique id..

Count Property

```
CellGroups.Count as Integer R/O
```

Returns the number of groups in the area.

Remove Method

```
CellGroups.Remove( Index as Value )
```

Removes the cell group specified by *Index*. The *Index* argument can be a numerical index or a unique id.

Clear Method

```
CellGroups.Clear()
```

Removes all groups from the area.

5.19 CELLGROUP

Item Property

```
CellGroup.Item[ Index as Value ] as Cell R/O (default)
```

Allows accessing a LayoutArea by a zero-based numerical index or by unique id..

Count Property

CellGroup.Count as Integer R/O

Returns the number of cells in the group.

ID Property

CellGroup.Id as ID R/O

Returns the ID of the group.

Name Property

CellGroup.Name as String R/W

Allows access to the name of the group. The name can be read or set.

plus all properties from Cell

See 5.19 Cell

5.20 CELL

Value Property

Cell.Value as String R/W (Default)

Gets or set the text shown in the cell.

FieldValue Property

Cell.FieldValue as Value R/W

Gets or set the value to or from the Input/Output of the cell.

UserValue Property

Cell.UserValue as String R/W

Gets or set the value to or from the Input/Output of the cell, through the cell formatting.

BackColor Property

```
Cell.BackColor as Integer R/W
```

Gets or set the back color of the cell.
Use the RGB functions to convert colors.

ClassName Property

```
Cell.ClassName as String R/W
```

Gets or set the class of the cell.

ClassFlags Property

```
Cell.ClassFlags as Integer/String R/W
```

Gets or set the class flags of the cell.
Optional use string keywords:
Active on, active off, disable on, disable off, error on, error off

Font Property

```
Cell.Font as Font R/W
```

Gets or set the font of the cell.

ReadOnly Property

```
Cell.ReadOnly as bool R/W
```

Gets or set the Read Only State of the cell.
If enabled not input is available.

TabStop Property

```
Cell.TabStop as Integer R/W
```

Gets or set the Tab Stop of the cell.
0 = Disabled, 1 and higher enabled and numbered.

EditLock Property

```
Cell.EditLock as bool R/W
```

Gets or set the EditLock State of the cell.
If enabled, no key input is available.

Active Property

Cell.Active as bool R/W

Gets or set the Active State of the cell.
If enabled, cell do not response anymore.

Visible Property

Cell.Visible as bool R/W

Gets or set the Visible State of the cell.

ColumnIndex Property

Cell.ColumnIndex as Integer R/O

Gets the column index of the cell in the area.

RowIndex Property

Cell.RowIndex as Integer R/O

Gets the row index of the cell in the area.

Width Property

Cell.Width as Integer/String R/W

Gets or set the width of the cell.
Use size as integer(twips) or as string – like “5 cm”

Height Property

Cell.Height as Integer/String R/W

Gets or set the height of the cell.
Use size as integer(twips) or as string – like “5 cm”

ID Property

Cell.ID as ID R/O

Gets the unique id of the cell.

ConfigValue Property

Cell.ConfigValue[Index as Integer] as Value R/O

Gets the configuration value stored from an element.

CreatePopup Method

```
Cell.CreatePopup( folder as folder, Layout as layout, <DataType as Integer>, <Structure as Record>, <Tube as Tube>, ... ) as bool
```

Creates a popup on the cell,
the clicked value of popup will be returned to the FieldValue property of this cell.

CreateChild Method

```
Cell.CreateChild( InputType as Integer, <OutputType as Integer> ) as bool
```

Creates a special functionality(code), specified with the types, on the cell.
Internal use, only.

5.21 FONT (CELL)

Name Property

```
Font.Name as String R/W (default)
```

Sets or get the name of font.

Size Property

```
Font.Size as Integer R/W
```

Sets or get the size of font.

Color Property

```
Font.Color as Integer R/W
```

Sets or get the color of font.
Use the RGB functions to convert colors.

Style Property

```
Font.Style as Integer/String R/W
```

Sets or get the style of font.
Optional you can use a combination of text constants separated with blank:
Normal, bold, italic, underline

Alignment Property

```
Font.Alignment as Integer/String R/W
```

Sets or get the alignment of font.

Optional you can use a combination of text constants separated with blank:

Center, left, right, top, bottom

5.22 DRAG/DROP

```
UQDO_TYPE_TEXT = 1,  
UQDO_TYPE_FILE = 2,  
  
UQDO_TYPE_IDS = 3,  
UQDO_TYPE_RECORDID = UQDO_TYPE_IDS = 3,  
UQDO_TYPE_LAYOUTID = 4,  
UQDO_TYPE_FIELDID,  
UQDO_TYPE_STRUCTUREID,  
UQDO_TYPE_VALUEFIELDID,  
UQDO_TYPE_LAYOUTAREAID,  
UQDO_TYPE_CELLELEMENTSID,  
UQDO_TYPE_TUBEID,  
UQDO_TYPE_IDS_MAX,  
  
UQDO_TYPE_DATABASE,  
UQDO_TYPE_PRIORITY,  
UQDO_TYPE_UQVALUE = 14,  
UQDO_TYPE_FLAGS,  
UQDO_TYPE_FLAGID,  
UQDO_TYPE_FOLDERID,  
UQDO_TYPE_ACCESSFLAGS,  
UQDO_TYPE_INSTANCEID,  
UQDO_TYPE_BITMAP,  
  
UQDO_TYPE_SCRIPT,
```

PrimaryType Property

```
DragDrop.PrimaryType as Integer R/W (default)
```

Sets or get the primary type of object.

