Edelwise™

beWISE Programming Guide

Title: User Manual for beWISE Products.

Type of documentation: Manual

Date of creation: 02/09/2007

This technical description replaces all previous versions.

Document history

Date	Name	Revision	Modification
2/9/2007	edelweiss	A	First version
2/15/2007	edelweiss	B	Some changes

1 TABLE OF CONTENTS

2 IMPORTANT CONSIDERATIONS

2.1 Before Installation

It is important to install the product as Administrator or as user with administrative rights.

2.2 After Installation

The product creates a directory .\Edelwise under the *ProgramFiles* folder during installation. In the English version of Windows the *ProgramFiles* folder is C:\Program Files\. There you will find all sample source codes, tools and documentation. The beWISE DLL and essential background programs are copied into the SYSTEM32 directory.

2.3 Removing an Installation

To uninstall a beWISE product you have to use the "Add or Remove Programs" function in the Control Panel. This will remove all programs from the SYSTEM32 directory, but not from the .\Edelwise folder. To completely remove the product you have to delete the .\Edelwise folder after uninstallation.

BEFORE removing an installed beWISE product make sure that the *beWISE-Core* is not loaded, otherwise the product cannot be removed correctly. To unload the *beWISE-Core* use the program .\Edelwise\beWISEstop.exe or reboot the system.

2.4 Trial Period

This product comes with a 14 day trial period during which you can test the program. During the trial period ***do not change the DATE*** on your computer in order to extend the trial period. The protection mechanism will detect such changes and as a result the product will not function anymore.

2.5 'Fast User Switching' Feature of Windows XP

'Fast User Switching' is currently NOT supported.

On a single computer only ONE instance of the beWISE-Core can be active (loaded) at any given time. The beWISE-Core is loaded in the user's context that initiated it and is only available to that user. If one user loads the beWISE-Core (e.g. by creating a beWISE variable) no other user in the system can use beWISE variables until this beWISE instance is shut down (normally this happens when the system restarts, when the user logs off or when beWISE is unloaded using the Dashboard system function).
This is not of any concern unless the 'Fast User Switching' feature of Windows XP is used. Here a new user can logon without the old user logging off. In this case only the user that first loads the beWISE-Core will have access to it. Other users will not be able to use any beWISE functions until this user shuts down the active beWISE instance.

3 PRODUCT INFORMATION

3.1 What is beWISE?

beWISE is a product that provides Inter-Process Communication (IPC) capabilities for your Visual Basic programs. In other words, beWISE makes it very easy for your Visual Basic programs to talk to each other.

beWISE does that by providing two classical methods of IPC (Inter-Process Communication); shared memory and message passing. We call them beWISE VARIABLES (sometimes also Shared Variables) and beWISE PIPES (named pipes). Although operating systems like Windows and Unix provide APIs that allow you to utilize shared memory and message passing, it is often a very painstaking process to implement Inter-Process Communication using these low level APIs. Not only do you have to know the API well, but you also have to consider things like resource sharing, priority inversion and deadlocks. beWISE takes away that pain from you by providing an easy-to-use interface that takes care of all these problems.

The product consists of a DLL, which you have to reference in you VB application, and a few essential background tasks which are automatically started when the beWISE DLL is loaded.

3.2 Different beWISE Products

The beWISE product comes in different flavors. You can choose between a BASIC, ADVANCED, PROFESSIONAL and a LAN version. There is also a FREE version. All, except the FREE product, are available for VB6 and VB.NET. The FREE version is available for VB6 only. All versions come with sample programs. An optional C library is available for some versions. It allows conventional C programs to access beWISE Variables. For more technical information, white-papers and best practices regarding beWISE please check our FAQ page and our Forum. We encourage you to register as a forum user to be able to post messages. Registration is free and does not require purchase of a product.

3.2.1 beWISE BASIC

This is the low-cost entry into the world of Inter-Process Communication. Choose this product if you have only a limited need for task-to-task communication. This product allows you to create a maximum of 32 shared beWISE variables.

3.2.2 beWISE ADVANCED

If you need more than just a few variables, this is the right product. It supports a maximum of 5,120 shared beWISE variables. It allows you to dynamically create and delete variables, to organize and manage them in lists, and to store additional attributes (dictionary information) with each variable. Choose this product if you need to share a lot of information among programs, but have no need for data streams or serialized data exchange.

3.2.3 beWISE PROFESSIONAL

This product includes all features of the ADVANCED version plus PIPES and an optional C interface. Pipes allow you to exchange streams of data between applications. Using pipes you can exchange messages (e.g. between a driver and an application) or large amounts of data (e.g. the serialized content of a web-site) among multiple programs. The optional C interface allows sharing of variables between VB and conventional C programs.

3.2.4 beWISE LAN

If you are developing a distributed system, with applications running on different computers in a LAN, then this is the product you need. It includes all features of the PROFESSIONAL version, plus it provides LAN capable variables and pipes. These variables and pipes are shared among all your beWISE applications in the LAN. If you use only beWISE LAN-variables and beWISE LAN-pipes for you Inter-Process Communication, you can take any application and put it on any computer in your network; and all the programs will still be able to communicate as if they where on one computer. If you plan implementing fully scalable, distributed systems, then this is the right product.
This product is currently under development and not available.

3.3 Feature Comparison

beWISE features	*Free*[1]	*Basic*	*Advanced*	*Professional*	*LAN*[2]
Variables (number)	1	32	5,120	5,120	5,120
LIST Objects			✓	✓	✓
PIPE Objects				✓	✓
C-Library				optional	optional
Network Variables					✓
Network Pipes					✓
Platforms	VB6	VB6 / NET	VB6 / NET	VB6 / NET	VB6 / NET

[1] The FREE version supports only one variable of type DOUBLE
[2] The LAN version is currently not available.

4 SAMPLE PROGRAMS

Each beWISE product comes with Visual Basic examples that demonstrate the use of the beWISE variables and objects. These sample programs are provided for VB6 and VB.NET and can be found in the folders

ProgramFiles\Edelwise\VB.NET.Samples\ and

ProgramFiles\Edelwise\VB6.Samples\

Full source code is provided with all sample programs.

Not all sample programs are delivered with each product. Since they require product specific features, only the sample programs that demonstrate features supported by the product are shipped. The following chart show which sample programs are shipped with which product version.

4.1 Example / Products Chart

Example Program	*Basic*	*Advanced*	*Professional*
Counting	✓	✓	✓
Discussing	✓	✓	✓
Listening		✓	✓
Messaging			✓

4.2 EXAMPLE "*Counting*"

This example shows how one VB program constantly increments a variable using a timer while another program displays the value of the variable using a beWISE CHANGED event.

4.3 EXAMPLE "*Discussing*"

This example shows how two programs can communicate both ways using two beWISE variables. For demonstration purposes the two variables used have a different data type.

4.4 EXAMPLE "*Listening*"

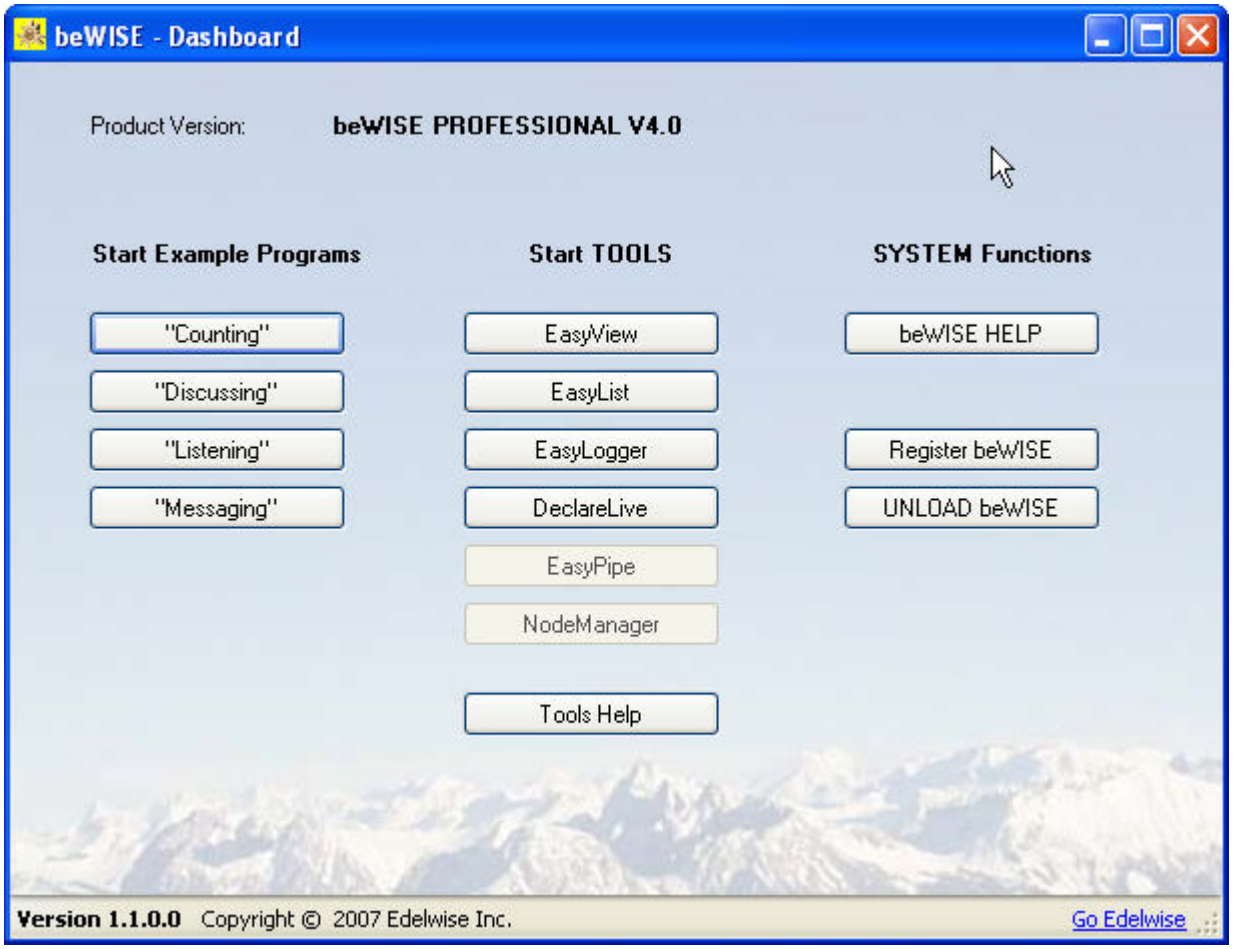
This example demonstrates the use of the LIST object. It shows how two programs can communicate large amounts of data using LIST objects.

