

---

---

# INTERFACE SPECIFICATION

## *isystem.connect*

### **Purpose**

This technical paper describes the *isystem.connect* port. It is designed to facilitate implementation of client side objects.

### **The *isystem.connect* Port**

The *isystem.connect* port was designed to allow out-of-process communication and control over a running instance of winIDEA with minimal complexity imposed on the client application.

The client application makes use of the *isystem.connect* port through the iCONNECT.DLL providing the IConnectXX COM interfaces. The DLL marshals the IConnectXX function calls via TCP/IP sockets to the specified winIDEA instance, which after processing them, returns them using the same transport path.

The following steps are required to make use of the *isystem.connect* interface:

- Optionally implement the *IConnectSink* interface through which the server can post notification messages.
- Include the provided *CICConnectClient* class in your application.
- Create an Instance of the *CICConnectClient*, call its *LinkToiCONNECT* member function to link to iCONNECT.DLL, *Connect* member function to open the connection and *Disconnect* to close it.
- Once the connection is established, all functions of IConnectXX interfaces can be used.

Example:

```
#include "stdafx.h"
#include "IConnectClient.h"
#include "MemArea.h"
int main()
{
    CICConnectClient ICC;
    if (!ICC.LinkToiCONNECT(ICC.GetMRUiCONNECT()))
        return 0;
    // test on local PC: pszIPAddress = NULL
    // TCP port as configured in winIDEA, for example 5315
    // no IConnectSink implemented: pIConnectSink = NULL
    if (ICC.Connect(NULL, 5315, NULL))
    {
        // read out a byte from address 0x1234, increment it and write back
        BYTE abyBuf[1];
        ICC.m_pIConnectDebug->ReadMemory(IConnectDebug::fMonitor, maPhysicalARM, 0x1234, 1,
                                        abyBuf, 1, NULL);

        abyBuf[0]++;
        ICC.m_pIConnectDebug->WriteMemory(IConnectDebug::fMonitor, maPhysicalARM, 0x1234, 1,
                                        abyBuf, 1, NULL);
    }
    return 0;
}
```

# Connection Mechanism

## Binding

Connection to winIDEA is maintained by an *isystem.connect* object which is created inside the *iCONNECT.DLL* upon a call to:

```
HRESULT iCONNECT_CreateObject(REFIID riID, void **ppv);
```

*riID*

Specifies the class GUID. This should always be *CLSID\_iCONNECT*.

*ppv*

is a pointer to *IUnknown* interface of the created object. If the creation failed, *ppv* will be *NULL*.

*Return value*

S_OK	success
CLASS_E_CLASSNOTAVAILABLE	object of class specified by riID is not supported

*Remarks*

Once the *IUnknown* pointer is obtained, *QueryInterface* should be used to obtain the *IConnect* interface.

Example:

```
// create an iCONNECT object
IUnknown * pIUnknown = NULL;
if (!SUCCEEDED(iConnect_CreateObject(CLSID_iCONNECT, (void **)&pIUnknown)))
    return;
// obtain IConnect interface
IConnect * pIConnect = NULL;
pIUnknown->QueryInterface(IID_IConnect, (void**)&pIConnect);
// release pIUnknown and check for success
pIUnknown->Release();
if (NULL != pIConnect)
{
    m_pIConnect->QueryInterface(IID_IConnectDebug, (void**)&m_pIConnectDebug);
}
```

The *isystem.connect* object is accessible only via the returned interfaces. Once no outstanding references are open, the object will be destroyed.

## *CIConnectClient* class

To facilitate binding, the *CIConnectClient* class is provided which encapsulates the object creation and obtains all *IConnectXX* interfaces. The class header and cpp files must be included in the client application.

```
LPCTSTR GetMRUiCONNECT ();
```

Returns the path to *iConnect.DLL* of winIDEA that was most recently started. The returned value can be *NULL* if the winIDEA version log file could not be found.

## **BOOL LinkToiCONNECT(LPCTSTR pszCONNECTPath = NULL);**

Dynamically loads the *iConnect.DLL*.

*pszCONNECTPath*

The path to *iConnect.DLL* to be used. If this parameter is *NULL*, standard windows DLL search mechanism will be used.

You can use this parameter if multiple versions of winIDEA are installed on a system and you want to control which version is used.

## **BOOL UnlinkFromiCONNECT();**

Unloads the *iConnect.DLL*.

Use this function if you wish to load a different version of *isystem.connect* next.

## **BOOL Connect(LPCTSTR pszIPAddress, WORD wTCPPort, IConnectSink \* pIConnectSink, void (\*pfnOnConnectionLost)(void\*) = NULL, void \* pOnConnectionLostInfo = NULL);**

Connects to winIDEA.

*pszIPAddress*

The IP address of the machine where winIDEA is running. This can be either dotted decimal or an URL.

If *pszIPAddress* is *NULL*, the local host is considered.

*wTCPPort*

The TCP port address to use. This should be the same value as configured in winIDEA to which the connection is attempted.

*pIConnectSink*

Pointer to the *IConnectSink* interface. If one is provided, the *isystem.connect* object will forward notifications received from the server to it. If this parameter is *NULL*, no notifications will be delegated to the application.

*pfnOnConnectionLost*

Pointer to a function to be called when connection is lost. If this parameter is *NULL*, no function will be called upon connection loss.

The function must use the following prototype:

```
void OnConnectionLostHandler(void * pOnConnectionLostInfo);
```

The client application can throw an exception in the *pfnOnConnectionLost* function.

*pOnConnectionLostInfo*

Pointer to be passed to *pfnOnConnectionLost* function.

## **void Disconnect(DWORD dwDetachFlags = 0);**

Disconnects from winIDEA. See *IConnect::Detach* for *dwDetachFlags* description.

# Interfaces

All `isystem.connect` interface functions return a `HRESULT` type identifying the result of the operation. Common return codes are:

<code>S_OK</code>	Success
<code>ICONNECT_E_NOT_CONNECTED</code>	The <i>isystem.connect</i> object is not attached to winIDEA.
<code>ICONNECT_S_SIZE</code>	The output parameter specified could not accommodate the entire return value (for example file path was longer than the provided string)

## *IConnect* Interface

The *IConnect* interface provides functions that control connection to winIDEA. If you use the `CConnectClient` class you don't need to use this interface.

### **HRESULT Attach(LPCTSTR pszIPAddress, WORD wTCPPort, SAttach \* pAttach);**

Establishes the connection to winIDEA.

#### *pszIPAddress*

The IP address of the machine where winIDEA is running. This can be either dotted decimal or an URL.

If *pszIPAddress* is `NULL`, the local host is considered.

#### *wTCPPort*

The TCP port address to use. This should be the same value as configured in winIDEA to which the connection is attempted.

#### *pAttach*

Pointer to an *SAttach* structure:

```
struct SAttach
{
    IConnectSink * m_pIConnectSink; // [IN]
    // Pointer to the IConnectSink interface.
    DWORD m_dwVersionClient; // [IN]
    // Interface version used by the client.
    DWORD m_dwVersionServer; // [OUT]
    // Interface version used by the server.

    // connection lost hook
    void (*m_pfnOnConnectionLost)(void *);
    // this function will be called when connection is lost
    void * m_pOnConnectionLostInfo;
    // this parameter will be passed to it
    char * m_szResult[256]; // [OUT]
    // result of the connection
};
```

#### *Return value*

<code>ICONNECT_E_CONNECT_FAILED</code>	The server is not reachable on the specified IP address and port.
<code>ICONNECT_E_VERSION_CONFLICT</code>	The client and server versions are incompatible.
<code>ICONNECT_E_ALREADY_CONNECTED</code>	The <i>isystem.connect</i> object is already attached to winIDEA

## HRESULT Detach (DWORD dwDetachFlags);

Closes the interprocess channel to winIDEA. After that, all *ICconnectXX* function calls other than *ICconnect::Attach* will fail.

### *dwDetachFlags*

dfCloseServerIfLastClient	winIDEA will exit if this is the last client attached
dfCloseServerUnconditional	winIDEA will exit unconditionally
dfCloseAutoSaveDefault	when closing default Prompts will be displayed
dfCloseAutoSaveAll	all documents and workspaces will be saved without prompting
dfCloseAutoSaveNone	all changes to documents and workspace will be discarded

## HRESULT Launch (DWORD dwLaunchFlags, SLaunch \* pLaunch);

Launches a new winIDEA instance, identifies an existing (running) instance or enumerates all running instances. Running instance discovery is implemented via directed UDP broadcasts on *pLaunch->m\_wDiscoveryPort*. If the default discovery port is blocked, configure a different port in winIDEA.

Note that running instances can be identified and enumerated on remote hosts too, but new instances can be launched only on local host.

### *dwLaunchFlags*

The *IfStartXXXX* flags determine under what conditions to start a new instance of winIDEA.