Item Property

```
DragDrop.Item[ Index as Value, <Type as Integer> ] as Value R/W (default)
```

Sets or get the values of object.

Usually leave the type prop blank or use the return of property PrimaryType as Type here.

Count Property

```
DragDrop.Count[<Type as Integer>] as Integer R/O
```

Returns the count of type specified with Type.

Remove Method

```
DragDrop.Remove( Index as Value, <Type as Integer> )
```

Removes the objects specified by *Index*. The *Index* argument can be a numerical index or a unique id.

Clear Method

```
DragDrop.Clear( )
```

Removes all objects from the area.

LoadFromClipboard Method

```
DragDrop.LoadFromClipboard( )
```

Load the properties from clipboard.

SaveToClipboard Method

```
DragDrop.SaveToClipboard( )
```

Save the properties to clipboard.

5.23 DIALOG BOX

OnOpenDialog()

OnCloseDialog(returnvalue)

returnvalue 0 TRUE/FALSE

Title Property

```
DialogBox.Title as String R/W
```

Sets or get the title of the dialog box.

Folder Property

```
DialogBox.Folder as Folder R/W
```

Sets or get the folder of the dialog box.

Layout Property

```
DialogBox.Layout as Layout R/W
```

Sets or get the layout of the dialog box.

Focus Property

```
DialogBox.Focus as Bool R/W
```

Set the focus at the newly opened dialog box.

DoModal Method

```
DialogBox.DoModal() as Integer
```

Show the dialog box and returns if the box is closed.
The return code is 0 or any value set with the CloseDialog(ExitCode as Integer) method.

Data::OnDialogOpen Event

```
Data::OnDialogOpen()
```

This event script has to be in the layout shown in dialog.

Data::OnDialogClose Event

```
Data::OnDialogClose( ExitCode as Integer )
```

This event scripts have to be in the layout shown from dialog.

5.24 TIMER

Timer objects can be instantiated in the Tube, Layout and Dataarea, but not in the Cell context.

```
dim withevents myTimer as new timer
```

SetTimer Method

```
Timer.SetTimer( Time as Integer, <Once as bool>, <TimerId as Integer> ) as bool
```

Sets and start the timer with Time in milliseconds.
Normally the timer runs until it is Killed. When Once is set to TRUE the timer event comes only one time. Set an optional TimerID to identify multiple timers.

KillTimer Method

```
Timer.KillTimer( <TimerId as Integer> ) as bool
```

Kill a timer.

_OnTimer Event

```
_OnTimer( TimerId as Integer )
```

The TimerId is the given integer in the SetTimer method.

The Function name must be prefixed with the corresponding timer object name, eg:

```
MyTimer_OnTimer()
```

5.25 ImportCsv

CsvHeaderName Property

```
Import.CsvHeaderName[ Index as Integer ] as String R/O
```

Get the header name of csv field.

CsvFieldCount Property

```
Import.CsvFieldCount as Integer R/O
```

Get the field count of the import.

EnableHeader Property

```
Import.EnableHeader as Bool R/W
```

Get or sets if import should care about csv header.

If enabled the import callbacks are called for the header line, but the record ID is zero. TBD

Match Property

```
Import.Match as Bool R/W
```

Force switch on/off matching records on import with keys.

Optional parameter, don't use if unsure. TBD

CountRecordsDone Property

```
Import.CountRecordsDone as Integer R/O
```

Get the count of records imported. After the import is completed. TBD.

CountRecordsMerged Property

```
Import.CountRecordsMerged as Integer R/O
```

Get the count of records merged in existing data.

MaxRecords Property

```
Import.MaxRecords as Integer R/W
```

Get or sets the max records to import.

Folder Property

```
Import.Folder as Folder R/W
```

Get or sets the folder to import.

Structure Property

```
Import.Structure as Value/Record R/W
```

Get or sets the structure of import.

StatusMessage Property

```
Import.StatusMessage as bool R/W
```

Enable/Disable Status Messages.

MaxRecordCount Property

```
Import.MaxRecordCount as Integer R/W
```

Set optional an max record count to import.

AddFieldDecl Method

```
Import.AddFieldDecl( Field as Value, CsvField as Integer, <Keyword as Value>, <InputFormat as String> ) as bool
```

Set the dependency between csv fields and cTubes fields.

Add a dependency from a database "Field" to a "CsvField".

Keywords are : "key", "ignorematch"

If an added Field is not found in the specified structure the field is added to the structure. TBD.

ClearFieldDecl Method

```
Import.ClearFieldDecl() as bool
```

Clear all set field declarations from function above.

Import Method

```
Import.Import( File as String ) as bool
```

Import the specified file.

Before calling this function, set at least the structure and folder.