4.5 EXAMPLE "*Messaging*"

This example demonstrates the use of PIPES. It achieves essentially the same functionality as the example "*Listening*", but does this solely with PIPES (no use of variables at all).

4.6 Starting the *Sample Programs*

In the VB.NET version the sample programs can be started using the beWISE Dashboard tool.



In the VB6 version there are no tools. The sample programs have to be started manually.

5 USING beWISE IN YOUR PROGRAM

5.1 Adding the beWISE DLL to Your Project

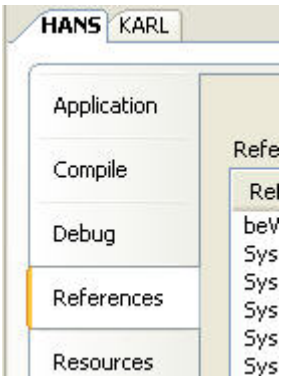
Before you can use any beWISE variables and objects you have to add the beWISE DLL to your VB project. The following example demonstrates how the beWISE PROFESSIONAL DLL (beWISEpro.dll) is loaded into a VB.NET or VB6 project.
If you have a product other than the PROFESSIONAL version, the product and file name will be different.

The following shows the names for the different beWISE products:

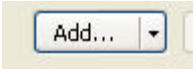
DLL name	*product name*
beWISEbas.dll	beWISE BASIC
beWISEadv.dll	beWISE ADVANCED
beWISEpro.dll	beWISE PROFESSIONAL
beWISElan.dll	beWISE LAN

5.1.1 Adding the beWISE DLL in Visual Basic .NET

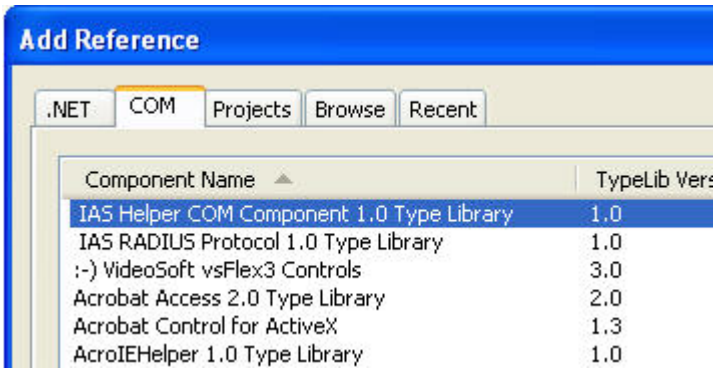
In your Project Properties tab select References



Click the Add button

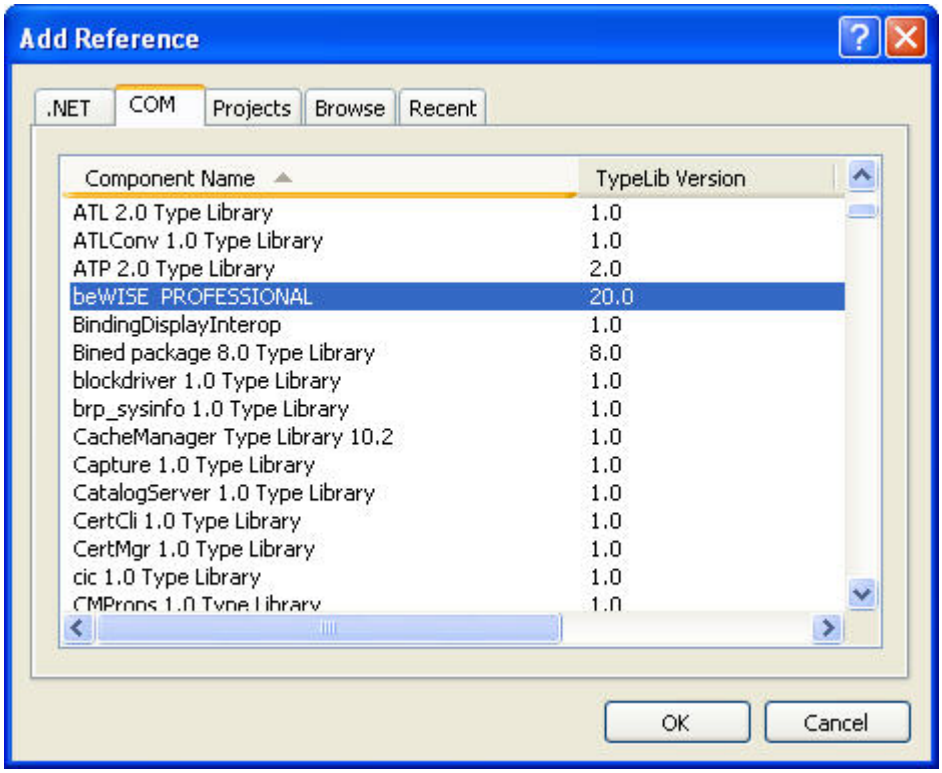


Select the COM tab



Scroll down and select beWISE PROFESSIONAL

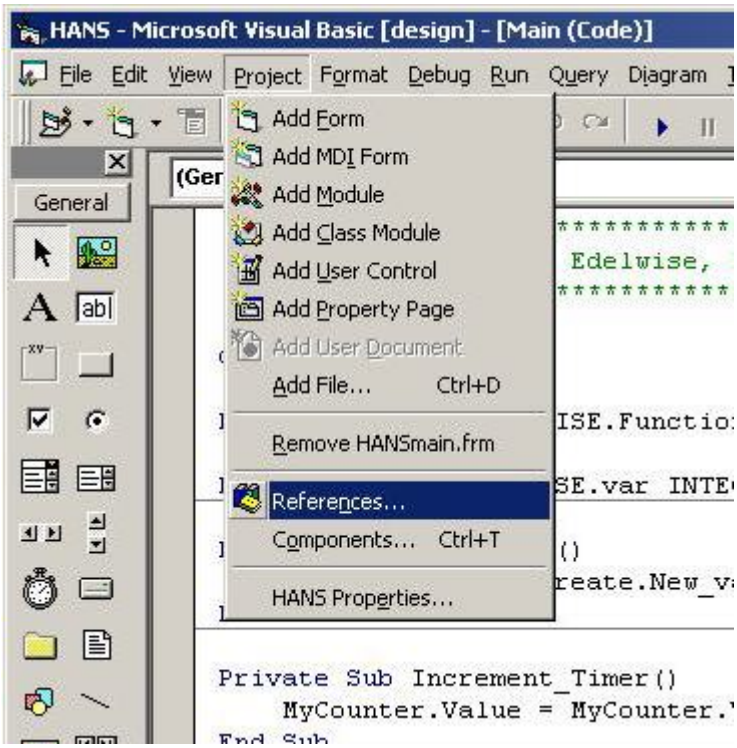
Note: When you have a different beWISE product the name and version will be different.



