ActiveX Guide

ActiveX Overview

Attention!
This module is licensed separately and can be not included in your package.

ActiveX allows to connect to the satellite tracking server Wialon™ through TCP/IP session (Internet or LAN) from different OLE applications such as 1C, Excel, Visual Basic, Word, or any HTML page.

ActiveX can be used also to create client connections which work with Wialon™ from different IDE such as Visual C++, Visual Basic, Visual Basic for Application, Visual .NET, Delphi, etc.

It is possible to maintain simultaneously an unlimited number of clients connected to different Wialon™ servers through it.

WialoActiveX is designed as ActiveX COM server in the form of system DLL. The program is ready to use right after the installation. It does not need any additional configuration.

This component is available in two versions: for 32-bit applications - *WialonActiveX* and for 64-bit applications - *WialonActiveX64*. Both of them can be installed simultaneously on Windows x64 OS in order to be used in different kinds of applications.

Connecting ActiveX to Wialon

To connect to ActiveX COM server, you have to construct the object *WialonActiveX.WialonConnection* that implements IWialonConnection interface.

Here is an example of a code in Visual Basic. It contains comments but does not contain error check.

```
        ' The main component of the system that provides connection to
Wialon
        Dim Wialon As Object
        ' The collection of available units
        Dim Units As Object
        ' A certain unit
        Dim Unit As Object
        ' Wialon error string
        Dim ErrStr As String
        ' The collection of messages from a unit
        Dim Msgs As Object
        ' Counter
        Dim i As Long
        ' The number of units in the collection
        Dim CountOfUnits As Long

        ' Creating an object to connect to Wialon (for 64-bit applications -
WialonActiveX64)
        Wialon = CreateObject("WialonActiveX.WialonConnection")
        ' Connection check
        If Wialon Is Nothing Then
            ' In case of error, the alert is generated
            MsgBox("No Object")
            Return
        End If

        ' Adjusting connection settings through
        Wialon.SetProxyMode "ProxyHost", 8080, "Login:Passw"
        ' Getting all units available to the "user" with the "passw" from
the server
        ' "https://activex.gurtam.com" (prefix is required for secure
connection)
        ' at 443 port without proxy server
        Units = Wialon.Login("https://activex.gurtam.com", 443, "user",
"passw")
        ' Check for available units
        If Units Is Nothing Then
            ' If units are not available, ActiveX error is reported as well
as Wialon error
            MsgBox("Error = " + Err.Number.ToString() + ": " +
Err.Source.ToString() + " (" + Err.Description.ToString() + ")")
            ' Getting error from Wialon server
            ErrStr = Wialon.GetLastError()
            MsgBox("Wialon error: " + ErrStr)
            Return
        End If

        ' Getting the number of available units
        CountOfUnits = Units.Count
        ' Displaying a message box with the number of available units
        MsgBox("Units = " + CountOfUnits.ToString())

        ' Starting the cycle for units search
        For i = 1 To CountOfUnits
            ' Getting a unit from the collection
            Unit = Units.Item(i)
            ' Check if the unit is available
            If Unit Is Nothing Then
                ' If the unit is not available, report the error
                MsgBox("Not unit")
                Return
            End If
            ' Adjusting the flag to get addresses by coordinayed. False -
without addresses (fast). True - with addresses (slow).
            Unit.ResolveLocations (False)
            ' Getting messages from a unit for the indicated period (time in
UNIX format)
            Msgs = Unit.GetMessages(1255112326, 1256312326)
            ' Check for messages availability
            If Msgs Is Nothing Then
                ' If there are no messages, display the alert
                MsgBox("No messages for unit: " + Unit.Name)
            Else
                ' Display the number of messages found
                MsgBox("Messages = " + Msgs.Count.ToString())
                ' Decrement the reference count for the given invoked
environment frame CLR
                ' (if explicit memory release is needed after using an
object)
                System.Runtime.InteropServices.Marshal.ReleaseComObject(Msgs)
                ' Releasing unit with messages
                Msgs = Nothing
            End If
            ' Decrement the reference count for the given invoked
environment frame CLR
            ' (if explicit memory release is needed after using an object)
            System.Runtime.InteropServices.Marshal.ReleaseComObject(Unit)
            ' Releasing unit with unit :)
            Unit = Nothing
        Next i
        ' Decrement the reference count for the given invoked environment
frame CLR
        ' (if explicit memory release is needed after using an object)
        System.Runtime.InteropServices.Marshal.ReleaseComObject(Units)
        ' This is the end of the program, we're not going to work with these
objects anymore, releasing
        Units = Nothing
        Wialon = Nothing
        ' End
        MsgBox("End")
```

More detailed example is available in Excel file delivered with ActiveX.

ActiveX API

To see the description of all available interfaces and their features, use available DLL sources viewer.