ImportTemplate Method

```
ImportTemplate.Import( TemplateName as String, <File as String>,  
<TubeTemplate as Tube> ) as bool
```

Import the specified template.

Optionally overload the File and Tube(prop on object) settings from template.

TubeTemplate is the tube storing the template (optional).

LoadHeader Method

```
ImportTemplate.LoadHeader(File as String ) as bool
```

Load only header information of file.

Property CsvHeaderName[] and CsvFieldCount are available.

_OnUpdate Event

```
_OnUpdate( ByRef Record as Record, Folder as Folder )
```

Called if a record will be update to the folder.

Return true if the record should be NOT updated.

If an import record is matched to an existing without any change this function is not called.

_OnInsert Event

```
_OnInsert( ByRef Record as Record, Folder as Folder )
```

Called if a record will be inserted in the folder.

If no folder was given in the object, the Folder argument will be undef.

Return true if the record should be NOT inserted.

__OnPostInsert Event

```
__OnPostInsert( ByRef Record as Record, Folder as Folder )
```

Call after record was inserted.

__OnMatch Event

```
__OnMatch( ByRef Record as Record, Folder as Folder )
```

The record matched with an existing record, so doing nothing.

5.26 FileIndex

Which fields must be specified for file recognition to work ? TBD

After each update the field Action holds the action value:

```
ACTION_NONE = 0, ACTION_ERROR =1, ACTION_CONFLICT=2,  
ACTION_CHANGE =3, ACTION_MOVE=4, ACTION_RENAME=5,  
ACTION_NEW=6, ACTION_BROKEN=7, ACTION_COPY_DEL=8,  
ACTION_MOVE_CHANGE=9, ACTION_RENAME_CHANGE=10,
```

MD - File Modify Date, CD - File Create Date

Recognized cases:

- 1) Change - MD and or Size change (UPDATE)
- 2) Rename - Name change (UPDATE)
- 3) Move - Path change (UPDATE)
- 4) Copy - File with new CD at new path (UPDATE-UserDaten werden von alten File übernommen)
- 5) Copy/Del - File with new CD at new path AND old file has been deleted (UPDATE)
- 6) Move/Change – Path change (UPDATE)

**** 7) Rename/Change - Namen cahnge and MD/Size ** turned off

Broken links are reactivated if a File with the same name, CD, MD, Size (regardless of Path) appears again!

Folder Property

```
FileIndex.Folder as Folder R/W
```

Get or set the folder.

Structure Property

```
FileIndex.Structure as Record R/W
```

Get or set the structure.

RootPath Property

```
FileIndex.RootPath as String R/W
```

Get or set the file root path. If empty the default Tube path is taken. TBD

ExcludeFilter Property

```
FileIndex.ExcludeFilter as String R/W
```

Get or set the exclude filter. Only one filter can be used at a time.

Example: .exe;.txt;.bmp

Include .ctu , Exclude _ctu does not only filter ctu files not starting with _ TBD

IncludeFilter Property

```
FileIndex.IncludeFilter as String R/W
```

Get or set the include filter.

Example: .exe;.txt;.bmp

AddFieldDecl Method

```
FileIndex.AddFieldDecl( Field as Value, Type as Integer ) as bool
```

Adds a field declaration – every field gets a special value:
up to Version 2.1.4. **This value +1 ! correction TBD.**

```
ERF_NAME = 1, ERF_FULLNAME = 2, ERF_RELPATH = 3, ERF_ABSPATH = 4, ERF_EXT =  
5, ERF_MD5 = 6, ERF_MODTIME = 7, ERF_CREATETIME = 8, ERF_SIZE = 9,  
ERF_ACTION = 10, ERF_ROOT = 11, ERF_MAX = 12
```

ClearFieldDecl Method

```
FileIndex.ClearFieldDecl() as bool
```

Removes all fields.

UpdateIndex Method

```
FileIndex.UpdateIndex() as bool
```

Returns true if an update was possible.

5.27 File

Item Property

```
File.Item[ Index as Integer ] as File (default) R/O
```

The content items of a file folder. TBD

Count Property

```
File.Count as Integer R/O
```

Gets the count of containing files, only available if it is a folder.
See IsFolder property.

IsFolder Property

```
File.IsFolder as Bool R/O
```

Returns true if the object is a folder.

Name Property

```
File.Name as String R/O
```

Returns the name of object, without path.

Path Property

```
File.Path as String R/O
```

Returns the path of object.
To get the complete path use : File.Path + File.Name

Extension Property

```
File.Extension as String R/O
```

Returns the extension of object.
Example: exe

Size Property

```
File.Size as Integer R/O
```

Returns the file size of object.

DateAccess Property

```
File.DateAccess as Date R/O
```

Returns the date of the last access.

DateModify Property

```
File.DateModify as Date R/O
```

Returns the date of the last modified.

DateCreation Property

```
File.DateCreation as Date R/O
```

Returns the date of creation.

Open Method

```
File.Open( <Path as String>, <ExtensionFilter as String> ) as bool
```

To initialize the object call Open to a file or directory.

Is the object a child from a parent file object, just call Open() without an argument.