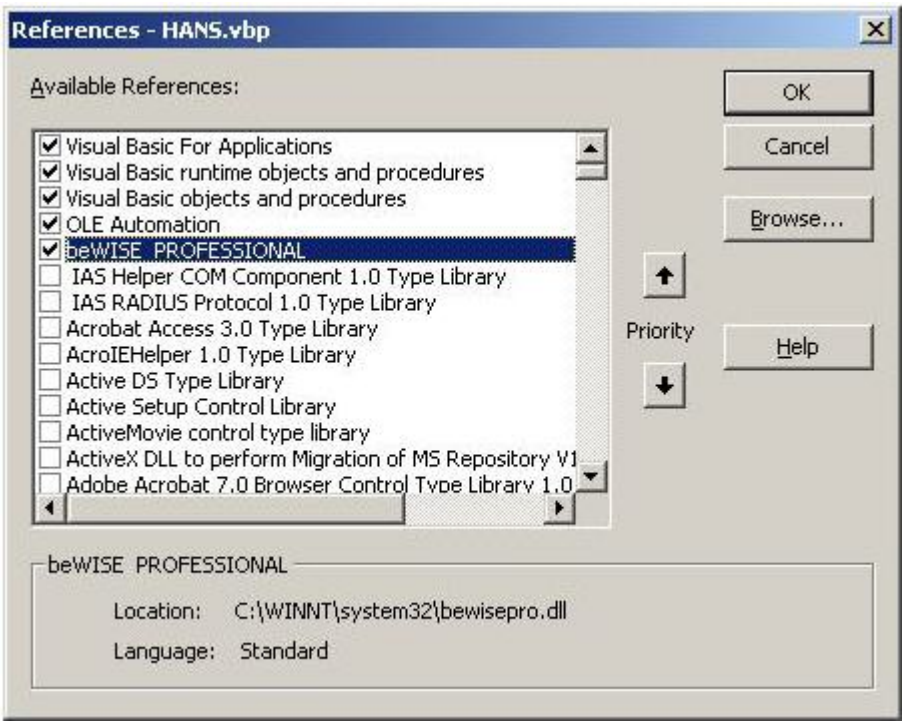
Click the OK button to add the library to your project.

5.1.2 Adding the beWISE DLL in Visual Basic 6.0

In Project select References



Scroll down, select beWISE PROFESSIONAL and click OK to add it

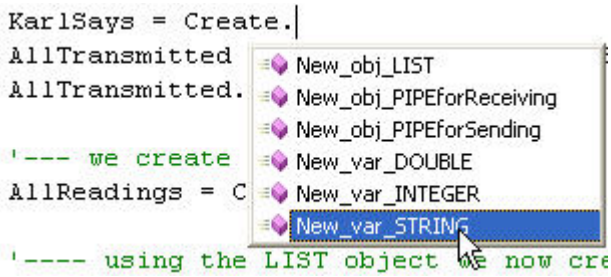


5.2 Required Source Code Declarations

beWISE provides a separate class with functions for creation of new Variables, Lists and Pipes. In order to access these functions the class has to be created first. To do this you have to include the following statement at the beginning of your source code:

```
Dim Create As New beWISE.Functions 'allows access to all CREATE functions
```

This will allow us to select all create functions by just typing `Create.`



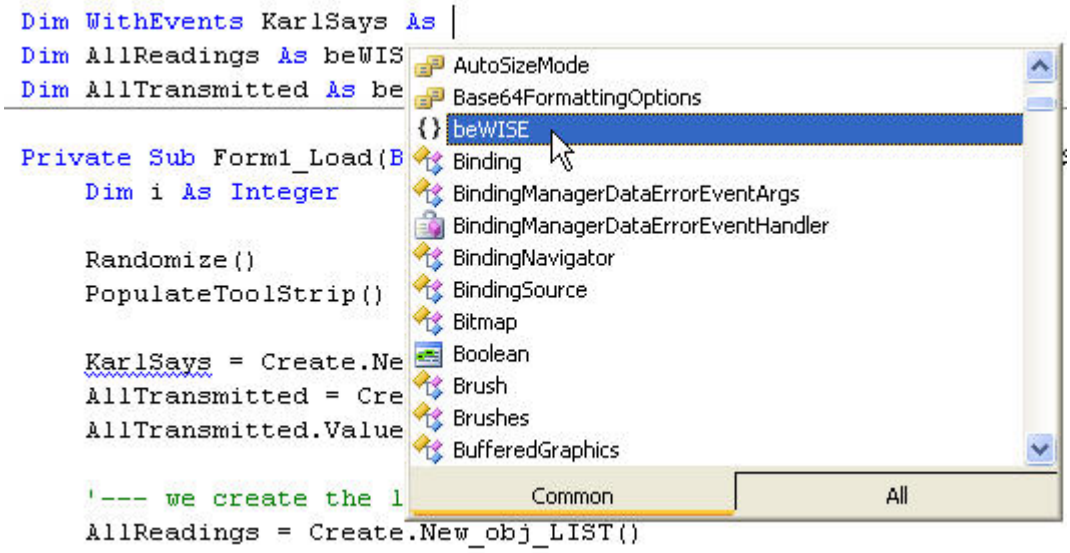
5.3 beWISE VARIABLES

beWISE VARIABLES allow your Visual Basic programs to communicate with each other. Every program can read and update the values of beWISE VARIABLES. Communicating via beWISE VARIABLES is like communicating via walkie-talkies; everyone (every program) can hear what everybody else is saying - as long as everybody is on the same frequency (as long as all programs use the same variables).

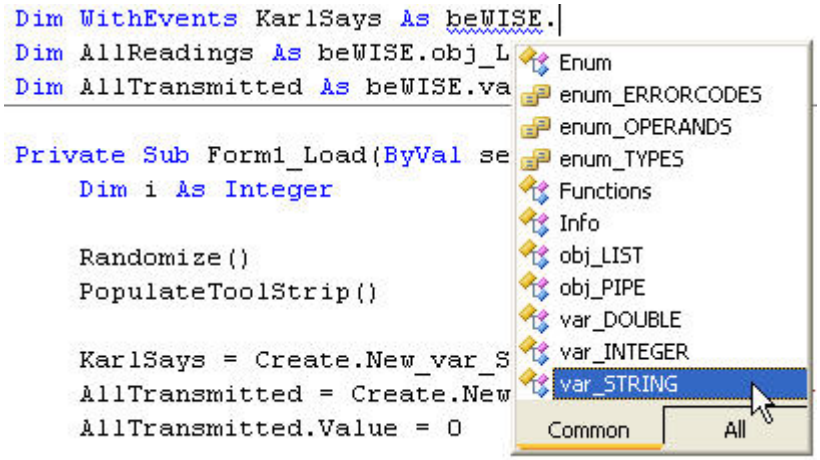
In this chapter we will demonstrate how to declare, create and use beWISE VARIABLES.

5.3.1 Declaring beWISE VARIABLES

To do declare it select the class beWISE after the As clause



Then type a dot (.) to get a list of possible beWISE variable types. They start with var_



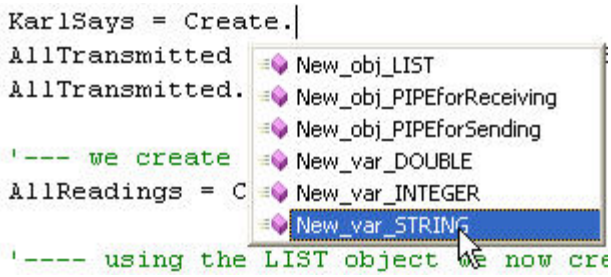
Select the var_*type* that you want for your variable.

In the example below we declare a beWISE variable of type var_STRING.

```
Dim WithEvents KarlSays As beWISE.var_STRING
```

5.3.2 Creating beWISE VARIABLES

After we declare a beWISE variable it does not exists yet, we have to create it first. To create it we use the Create. class (remember that is the class that we have to declare at the beginning of each program; for details see Required Source Code Declarations).



We have to select the correct New_var_*type* function; *type* has to match the data type of our variable. Since KarlSays is defined as STRING, we select the New_var_STRING function.

The syntax is slightly different between VB6 and VB.NET as shown below:

```
KarlSays = Create.New_var_STRING("KARLtoHANS")  'VB.NET Syntax

Set KarlSays = Create.New_var_STRING("KARLtoHANS") 'VB6 Syntax
```

The New_var_*type* functions require a string input parameter. In our above example this parameter is "KARLtoHANS". It is a very important parameter because it designates the name of the beWISE variable - we call this the *beWISEname*.

KarlSays is just the name of our Visual Basic variable. It's only known inside our VB program or project (depending on the scope) but it is not important for beWISE. It can be any name - we call it the *VBname*.

Different programs can share a beWISE variable by creating it with the same *beWISEname*.

5.3.3 Using beWISE VARIABLES

Once we have declared and created a beWISE variable, we can use it. In the following example we assign the value `"Start"` to our variable.

```
KarlSays.Value = "Start"
```

We are using the .Value property of the beWISE variable to set its value.

Using this property we can read and write the value of a variable.

If the variable "KARLtoHANS" is shared by multiple programs, each of these programs will receive a Changed event when the value of the variable changes. The example below demonstrates how such an event handler could be implemented.

```
Private Sub KarlSays_Changed() Handles KarlSays.Changed 'this is VB.NET Syntax
    Select Case KarlSays.Value
        Case "Start"
            Increment.Enabled = True
            Todo.Text = "All right, I start reporting now - OVER!"

        Case "Stop"
            If Increment.Enabled = True Then
                Todo.Text = "OK, I am not reporting anymore - OVER!"
            Else
                Todo.Text = "It's allright, I said I am NOT reporting anymore!"
            End If
            Increment.Enabled = False
    End Select
End Sub
```

5.3.4 Naming Conventions for beWISE VARIABLES

The *beWISEname* is limited to 128 bytes. It can only consist of the numbers 0 to 9, letters A to Z (upper or lower case) and the underscore character "_". It **cannot start** with a number or the underscore character.