IfStartIfRequired	Starts winIDEA if an instance isn't running yet
IfStartAlways	Always launch a new instance
IfStartExisting	Find an existing instance, do not launch new.
IfStartEnumerate	Enumerate all running instances

The *IfWaitXXXX* flags determine how long the *isystem.connect* should wait for running instances to respond.

IfWaitDefault	Currently one second
IfWait30ms	30 milliseconds
IfWait100ms	100 milliseconds
IfWait300ms	300 milliseconds
IfWait1s	1 second
IfWait3s	3 seconds
IfWait10s	10 seconds
IfWait30s	30 seconds

Note that if any *IfStartXXXX* flags other than *IfStartEnumerate* are used, *isystem.connect* will finish discovery as soon as the first matching instance is found.

The *IfShowXXXX* flags determine how the newly launched instance will start

IfShowDefault	No special provision
IfShowMinimized	winIDEA is started minimized
IfShowMaximized	winIDEA is started maximized

### *pLaunch*

```

struct SLaunch
{
    LPCTSTR m_pszIPAddress; // [IN]
    // IP address of the PC where the winIDEA is running or enumeration is requested
    LPCTSTR m_pszID; // [IN]
    // the winIDEA instance ID by which it can be identified later
    LPCTSTR m_pszWorkspace; // [IN]
    // the workspace file path currently opened
    LPCTSTR m_pszCmdLine; // [IN]
    // if new instance is launched, this command line is used in addition to specified
    m_pszWorkspace and m_pszID
    WORD m_wDiscoveryPort; // [IN]
    // specifies the UDP discovery port to be used. If this value is 0, winIDEA default is
    used
    WORD m_wTCPPort; // [OUT]
    // returns the port on which the instance is listening. This value should be used on a
    subsequent call to IConnect::Attach

    // enumerator
    struct SInstanceInfo
    {
        LPCTSTR m_pszID; // the ID of this instance
        LPCTSTR m_pszWorkspace; // the currently open workspace of this instance
        WORD m_wTCPPort; // TCP port on which this instance is listening
    };
    void (*m_pfnInstanceEnumerator)(void *, const SInstanceInfo *);
    // this function will be called when connection is lost
    void * m_pInstanceEnumeratorInfo;
    // this parameter will be passed to it along with respective instance info
};

```

### *Return value*

ICONNECT_E_ERROR	Discovery could not be initiated
ICONNECT_E_BAD_ID	Enumeration was requested, but <i>m_pfnInstanceEnumerator</i> is NULL
E_FAIL	winIDEA could not be launched.

### **HRESULT GetLastError (DWORD dwGetLastErrorFlags, LPTSTR pszError, DWORD dwErrorLen);**

An *isystem.connect* call can cause an error condition in winIDEA which under normal circumstances brings up a modal message box. Such message boxes require user intervention which is not desired during *isystem.connect* operation. winIDEA therefore hooks all such messages, taking default decisions when they occur. The text of the message can be obtained after the call failed by calling this function.

### *dwGetLastErrorFlags*

Reserved should be zero.

### *pszError*

Pointer to string to accept last error description

*dwErrorLen*

Length of string *pszError* points to.

*Return value*

ICONNECT_S_MSG_FAIL	The previous call generated a message box
S_OK	The previous call generated no message box

## ***IConnectIDE Interface***

The *IConnectIDE* interface provides access to IDE functions exported by winIDEA.

**HRESULT Document (DWORD dwFileFlags, LPCTSTR pszFileName, LPCTSTR pszParameter = NULL, DWORD dwParameter = 0);**

Manages document files and windows in winIDEA.

### *dwFileFlags*

ffClose	Close
ffCloseAll	Close all
ffSave	Save under its original name
ffSaveAs	Save under <i>pszParameter</i>
ffSaveAsOverwrite	Save under <i>pszParameter</i> , overwrite an existing file
ffSaveAsPrompt	Save using <i>Save As...</i> prompt
ffSaveAll	Save all
ffNew	Create new named <i>pszFileName</i>
ffNewPrompt	Create new prompt for name and folder
ffOpen	Open <i>pszFileName</i>
ffOpenPrompt	Prompt to open

Document actions:

dfFocus	Focus the file window..
dfSpecialWindow	This call refers to a non document window, defined by the dwXXXX value.
dmMarkSet	Puts a marker on the requested line ( <i>dwParameter</i> )
dmMarkClear	Clears the global editor marker. <i>pszFileName</i> is ignored.
dmMarkPos	Positions the document on the specified line.
daStart	Starts the document action (start trace, script, etc.)
daStop	Stop the document action completely (stop trace, etc.) After this all the document data is valid and status is idle
daResume	Resume the document action.
daDeactivate	Stop the active document action (stop HW sampling). After this the document status will remain busy until all data is valid.
daSelect	Select a property of the document specified by <i>pszParameter</i> and <i>dwParameter</i>
daStatus	Probe status of the document action. <b>S_OK</b> indicates idle/finished, <b>S_FALSE</b> indicates busy state.
daExport	Export the document to a different format. <i>pszParameter</i> specifies the file to export to, <i>dwParameter</i> specifies the scope and export format to use. The export format values follow the document specific export format list starting with 0. Value of 0xFFFFFFFF indicates that the last configured

	export format for the document should be used.
daExportV	Same as <i>daExport</i> , but the OS associated viewer is launched after export
daSetExportFormat	Set default export format for this document. The format name is passed in <i>pszParameter</i> .

Document type (used only with *ffNew*)

dtText	ASCII Text file
dtAnalyzer	winIDEA Analyzer type
dtCodeCoverage	winIDEA Code Coverage type
dtDataCoverage	winIDEA Data Coverage type

Special window type

dwTerminal	Terminal window, supports <i>daStart</i> , <i>daStop</i> and <i>daStatus</i> actions
------------	--

*pszFileName*

File path. If this path is relative, winIDEA will consider the workspace file directory as the root.

*pszParameter*

Parameter to the requested action, depending on the action (*ffSaveAs*, *daSelect*)

*dwParameter*

Parameter for the requested action (*dfFocus*, *dmMarkSet*, *daSelect*, *daExport*). Use with matching flags.

*Return value*

S_FALSE	Indicates busy state when querying status. Returned also on <i>daDeactivate</i> and <i>daStop</i> to indicate that the document was already inactive or stopped.
ICONNECT_E_FILE_NOT_FOUND	File not found
ICONNECT_E_ALREADY_EXISTS	Creation of requested file failed because a file with the same name already exists
ICONNECT_E_NOT_AVAILABLE	The requested operation is not available
ICONNECT_E_ERROR	An error occurred performing the requested operation
ICONNECT_E_BAD_ID	The specified export format was is not supported.
E_NOTIMPL	The requested document or window type is not supported

**HRESULT Workspace (DWORD dwFileFlags, LPCTSTR pszWorkspaceName);**

Opens a source file in winIDEA.

*dwFileFlags*

See *ffXXXX* flags for *Document*.

*pszWorkspaceName*

Workspace file path. If this path is relative, winIDEA will consider the workspace file directory as the root.

### Return value

S_FALSE	The specified workspace is already open
ICONNECT_E_ERROR	An error occurred performing the requested operation

## HRESULT Application (DWORD dwApplicationFlags, SApplication \* pApplication);

Manipulates the attached winIDEA.

### dwApplicationFlags

afWindowActivate	Activate (bring to foreground) the application
afWindowMinimize	Minimize
afWindowRestore	Restore size
afWindowMaximize	Maximize
afWindowMove	Move to <b>pApplication-&gt;m_Rect</b>
afHookMsgBoxSet	hook message box when called over iConnect
afHookMsgBoxClr	display regular message box when called over iConnect
afHookMsgBoxGet	gets the hook state, returns S_OK if hooked, S_FALSE if not hooked

### pApplication

This parameter is used when *afWindowMove* is used. On return it always contains the current position of the winIDEA window.

```
struct SApplication
{
    struct SRect
    {
        LONG m_dwLeft;
        LONG m_dwTop;
        LONG m_dwRight;
        LONG m_dwBottom;
    } m_Rect; // [IN] new coordinates if afWindowMove is requested.
              // [OUT] current window coordinates
    DWORD m_dwPID; // [OUT] Process ID
};
```

## ***IConnectDebug* Interface**

The *IConnectDebug* interface provides access to debug functions exported by winIDEA.

### **HRESULT GetInfo (SInfo \* pInfo);**

Obtains information about the CPU.

*pInfo*

Pointer to structure that will receive the information. The structure has the following members:

```
struct SInfo
{
    CCPUIInfo m_CPUInfo; // type of CPU
};
```

### **HRESULT GetAddress (DWORD dwAccessFlags, LPCTSTR pszExpression, BYTE \* pbyMemArea, ADDRESS \* paAddress, ADDRESS \* paSizeMAUs, SType \* pType);**

Returns address and size of an object, like a global symbol, function, etc.