Sort Method

```
File.Sort( Type as Value, <Descend as bool> ) as bool
```

After opening a folder you may sort them by a specified type:

- 1 – Name
- 2 – Path only
- 3 – Path and and name
- 4 – size
- 5 – extension
- 6 – folder / file
- 7 – Date access
- 8 – Date modify
- 9 – Date creation

Rename Method

```
File.Rename( FileName as String ) as bool
```

Rename the actual object to "FileName".

Remove Method

```
File.Remove() as bool
```

Removes the current object.

Error handling all file methods TBD

Move Method

```
File.Move( FilePath as String ) as bool
```

Moves the actual object to another path.

Copy Method

```
File.Copy( FilePath as String ) as bool
```

Copies the current object to a file path.

Load Method

```
File.Load() as Binary
```

Returns a binary loaded from file, or undef if it's a folder or an error occurs.

Save Method

```
File.Save( Binary as Binary ) as bool
```

Saves the given binary to file.

6 Database Objects

6.1 FOLDER

Item Property

```
Folder.Item[Index as value] as Record R/W (default)
```

Sets or get the record of index.

This function does not change the position.

Available index: position, record id, fields(filter), string(name)

Note:

This lookup internally uses a query (even when using the record ID as index), so the folder's current query settings apply! Deleted records are NOT returned.

Id Property

```
Folder.Id as ID R/W
```

Sets or get the actual folder id.

Name Property

```
Folder.Name as String R/W
```

Sets or get the actual folder name.

Structure Property

```
Folder.Structure as Record R/W
```

Sets or get the current structure of folder.

This structure works as query filter. TBD.

IMPORTANT: set the Structure when initializing a query, otherwise you will get the records, but without fields!

Record Property

```
Folder.Record as Record R/W
```

Sets or get the record representing the folder itself.

Count Property

```
Folder.Count as Integer R/O
```

Gets the current count of records in folder.

Insert Method

```
Folder.Insert( Record as Record, <ByRef RecOut as Record > ) as Id
```

Insert a new record in the folder.

Optionally the new record is in the output RecOut.

Set RecOut to true, and the record id is taken from Record.

Note: In this case the TRUE value must be passed as a variable, not directly!

Returns the id of inserted record if success.

InsertFolder Method

```
Folder.InsertFolder( Folder as Folder, <OnlyChilds as bool>, <KeepIds as bool> ) as Value
```

Insert a complete folder with its childs.

Set OnlyChilds to true if you only want insert the childs.

Returns the new folder if inserting the complete folder, and returns a bool if inserting only childs.

AddLink Method

```
Folder.AddLink( Record as Value, <History as bool> ) as Bool
```

Add a link to an existing record (Multi Parent).

Returns true if success.

AddLinkFolder Method

```
Folder.AddLinkFolder( Folder as Value, <OnlyChilds as bool>, <History as bool> ) as Bool
```

Add a link to an existing record (Multi Parent).

Returns true if success.

Update Method

```
Folder.Update( <Record as Record> ) as Bool
```

Update an existing record or the folder itself, if no parameters given.

Returns true if success.

Remove Method

```
Folder.Remove( Record as Value, <AllLinks as bool> ) as Bool
```

Remove an existing record in the folder.

Returns true if success.

Clear Sub Method

```
Folder.Clear()
```

Remove all records in the folder.

MoveTo Method

```
Folder.MoveTo( Index as value ) as Bool
```

Change the position in the folder.

Available index: position, record id, fields(filter), string(name)

Returns true if success.

NextItem Method

```
Folder.NextItem( ByRef Record as Record ) as Bool
```

AddFieldOption Method

```
Folder.AddFieldOption( Field as Value, SortUp as bool, <Filter as Value> )  
as Bool
```

Set optionally a sort, a filter or the two properties together.

Set undef if you don't want set the property.

ClearFieldOptions Method

```
Folder.ClearFieldOptions( ByRef Record as Record ) as Bool
```

Clear all settings from AddFieldOption() method.

GetStory Method

```
Folder.GetStory() as Folder
```

Returns a new folder object containing the story records.

InvalidateIndex Sub

```
Folder.InvalidateIndex( field as value )
```

See also Data.InvalidateIndex TBD

6.2 RECORD

Id Property

`Record.Id` as ID (default) R/W

Sets or get the actual record id.

Note: you can not directly call `Record.Id.Unique()`, instead set `Record.ID = mynewID` TBD.

Structure Property

`Record.Structure` as ID R/W

Sets or get the current structure.

Fields Property

`Record.Fields` as Fields R/W

Sets or get the Fields collection.

Modified Property

`Record.Modified` as Date R/W

Sets or get the actual Modify date.

Owner Property

`Record.Owner` as ID R/W

Sets or get the last modifier (user) id.

Creator Property

`Record.Creator` as ID R/W

Sets or get the record's creator (user) id.

Name Property

`Record.Name` as String R/W

Sets or get the name field.

6.3 FIELDS

Item Property

```
Fields.Item[Index as Value] as Field (default) R/W
```

Sets or get the field from index.