Below are examples of valid and invalid beWISEnames:

Valid Names:

```
ExampleString = Create.New_var_STRING("This_cannot_be_longer_than_128_characters")
ExampleInteger = Create.New_var_INTEGER("Employee_Mayer_Age")
ExampleDouble = Create.New_var_DOUBLE("Pump01_Motor_Current_Value3")
```

Invalid Names:

```
ExampleString = Create.New_var_STRING("128_characters_max")
```
Invalid because it starts with a number

```
ExampleInteger = Create.New_var_INTEGER("_Employee_Mayer_Age")
```
Invalid because it starts with the underscore

```
ExampleDouble = Create.New_var_DOUBLE("Pump01.Motor.Current.Value3")
```
Invalid because it uses the character "**.**" which is not allowed.

IMPORTANT: beWISEnames are 'case-sensitive'.

In the following example two *different* beWISE VARIABLES are created:

```
KarlSays1 = Create.New_var_STRING("KARLtoHANS")
KarlSays2 = Create.New_var_STRING("KarlToHans")
```

5.3.5 Supported Data Types

The beWISE VARIABLES support the following data types:

beWISE Type	*Size*	*Visual Basic 6.0 type*	*Visual Basic 2005 type*
`var_STRING`	256 bytes	`String`	`String`
`var_INTEGER`	32 bits	`Long`	`Integer`
`var_DOUBLE`	8 bytes	`Double`	`Double`

5.3.6 Declaration Syntax

Dim [**WithEvents**] *VBname* **As beWISE.***var_type*
Private [**WithEvents**] *VBname* **As beWISE.***var_type*
Public [**WithEvents**] *VBname* **As beWISE.***var_type*

The **Dim** statement syntax has these parts:

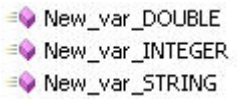
Part	**Description**
WithEvents	Optional. Keyword that specifies if an event handler is installed. If you want to receive Changed events for a beWISE variable, then this keyword has to be specified.
VBname	Required. Name of the Visual Basic variable used to access the beWISE variable.
var_type	Required. Can be **var_DOUBLE, var_INTEGER, var_STRING**

Remarks

Dim, **Private** or **Public** have to be used according to the Visual Basic documentation. Which one to use depends on the scope you want your Visual Basic variable to have.

5.3.7 Create. Functions for beWISE VARIABLES

The following Create. Functions are available for beWISE VARIABLES



Set *VBname* **= Create.New_var_ DOUBLE(***beWISEname***)**
Set *VBname* **= Create.New_var_ INTEGER(***beWISEname***)**
Set *VBname* **= Create.New_var_ STRING(***beWISEname***)**

The **Set** keyword is only required in VB6.

Part	Description
VBname	Required. Name of the Visual Basic variable used to access the beWISE variable.
beWISEname	Required. **STRING** Name of the beWISE VARIABLE. For details regarding the naming convention see Naming Conventions for beWISE VARIABLES.

Remarks

This function creates a Visual Basic variable (actually an object) that allows you to access the beWISE VARIABLE.

Examples

```
Set TellHans = Create.New_var_STRING("KARLtoHANS")
Set UpdateOk = Create.New_var_INTEGER("Ready")
Set MyReading = Create.New_var_DOUBLE("Reading")
```

5.3.8 Methods and Properties

beWISE VARIABLES provide the following methods and properties.

5.3.8.1 Methods

VBname.**FireChanged**

Return Value

None

Remarks

When called this method executes the Changed event handler of this variable in the current program (not in other programs that share this variable). This can be used to execute code that normally would be executed when the value of the variable changes. An example for the use of this function would be at program startup, when you want to execute code in the event handler (e.g. to display the variable) that is otherwise not executed until the variable changes.

5.3.8.2 Properties

VBname.**Name**
VBname.**Value**

Property	Description
Name	(read only). ASCII string representing the *beWISEname* of the variable.
Value	(read/write). Gets or sets the value of the beWISE variable. Depending on the declaration of the variable, the value is either of type **DOUBLE**, **STRING** or **INTEGER** (LONG in VB6).

Remarks

When the Value of a variable changes, the Changed event handler is executed in all programs that have an event handler installed for this variable (including the program changing the variable).

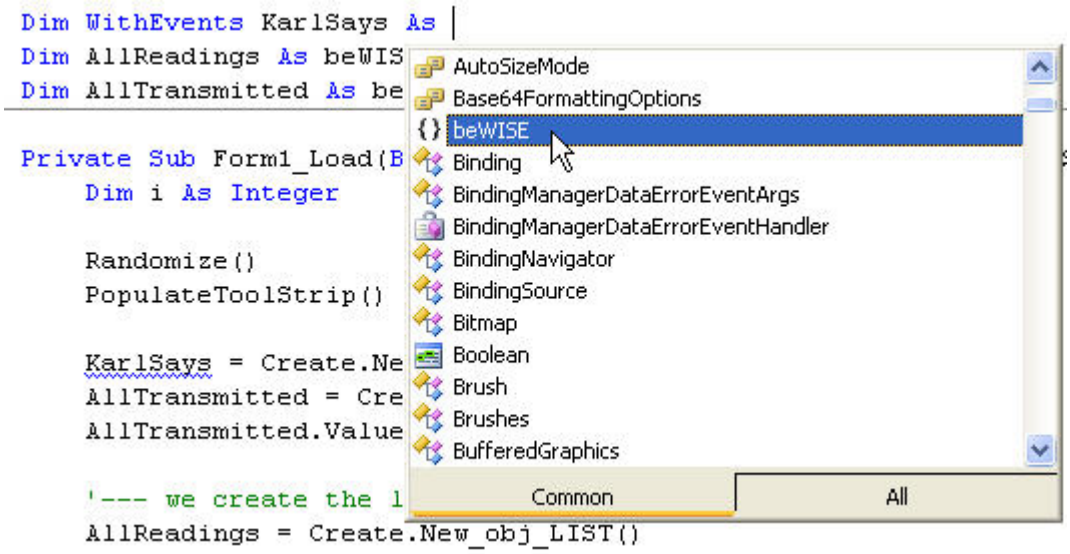
5.4 beWISE LIST Objects

The beWISE LIST Object allows you to dynamically Create and Delete beWISE VARIABLES, to organize and manage them in LISTS, and to store additional properties (dictionary information) with each beWISE VARIABLE. When you have to manage hundreds or even thousands of variables, the beWISE LIST Object comes in very handy.

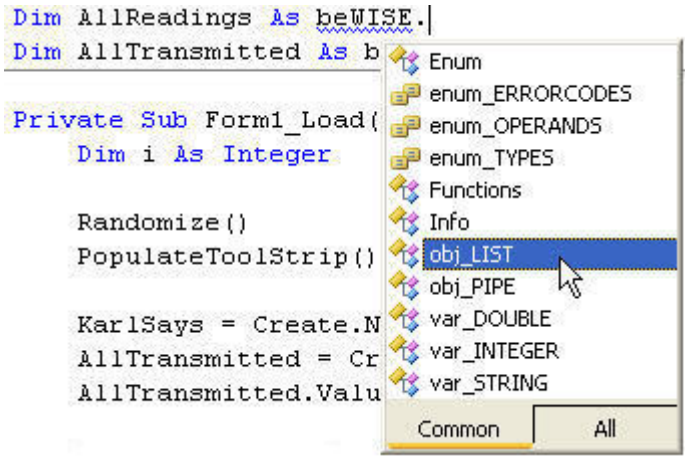
In this chapter we will demonstrate how to declare, create and use beWISE LIST Objects.

5.4.1 Declaring beWISE LIST Objects

To do declare a beWISE LIST Object select the class beWISE after the As clause



Type dot (.) to get a list of possible beWISE Objects (they start with obj_) and select obj_LIST

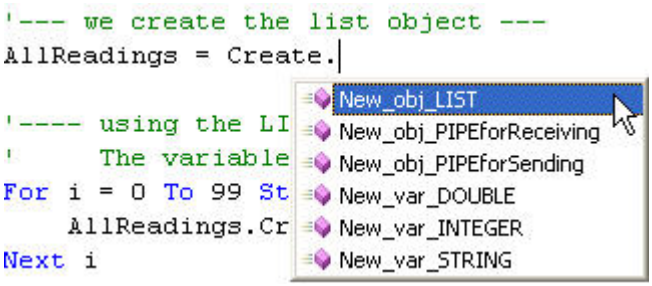


In the example below we declare AllReadings as a beWISE LIST Object.

```
Dim AllReadings As beWISE.obj_LIST
```

5.4.2 Creating beWISE LIST Objects

After we declare a beWISE LIST Object it does not exists yet, we have to create it first. To create it we use the Create. class (remember that is the class that we have to declare at the beginning of each program; for details see Required Source Code Declarations).



The syntax is slightly different between VB6 and VB.NET as shown below:

```
'--- we create the list object ---
AllReadings = Create.New_obj_LIST()   'VB.NET Syntax

'--- we create the list object ---
Set AllReadings = Create.New_obj_LIST() 'VB6 Syntax
```

The New_obj_LIST function has an optional parameter for the 'Filter'. But we don't have to worry about that in the moment. If no filter is specified the LIST will return ALL beWISE VARIABLES.