Note.
Do not forget to free the storage space each time after using an object (example code for VBA: Set Unit = Nothing).

Available interfaces:

- IWialonConnection – connection to ActiveX COM server.
- IWialonCollection – ActiveX collection of objects: IWialonUnit, IWialonUnitMsg, IWialonParam, IWialonReport, IWialonUnitGroup.
- IWialonUnit – units to be tracked.
- IWialonUnitMsg – messages from units.
- IWialonParam – message parameters.
- IWialonReport – reports.
- IWialonUnitGroup – unit groups.

IWialonConnection

IWialonConnection is the main interface which allows to connect to Wialon server and retrieve needed objects from there.

Method	Parameter	Return Value	Description
Login	*till version 1.6* BSTR **Host**, unsigned short **Port**, BSTR **UserName**, BSTR **Password**, BSTR **Proxy**, unsigned short **ProxyPort** *since version 1.7* BSTR **Host**, unsigned short **Port**, BSTR **UserName**, BSTR **Password**	IWialonCollection ****UnitsCol**	To get all objects available to the current user. It is required to check each return value to be in existence (Not Nothing) or check return result (Error). To get newer data, disconnect from the server using the function *Disconnect*Disconnect. Note: Since the version 1.7 parameters for connection through proxy server have been located in a separate function **SetProxyMode**.
GetLastError	-	BSTR* **Error**	To get the latest error from Wialon.
Disconnect	-	-	Disconnect from Wialon server in order to inquire fresh data later on (refresh).
GetReportsList	-	IWialonCollection ****ReportsCol**	To get all available report templates. It is required to check each return value to be in existence (Not Nothing) or check return result (Error).
GetReportByID	unsigned **From**, unsigned **To**, long long **UnitID**, int **TimeZoneOffset**, BSTR **Lang**, long long **ResourceID**, long long **ReportID**	BSTR* **ReportData**	To get a report by report or resource ID. This is an analogue of the function *Generate* from the IWialonReport interface. Starting and end time (*From - To*) is indicated in UNIX format, that is in seconds from the 1st of January 1970. Unit ID can be received from the collection of available units. Time zone (*TimeZoneOffset*) is indicated in seconds, for example, the value for Moscow is 10800, the shift from the prime meridian. The language *Lang* is indicated like domains

			(ru, en, etc). Report or resource ID can be received from the collection of available reports.
GetReportByIDU	int **From**, int **To**, int **UnitID**, int **TimeZoneOffset**, BSTR **Lang**, int **ResourceID**, int **ReportID**	BSTR* **ReportData**	This is an analogue of the function *GetReportByID* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.
GetUnitGroups	-	IWialonCollection **UnitGroups**	To get all available unit groups IWialonUnitGroup.
GetUnitGroupByID	long long **UnitGroupID**	IWialonUnitGroup **UnitGroup**	To get a unit group IWialonUnitGroup by its ID.
GetUnitGroupByIDU	int **UnitGroupID**	IWialonUnitGroup **UnitGroup**	The analogue of the function *GetUnitGroupByID* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.
SetProxyMode	BSTR **Proxy**, unsigned short **ProxyPort**, BSTR **ProxyUserPwd**	-	Set parameters to connect to proxy server (host, port, login:password). Note: The function is available since the version 1.7.
GetLocationsText	BSTR **Lats**, BSTR **Lons**, int **Count**	BSTR* **Text**	Get addresses by coordinates. Coordinates must be in text format separated with comma. Use dot as decimal delimiter. The number of coordinates is set in the third parameter, so the number of addresses, so the number of addresses returned will be as set. Addresses are returned in text format, separated with comma. Note: The function is available since the version 1.7.

IWialonCollection

This interface describes the collection of ActiveX objects. It can contain the following interfaces: IWialonUnit, IWialonUnitMsg, IWialonParam.

Property	**Parameter**	**Return Value**	**Description**
Item	long **Index**	IDispatch** **pVal**	To get an object from the collection by the given index. The indexes begin from one (1). It is required to check each return value to be in existence (Not Nothing) or check return result (Error).
Count	-	long ***pVal**	To get the number of units contained in the collection.

IWialonUnit

This interface contains the description of units.