Possible values for index are:

Name as String, FieldId as Id, Field as Field, Position as Integer

Count Property

```
Fields.Count as Integer R/W
```

Sets or get the count of fields.

Remove Method

```
Fields.Remove( Index as Value ) as bool
```

Removes the field from collection.

Index : see Item property.

Add Method

```
Fields.Add( Field as Value ) as Integer
```

Add fields to the collection.

Possible field values:

Field as Field, CountFields as Integer

Returns the count of fields added.

Clear Method

```
Fields.Clear()
```

Removes all fields from collection.

6.4 FIELD

Value Property

```
Field.Value as Value (default) R/W
```

Sets or get the value.

Name Property

```
Field.Name as String R/W
```

Sets or get the name of field.

Type Property

```
Field.Type as Long R/O
```

Gets the Type of field. 0= 1= Bool, 2= Long, 3= Float, 4 = ID, 5 = General (string), 6=DateTime, 7=Binary, 10=Array

Id Property

```
Field.Id as Id R/W
```

Sets or get the ID of field.

Format Property

```
Field.Format as String R/W
```

Sets or get the format (see Format String) of field.

Formula Property

```
Field.Formula as String R/W
```

Sets or gets the formula of field.

Options Property

```
Field.Options as Integer R/W
```

Sets or get the options of field.

6.5 GROUPS

Item Property

```
Groups.Item[Index as Value] as Group (default) R/O
```

Sets or get the group from index.

Possible values for index are:

Name as String, GroupId as Id, Position as Integer

Count Property

```
Groups.Count as Integer R/O
```

Get the count of groups.

Admins Property

```
Groups.Admins as Group R/O
```

A direct access to the system admin group.

TubeUsers Property

```
Groups.TubeUsers as Group R/O
```

A direct access to the system tube users group.

Creators Property

```
Groups.Creators as Group R/O
```

A direct access to the system creators group.

BlackList Property

```
Groups.BlackList as Group R/O
```

A direct access to the system blacklist group.

FindUser Method

```
Groups.FindUser( User as Value ) as User
```

Find a user by name or id.

Create Method

```
Groups.Create( <Name as String> ) as Group
```

Creates a new user group.

Remove Method

```
Groups.Remove( Group as Value ) as bool
```

Removes a user group.

Clear Method

```
Groups.Clear() as bool
```

Removes all user groups.

6.6 GROUP

Item Property

```
Group.Item[Index as Value] as User (default) R/O
```

Returns the user.

Possible values for index are:

Name as String, UserId as Id, Position as Integer

Id Property

```
Group.Id as Id R/O
```

The id of the group.

Name Property

```
Group.Name as String R/W
```

Sets or gets the name of group.

Count Property

```
Group.Count as Integer R/O
```

Get the count of groups.

IsSystem Property

```
Group.IsSystem as Bool R/O
```

Return true if the group is a system group.

IsAdmins Property

```
Group.IsAdmins as Bool R/O
```

Return true if the group is the system administrator group.

IsTubeUsers Property

```
Group.IsTubeUsers as Bool R/O
```

Return true if the group is the system tube users group.

IsCreators Property

```
Group.IsCreators as Bool R/O
```

Return true if the group is the system creators group.

IsBlackList Property

```
Group.IsBlackList as Bool R/O
```

Return true if the group is the system black list group.

Update Method

```
Group.Update() as bool
```

Update changes in the group.
Updates only for user groups possible.

Add Method

```
Group.Add( user as user ) as bool
```

Add a user to the group.

Remove Method

```
Group.Remove( user as user ) as bool
```

Removes the user from the group.
You can't remove a user from the tube users group.

Clear Method

```
Group.Clear() as bool
```

Removes all users from the group.

You can't remove users from the tube users group.

CheckRights Method

```
Group.CheckRights( Rights as Integer ) as bool
```

Returns true if the specified rights set on the object.
This function supports an array for the Rights argument.

SetRights Method

```
Group.SetRights( Group as Value, Rights as Integer, SetType as Value ) as bool
```

Set the right on the object with specified group.
SetType can be true, false or undef.
Returns true if no error occurs.
This function supports an array for the Rights argument.

GetRights Method

```
Group.GetRights( Group as Value, Rights as Integer ) as Integer
```

Returns a number defining the status of Rights.
This function supports an array for the Rights argument.
See Tube.GetRights() for return codes.

6.7 USER

Name Property

```
User.Name as String (default) R/W
```

Get the name of the user.

Id Property

```
User.Id as Id R/O
```

The id of the user.

IsEmpty Property

```
User.IsEmpty as bool R/O
```

Return true if user is not valid.

Groups Property

User.Groups as Groups	R/O
-----------------------	-----

Returns a new groups object of all groups where the user is a member of.

7 Events

7.1 Built in Events

For each context object there is a number of events that appear in the script editor.
For example you can execute code when a Cell is clicked:

```
function OnMouseClicked()  
    Rem do something  
end function
```

TBD: list of all events and field-update / record-update concept

Timer Events

Within the Tube, Layout (Data) and Layout Area you can define Timer objects.
The event function is then called with the name of your timer variable.
See also Timer object.

```
Dim WithEvents MyTimer as new Timer  
  
Function OnInit()  
    MyTimer.SetTimer( 500 )  
End Function  
  
Function MyTimer_OnTimer( TimerID as Integer )  
    Rem do something  
End function
```

8 Global Methods

8.1 Mathematics

Sin Method

```
Sin( Radian as Float ) as Float
```

Calculate the sinus of given radian.