*dwGetAddressFlags*

31	30 - 24	23 - 16	15 - 12	11 - 8	7 - 0
Use File	File Index	Enum Index	Enum Source	Reserved SBZ	EAccessFlags

#### **EAccessFlags**

see *ReadMemory*. Determines what kind of memory access is used if required to resolve the expression.

#### **Enum Source**

Defines which symbols to enumerate.

gafExpression	No enumeration, address of any L-value expression can be retrieved
gafVariables	Global variables (data objects)
gafLabels	Global code labels
gafFunctions	High level functions
gafType2	The pType parameter is of type <i>SType2</i> , which allows retrieval of extended type info. This flag can be used only with <i>gafExpression</i>

#### **Enum Index**

Index of the enumerated symbol to return.

#### **File Index**

Index of the symbol (download) file to use. Used only if **Use File** flag is set.

*pszExpression*

An expression that describes a location in memory (an L-value).

If symbol enumeration is used, the name of the global symbol.

### *pbyMemArea*

Pointer to a **BYTE** type variable that will receive the memory area of the object.

### *paAddress*

Pointer to an **ADDRESS** type variable that will receive the address of the object.

### *paSizeMAUs*

Pointer to an **ADDRESS** type variable that will receive the size of the object.

### *pType*

Pointer to a **SType** type variable that will receive the type of the object.

```
struct SType
{
    enum EType
    {
        tUndefined = 0x00,
        tUnsigned  = 0x01,
        tSigned    = 0x02,
        tFloat     = 0x03,
        tAddress   = 0x04,
        tCompound  = 0xFF,
    };
    BYTE m_byType;
    BYTE m_byBitSize;
};

struct SType2: public SType
{
    enum EType2
    {
        t2Regular = 0x00, // no extra type information available
        t2Bitfield = 0x01, // this is a bit field, m_dw1 specifies size and m_dw2 offset within
        container integer
    };
    BYTE m_byType2;
    BYTE m_byReserved;
    DWORD m_dw1;
    DWORD m_dw2;
};
```

### Example:

```
BYTE byMemArea;
ADDRESS aAddress, aSize;
SType2 Type2;

pIConnectDebug->GetAddress(IConnectDebug::gafType2, "bA.m_b3", &byMemArea, &aAddress, &aSize,
&Type2);

printf("Memory space: %d, Address: %X, Size: %X\n", byMemArea, aAddress, aSize);
printf("Type: %d, Size: %d\n", Type2.m_byType, Type2.m_byBitSize);

if (Type2.m_byType == SType2::t2Bitfield)
{
    printf("Bitfield, NumBits %d, Displacement %d\n", Type2.m_dw1, Type2.m_dw2);
}
```

*Return value*

ICONNECT_E_CAN_NOT_ACCESS	Evaluation required a memory access which could not be performed.
ICONNECT_E_BAD_ID	Expression could not be evaluated
S_FALSE	File or enum index out of range

**HRESULT GetSymbol (DWORD dwSymbolFlags, BYTE byMemArea, ADDRESS aAddress, LPTSTR pszName, DWORD dwNameLen);**

Returns the name of the symbol on the specified address.

*dwSymbolFlags*

Specifies symbol classes to consider.

sVariables	Global variables (data objects)
sLabels	Global code labels
sFunctions	High level functions
sLines	Source lines
sConstants	Global constants

*byMemArea*

Memory area of the object.

*aAddress*

Address of the object.

*pszName*

Pointer to a **TCHAR** buffer that will receive the name of the symbol.

*dwNameLen*

Length of the *pszName* buffer.

*Return value*

S_FALSE	No symbol on the specified address. <i>pszName</i> is set to empty string.
---------	--

**HRESULT GetSourceAddress (DWORD dwSourceFlags, LPCTSTR pszFileName, DWORD dwLine, ADDRESS \* paAddress, DWORD \* pdwNumAddresses);**

Returns the address(es) of a source line.

*dwSourceFlags*

Reserved, must be zero.

*pszFileName*

Name of the file.

*dwLine*

Line number.

*paAddress*

Pointer to an array of **ADDRESS** type variables that will receive the addresses to which the source line translates.

*pdwNumAddresses*

Points to a **DWORD** type variable that

- on input specifies the size of the *paAddresses* array,
- on output, the number of entries placed into the *paAddresses* array.

*Return value*

S_FALSE	No source line symbol found
ICONNECT_E_SIZE	On input the <i>pdwNumAddresses</i> was specified as zero

**HRESULT GetAddressSource (DWORD dwSourceFlags, ADDRESS aAddress, LPTSTR pszFileName, DWORD dwFileNameLen, DWORD \* pdwLine);**

Returns the source matching to an address.

*dwSourceFlags*

Reserved, must be zero.

*pszFileName*

Pointer to **TCHAR** array buffer that will receive the file name.

*dwFileNameLen*

Length of the array pointed to by *pszFileName*.

*pdwLine*

Pointer to **DWORD** type variable that will receive the line number.

*Return value*

S_FALSE	No source line symbol found
---------	-----------------------------

## HRESULT Evaluate (DWORD dwAccessFlags, LPCTSTR pszExpression, LPTSTR pszResult, DWORD dwResultLen, SValue \* pValue, SType \* pType);

Evaluates an expression.

### *dwAccessFlags*

Same as for *ReadMemory*. This determines what kind of memory access is permitted if required to resolve the expression.

### *pszExpression*

Any C syntax expression.

### *pszResult*

Pointer to a TCHAR buffer that will receive the result of the evaluation. If this parameter is NULL no string value is returned.

### *dwResultLen*

Length of the *pszResult* buffer.

### *pValue*

Pointer to a SValue type variable that will receive the value. If this parameter is NULL no simple type value is returned. If this parameter is non NULL and the *pszExpression* doesn't evaluate to a simple type, the function will fail.

```
struct SValue
{
    union
    {
        BYTE        m_UINT8;
        char        m_INT8;
        WORD        m_UINT16;
        short       m_INT16;
        DWORD       m_UINT32;
        LONG        m_INT32;
        QWORD       m_UINT64;
        QLONG       m_INT64;
        float       m_FLOAT32;
        double      m_FLOAT64;
        CAddress    m_adrAddress;
    };
};
```

### *pType*

Pointer to a SType type variable that will receive the type.

### *Return value*

ICONNECT_E_CAN_NOT_ACCESS	Evaluation of the expression required memory access which could not be performed.
ICONNECT_E_BAD_ID	Expression could not be evaluated

**HRESULT Modify (DWORD dwAccessFlags, LPCTSTR pszExpression, LPTSTR pszResult, DWORD dwResultLen, const SValue \* pValue, const SType \* pType);**

Modifies an expression. The expression must evaluate to a lvalue.

*dwFlags*

Same as *Evaluate* with addition of

mfModifyToResult	if set, pValue and pType are ignored, pszResult must specify the new value
------------------	--

*pszExpression*

Any C syntax expression that evaluates to a lvalue.

*pszResult*

Pointer to a TCHAR buffer that will receive the result of the evaluation. If this parameter is NULL no string value is returned. In case of an error, the *pszResult* will contain the error string.

If *mfModifyToResult* flag is specified the *pszResult* is evaluated first and the resulting value is used to modify the *pszExpression*.

*dwResultLen*

Maximum length of the *pszResult* buffer returned. This value can be zero, even when *pszResult* is non-NULL (used with the *mfModifyToResult* flag)

*pValue*

Pointer to a SValue type variable containing the value to write to *pszExpression*. This parameter is not used if *mfModifyToResult* flag is used.

*pType*

Pointer to a SType type variable containing the type info of *pValue*. This parameter is not used if *mfModifyToResult* flag is used.

*Return value*

ICONNECT_E_CAN_NOT_ACCESS	Evaluation of the expression required memory access which could not be performed.
ICONNECT_E_BAD_ID	Expression could not be evaluated

**HRESULT ReadMemory (DWORD dwAccessFlags, BYTE byMemArea, ADDRESS aAddress, ADDRESS aNumMAUs, BYTE byBytesPerMAU, BYTE \* pbyBuf, BYTE \* pbyAccessInfo);**

Reads memory from the target system.

*dwAccessFlags*

fMonitor	Use regular memory access
fRealTime	Use real time access
fCacheNever	Do not cache the access
fCacheDefault	Cache according to winIDEA settings
fCacheStop	Cache while CPU is stopped
fCacheCode	Get from downloaded code mirror
fNoRefresh	Do not refresh winIDEA GUI after write operation

*byMemArea*

Memory space. Definitions for individual CPUs are provided in *MemArea.h*

*aAddress*

Address of the memory location of the first access.

*aNumMAUs*

Number of MAUs (memory addressable units) to read.

*byBytesPerMAU*

Number of BYTE type locations required to fit one MAU (memory addressable unit). The number of bytes required for every MAU is:

MAU Size	Bytes Required per MAU
1 - 8	1
9 - 16	2
17 - 32	4

winIDEA will check this parameter and return ICONNECT\_E\_SIZE in case of mismatch.