5.4.3 Using beWISE LIST Objects

Once we have declared and created a beWISE LIST Object, we can use it.

In the following example we use the list object to create 100 beWISE VARIABLES of type DOUBLE. The variables we create have the *beWISEnames* `"Reading00"` through `"Reading99"`. The initial value of these variables is set to 0.0 (a detailed description of the *beWISElist*.Create function can be found in the chapter Create. Function for beWISE LIST Objects).

```
For i = 0 To 99 Step 1
    AllReadings.Create("Reading" & Format(i, "00"), beWISE.enum_TYPES.bwDOUBLE, 0.0)
Next i
```

The next example demonstrates how the filter property can be used to work only with a subset of variables. By setting the filter to `"Reading0"` the LIST Object will return only 10 variables; the

variables "Reading00" through "Reading09". We will 'browse' thru the list, positioning the internal list cursor over each of the 10 variables, adding the name and value of each variable to ListBox1.

```
AllReadings.Filter = "Reading0"    ' set FILTER to the subset we want to see
If AllReadings.FindFirst Then      ' position to first varaible in list
    While AllReadings.varIsValid   ' browse thru list
        ListBox1.Items.Add(AllReadings.varName & Format(AllReadings.varValue, " 00.0"))
        AllReadings.FindNext()     ' position list cursor to NEXT variable in list
    End While                      ' end of list reached
End If
```

5.4.4 Declaration Syntax

Dim [**WithEvents**] *beWISElist* **As beWISE.obj_LIST**
Private [**WithEvents**] *beWISElist* **As beWISE.obj_LIST**
Public [**WithEvents**] *beWISElist* **As beWISE.obj_LIST**

The **Dim** statement syntax has these parts:

Part	Description
WithEvents	Optional. Keyword that specifies if an event handler is installed for the beWISE LIST Object. If you want to receive Changed events for the current beWISE variable (i.e. the beWISE variable currently under the list curser), then this keyword has to be specified.
beWISElist	Required. Name of the Visual Basic variable used to access the beWISE LIST Object.

Remarks

Dim, **Private** or **Public** have to be used according to the Visual Basic documentation. Which one to use depends on the scope you want your Visual Basic variable to have.

5.4.5 Create. Function for beWISE LIST Objects

The following Create. Function is available for beWISE LIST Objects

Set *beWISElist* **= Create.New_obj_ LIST(** [*Filter*] **)**

The **Set** keyword is only required in VB6.

Part	Description
beWISElist	Required. Name of the Visual Basic variable used to access the beWISE LIST Object.
Filter	Optional. **STRING** If specified this contains the partial or full name of a beWISE variable. Naming conventions for beWISE VARIABLES apply. For details regarding the naming conventions see Naming Conventions for beWISE VARIABLES.

Remarks

This function creates a Visual Basic variable (actually an object) that allows you to access the beWISE LIST Object.

If *Filter* is specified *beWISElist* object will only return variables that match *Filter*. Only leading characters are compared (no wildcard characters or regular expression are allowed).

If *Filter* specifies the full name (*beWISEname*) of an existing variable, the *beWISElist* object will only return this one variable.

If *Filter* omitted, empty (“”) or **vbNullString**, the *beWISElist* object will return ALL existing beWISE VARIABLES.

Examples

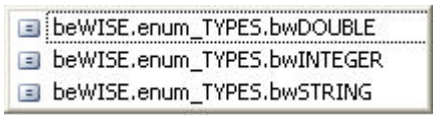
```
Set AllReadings = Create.New_obj_LIST("Reading")
Set AllVariables1 = Create.New_obj_LIST()
Set AllVariables2 = Create.New_obj_LIST("")
Set AllVariables3 = Create.New_obj_LIST(vbNullString)
Set Just1Variable = Create.New_obj_LIST("KARLtoHANS")
```

5.4.6 Methods and Properties

5.4.6.1 Methods

*beWISElist***.Create(** *beWISEname, beWISEtype, InitialValue* **)**

The **Create** method has these parameters:

Parameter	Description
beWISEname	Required. **STRING** Name of the beWISE VARIABLE. For details regarding the naming convention see Naming Conventions for beWISE VARIABLES.
beWISEtype	Required. **beWISE.enum_TYPES**  Determines the type of variable being created.
InitialValue	Required. **Variant** (VB6) / **Object** (VB.NET) Initial value of the variable. The value has to be of same type as defined in *beWISEtype*. For variables of type **bwSTRING** a value of **vbNullString** is ***NOT allowed***, however a zero length string ("") is allowed.

Return Value

Returns a **Boolean** value indicating whether this was the *INITIAL create* of the variable or not.

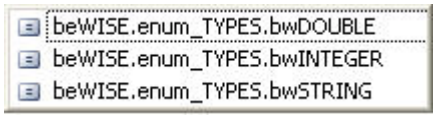
Remarks

If the method returns **True** then the variable was created for the very first time in the system (*INITIAL create*) with the *beWISEtype* and *InitialValue* assigned to it. The current *list cursor* is set to this variable, regardless of any *Filter* that may be specified. The ***.varIsValid*** property is set to **True**.

If **False** then the variable existed already and *beWISEtype* and *InitialValue* are ignored. This does not indicate an error. It only indicates that the variable existed already and may have a different type and value.
To check for any error condition use the ***.varIsValid*** property. If ***.varIsValid*** is **True** then the current *list cursor* points to this variable. In this case the actual type and value of the variable can be determined using the ***.varType*** and ***.varValue*** properties.
If ***.varIsValid*** is **False** then an error occurred and the variable could not be created. The most likely cause is an invalid *beWISEname*.

beWISElist.**CreateHandle(** *beWISEname, beWISEtype, InitialValue* **)**

The **CreateHandle** method has these parameters:

Parameter	Description
beWISEname	Required. **STRING** Name of the beWISE VARIABLE. For details regarding the naming convention see Naming Conventions for beWISE VARIABLES.
beWISEtype	Required. **beWISE.enum_TYPES**  Determines the type of variable being created.
InitialValue	Required. **Variant** (VB6) / **Object** (VB.NET) Initial value of the variable. The value has to be of same type as defined in *beWISEtype*. For variables of type **bwSTRING** a value of **vbNullString** is ***NOT allowed***, however a zero length string ("") is allowed.

Return Value

Returns an **INTEGER** value (Long in VB6) representing the system-wide (computer-wide) unique *handle* of the beWISE VARIABLE.

Remarks

If successful the method returns a positive number representing the system-wide (computer-wide) unique *handle* of the beWISE VARIABLE. This number (*handle*) can be used with the ***.FindHandle*** method to position the *list cursor* to this variable.
If a handle is returned the ***.varIsValid*** property is set to **True** and the current *list cursor* is set to this variable, regardless of any *Filter* that may be specified.

If unsuccessful the method returns a negative number and the ***.varIsValid*** property is **False**. The *list cursor* is invalid.

No indication is given if this was an *INITIAL create* or not. The application has to make sure the variable under the cursor has the desired data type by checking the ***.varType*** property.

beWISElist.**DELETE()**

Return Value

Returns a **Boolean** value indicating whether or not the current variable was successfully deleted.

Remarks

If the method returns **True** then the variable under the current *list cursor* was successfully deleted. The ***.varIsValid*** property is set to **False** and the *list cursor* is invalid.

If the method returns **False** then an error occurred during the DELETE operation. The most likely cause is, that variable has been deleted by another program. The ***.varIsValid*** property is set to **False** and the *list cursor* is invalid.

beWISElist.**FindFirst()**

Return Value

Returns a **Boolean** value indicating whether a variable was found or not.

Remarks

To understand this function it is important to know, that the variables in a beWISE LIST Object are sorted alphabetically in ascending order by the name of the variable (by *beWISEname*).

When called this method positions the cursor to the ***First*** beWISE VARIABLE that matches *Filter*. If no *Filter* is specified or *Filter* is empty (""), then *FindFirst()* positions the cursor to the very first variable.

If the method returns **True** then a variable was found. The current *list cursor* is set to this variable and the ***.varIsValid*** property is set to **True**.

If the method returns **False** then the no variable was found. The ***.varIsValid*** property is set to **False** and the *list cursor* is invalid.

Since variables can be created and deleted dynamically at all times and by any program, it is possible that consecutive calls of the *FindFirst()* method return different results.

beWISElist.**FindHandle(** *beWISEhandle* **)**

The **FindHandle** method has these parameters:

Parameter	Description
beWISEhandle	Required. **Integer** (VB.NET) / **Long** (VB6) Number representing the system-wide (computer-wide) unique *handle* of the beWISE VARIABLE.

Return Value

Returns a **Boolean** value indicating whether the variable identified by *beWISEhandle* was found or not.

Remarks

When called this method positions the *list cursor* to the beWISE variable that is associated with the *beWISEhandle*.

If the method returns **True** then the variable was found. The current *list cursor* is set to the variable and the ***.varIsValid*** property is set to **True**.