Cos Method

```
Cos( Radian as Float ) as Float
```

Returns the cosinus of given radian.

Tan Method

```
Tan( Radian as Float ) as Float
```

Returns the tangens of given radian.

HSin Method

```
HSin( Radian as Float ) as Float
```

Returns the hyperbolic sinus of given radian.

HCos Method

```
ACos( Radian as Float ) as Float
```

Returns the hyperbolic cosinus of given radian.

HTan Method

```
HTan( Radian as Float ) as Float
```

Returns the hyperbolic tangens of given radian.

Sqr Method

```
Sqr( Number as Value ) as Value
```

Returns the square root of the given number.

Log Method

```
Log( Number as Value ) as Value
```

Returns the natural logarithm of the given number.

Log10 Method

```
Log10( Number as Value ) as Value
```

Returns the base 10 logarithm.

Exp Method

```
Exp( Number as Value ) as Value
```

Returns the exponential e^x

DegToRad Method

```
DegToRad( Number as Value ) as Value
```

Converts Number in degrees to radiant.

RadToDeg Method

```
RadToDeg( Number as Value ) as Value
```

Converts Radiant to degrees.

MinMax Method

```
MinMax( Number as Value, Min as Value, <Max as Value> ) as Value
```

Returns Value, but not less than Min and not greater than Max.

RGB Method

```
RGB( Color/Red as Integer, <Green as Integer>, <Blue as Integer> ) as Integer
```

Returns the color code. You can either specify the components as R,G,B (range 0 .. 255) or the 'html' color as a single parameter.

cell.backcolor = RGB(255, 0, 0) sets the color to red

cell.backcolor = RGB(0x1CE500) sets the color to a shade of green (0x denotes hex value)

cell.backcolor = -1 sets transparent color, no need for the rgb() function

The valid range of red, green and blue values are between 0 and 255.

8.2 File Handling

FileCopy Method

```
FileCopy( SrcPath as String, DstPath as String, <Dir as bool> ) as bool
```

Returns true if the file or directory (Dir argument) in SrcPath have been copied to DstPath.

FileRename Method

```
FileRename( SrcPath as String, DstPath as String, <Dir as bool> ) as bool
```

Returns true if the file or directory (Dir argument) in SrcPath have been renamed to DstPath.

FileRemove Method

```
FileRemove( Path as String, <Dir as bool> ) as bool
```

Returns true if the file or directory (Dir argument) in Path have been removed.

FileSize Method

```
FileSize( Path as String ) as Integer
```

Returns the file size in bytes, 0 if an error.

OpenFileDialog Method

```
OpenFileDialog( Title as String, <IsSaveDialog as bool>, <InitDir as String>, <Filter as Array>, <SelectedFile as String> ) as String
```

Returns the selected file in the dialog, or returns undef if dialog canceled.

Filter is a pair of strings containing description and filter.

Example, two filters: ["Text File (*.csv;*.txt)", "*.csv;*.txt", "cTubes File (*.ctu)", "*.ctu"]

OpenFolderDialog Method

```
OpenFolderDialog( Title as String, <InitDir> as String ) as String
```

Returns the selected folder in the dialog, or returns undef if dialog canceled.

OpenInputDialog Method

```
OpenFolderDialog( Title as String, <Default> as String ) as String
```

Opens a input dialog and returns the value.

The return value is undef, if dialog was canceled.

8.3 Miscellaneous

Beep Method

```
Beep(sound as value)
```

If sound is not specified the standard beep message on your windows. If sound is specified:

HAND	1
QUESTION	2
EXCLAMATION	3
ASTERISK	4

LastError Method

```
LastError() as Error
```

If an error is stored, an error object is returned.

ExecuteShellCmd Method

```
ExecuteShellCmd( ExePath as String, <Parameters as String> )
```

Calls the program at ExePath with the given command line parameters.

Note: `executeshellcmdsync`

Print Method

```
Print( Output as String )
```

Prints the output in:

- 1) cTubes : Statusbar
- 2) ctServer : Log, Shell output if not running as service

Trace Method

```
Trace( Output as String )
```

Stores the string in a tracefile or/and logfile.

IsLicense Method

```
IsLicense( License as String ) bool
```

Check if a license is available.

CloseDialog Method

```
CloseDialog( <ExitCode as Integer> )
```

Use this method to close a dialog box (in the dialog box layout).
The ExitCode will be the return value of `DialogBox.DoModal()`, see object `DialogBox`.

CopyToClipboard Method

```
CopyToClipboard( Value as Variable )
```

Stores the value into clipboard.

PrinterPage Method

```
PrinterPage() as Integer
```

Returns the current page of printing during printing.
Only available when the layout is printing.

PrinterPageCount Method

```
PrinterPageCount() as Integer
```

Returns the count of pages during printing.
Only available when the layout is printing.