*pbyBuf*

The buffer to receive the read out data. Note that for memory areas with MAU size other than 8 bits (PIC Program, ARM CP, etc), the *pbyBuf* must be large enough to accommodate all data. Data is placed in consecutive locations in the buffer, i.e. the MAU from *aAddress* is placed first, MAU from *aAddress+1* is placed next etc.

Within a MAU the byte ordering is LSB. For the 14 bit maProgramPIC16XX area a read from address 10 would yield the following format:

ADDR = 10		ADDR = 11		ADDR = 12	
(7-0) pbyBuf [0]	xx(13-8) pbyBuf [1]	(7-0) pbyBuf [2]	xx(13-8) pbyBuf [3]	(7-0) pbyBuf [4]	xx(13-8) pbyBuf [5]

*pbyAccessInfo*

The buffer to receive information about access to every individual location. One byte must be allocated per MAU. If this parameter is NULL, no access information will be returned.

*Return value*

ICONNECT_E_CAN_NOT_ACCESS	Memory can not be accessed. This can be caused by bad memory area specification, out of range address, unavailability of the requested access mode (real-time) or the CPU is not in a mode that allows memory access.
ICONNECT_E_SIZE	The <i>byBytesPerMAU</i> value is incorrect for the specified memory area.

**HRESULT WriteMemory (DWORD dwAccessFlags, BYTE byMemArea, ADDRESS aAddress, ADDRESS aNumMAUs, const BYTE \* pbyBuf, BYTE byBytesPerMAU, BYTE \* pbyAccessInfo);**

Writes memory to the target system.

For description of arguments and return value see *ReadMemory*. Note that *fCacheXXXX* access flags do not apply.

**HRESULT ReadValue (DWORD dwAccessFlags, BYTE byMemArea, ADDRESS aAddress, SValue \* pValue, const SType \* pType);**

Reads a value from the target system. This function resembles *ReadMemory* but additionally it formats the value according to the type specified (including endian conversions).

*dwAccessFlags*

See *ReadMemory*.

*byMemArea*

See *ReadMemory*.

*aAddress*

See *ReadMemory*.

*pValue*

Pointer to a SValue type variable that will receive the value.

*pType*

Pointer to a SType type variable containing the type info of *pValue*.

**HRESULT WriteValue (DWORD dwAccessFlags, BYTE byMemArea, ADDRESS aAddress, const SValue \* pValue, const SType \* pType);**

Writes a value to the target system. This function resembles *WriteMemory* but additionally it formats the value according to the type specified (including endian conversions).

*dwFlags*

See *WriteMemory*.

*byMemArea*

See *WriteMemory*.

*aAddress*

See *WriteMemory*.

*pValue*

Pointer to a SValue type variable containing the value to write.

*pType*

Pointer to a SType type variable containing the type info of *pValue*.

**HRESULT ReadRegister (DWORD dwAccessFlags, LPCTSTR pszRegisterName, SValue \* pValue, SType \* pType);**

Reads the specified register.

*dwAccessFlags*

Specify which register domain to access:

fCore	CPU core register
fSFR	Special function register
fMonitor	Use regular memory access
fRealTime	Use real time access
fCacheNever	Do not cache the access
fCacheDefault	Cache according to winIDEA settings
fCacheStop	Cache while CPU is stopped

*pszRegisterName*

Name of the register to read.

*pValue*

Pointer to a SValue type variable that will receive the value.

*pType*

Pointer to a SType type variable containing the type info of *pValue*.

*Return value*

ICONNECT_E_BAD_ID	The specified register does not exist.
ICONNECT_E_SIZE	The specified register is too long to fit SValue.
ICONNECT_E_CAN_NOT_ACCESS	The register can not be accessed in the current CPU mode.

## **HRESULT WriteRegister (DWORD dwAccessFlags, LPCTSTR pszRegisterName, const SValue \* pValue, const SType \* pType);**

Writes the specified register.

For description of arguments and return value see *ReadRegister*.

## **HRESULT GetStatus (DWORD dwStatusFlags, SStatus \* pStatus);**

Obtains status of the CPU.

### *dwStatusFlags*

Specifies which status is obtained.

tCPU	Status of the CPU
rCurrent	Last status obtained by winIDEA
rRefresh	Status refresh is forced
sWantStopReason	If this flag is set, stop reason is returned in <i>m_byStatus</i>

### *pStatus*

Pointer to a *SStatus* type variable that will receive the status.

```
struct SStatus
{
    enum EStatus
    {
        stMustInit    = 0x00, // the debug system must initialize
        stStopped     = 0x01, // CPU is stopped
        stRunning     = 0x02, // CPU is running
        stReset       = 0x03, // CPU is held in reset
        stHalted      = 0x04, // CPU is halted by target
        stWaiting     = 0x05, // CPU is halted by emulator
        stAttach      = 0x06, // debugger is initialized and waiting for hot attach on the target
        stIdle        = 0x07, // CPU idle

        srExplicit    = 0x10, // CPU is stopped explicitly by the user
        srBP          = 0x20, // CPU is stopped after execution breakpoint hit
        srStep        = 0x30, // CPU is stopped after a step/over/until/until return command
        srHW          = 0x40, // CPU is stopped after a hardware breakpoint hit
        srMask        = 0x70,

    };
    BYTE m_byStatus;
    BYTE m_byExecutionArea;
    ADDRESS m_aExecutionPoint; // current execution point. Valid when stStopped
};
```

**HRESULT SetBreakpoint (DWORD dwBreakpointFlags, BYTE byMemArea, ADDRESS aAddress, LPCTSTR pszAddress, DWORD dwLine);**

Controls breakpoint setting.

*dwBreakpointFlags*

Determines the breakpoint operation:

bAddress	Use the aAddress as breakpoint address
bSymbol	Use the pszAddress as breakpoint address
bSource	Use the pszAddress and dwLine as breakpoint address

bSet	Set a breakpoint
bClear	Clear a breakpoint
bEnable	Enable a breakpoint
bDisable	Disable a breakpoint
bAll	Use in combination with <i>bClear</i> , <i>bDisable</i> and <i>bEnable</i> . When used, it applies to all configured breakpoints.

*byMemArea*

Determines the address space of the breakpoint. This value is used only if *bAddress* flag is used.

*aAddress*

Determines the address of the breakpoint. This value is used only if *bAddress* flag is used.

*pszAddress*

Determines the address of the breakpoint.

- If *bSymbol* flag is used this string is interpreted as a symbol (function name, code label).
- If *bSource* flag is used, this string is interpreted as file name.

*dwLine*

If *bSource* flag is used, *dwLine* specifies the line number.

*Return value*

ICONNECT_E_BAD_ID	The <i>pszAddress</i> does not evaluate to a location.
ICONNECT_E_ERROR	Breakpoint could not be set.
ICONNECT_E_NO_BREAKPOINT	No breakpoint was found at the specified address.

## HRESULT RunControl(DWORD dwRunControlFlags, BYTE byMemArea, ADDRESS aAddress);

Performs a run control operation.

### *dwRunControlFlags*

The *rRunMask* fields specify the operation to be performed:

rReset	Reset the CPU
rResetAndRun	Reset and runs the CPU
rDownload	Download the executable
rStop	Stop the CPU
rRun	Run the CPU
rStep	Execute a single instruction step or high level step according to context
rStepOver	Same as <i>rStep</i> , do not enter subroutine calls
rStepInst	Execute a single instruction step
rStepOverInst	Same as <i>rStepInst</i> , do not enter subroutine calls
rStepHigh	Execute a high level step
rStepOverHigh	Same as <i>rStepHigh</i> , do not enter function calls
rRunUntil	Run until <i>aAddress</i>
rRunUntilReturn	Run until current function returns
rGoto	Preset execution point to <i>aAddress</i>

The *rPollingMask* fields allow enabling and disabling of periodic status polling:

rPollingOff	Disable periodic polling
rPollingOn	Enable periodic polling (default)

### *byMemArea*

Determines the address space of the breakpoint.

### *aAddress*

Address used by *rRunUntil* and *rGoto* commands.

### *Return value*

ICONNECT_E_NOT_AVAILABLE	The requested operation is not available.
--------------------------	---

### *Remarks*

Note that *rRun*, *rRunUntil* and *rRunUntilReturn* functions are ‘non-blocking’ – when they return the CPU might still be running.

## ***IConnectProfiler* Interface**

The *IConnectProfiler* interface provides access to real-time measurement functions exported by winIDEA.

### **HRESULT GetInfo (DWORD dwSize, SInfo \* pInfo);**

Obtains information about the profiler capabilities.

*dwSize*

Specifies the size of the structure pointed to by pInfo.