If the method returns **False** then the variable was not found. The ***.varIsValid*** property is set to **False** and the *list cursor* is invalid.

beWISElist.**FindNext()**

Return Value

Returns a **Boolean** value indicating whether a *next* variable was found.

Remarks

To understand this function it is important to know, that the variables in a beWISE LIST Object are sorted alphabetically in ascending order by the name of the variable (by *beWISEname*).

When called this method positions the cursor to the ***Next*** beWISE VARIABLE that matches *Filter* if *Filter* is specified. If no *Filter* is specified the method positions the cursor to the ***Next*** beWISE VARIABLE in the system.

If the method returns **True** then a variable was found. The current *list cursor* is set to this variable and the ***.varIsValid*** property is set to **True**.

If the method returns **False** then the no variable was found. The ***.varIsValid*** property is set to **False** and the *list cursor* is invalid.

The *FindNext ()* method works only in conjunction the *FindFirst()* method. A successful call to *FindFirst()* is required prior to any call to *FindNext()*. Consecutive calls to *FindNext()* are allowed.

The following example shows how *FindFirst()* and *FindNext()* can be used to browse through a list of variables, positioning the internal *list cursor* over each of the variables and adding the name and value of each variable to ListBox1.

```
AllReadings.Filter = "Reading0"    ' set FILTER to the subset we want to see
If AllReadings.FindFirst Then      ' position to first varaible in list
    While AllReadings.varIsValid   ' browse thru list
        ListBox1.Items.Add(AllReadings.varName & Format(AllReadings.varValue, " 00.0"))
        AllReadings.FindNext()     ' position list cursor to NEXT variable in list
    End While                      ' end of list reached
End If
```

beWISElist.**FireChanged**

Return Value

None

Remarks

When called this method executes the Changed event handler of the list in the current program (not in other programs that share this variable). This can be used to execute code that normally would be executed when the value of the variable changes. An example for the use of this function would be at program startup, when you want to execute code in the event handler (e.g. to display the variable) that is otherwise not executed until the variable changes.

beWISElist.**SetProperties(***Description, RefreshRate, Resolution, MaxValue, MinValue***)**

The **SetProperties** method has these parameters:

Parameter	Description
Description	Required. **String** (256 bytes maximum) Sets the *.varDescription* property of the *current variable*.
RefreshRate	Required. **Integer** (Long in VB6) Sets the *.varRefreshRate* property of the *current variable*.
Resolution	Required. **Integer** (Long in VB6) Sets the *.varResolution* property of the *current variable*.
MaxValue	Required. **Double** Sets the *.varMaxValue* property of the *current variable*.
MinValue	Required. **Double** Sets the *.varMinValue* property of the *current variable*.

Return Value

Returns a **Boolean** value indicating whether the function was successful and all properties have been set.

Remarks

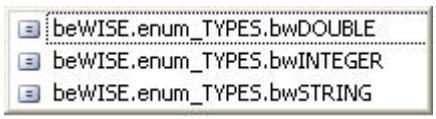
This method is used to set all *informational properties* of beWISE VARIABLEs.

Informational properties allow the programmer to store additional information with each variable. They are not processed by beWISE in any way.

5.4.6.2 Properties

beWISElist.**Filter** ..... partial or full name of a beWISE variable
beWISElist.**varDescription** ..... additional description for current variable (*informational*)
beWISElist.**varHandle** ..... system-wide unique handle of current variable
beWISElist.**varInitValue** ..... initial value of current variable
beWISElist.**varIsValid** ..... indicates whether the list cursor has a current variable
beWISElist.**varMaxValue** ..... maximum allowed value of current variable (*informational*)
beWISElist.**varMinValue** ..... minimum allowed value of current variable (*informational*)
beWISElist.**varName** ..... name of the current variable
beWISElist.**varRefreshRate** ..... refresh rate of current variable (*informational*)
beWISElist.**varResolution** ..... resolution of current variable (*informational*)
beWISElist.**varType** ..... data type of the current variable
beWISElist.**varValue** ..... value of the current variable

Property	**Description**
Filter	(read/write). **String** The partial or full name of a beWISE variable. *Filter* is compared against the *beWISEname* of all existing variables. Only leading characters are compared (no wildcard characters or regular expression are allowed). The *beWISElist* object will return ALL variables that MATCH *Filter*. If no variables match *Filter* the *beWISElist* object will return NO variables. If *Filter* specifies the full name (*beWISEname*) of an existing variable, the *beWISElist* object will return only this ONE variable. If *Filter* is empty (""), the *beWISElist* object will return ALL existing variables. *Filter* can be changed at any time to change the content of the *beWISElist* object (i.e. the variables that it will be return). When *Filter* is changed the **FindFirst()** method is automatically invoked to position the *beWISElist* object to the first variable. If no first variable can be found (if the list is empty) the ***.varIsValid*** property of the *beWISElist* object is **False** and the *list cursor* is invalid. If a first variable is found the ***.varIsValid*** property of the *beWISElist* object is **True** and the *list cursor* is set to this variable. The naming conventions of beWISE VARIABLES apply for *Filter*. For details regarding the naming conventions see Naming Conventions for beWISE VARIABLES.
varDescription	(read only). **String**

	varDescription of the *current variable*. This is an *informational* property.
varHandle	(read only). **Integer** (Long in VB6) System-wide (computer-wide) unique *handle* of the *current variable*. The *handle* can be used with the ***.FindHandle*** method to position the *list cursor* to this variable. A typical application would be to store the handles of variables (hidden from the user) with the items of a ListBox that displays these variables. The *handle* can then be used to quickly find the variable that each item in the ListBox represents.
varInitValue	(read only). Initial value of the *current variable*. This is the value that the variable received the very first time when it was created in the system. Depending on the declaration of the variable, the value is either **Double**, **String** or **Integer** (Long in VB6).
varIsValid	(read only). **Boolean** Indicates whether the *list cursor* points to a valid variable (i.e. the list has a *current* variable). If **True** the list *has a current variable*. If **False** the list does *not have a current variable*.
varMaxValue	(read only). **Double** varMaxValue of the *current variable*. This is an *informational* property.
varMinValue	(read only). **Double** varMinValue of the *current variable*. This is an *informational* property.
varName	(read only). **String** The *beWISEname* of the current variable.
varRefreshRate	(read only). **Integer** (Long in VB6) varRefreshRate of the *current variable*. This is an *informational* property.
varResolution	(read only). **Integer** (Long in VB6) varResolution of the *current variable*. This is an *informational* property.
varType	(read only). The type of the *current variable*. The following types are possible: 
varValue	(read/write). Gets or Sets the value of the *current variable*. Depending on the declaration of the variable, the value is either **Double**, **String** or **Integer** (Long in VB6).

5.5 beWISE PIPE Objects

beWISE PIPE Objects allow message passing between applications. A beWISE PIPE is unidirectional, i.e. it can pass messages only in one direction. One program (the RECEIVER) owns the PIPE, other programs (SENDERS) can send massages to the RECEIVER via this PIPE.

Since beWISE PIPES are unidirectional, each application needs its own RECEIVER PIPE in order for programs to exchange messages bidirectional. A program can create one ore more RECEIVER PIPEs.

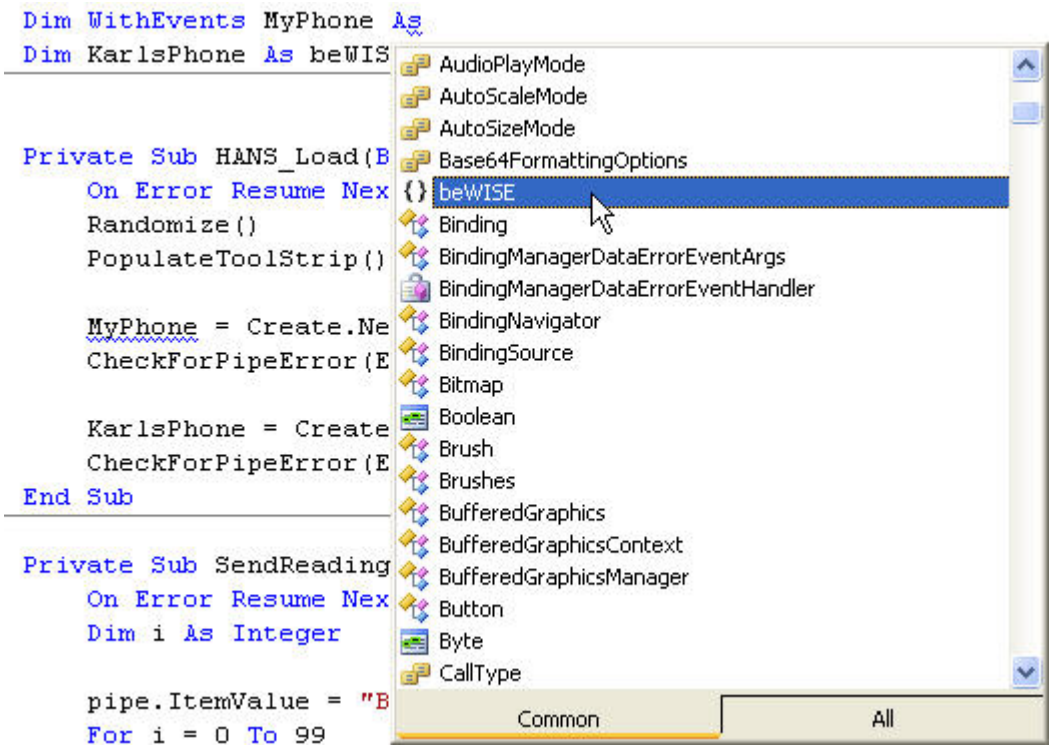
To send messages to other programs an application has to create SENDER PIPES with the names of these PIPES (beWISE PIPES are *named pipes*). Applications have to create SENDER PIPEs for each program they want to send messages to.