PrinterPageText Method

```
PrinterPageText() as String
```

Returns a complete finished text with page and count information.
Only available when layout is printing.

FormatOut Method

```
FormatOut( Value as Value, <Format as String> ) as String
```

Returns a converted string from given value.
Use this method to flexibly convert values to formatted strings.
For description of the Format argument see Format section.

FormatIn Method

```
FormatIn( Value as String, <Format as String> ) as Value
```

Returns a Value converted from the string value.
Use this method to flexibly convert from strings to internal values.
For description of the Format argument see Format section.

RegExp Method

```
RegExp( Value as String, RegExp as String ) as Array
```

Scans Value for any matches given in RegExp and returns them as Array.
Value is the string to scan, and RegExp is the regular expression.
The return will be an array with strings, results from the regular expression.
Examples TBD

Text Method

```
Text( Value as Value ) as String
```

This method tries to find a string in the value(or object).
Examples: `r = Text(record)` returns the name if a standard name field exists or even the language independent name of the record if it exists.

MessageBox Method

```
MessageBox( Text as String, <Title as String> )
```

Shows a "Ok" Message box.

Chr Method

```
Chr( CharCode as Integer )
```

Shows a "Ok" Message box.

Now Method

```
Now() as Date
```

Returns the current date and time.

8.4 Alternative Global Functions

Many functions already available for objects are also available als global functions.

As compared to the obejct functions with global functions the obejct is supplied as additional first parameter.

Available functions:

```
Len, Mid, Instr Left, Right, Len, Mid, Instr, Right, FormatDate, IsId,  
IsSystemID, IdSysValue, AddSysValue, IdHash, Isundef, IsEmpty, IsNull,  
Array
```

Example:

```
dim s as string = "abcdef"  
x= s.left(3)  
#alternative:  
x= left(s,3)  
# both return 'abc'
```

9 System Names

A System Name in cTubes is a simple way to get a reference to an object.
There are several types of system names:

Tube, Layout, Cell, CellGroup, Structure, Field, View, Lang, Value Lists.

A system name begins always with a dollar '\$'.
Optionally beginning with a system name type, and following with the object path.
The type and path segments are separated with ':'

System Name Examples:

\$Cell:F5	Reference to the cell in column "F" and row "5".
\$Field:Tom named "Tom".	Reference to a field (in the context record) named "Tom".
\$Field:Person:Name structure.	Reference to a field named "Name" in the "Person" structure.
\$Structure:Person	Reference to the Structure named "Person".
\$Layout:MyLayout	Reference to the "MyLayout" Layout

System Views:

[PRINTER]	Printer view (only while printing)
[VIEWTOOLBAR]	Own Toolbar
[MODALBOX]	Currently open modalbox
[EDITPROP]	Editproperty in editor
[AUTOVIEW]	Current autoview
[ACTIVE]	Currently active view
[LASTUSERVIEW]	last active user view – to know in an autoview where we came from

Plus all Views as specified with the ListBox Element

The names of the currently open user views.

```
set view = views["name"]
```

Script Examples:

```
$Cell:F5.backcolor = RGB( 200, 0, 0 ) Setting the back color of the cell  
F5 to red.  
R = $Field:Tom Getting the value of the field "tom"  
(standard property)  
T = $Lang:LangName Getting the language entry string named  
"LangName"
```

10 Example of COM use

Login to Website and get data from page.
We simply use the COM object "WinHttpRequest" from within UqScript.

WinHttpRequest

see description at: <http://msdn2.microsoft.com/en-us/library/aa384106.aspx>

Example cTubes admin :

```
function OnMouseClicked()  
  
set r = CreateObject( "WinHttp.WinHttpRequest.5.1" )  
  
url = "http://adminix.ctubes.at/"  
  
r.Open( "POST", url & "index.php" )  
r.setRequestHeader( "Content-Type", "application/x-www-form-urlencoded" )  
r.Send( "login=mid&passwd=myspasswd" )  
# logged in from here !  
  
r.Open( "GET", url & "nav.php" )  
r.Send()  
  
value = r.ResponseText  
  
end function
```

Das upload thema ist eingebaut (aber noch nicht getestet mit php).
Sorces eingecheckt - benötigt recompile.

Hier das script :

```
function OnMouseClicked()  
  
dim bin1 as binary  
dim bin2 as binary  
  
bin2.load( "c:\file" )  
  
set r = CreateObject( "WinHttp.WinHttpRequest.5.1" )  
  
# WEBSERVER  
#  
url = "http://www.url.at/index.php"  
  
# content-type und variablen setzen  
# man kann hier natürlich variablen hinzufügen # mp = '-----  
-----4277187999986  
Content-Disposition: form-data; name="tom"; filename="standard.iss"  
Content-Type: text/plain  
  
,  
  
bin1 = mp.ToMultibyte  
bin1 += bin2  
  
r.Open( "POST", url )  
r.SetRequestHeader( "Content-Type", "multipart/form-data; boundary=-----  
-----4277187999986" ) r.Send( bin1 )  
  
end function
```

11 Formatting / Format Strings

Format strings define how values are displayed and how entered strings are converted into stored values. The same format strings are used in the Format properties of Fields and Cells and in the FormatIn and FormatOut Functions.