*pInfo*

Pointer to structure that will receive the information. The structure has the following members:

```
struct SInfo
{
    enum ERequiredInfo
    {
        riCapabilities = 0x00000001,
    };
    DWORD m_dwRequiredInfo; // info to be returned.
                                // on return it contains flags about the info actually returned
    struct SCapabilities
    {
        enum EFlags
        {
            flAvail = 0x00000001, // profiler available
            flCanUseStartingPoint = 0x00000002,
            flCanProfileExec = 0x00000004,
            flCanProfileFuncLines = 0x00000008,
            flCanProfileData = 0x00000010, // in which mode data can be profiled
            flSingleData = 0x00000020, // can profile single data
            flSingleDataAddress = 0x00000040, // single data requires address specification
            flSingleDataSize = 0x00000080, // single data requires size specification
        };
        DWORD m_dwFlags;
        DWORD m_dwReserved;
        DWORD m_dwNumExecAreas; // number of exec areas
        DWORD m_dwNumDataAreas; // number of data areas
        BYTE m_bySingleDataWidth; // MAUs
        BYTE m_byReserved[3];
    };
    SCapabilities m_Capabilities;
};
```

## HRESULT StartConfig (DWORD dwStartConfigFlags, const SProfilerStartingPoint \* pStartingPoint)

Begins configuration for a new session of the profiler. Once a profiler is configured, it can be activated any number of times.

### *dwStartConfigFlags*

cfTimeStampTime	Default time resolution
cfTimeStampM0	Timestamp resolution in magnitude order of 1 second
cfTimeStampM1	Timestamp resolution in magnitude order of 100 milliseconds
...	
cfTimeStampM9	Timestamp resolution in magnitude order of 1 nanosecond

### *pStartingPoint*

Specifies the condition that triggers the profiler after it has been activated.

```
struct SProfilerStartingPoint
{
    enum EEvent
    {
        eAnything          = 0x00, // start immediately
        eExecution         = 0x01, // execution from a specified address
        eRead              = 0x02, // read access
        eWrite             = 0x03, // write access
        eRW                = 0x04, // read or write access
        eAUX_State        = 0x05, // state on a certain AUX inputs. m_dwDataAUX, m_dwDataAUXRM
                               // determine value/mask
        eAUX_Edge         = 0x06, // transition on certain AUX input. m_dwDataRM configures a
                               // single bit only, m_dwData value 0 determines falling, 1 rising
        eMaskEvt          = 0x0F,

        eAddressRange     = 0x00, // starting point is execution or access within a range
                               // of addresses [m_aAddress - m_aAddressRM]
        eAddressMask      = 0x10, // starting point is execution or access on specific mask:
                               // (m_aAddress & m_aAddressRM)

        eDataRange        = 0x00, // starting point is read/write of data within the range
                               // [m_dwData - m_dwDataRM]
        eDataMask         = 0x20, // starting point is read/write of data with a specific mask:
                               // (m_dwData & m_dwDataRM)
    };
    BYTE m_byEvent;
    BYTE m_byMemArea; // memory area of m_aAddress
    BYTE m_byDataSize; // number of MAUs to consider. 0 = full
    BYTE m_byReserved;

    ADDRESS m_aAddress; // used for eRead, eWrite, eRW
    ADDRESS m_aAddressRM; // -||-

    DWORD m_dwDataAUX; // m_dwDataAUXRM masks bits to be considered and m_dwDataAUX
    DWORD m_dwDataAUXRM; // specifies the value. If data is of no interest, m_dwDataAUXRM
                          // should be zero
};
```

---

Note: event availability depends on the emulation capabilities of the emulator/CPU. Mostly only *eAnything* and *eExecution* are available.

---

**HRESULT AddArea (DWORD dwAreaFlags, DWORD \* pdwHandle, LPCTSTR pszName, BYTE byMemArea = 0, ADDRESS aAddress = 0, ADDRESS aSize = 0, DWORD dwNumExits = 0, ADDRESS \* paExits = NULL);**

Adds an area to the current profiler configuration.

*dwAreaFlags*

afValueTypeMask	lower 4 bits used as by EProfilerDataValueType
afTypeFunction	The (execution) area is a high level function
afTypeRoutine	The (execution) area is a low level routine
afTypeVariable	The (data) area is variable specified by its name
afTypeRegion	The (data) area is a region of memory specified by address and size
afFunctionIncludeLines	valid with afTypeFunction - includes function lines of the specified function
afDataTaskID	valid with afTypeVariable, afTypeRegion, identifies that the specified area is the task ID object
afDataSingleData	valid with afTypeVariable, afTypeRegion - area is the 'single data' (OTM style) object. if CPUs single data resource is configurable, byMemArea/aAddress and aSize should be specified

*pdwHandle*

Specifies the location of a **DWORD** type variable that will accept the handle of the new area. This handle is subsequently used in calls to *GetStatistics* and *GetHistory*.

As of version 9.9.52 a *hpInvalid* (0xFFFFFFFF) is returned. The *pdwHandle* can be NULL. To obtain handles for result access, use the **IconnectProfiler2** interface.

*pszName*

Specifies the name of the area. For *afTypeFunction* and *afTypeVariable*, this name is used to obtain the address. For *afTypeRoutine* and *afTypeRegion* the name will be displayed in the profiler window.

*byMemArea, aAddress*

Specify the entry point for *afTypeRoutine* and starting address for *afTypeRegion*.

*aSize*

Specifies the MAU size for *afTypeRegion*.

*dwNumExits*

Specifies the number of exits for *afTypeRoutine*.

*paExits*

Points to an array of **ADDRESS** type objects, each specifying an exit point from *afTypeRoutine* object. The array must contain *dwNumExits* elements.

*Return value*

ICONNECT_E_BAD_ID	The area could not be added. Either an area with that name already exists, or parameters are not valid.
-------------------	---

## HRESULT Activate (DWORD dwActivateFlags)

(De)activates the profiler. When activating the profiler, the configured areas are evaluated at this time. Activation can fail if an area symbol does not evaluate or number of resulting monitoring points exceeds hardware capabilities.

### *dwActivateFlags*

aStart	Activates (arms) the profiler
aStop	Deactivates the profiler – note the profiler deactivates automatically when the trace buffer is filled.

### *Return value*

ICONNECT_E_ERROR	The requested operation could not be performed. Either the configuration is invalid or the profiler is not available.
S_FALSE	Activate(aStop) was called but the profiler is not active

## HRESULT GetStatus(DWORD dwGetStatusFlags, SStatus \* pStatus)

Polls the profiler status.

### *dwGetStatusFlags*

gsfNumRecorded	m_dwNumAvailable specifies the number of recorded samples
gsfNumAnalyzed	m_dwNumAvailable specifies the number of analyzed samples

### *pStatus*

Specifies the location of the structure that will receive the current status.

```
struct SStatus
{
    enum EStatus
    {
        stMustInit,           // the profiler is not configured
        stIdle,               // the profiler is configured, but not active
        stWaiting,           // the profiler is active, waiting for starting point
        stActive,            // the profiler is active and recording data
    };
    BYTE m_byStatus;
    enum EStatusEx
    {
        stexErrorsInBuffer = 0x01, // some recordings are incomplete due to OCT bottleneck
        stexBufferOverrun = 0x02, // session was terminated automatically when buffer was full
        stexLoading = 0x04, // the buffer is being loaded
    };
    BYTE m_byStatusEx;
    BYTE m_byReserved[2];
    DWORD m_dwNumAvailable; // number of recorded entries
};
```

**HRESULT GetStatistics (DWORD dwResultFlags, DWORD dwHandle, DWORD dwTask, DWORD dwValue, DWORD \* pdwNumStatistics, SStatistic \* pStatistics)**

Retrieves statistic results for specified area(s).

*dwResultFlags*

rfByHandle	retrieve data for the area specified by dwHandle only
rfAllAreas	retrieve data for all areas
rfFilterTask	include only data within specified task. Use with rfByHandle or rfAllAreas
rfFilterValue	include data results for a specific value of a data area only. Use with rfByHandle.

*dwHandle*

Handle of the area for which the statistics should be retrieved. Used with *rfByHandle*.

*dwTask*

Task ID for which the statistics should be retrieved. Used with *rfFilterTask*.

*dwValue*

Data value for which the statistics should be retrieved. Used with *rfFilterValue*.

*pdwNumStatistics*

On input, specifies the size of array *pStatistics* points to (i.e. the maximum number of statistics to retrieve).

On output, specifies the number of statistics placed into *pStatistics*.

*pStatistics*

Specifies the location of array of *SStatistic* objects that will receive the statistics results.

```

struct SStatistic
{
    DWORD m_dwHandle; // handle of the area for which this statistic is for
    DWORD m_dwValue; // (data areas), identifies the value for which the statistic is for
    QWORD m_qwTotalTime; // total time spent in the area
    QWORD m_qwMaxTime; // max time spent in a single hit
    QWORD m_qwMinTime; // min time spent in a single hit
    DWORD m_dwNumHits; // number of hits
};
    
```

*Return value*

ICONNECT_E_ERROR	Profiler session was not started. There are no valid results available.
------------------	---

## HRESULT GetHistory (DWORD dwResultFlags, DWORD dwHandle, DWORD dwTask, DWORD dwValue, DWORD dwHistoryBase, DWORD \* pdwNumHistories, SProfilerHistory \* pHistories)

Retrieves history of invocations for specified area(s).