PIPEs are implemented as FIFO (First In, First Out).

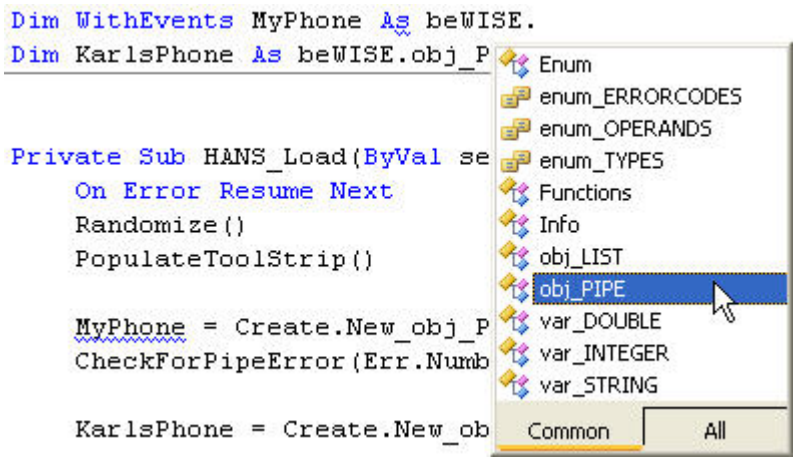
In this chapter we will demonstrate how to declare, create and use beWISE PIPE Objects.

5.5.1 Declaring beWISE PIPE Objects

To do declare a beWISE PIPE Object select the class beWISE after the As clause



Type dot (.) to get a list of possible beWISE Objects (they start with obj_) and select obj_PIPE



In the example below we declare MyPhone as a beWISE PIPE Object.

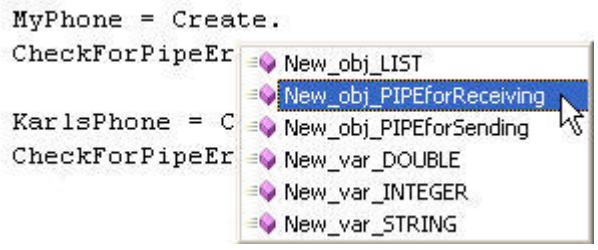
```
Dim WithEvents MyPhone As beWISE.obj_PIPE
```

5.5.2 Creating beWISE PIPE Objects

After we declare a beWISE PIPE Object it does not exists yet, we have to create it first. To create it we use the Create. class (remember that is the class that we have to declare at the beginning of each program; for details see Required Source Code Declarations).

5.5.2.1 Creating a RECEIVER PIPE

In the following example we create a RECEIVER PIPE. This pipe will allow us to receive messages that other programs send us.



The syntax is slightly different between VB6 and VB.NET as shown below:

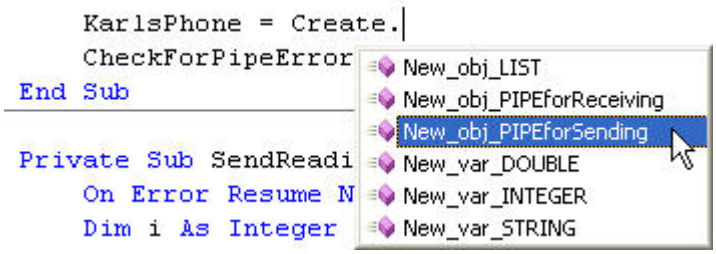
```
'--- we create the pipe object ---
MyPhone = Create.New_obj_PIPEforReceiving("HANS") 'VB.NET Syntax

'--- we create the pipe object ---
Set MyPhone = Create.New_obj_PIPEforReceiving("HANS") 'VB6 Syntax
```

The only required parameter for the New_obj_PIPEforReceiving function is *PipeName*. The other parameters are optional and only necessary if we want to fine-tune our pipe. The default settings for these parameters are sufficient for most applications.

5.5.2.2 Creating a SENDER PIPE

In the following example we create a SENDER PIPE. This pipe will allow us to send messages to the RECEIVER of the PIPE named “KARL”. The RECEIVER is the program that created the pipe “KARL” using the New_obj_PIPEforReceiving function.



The syntax is slightly different between VB6 and VB.NET as shown below:

```
'--- we create the pipe object ---
KarlsPhone = Create.New_obj_PIPEforSending("KARL") 'VB.NET Syntax

'--- we create the pipe object ---
Set KarlsPhone = Create.New_obj_ PIPEforSending("KARL") 'VB6 Syntax
```

The only required parameter for the New_obj_PIPEforSending function is *PipeName*. The other parameters are optional and only necessary if we want to fine-tune our pipe. The default settings for these parameters are sufficient for most applications.

Before we can use *KarlsPhone* it has to be declared as a pipe object:

```
Dim KarlsPhone As beWISE.obj_PIPE
```

5.5.3 Using beWISE PIPE Objects

Once we have declared and created a beWISE PIPE Object, we can use it.

In the following example demonstrates how program HANS sends String and Double items thru the pipe named “KARL” to the *receiver* of that pipe.

Program HANS:

```
Dim KarlsPhone As beWISE.obj_PIPE
...

Private Sub Form_Load(...
    KarlsPhone = Create.New_obj_PIPEforSending("KARL") 'that is KARLS's pipe

    KarlsPhone.ItemValue = "BEGIN"                ' put "START" into the pipe
    For i = 0 To 99                               ' send 100 random numbers
        KarlsPhone.ItemValue = CDbl(Rnd() * 99)   ' put a random number into pipe
    Next i
    KarlsPhone.ItemValue = "END"                  ' put "END" into the pipe
    KarlsPhone.SendTrigger()                      ' wake-up trigger
End Sub
```

Below we see how program KARL creates the pipe named “KARL” as *receiver pipe* and picks up the items that program HANS sent into this pipe. When KARL receives the “BEGIN” item he clears ListBox1. When he receives the “END” item he adds `"All Values Received -----"` to ListBox1. In all other cases he adds *the random number* to ListBox1.

Program KARL:

```
Dim WithEvents MyPhone As beWISE.obj_PIPE
...
Private Sub Form_Load(...

    MyPhone = Create.New_obj_PIPEforReceiving("KARL") 'that is MY pipe

End Sub
                              'VB.NET syntax
Private Sub MyPhone_Changed() Handles MyPhone.Changed 'we have data in our pipe
Dim message As Object

    While (MyPhone.PipeIsEmpty = False)      'while pipe not empty
        message = MyPhone.ItemValue          'get item from pipe

        If message = "BEGIN" Then
            ListBox1.Items.Clear()
        ElseIf message = "END" Then
            ListBox1.Items.Add("All Values Received -----")
        Else
            ListBox1.Items.Add(Format(message, "000.00"))
        End If

    End While

End Sub
```

5.5.4 Naming Conventions for beWISE PIPE Objects

The name of a beWISE PIPE is limited to 100 bytes. It can only consist of the numbers 0 to 9, letters A to Z (upper or lower case) and the underscore character "_". It **cannot start** with a number or the underscore character.

Below are examples of valid and invalid beWISEnames:

Valid Names:

```
Example1 = Create.New_obj_PIPEforSending("This_cannot_be_longer_than_100_characters")
Example2 = Create.New_obj_PIPEforReceiving("HANS_Pipe1")
Example3 = Create.New_obj_PIPEforReceiving("KARL")
```

Invalid Names:

```
Example1 = Create.New_obj_PIPEforSending("128_characters_max")
```
Invalid because it starts with a number

```
Example2 = Create.New_obj_PIPEforReceiving("_HANS_Pipe1")
```
Invalid because it starts with the underscore

```
Example3 = Create.New_obj_PIPEforReceiving("HANS.Pipe1")
```
Invalid because it uses the character "**.**" which is not allowed.

IMPORTANT: beWISE PIPE names are 'case-sensitive'.

In the following example two *different* beWISE PIPES are created:

```
Example1 = Create.New_obj_PIPEforReceiving("HANS")
Example2 = Create.New_obj_PIPEforReceiving("Hans")
```

5.5.5 Lifetime of beWISE PIPE Objects

A beWISE PIPE starts to exist as soon as the RECEIVER creates the pipe. SENDER programs that start up before the RECEIVER can successfully call the New_obj_PIPEforSending function for this pipe, but they cannot send items thru the pipe until the RECEIVER has first created the pipe. Any attempt to send before the pipe exists will raise error #5 in the sender program.

The pipe exists as long as the receiver program or any sender program is active. Sender programs will be able to send items thru this pipe even if the receiver program stops. This allows e.g. the receiver to recover from errors (e.g. thru microreboot) without effecting the sender programs.