Method/Property	Parameter	Return Value	Description
Name (property)	-	BSTR* **Name**	To get unit name.
GUID (property)	-	BSTR* **GUID**	To get unit GUID (global unique identifier).
ID (property)	-	long long* **ID**	To get unit local ID.
IDU (property)	-	int* **ID**	The analogue of the function *ID* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.
Phone (property)	-	BSTR* **Phone**	To get the phone number of the SIM card installed on a unit.
LastPosition (property)	-	IWialonUnitMsg ****LastPos**	To get the latest message from the unit with its location.
GetMessages (method)	unsigned **From**, unsigned **To**	IWialonCollection ****MsgCol**	To get the collection of messages for the given period. The time is indicated in UNIX format, that is in seconds beginning from the 1st of January 1970. It is required to check each return value to be in existence (Not Nothing) or check return result (Error). To speed up server work, it is recommended to inquire messages for up to 30 days.
GetMessagesU (method)	int **From**, int **To**	IWialonCollection ****MsgCol**	The analogue of the function *GetMessages* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.
GetLastError (method)	-	BSTR* **Error**	To get the latest error from Wialon.
ResolveLocations (method)	BOOL **ResolveLocationsFlag**	-	Set the option to define location by coordinates when getting message. If the option is on, it considerably enlarges time to get messages. Note: The function is available since the version 1.7.

IWialonUnitMsg

The interface containing the description of one message from a unit.

Property	Parameter	Return Value	Description
Time	-	unsigned* **Time**	To get message time. The time is indicated in UNIX format, that is in seconds beginning from the 1st of January 1970.
TimeU	-	int* **Time**	The analogue of the property *Time* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.
Speed	-	int* **Speed**	To retrieve speed from a message. It is required to check the result of function operation because there can be no speed in a message.
Course	-	int* **Course**	To get movement direction from a message. It is required to check the result of function operation because there can be no course in a message.
X	-	double* **X**	To get longitude. It is required to check the result of function operation because there can be no location information in a message.
Y	-	double* **Y**	To get latitude. It is required to check the result of function operation because there can be no location information in a message.
Z	-	double* **Z**	To get altitude. It is required to check the result of function operation because there can be no location information in a message.
Type	-	BSTR* **Type**	To get message type: SMS, Data, CMD, etc. - *udp* - message contains location and data such as speed, course, number of satellites, I/O, driver ID. - *ud* - message contains only data from a device such as I/O and driver ID. - *ucr* - message contains information about an executed command (command name, parameters, user, link type, connector name, execution time). - *us* - message contains information about a received SMS message (SMS text, phone number). - *evt* - message contains location of an event happened.
CountSats	-	int* **CountSats**	To get the number of satellites which is important to estimate coordinates accuracy. If the number is 255, it means satellites are locked successfully but their number is unknown. It can happen if the device used

			does not send such information in general. It is required to check the result of function operation because there can be no location information in a message.
Param	int **Number**	IWialonParam ****Param**	To get messages parameter by its number. It is required to check each return value to be in existence (Not Nothing) or check return result (Error).
Location	-	BSTR* **Location**	To get unit location in the from of address.
ParamCount	-	long* **ParamCount**	To get the number of parameters in a message.
ParamByName	BSTR **ParamName**	IWialonParam ****Param**	To get a parameter by its name. It is required to check each return value to be in existence (Not Nothing) or check return result (Error).
SMSText	-	BSTR* **SMS**	To get SMS text (only for SMS messages).
Driver	-	BSTR* **Driver**	To get driver's name if known.
CMDName	-	BSTR* **CMDName**	To get command name if known.
CMDParam	-	BSTR* **CMDParam**	To get command parameters if known.
UserGUID	-	BSTR* **UserGUID**	To get the name the user who executed the command (if known).
LinkName	-	BSTR* **LinkName**	To get the name of hardware which were used to connect to the unit for command execution (if known).
LinkType	-	BSTR* **LinkType**	To get link type (UDP, TCP, GSM) used to execute a command (if known).
ModemPhone	-	BSTR* **ModemPhone**	To get the number of the modem used to execute a command (if known).
EventText	-	BSTR* **EventText**	To get event text if there is such.

IWialonParam

The interface contains the description of parameters in a message.

Property	**Param**	**Return Value**	**Description**
Type	-	BSTR* **Type**	To get message type (int, double, string).
Name	-	BSTR* **Name**	To get parameter name.
Value	-	VARIANT* **Value**	To get parameter value. Previously, you should get to know data type in VARIANT - to do this use the function *Value.Type* or *VarType(Value)*.
ValueStr	-	BSTR* **Value**	The analogue of the property *Value* compatible with applications which do not support VARIANT data type.

IWialonReport

This interface allows to retrieve reports from the monitoring site. Only ready report templates can be used. Creating new report templates is impossible here. To generate a report, the function *GetReportByID* from IWialonConnection interface can be used instead.