*dwResultFlags*, *dwHandle*, *dwTask*, *dwValue*

See *GetStatistics*.

*dwHistoryBase*

Specify the zero based offset of the history. Use this in consecutive calls to *GetHistory*:

```
SProfilerHistory aHistories[0x100];
for (DWORD dwHistoryBase = 0; ; )
{
    DWORD dwNumHistories = 0x100;
    pIConnectProfiler->GetHistory(IConnectProfiler::rfAllAreas, 0, 0, 0,
                                dwHistoryBase, &dwNumHistories, aHistories);

    if (0 == dwNumHistories)
        break;
    ProcessHistories(dwNumHistories, aHistories);
    dwHistoryBase += dwNumHistories;
}
```

*pdwNumHistories*

On input, specifies the size of array *pHistories* points to (i.e. the maximum number of histories to retrieve). On output, specifies the number of histories placed into *pHistories*.

*pHistories*

Specifies the location of array of *SProfilerHistory* objects that will receive the history results.

```
struct SProfilerHistory
{
    DWORD m_dwHandle; // handle of the area for which this history is for
    DWORD m_dwValue; // value for which the statistic is for
    QWORD m_qwTime; // hit time in nanoseconds
    enum EValue // history for an individual area where Value is implicitly known
    {
        // (Execution area, vtState, ...) m_dwValue holds the bellow to indicate
        valExecActive = 0x80000000, // the area is currently active (executing)
        valExecInactive = 0x00000000, // the area is not active, but might be on stack
        valExecNestingMask = 0x7FFFFFFF, // nesting level mask - the difference between number
        // of entries and exits
    };
};
```

---

Note: *SProfilerHistory::m\_dwValue* will hold values as by *EValue* enum for all execution areas (*afTypeFunction*, *afTypeRoutine*) and for data areas (*afTypeVariable*, *afTypeRegion*) of type *vtState*, *vtLSB\_Exit\_Entry*, *vtZero\_Exit\_Entry* if history is obtained with *rfFilterValue*. For *vtRegular* or data areas obtained without *rfFilterValue*, the value holds the value written to the object.

Note: The first item for any history will always be on time 0. This item is inserted explicitly. If histories for all areas are retrieved, the first item will have handle value *hpInvalid*.

---

*Return value*

ICONNECT_E_ERROR	Profiler session was not started. There are no valid results available.
------------------	---

## ***ICoconnectProfiler2* Interface**

The *ICoconnectProfiler2* interface is a polymorphic extension to the *ICoconnectProfiler* interface - all functions in the *ICoconnectProfiler* are also available through the *ICoconnectProfiler2*. This section describes only the additional functions.

### **HRESULT GetResultInfo (DWORD dwFlags, DWORD dwSize, SResultInfo \* pResultInfo)**

Returns information about the results of the last profiler session. These results are valid after the profiler has been activated.

*dwFlags*

Reserved. Must be zero.

*dwSize*

Size of the structure pointed to by *pResultInfo*.

*pResultInfo*

Pointer to a structure which will receive the result information.

```
struct SResultInfo
{
    DWORD m_dwNumAreas; // number of result areas
};
```

---

Note: Area handles are assigned starting from zero. Thus handles in range [0, *m\_dwNumAreas*-1] are valid.

---

*Return value*

ICONNECT_E_ERROR	Profiler session was not started. There are no valid results available.
------------------	---

### **HRESULT GetResultArea (DWORD dwFlags, DWORD dwSize, SResultArea \* pResultArea)**

Returns information about a specific area of the last profiler session. These results are valid after the profiler has been activated.

*dwFlags*

```
enum EGetResultAreaFlags
{
    grafByHandle = 0x00000000, // return area info of the area specified in m_dwHandle
    grafByName   = 0x00000001, // return area info of the area specified in m_szName
    grafByAddress = 0x00000002, // return area info of the area specified in m_aAddress
    grafByMask   = 0x0000000F,

    grafTypeExec = 0x00000000, // requesting exec area - for grafByAddressOnly
    grafTypeData  = 0x00000010, // requesting data area - for grafByAddressOnly
    grafTypeMask  = 0x00000030,
};
```

*dwSize*

Size of the structure pointed to by *pResultInfo*.

## *pResultArea*

Pointer to a structure which will receive the result information.

```
struct SResultArea
{
    TCHAR    m_szName[256]; // name of the area
    DWORD    m_dwHandle;    // handle of the area
    ADDRESS  m_aAddress;    // address of the area
};
```

### *Return value*

ICONNECT_E_BAD_ID	No area matching the specified search parameter was found.
ICONNECT_E_ERROR	Profiler session was not started. There are no valid results available.

## ***IConnectProject* Interface**

The *IConnectProject* interface provides access to build manager functions exported by winIDEA.

### **HRESULT GetInfo (DWORD dwSize, SInfo \* pInfo)**

Obtains information about the build manager capabilities.

*dwSize*

Specifies the size of the structure pointed to by pInfo.

*pInfo*

Pointer to structure that will receive the information. The structure has the following members:

```
struct SInfo
{
    enum ERequiredInfo
    {
    };
    DWORD m_dwRequiredInfo; // info to be returned.
                           // on return it contains flags about the info actually returned
};
```

*Return value*

S_OK	Success
ICONNECT_E_NOT_CONNECTED	The <i>isystem.connect</i> object is not attached to winIDEA.

### **HRESULT Operation (DWORD dwOperationFlags, LPCTSTR pszOperation, LPCTSTR pszParameters)**

Invokes a build manager operation.

*dwOperationFlags*

ofCustom	Run custom operation specified by pszOperation
ofCompile	Compile the file specified by pszParameters. The file must be included in the active project.
ofLink	Link the active project.
ofMake	Make the active project.
ofBuild	Build the active project.
ofDependencies	Update dependencies.
ofStop	Stop the current operation.
ofSuppressPostLinkActions	Do not run any automatic post-link actions after successful link operation

*pszOperation*

Identifies the custom operation to perform. The specified operation must be defined as custom operation in winIDEA. This parameter is ignored if *ofCustom* flag is not set.

*pszParameters*

Specifies the additional parameters for the requested operation.

*Return value*

ICONNECT_E_NOT_AVAILABLE	The requested operation is not available.
ICONNECT_E_FILE_NOT_FOUND	The specified file is not a part of the active project.
E_NOTIMPL	The requested operation is not implemented.

**HRESULT GetStatus (DWORD dwStatusFlags, SStatus \* pStatus)**

Obtains the status of the build manager.

*dwStatusFlags*

Reserved, must be zero.

*pStatus*

Pointer to a structure that will receive the current status.

```
struct SStatus
{
    enum EStatus
    {
        stIdle,           // build manager is idle
        stIdleError,     // build manager is idle, last operation finished with errors
        stConfigError,   // build manager is misconfigured
        stActive,        // build manager is active
    };
    DWORD    m_dwStatus;
    DWORD    m_dwNumErrors;
    DWORD    m_dwNumWarnings;
};
```

**HRESULT Option (DWORD dwOptionFlags, LPCTSTR pszScope, LPCTSTR pszOption, LPTSTR pszValue, DWORD dwValueLen)**

Sets or retrieves a build manager configuration option.

*dwOptionFlags*

The following options are used exclusively (i.e. specify only one)

ofaGet	Retrieves an option value
ofaEnum	Used with ofaGet, when the flag is set, the lower 16 bits of dwOptionFlags ( <i>ofaEnumIndexMask</i> ) should contain the index number of the option (e.g. to enumerate project files, include directories, etc.).  If the index is larger than the entity size, the function will return <i>S_FALSE</i> .
ofaSet	Sets an option value
ofaAdd	Adds an option value (e.g. a project file)
ofaRmv	Removes an option (e.g. a project file)
ofaMov	Moves an option to a different location (e.g. move a project file to a different group)

The following options can be used concurrently with the above.

ofaRelativePaths	This flag indicates that the paths should be returned in relative form rather than absolute.
------------------	--

## *pszScope*

Identifies the option location. The format of the scope string is:

">project>group>file name|translator|target"

Alternatively, the group can be omitted when referencing a project file already in the project:

">project>file path|translator|target"

In case scope path parts are missing, the location is interpreted using the below table:

Scope format	Location	Example	Example pszOption	Example pszValue
""	Global	""	"BeepWhenDone"	"1"
">project>"	Project	">Files>"	"ToolsetDir"	"\$(EXEDIR)\gcc"
">project>group>"	Group	">Files>Source Files>"	"Name"	"Sources"
">project>group>file name" ">project>file path"	File	">Files>Source Files>main.c" ">Files>c:\test\main.c"	"ExcludeFromMake"	"0"
">project> translator"	Project	">Files Compile"	"ExtInput"	"c;cpp"
">project> translator target"	Project	">Files Compile Debug"	"CmdLine"	"-g"
">project>file translator target"	File	">Files>c:\test\main.c Compile Debug"	"CmdLine"	"-g -o"
<i>etc.</i>				

If project or target are blank, the current project/active target is assumed.