11.1 Numbers / Currency

The number and currency formatting can be fully controlled.

```
N{g3.n,p4}C{"€ "g3.n,p2}
```

g-group digit, n-number, p-fractional part of number

minimal examples: N{n} - N{np} - std: N,

it is better not to use '.' and ',' (look and feel) ex: N{g3np4}

holder for std currency : #, ex : C{"#"pn }

More examples TBD.

11.2 Date / Time

Date/Time formatting can either use the Windows settings or use formatting defined within cTubes.

```
D{dd:MM:yy, hh:mm:ss}
```

d-day, M-month, y-year, h-hour(12)(hh-leading zero), H-hour(24)(HH-leading zero), m-minute(mm-leading zero), s-seconds(ss-leading zero), x-milli seconds(xx-leading zero), t-am/pm

MM - month number, MMMM - month text, dd - day number, dddd - weekday text

SHORT FORMAT : D{DL T} - use std Date Long and std Time

DL - Date long, DS - Date short, DY - Date year, month, T - time

Warning : use a blank instead ',' or something else as separator between date and time(example) !

D{DL, T} better-> D{DL T} because the user types always a blank not the comma!

```
= FormatOut( t, 'N{n" "[sec:*]}' )
```

```
= FormatIn( input, 'N{n" "[sec:*]}' )
```

DateTime TimeZones :

Usually a date/time will be stored as local time and not transformed to UTC time, this means the time is shown as the same value around the world.

You can modify the INPUT behavior of datetime:

```

INPUT :
example1) D{DL T[LU]} input in local time, transformed and stored in UTC
time
example2) D{DL T[UU]} input in UTC time, stored in UTC time
example3) D{DL T[LL]} input in LOCAL time, stored in LOCAL time (standard
input)
OUTPUT :
example1) D{DL T[LU/U]} input in local time, transformed and stored in UTC
time - output is in UTC time
example2) D{DL T[LU/L]} input in local time, transformed and stored in UTC
time - output is in LOCAL time (standard output)

```

```

groups:
D - date/time
N - number
C - Currency
+ - force this group

```

Usually there can be a few types of formats given in format string.
It depends on the value type which format string is taken.
To force a special type, put a plus behind the group character.

```

ex: N+ or N+{np4) or C+ or D+{MM}
in this case a UqValue will be always transformed in the specified type.

```

11.3 Extensions for Numbers and Currency - Measurement module

Values can be converted automatically. For example input may accept a size in mm, cm or m. The size may be stored normalized to mm. On Output the size may be displayed as appropriate for the value.

Full documentation TBD.

```

N{np3[cm]}           base unit: cm
N{np3[cm:5-mm/100-m]} value less than 5, output in mm / value greater
100 output in m
N{np3[cm:*/10-dm]}   default useful unit / value greater 10 in dm
or:
N{np3[TWIPS:cm]}     base unit: twips, output in cm

```

11.4 Special format for value lists

```
V+{ValueName/ID/ConfigValueNumber}
or
V+{ValueName/ID/ConfigValueNumber:ForceFlag} Force FLag see eUqMenuFlags
in uq_application.h

E+  always empty text/value

B+  Formats out a BARCODE value

S+  Script formatting - script id's converted to names on ouptput, and
names converted to id's on input !

X+  Binary formatting - not defined yet, if picture it will shown

A+  Array formatting -    {number of levels, separator(s), type,
                           subformat}
                           if type is undef, the type will be guessed
examples: A{1;integer} A{2[ ];undef} A{2;,id} A{3[ ];,string}
          A{3[ ];,integer g3np2}
standard format:  A{1;undef}
```

12 Elements

12.1 Element ValueBar

The value Bar Element resizes a group of cells according to their field values ('OnOutput).

It can be used to show horizontal bar-graphs, with any number of segments.

There are 2 config values:

Number of Cells

Defines how many cells are part of the bar. Number can be from 2 to n.

Type

Defines where the total value for the bar is taken from:

Type 0: x,y,z,... The sum of all cells defines 100%

Type 1: x,y,z,..., total The last cell value defines 100%

Note: The cells are painted in the background color as defined in the layout. OnRefresh() is not called for cells which are part of a value bar. (At least the bgcolor can not be set this way.)

13 Other

Complete EventCalls description TBD

Diverse System Values

```
#define SYSVALUE_PSEUDO_FIRST (0x180)

#define SYSVALUE_STRUCTID      (0x180)
#define SYSVALUE_OWNERID      (0x181)
#define SYSVALUE_MODIFIED     (0x182)
#define SYSVALUE_RECORDID     (0x183)
#define SYSVALUE_STORYAGE     (0x184)
#define SYSVALUE_STORYDEPTH   (0x185)
#define SYSVALUE_CREATORID    (0x186)
#define SYSVALUE_CHANGEID     (0x187)

#define SYSVALUE_ALLFIELDS    (0x18e)

#define SYSVALUE_PSEUDO_LAST  (0x18f)
```