Method/Property	Parameter	Return Value	Description
Generate (method)	unsigned **From**, unsigned **To**, long long **UnitID**, int **TimeZoneOffset**, BSTR **Lang**	BSTR* **XMLData**	To generate a report on the server and get it in the form of XML string. Starting and end time (*From* - *To*) is indicated in UNIX format, that is in seconds from the 1st of January 1970. Unit ID can be received from the collection of available units. Time zone (*TimeZoneOffset*) is indicated in seconds, for example, the value for Moscow is 10800, the shift from the prime meridian. The language *Lang* is indicated like domains (ru, en, etc).
GenerateU (method)	int **From**, int **To**, int **UnitID**, int **TimeZoneOffset**, BSTR **Lang**	BSTR* **XMLData**	The analogue of the function *Generate* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.
Name (property)	-	BSTR* **ReportName**	To get report name. Not available in 64-bit version.
ReportID (property)	-	long long* **ReportID**	To get report unique ID.
ReportIDU (property)	-	int* **ReportID**	The analogue of the property *ReportID* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.
ResourceID (property)	-	long long* **ResourceID**	To get the unique ID of the resource where a report template belongs to.
ResourceIDU (property)	-	int* **ResourceID**	The analogue of the property *ResourceID* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.

IWialonUnitGroup

The interface contains the description of unit groups. To get unit groups collection see IWialonConnection.

Method/Property	Parameter	Return Value	Description
Name (property)	-	BSTR* **UnitGroupName**	To get the name of current unit group.
ID (property)	-	long long* **UnitGroupID**	To get unit group unique ID.
IDU (property)	-	int* **UnitGroupID**	The analogue of the property *ID* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.
GetUnits (method)	-	IWialonCollection **UnitsCol**	To get collection of the units in the group.
CheckUnitInGroup (method)	long long **UnitID**	BOOL* **UnitInGroup**	To check whether a unit with the given ID belongs to the given group.
CheckUnitInGroupU (method)	int **UnitID**	BOOL* **UnitInGroup**	The analogue of the function *CheckUnitInGroup* compatible with applications which do not support 64-bit integer numbers. Not available in 64-bit version.

Compatibility

The functions with the ending **U** are intended to create compatibility with 64-bit integer numbers (1C 7th version). This ending indicates that the function will get or return all values as usual 32-bit integer numbers (signed integer).

All duplicated functions and properties are interchangeable. If your application can support 64-bit integer numbers, it is recommended to use the functions which support this digital capacity, that is without U ending.

Always use homogeneous functions. It means, if you are working with 64-bit integer numbers, use them throughout the application. The same is for 32-bit systems.

For 64-bit operating systems there is a special component - *WialonActiveX64*. It works only with 64-bit applications. For 32-bit applications installed on 64-bit OS, use 32-bit component *WialonActiveX*.

Garbage Collector

System garbage collector is a service that automatically reclaims unused memory. See GC Class for details.

If using ActiveX you notice that the garbage collector rarely cleans the main memory, in your program code you can decrement the count of references to object before the garbage collector is activated. To do this, use the following code (VB):

```
System.Runtime.InteropServices.Marshal.ReleaseComObject(object)
object = Nothing
```

It will allow you to free the memory beeing used by an object if its references count is zero. See Marshal.ReleaseComObject Method for more details.

An alternative way to clean the memory is a method of forced garbage collection (VB):

```
GB.Collect()
```

See GC.Collect Method for the full description of the method.

Errors

Each function or method activated in the program returns an error code. Below see the list of more frequent errors which can help you to detect the problem:

Error Code	**Description**
0	Succeeded.
1	Unknown error.
-1	Non-supported protocol.
-2	Error initializing connection. It can happen when trying to create an object IWialonConnection repeatedly.
-3	Invalid URL address.
-5	Error to resolve proxy server name.
-6	Error to resolve server name.
-7	Error connecting to the given address. It can happen when firewall or antivirus program are set incorrectly.
-9	Access to server denied.
-18	Insufficient data. It is possible if connection to server is broken or in case of server abnormal termination.
-22	Http error.
-27	Internal error while executing query.
-28	Execution timeout is exceeded.
-34	POST query error.
-35	Error connecting through a protected connection (SSL).
-36	It is impossible to continue loading. Connection may be broken.
-47	Too many redirections.
-51	Certificate error.
-55	Data communication error.
-56	Data acquisition error.
-80	Incorrect SSL connection termination.
-81	The socket is not ready to data sending or acquisition. Maybe, all sockets are busy - try to upload unusable applications.
-95	Wialon server error. Use the function *GetLastError* to retrieve the error text.
-96	Error when operating file input/output. It can happen if the user who is executing the program does not have enough access to read and record data in the temporary folder.
-97	Error processing data. Maybe, the server returned invalid data or was abnormally terminated while executing the inquiry.
-98	Error processing data. Maybe, the server returned nothing or was abnormally terminated while executing the inquiry.
-99	Inquiry to the server is impossible because of incorrect product settings. Please, reinstall ActiveX.
-300	Error when parsing data received from the server.

-301	Error when parsing data received from the server.
-302	Error when parsing data received from the server.
-303	Error when parsing data received from the server.
-304	Error while creating objects. It may be caused by memory lack or if you inquire too many messages. Close other applications and retry.
-305	Not enough memory to unpack data. Close other applications and retry.