## *pszOption*

Identifies the option to access. The below table lists all available options.

**Type, Operation** column specifies the type of the **pszValue** parameter and the available operations.

Types:

- B - BOOL "0" or "1".
- S - string, value as in *pszValue*.
- -, *pszValue* is not used

Operation:

- E - ofaEnum
- G - ofaGet
- S - ofaSet
- A - ofaAdd
- R - ofaRmv
- M - ofaMov

Global options

Scope	Option	Type, Oper.	Description
	BeepWhenDone	B, GS	<b>Settings/Customize/Beep when done</b>

## Project options

Scope	Option	Type, Oper.	Description
>project>	OutputFile	S, GS	<b>Settings/General/Output file name</b>
>project>	RootDir	S, GS	<b>Settings/General/Root directory</b>
>project>	ToolsetDir	S, GS	<b>Settings/General/Compiler toolset path</b>
>project>	RunWithRelativePaths	B, GS	<b>Settings/Customize/Run translators with relative paths</b>
>project>	CopyToTargetDir	B, GS	<b>Settings/Customize/Copy output files to target directory</b>
>project>	ChangeWorkingDir	B, GS	<b>Settings/Customize/Change working directory</b>
>project>	DisplayParam	B, GS	<b>Settings/Customize/Display tool parameters</b>
>project>	ShowTool	B, GS	<b>Settings/Customize/Show tool window</b>
>project>	WarnIncludeNotFound	B, GS	<b>Settings/Includes/Warn if include file is not found</b>
>project>	Environment	S, GSAR	<b>Settings/Customize/Environment settings</b> <i>ofaGet/ofaSet</i> retrieve/set full configuration, individual strings are separated with <b>NL (0x0A)</b> characters. <i>ofaAdd/ofaRmv</i> remove individual environment variables entries. An environment entry must be of the form: <variable name>=<value> Example: PATH=C:\Compiler;%PATH%
>project>	IncludePath	S, E	<b>Settings/Includes/Includes search paths</b> enumeration.
>project>	IncludePath>path	-, AR	<b>Settings/Includes/Includes search paths</b> Add or Remove a path
>project>	IncludePath>path	B, GS	<b>Settings/Includes/Includes search paths</b> Defines the <b>Search subdirectories</b> option for the specified <i>path</i> .
>project>	Name	S, GS	Project name
>project>	Filter	S, GS	<b>Settings/General/Error Filter</b>
>project>	Target	S, E	Available targets for the project.
>project>	Group	S, E	Groups in the project.
>project>	File	S, E	Enumerate files in the project.

## Target options

Scope	Option	Type, Oper.	Description
>project>  target	Name	S, GS	Target name
>project>  target	Target	-, GSR	<i>ofaGet/ofaSet</i> retrieve/set active target.
>project>  target	Target	S, A	Add a target. If <b>pszValue</b> is not empty, it specifies the target from which the default translator settings are copied.
>project>  target	OutputDir	S, GS	<b>Settings/General/Exec.directory</b>
>project>  target	UseIntermediateDir	B, GS	<b>Settings/General/Intermediate files directory (enable/disable)</b>
>project>  target	IntermediateDir	S, GS	<b>Settings/General/Intermediate files directory</b>

## Translator options

Scope	Option	Type, Oper.	Description
>project> translator	Mode	S, GS	<b>Settings/Build.</b> 0 for internal, 1 for external.
>project> translator	Path	S, GS	Compiler, Assembler, Linker, External Make and External Build executable paths.
>project> translator	ExtInput	S, GS	<b>Settings/File Type/Input Ext</b>
>project> translator	ExtOutput	S, GS	<b>Settings/File Type/Output Ext</b>
>project> translator	ExtInclude	S, GS	<b>Settings/File Type/Header Ext</b>
>project> translator	ExtAdditional	S, GS	<b>Settings/File Type/Extra Ext</b>
>project> translator	RunAfterPath	S, GS	<b>Settings/Customize/Run Before/After/Path</b>
>project> translator	RunAfterParam	S, GS	<b>Settings/Customize/Run Before/After/Command line</b>
>project> translator	RunBeforePath	S, GS	<b>Settings/Customize/Run Before/After/Path</b>
>project> translator	RunBeforeParam	S, GS	<b>Settings/Customize/Run Before/After/Command line</b>
>project> translator	CaptureOutput	B, GS	<b>Settings/Customize/Translator Execution/Capture translator output</b>
>project> translator	UseCMD	B, GS	<b>Settings/Customize/Translator Execution/Use command interpreter</b>
>project> translator	CustomFilter	S, GS	<b>Settings/General/Advanced/Compiler Assembler Linker.</b> Blank value indicates usage of default error file.
>project>	CustomFilter	S, GS	<b>Settings/General/Advanced/Custom Filter.</b> Blank value indicates no external filter
>project> translator	PathSeparator	S, GS	<b>Settings/Linker/Path Separator.</b> <i>pszValue</i> must contain exactly one character.
>project> translator	DepScan	B, GS	<b>Settings/Includes/Check Assembler Includes</b>
>project> translator	DepReqBlanksFront	B, GS	<b>Settings/Includes/Require leading blank(s)</b>
>project> translator	DepAllowBlanksFront	B, GS	<b>Settings/Includes/Allow leading blank(s)</b>
>project> translator	DepKeyword	S, GS	<b>Settings/Includes/Include Keyword</b>
>project> translator	DepStartPath	S, GS	<b>Settings/Includes/Path start char</b>
>project> translator	DepEndPath	S, GS	<b>Settings/Includes/Path end char</b>
>project> translator	DefinesPrefix	S, GS	<b>Settings/Compiler/Prefix</b>
>project> translator	DefinesPostfix	S, GS	<b>Settings/Compiler/Postfix</b>

## Target related translator options

Scope	Option	Type, Oper.	Description
>project> translator target	CmdLine	S, GS	<b>Settings/translator/Options</b>
>project> Compile target	RunAssembler	B, GS	<b>Settings/Compiler/Run Assembler</b>
>project> Link target	IndirectionPath	S, GS	<b>Settings/Linker/Indirection File</b>
>project> Link target	CRLF	B, GS	<b>Settings/Linker/CR-LF after file name (one file name per line)</b>
>project> Link target	SkipObjectCheck	B, GS	<b>Settings/Linker/Skip object file existence check</b>
>project> Link target	TranslateChar	S, GS	<b>Settings/Linker/Translate character.</b> <i>pszValue</i> must contain exactly one character.

## Group options

Scope	Option	Type, Oper.	Description
>project>group>	Name	S, GS	Group name
>project>group>	Group	-, AR	Add/Remove group
>project>group>	File	S, E	Enumerate files in the group.

## File options

Scope	Option	Type, Oper.	Description
>project>group>file	File	-, AR	Add/Remove file
>project>group>file	File	S, M	Move file to a group specified in <i>pszValue</i>
>project>group>file translator target	CmdLine	S, GS	<b>Settings/translator/Options</b>
>project>group>file Compile target	RunAssembler	B, GS	<b>Settings/Compiler/Run Assembler</b>
>project>group>file  target	ExcludeFromMake	B, GS	<b>Settings/General/Exclude file(s) from make</b>
>project>group>file translator target	Defines	S, GSAR	<b>Settings/Compiler/Command Line Defines</b> <i>ofaGet/ofaSet</i> retrieve/set full configuration, individual strings are separated with <b>NL (0x0A)</b> characters. <i>ofaAdd/ofaRmv</i> remove individual command line define entries. A command line define entry must be of the form: <define name>[=<value>] Example: DEBUG VERSION=3

### *pszValue*

Pointer to the string that defines the new value or in case *ofaGet* or *ofaEnum* flags are set, will receive the value of the option.

### *dwValueLen*

Length of the *pszValue* buffer.

### *Return value*

ICONNECT_E_BAD_ID	Invalid option specified by the <i>pszValue</i> parameter.
ICONNECT_E_FILE_NOT_FOUND	The <i>pszScope</i> does not identify a valid option.
ICONNECT_E_NOT_AVAILABLE	Requested operation is not available.
ICONNECT_E_ERROR	The current build manager configuration is not valid for this operation.
S_FALSE	Enumeration index bigger than the number of enumerated items.