As soon as the receiver program and all sender programs stop, the pipe seizes to exist.

5.5.6 Declaration Syntax

Dim [**WithEvents**] *beWISEpipe* **As beWISE.obj_PIPE**
Private [**WithEvents**] *beWISEpipe* **As beWISE.obj_PIPE**
Public [**WithEvents**] *beWISEpipe* **As beWISE.obj_PIPE**

The **Dim** statement syntax has these parts:

Part	Description
WithEvents	Optional. Keyword that specifies if an event handler is installed for the beWISE LIST Object. This is only useful for pipes that are later created as RECEIVER PIPES. It allows the Changed event handler to be executed when messages are received for the pipe.
beWISEpipe	Required. Name of the Visual Basic variable used to access the beWISE PIPE Object.

Remarks

Dim, **Private** or **Public** have to be used according to the Visual Basic documentation. Which one to use depends on the scope you want your Visual Basic variable to have.

When the pipe is *declared* using the **WithEvents** keyword, a Changed event handler is installed for this pipe. The Changed event handler works only for pipes that are later *created* as receiver pipes, i.e. pipes that are *created* with the Create.New_obj_PIPEforReceiving() method.

The Changed event handler is called whenever the status of the pipe changes from **EMPTY** to **NOT EMPTY**, i.e. when the pipe *is empty* and one ore more items arrive.

The Changed event handler is *not activated* when the pipe has already one or more items and new items arrive. Therefore ALWAYS make sure to completely empty the pipe before you exit the event handler function, otherwise your program will not be notified anymore when new items arrive. The example below demonstrates the correct implementation:

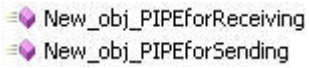
```
                              'VB.NET syntax
Private Sub MyPhone_Changed() Handles MyPhone.Changed 'we have data in our pipe
Dim message As Object

    While (MyPhone.PipeIsEmpty = False)      'while pipe not empty
        message = MyPhone.ItemValue          'remove item from pipe
       'message hast the item now;
       'do your processing here ...
    End While

End Sub
```

5.5.7 Create. Function for beWISE PIPE Objects

The following Create. Function is available for beWISE PIPE Objects



Set *beWISEpipe* **= Create.New_obj_PIPEforReceiving(** *Name*, [*MaxItems*], [*MaxSender*] **)**
Set *beWISEpipe* **= Create.New_obj_PIPEforSending(** *Name*, [*MaxItems*], [*MaxSender*] **)**

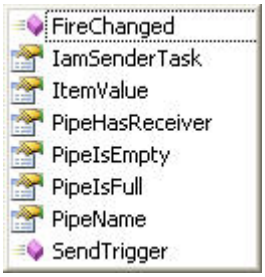
The **Set** keyword is only required in VB6.

Part	**Description**
Name	Required. **String** Name of the beWISE PIPE. For details regarding the naming conventions see Naming Conventions for beWISE PIPE Objects.
MaxItems	Optional. **Integer** (Long in VB6) **Create.New_obj_PIPEforReceiving:** Specifies the maximum number of items the pipe can buffer. If the receiver does not pick up the items from the pipe and sender programs continue inserting items into the pipe, the pipe will buffer up to *MaxItems* items before the pipe status will change to **FULL**. When the pipe status is set to **FULL** any attempt to insert further items will raise the error condition #6 (Overflow) for the sender programs. Sender and receiver programs can check the status of the pipe using the **.PipeIsFull** property. **Create.New_obj_PIPEforSending:** Currently not implemented.
MaxItems	Currently not implemented.

Remarks

The functions create beWISE PIPES for receiving or sending.

5.5.8 Methods and Properties



5.5.8.1 Methods

beWISEpipe.**FireChanged**

Return Value

None

Remarks

When called this method executes the Changed event handler of the pipe in the current program. This can be used to execute code that normally would only be executed when the status of the pipe changes from EMPTY to NOT EMPTY.

beWISEpipe.**SendTrigger**

Return Value

Returns a **Boolean** value indicating whether the pipe had items (**True**) or was empty (**False**).

Remarks

When called this method activates the Changed event handler in the receiver of the pipe. This will allow the receiver to read the items from the pipe without having to poll the pipe status constantly.

The method can only be used with sender pipes. When called for receiver pipes this method will raise the error condition #5 (Invalid call).

5.5.8.2 Properties

beWISEpipe.**IamSenderTask**
beWISEpipe.**ItemValue**
beWISEpipe.**PipeHasReceiver**
beWISEpipe.**PipeIsEmpty**
beWISEpipe.**PipeIsFull**
beWISEpipe.**PipeName**

Property	**Description**
IamSenderTask	(read only). **Boolean** Indicates whether the current program (the current task) has created this pipe as a sender pipe (**True**) or receiver pipe (**False**).
ItemValue	(read for receiver pipes / write for sender pipes). The following types are supported **Double**, **String, Integer** (Long in VB6). Allows sending or receiving items thru the pipe. Assigning a value to ItemValue will insert this value as an item into the pipe. **String** items cannot be longer than 2048 characters. Assigning ItemValue to another variable or using it in statements (e.g. If, Select Case) will remove the item from the pipe. If want to use ItemValue in multiple IF and ELSEIF statements, you have to assign it to a local variable (of type Object or Variant) and use this local variable instead.
PipeHasReceiver	(read only). **Boolean** Indicates whether the receiver program has already created the pipe and thus put the pipe into existence. Returns **True** if the pipe exists already. **False** if the receiver has not yet created the pipe. For details see about the lifetime of pipes see Lifetime of beWISE PIPE Objects.
PipeIsEmpty	(read only). **Boolean** Indicates whether the pipe is empty (has 0 items in it) or not. Returns **True** if the pipe is empty and **False** if the pipe has at least one item.
PipeIsFull	(read only). **Boolean** Indicates whether the pipe is full (number of items in pipe = *MaxItems*) and therefore cannot accept any more items. Returns **True** if the pipe is full and **False** if the pipe has at least one empty slot.
PipeName	(read only). **String** Name of the beWISE PIPE. For details regarding the naming conventions see Naming Conventions for beWISE PIPE Objects.

6 PRODUCT REGISTRATION

6.1 Obtaining a Registration Key

You have a 14 day evaluation period for this product. If you want to continue to use it beyond the 14 day evaluation period, you have to purchase a registration key.

For more information on how to obtain a registration key for a beWISE product please visit our web-site at www.edelwise.com

6.2 Receiving Your Registration Information

Shortly after purchasing the registration key you will receive an e-mail that contains your registration information, similar to the one shown below:

```
------------------- REGISTRATION INFORMATION -----------------

Enter your name and registration key exactly as shown below to
activate the license as soon as the program asks you for it.
(Press the "Enter Key" button when asked for the license).

Name: your name
Key:  000015-T8HG2X-42UKWX-Z2GH41-5SDQ0U-4AK5CU-07JFT3-3AY3EQ-0W3AER-Q8F9KM

------------------------------------------------------------
```

6.3 Entering Your Registration Information

Four (4) days into the evaluation period the following registration reminder begins to appear:

To continue in evaluation mode press the "OK" button. Pressing the "Enter Key" button opens a new screen that allows you to enter your registration information. Please enter the 'Name' and 'Key' information exactly as it appears in your e-mail (use cut and paste if possible).

You can open the registration dialog box *at any time* by starting the program beWISEreg.exe with the parameter REGISTER (`.\Edelwise\beWISEreg.exe REGISTER`)

If you have Visual Studio 2005 installed you can simply use the beWISE Dashboard to enter your registration information.

7 DISTRIBUTING YOUR APPLICATION WITH beWISE

7.1 Required Runtime Files

Note: In the description below beWISExxx.dll needs to be replaced with the actual name of the dll that you have. The name of the dll depends on the beWISE product you have.

DLL name	*product name*
beWISEbas.dll	beWISE BASIC
beWISEadv.dll	beWISE ADVANCED
beWISEpro.dll	beWISE PROFESSIONAL
beWISElan.dll	beWISE LAN

When distributing the beWISExxx.dll as part your product you have to include the following files in your distribution package and copy them into the system32 directory of the target computer:

```
beWISExxx.dll        ... this file needs to be registered (regsvr32 beWISExxx.dll)
nodectr.exe
taskcln.exe
rtdbdef.exe
wntlogsrv.exe
rtshare.dll
rtvbvar.dll
MSVCRTD.DLL
beWISEreg.exe
beWISEreboot.exe
beWISEstop.exe
```

You can find most of these files in your system32 directory and also in the beWISE.CAB file that came with the installation. The files beWISEreg.exe and beWISEstop.exe can be found in the directory C:\Program Files\Edelwise\

7.2 Silent Registration of beWISE During Your Installation

When you deploy beWISE components together with your application you need to register these components during the installation process. You can use the beWISEreg.exe program to do that using the following syntax:

```
beWISEreg QUIETREGISTER your name, your key
```

Enter the 'Name' and 'Key' information exactly as it appears in your e-mail. You can use your own licensing information for deployment of run-time versions of your product as long as your product is not used for software development. For details please see the 'Software License Agreement' that came with this product.