# W/32 OLE Automation

CET BASIC OLE allows user to manipulate OLE automation objects, such as MS WORD, from BASIC program. W/32 OLE Automation support consists of set of functions callable with CALL statement. These functions get BASIC programmer possibility to create, manipulate and destroy OLE Automation objects, in general, same way as from Visual Basic programs.

## *Function set*

**cetOleCreate( ADDROF(Object%), Name$ )**

Creates new OLE object **Name$** and returns reference to it in variable **Object%**

**cetOleGet( ADDROF(Object%), Name$ )**

Returns reference to existing OLE object **Name$** or creates new one, if no such object exists.

**cetOleDestroy( Object% )**

Destroys reference to OLE object (but does not destroys object itself), and all references to child objects

**cetOleDestroyChilds( Object% )**

Destroys reference to all child objects of **Object%**

**cetOleExecute( Object%, Method$, ADDROF({result%|result$|result}|0 ),...,{Count%|Types$} )**

Executes method **Method$** of OLE object **Object%**. Returns result in **result{%|$}**. If no results are expected, of to ignore result value, pass 0(zero) as third argument.
Forth and all subsequent arguments are method parameters. Function accepts variable number of arguments, argument counter must be passed as last argument. If no argument is required, argument counter 0(zero) must be passed.
Argument counter can be either integer value or string value. String argument counter is used to define types of method arguments, if they types must be converted some non-standard way.
Standard type conversion rules:
- integer argument (arg%) is passed to OLE object as Visual Basic type Long.
– decimal argument (arg) is passed to OLE object as Visual Basic type Double.
– string argument (arg$) is passed to OLE object as Visual Basic type String.
To pass arguments of different type, **count$** value must contain type definitions for arguments, one character for argument:
h - pass CET BASIC integer value as Visual Basic Short.
l - pass CET BASIC integer value as Visual Basic Long.
f - pass CET BASIC integer value as Visual Basic Float.
d - pass CET BASIC integer value as Visual Basic Double.
b - pass CET BASIC integer value as Visual Basic Boolean.
s - pass CET BASIC integer value as Visual Basic String.
o - pass CET BASIC integer value as Visual Basic Object.
In general, only ‘o’ conversion is required. All other conversions can be done automatically by OLE object.

**cetOleGetProperty( Object%, Property$, ADDROF({value%|value$|value}) )**

Returns value of **Property$** of OLE object **Object%** in variable **value{%|$}**. This call is used to get a property that does not require parameters. Properties with parameters (see **EXC** example below) must be retrieved with **cetOleExecute** call, while argument counter must be negative integer value to indicate that it is a property, not method.

**cetOleSetProperty( Object%, Property$, {value%|value$|value} )**

Sets new property value

**cetOleErrorInfo( ADDROF(code%), ADDROF(sysmsg$), ADDROF(objmsg$) )**

Returns information about last OLE error detected. This function can be used to retrieve detail information about last OLE error (err = 14006). **Code%** is the error code, **sysmsg$** is the system message associated with error code, **objmsg$** is the OLE object message, if any.

## *Error handling*

All the functions can raise BASIC errors with codes 14001..14007:

**14001 Ole: bad parameter type**
One of the obligatory arguments has wrong type

**14002 Ole: bad object reference**
**Object%** argument is not referenced to valid OLE object

**14003 Ole: bad type in Type$ argument**
Invalid character in **type$** argument

**14004 Ole: type conversion**
Argument passed can not be converted to required Visual Basic type

**14005 Ole: bad return type conversion**
Return value can not be converted from Visual Basic type

**14006 Ole: Internal COM/OLE error**
Use **cetOleErrorInfo** to get additional information

## *Notes*

1. In most cases, arguments of **cetOleExecute** and return value are converted to/from Visual Basic types automatically. Only when you pass integer value as reference to OLE object to **cetOleExecute** call. In this case you must explicitly define argument type as type 'o' in **Types$** argument.
2. OLE functions keep track of child objects returned from methods and property values. If you create many child objects, it can take some memory, so it would be good to call **cetOleDestroyChilds** from time to time. This function **does not** delete actual objects, only invalidates references. See EXC example.
3. **CetOleDestroy** destroys reference to main object, created with **cetOleCreate** or **cetOleGet**. It destroys all the child objects of this object as well as parent object. Again, only references will be destroyed, not objects.
4. Event handling is not supported yet. Currently there is no way to synchronize user activity in OLE object with BASIC program.
5. Enumerators and constants are not supported yet. Use integer variables or constants to pass constant values.

## *Examples*

The following examples are installed into CETSAMP\OLE directory. To compile the example type **Bl <name>** in the command line prompt, while SETSAMP\OLE is your current directory.
All the examples contain equivalent Visual Basic statements in REM statements.
All the examples are converted, with small changes, from original Visual Basic sources from the book "Using Visual Basic 5.0", 1997 QUE Corporation.

**Creating Microsoft Word Document - WORD.B**
The program creates MS Word object, creates new document, types some text into it, then, after user confirmation, closes Word without saving changes.
Please note that program calls **cetOleDestroy** even in case of error. It must be done to destroy all references to OLE object.

**Creating Microsoft Excel document – EXC.B**
This program creates MS Excel object, creates new table, fills table with some values, then opens chart view for this table. Finally program marks table as if it was not changed, to prevent Excel to ask user to save changes, and closes Excel. Please note **cetOleDestroyChild** calls. It is not required, but when program intensive creates child objects (for example, if you create big table), it helps keep size of internal table small.