### *Example: Set toolset directory*

```
TCHAR szValue[300] = "C:\\Compiler";
pIConnectProject->Option(IConnectProject::ofaSet, ">", "ToolsetDir", szValue, 300);
```

### *Example: Add a target*

```
// Add target 'Release', copy settings from existing target 'Debug'
TCHAR szValue[300] = "Debug";
pIConnectProject->Option(IConnectProject::ofaAdd, ">||Release", "Target", szValue, 300);
```

### *Example: Retrieve all project files*

```
TCHAR szValue[300] = "Debug";
for (WORD wEnumIndex = 0; ; wEnumIndex++) {
    hr = pIConnectProject->Option(IConnectProject::ofaEnum | wEnumIndex, ">", "File", sz, 300);
    if (S_OK != hr) break;
}
```

### *Example: Add a project file*

```
TCHAR szValue[300] = "";
pIConnectProject->Option(IConnectProject::ofaAdd, ">>Source Files>main.c", "File", sz, 300);
```

*Example: Set command line options for a file*

```
TCHAR sz[300] = "-g -O3 $(EDNAME)";  
pIConnectProject->Option(IConnectProject::ofaSet, ">>main.c", "CmdLine", sz, 300);
```

*Example: Set environment entry*

```
// set new PATH value. If a 'PATH' entry already exists, it will be overwritten  
TCHAR sz[300] = "PATH=C:\\Compiler;%PATH%";  
pIConnectProject->Option(IConnectProject::ofaAdd, ">", "Environment", sz, 300);
```

# Appendix

## Connection Troubleshooting

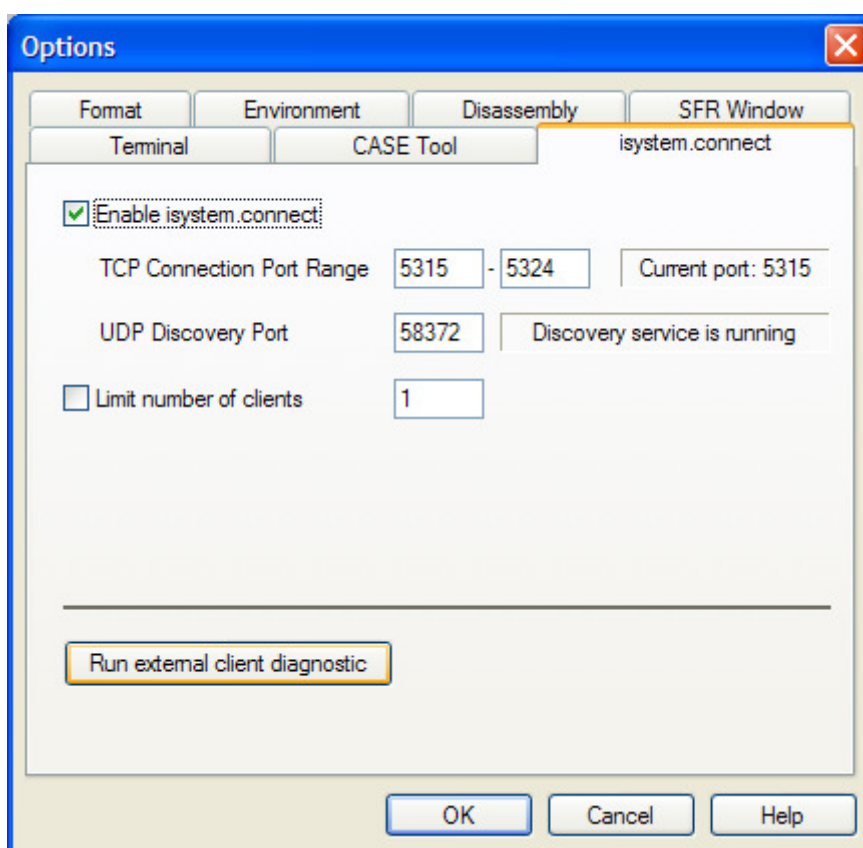
The *isystem.connect* interface operates via TCP/IP protocol. Many firewall and anti-virus software packages can block the UDP discovery or TCP connection ports. winIDEA configures the Windows firewall to allow *isystem.connect* traffic, but it does not do so with third party applications.

Another problem can be caused if instances of different version of winIDEA are running simultaneously.

If the *isystem.connect* client application can not connect to winIDEA, winIDEA can run a diagnostic utility which can help you determine the cause of the problem.

## Running the diagnostic

Diagnostic is started from the *Tools/Options/isystem.connect* dialog



The 'Run external diagnostic' button launches *iConnectDiag.exe* which is using our standard *isystem.connect* utility classes to enumerate all running instances of winIDEA and attempts to connect to each of them:

```
C:\ E:\winIDEA\iConnectDiag.exe
**** isystem.connect diagnostic ****
Loading iConnect.DLL: E:\winIDEA\iConnect.dll

Enumerating all running instances of winIDEA
*** winIDEA instance ****
Instance ID:
Workspace: E:\PRO\Sample\Active\Emu12\HC12\DP512\MC9S12Dx512\Nobank\Cosmic\Sample.jrf
TCP Port: 5315
Connect: OK
*****

*** winIDEA instance ****
Instance ID:
Workspace: E:\PRO\PPC\55xxCoverage\IntrAM\Sample_iTracePro.jrf
TCP Port: 5316
Connect: FAILED
VERSION CONFLICT
ERROR: <client> RC1: Can not connect. Error: 0. Connection refused by server
<server> RS2: Incompatible versions.
Client:9.7.114
Server:9.7.113
*****

Connect via port 5315
Connect: OK

Press any key to continue...
```

The above screenshot shows diagnostics when two different winIDEA build instances are active.

```
C:\ E:\winIDEA\iConnectDiag.exe
**** isystem.connect diagnostic ****
Loading iConnect.DLL: E:\winIDEA\iConnect.dll

Enumerating all running instances of winIDEA
Connect via port 5315
Connect: OK

Press any key to continue...
```

This screenshot shows firewall blocking UDP discovery but a direct TCP connect is still working.

```
C:\ E:\winIDEA\iConnectDiag.exe
**** isystem.connect diagnostic ****
Loading iConnect.DLL: E:\winIDEA\iConnect.dll

Connecting directly via TCP port 5315
Connect: OK

Enumerating all running instances of winIDEA
*** winIDEA instance ****
Instance ID:
Workspace: E:\PRO\U850\Fx3APRO\Fx3.jrf
TCP Port: 5316
Connect: OK
*****

*** winIDEA instance ****
Instance ID:
Workspace: E:\9.7\src\iC3kkrnl\iC3kkrnl.jrf
TCP Port: 5315
Connect: OK
*****

Press any key to continue...
```

This screenshot was taken with two instances running, with no UDP or TCP blocking.

# Revision History

Version	Date	Revisions
9.6.1	13.04.2005	Created
9.6.45	07.10.2005	Added IConnectProject interface
9.6.103	05.05.2006	IConnectDebug::GetStatus <ul style="list-style-type: none"> <li>Added <i>sWantStopReason</i> to <i>dwStatusFlags</i></li> <li>Added <i>srXXXX</i> flags to <i>SStatus::m_byStatus</i></li> </ul> IConnectProject::Option <ul style="list-style-type: none"> <li>Added access to command line defines via “Defines” option</li> </ul>
9.6.104	05.05.2006	IConnectProject::Option <ul style="list-style-type: none"> <li>Added access to command line defines Prefix via “DefinesPrefix” option</li> <li>Added access to command line defines Postfix via “DefinesPostfix” option</li> </ul>
9.7.9	11.07.2006	IConnectProject::Option <ul style="list-style-type: none"> <li>Added access to <i>Linker/Skip object file existence</i> via “SkipObjectCheck” option</li> </ul>
9.7.66	09.03.2007	IConnectIDE::Document <ul style="list-style-type: none"> <li>Added <i>daStart/daStop/daStatus</i> flags</li> <li>Added <i>dtCodeCoverage/dtDataCoverage</i> document types</li> </ul>
9.7.76	17.04.2007	IConnectDebug::Modify <ul style="list-style-type: none"> <li>Added <i>mfModifyToResult</i> flag</li> </ul>
9.7.95	27.06.2007	IConnectDebug::Document <ul style="list-style-type: none"> <li>Changed <i>pszFileNameNew</i> to <i>pszParameter</i> and <i>dwLine</i> to <i>dwParameter</i>.</li> <li>Added <i>daSelect</i> and <i>daResume</i> flags</li> </ul>
9.7.96	04.07.2007	IConnectDebug::Document <ul style="list-style-type: none"> <li>Added <i>dfSpecialWindow</i> flag.</li> <li>Added <i>dwTerminal</i> for Terminal window access</li> </ul>
9.7.104	13.08.2007	IConnectDebug::GetAddress <ul style="list-style-type: none"> <li>Added enumeration capabilities</li> </ul>
9.7.115	15.10.2007	IConnectProfiler::GetStatus <ul style="list-style-type: none"> <li>Added <i>SStatus::m_byStatusEx</i> information</li> </ul>
9.7.121	14.11.2007	IConnectProfiler::Operation <ul style="list-style-type: none"> <li>Added <i>ofSuppressPostLinkActions</i> flag</li> </ul>
9.7.133	28.12.2007	Added IConnect::GetLastError function
9.7.135	07.01.2008	IConnectIDE::Document <ul style="list-style-type: none"> <li>Added <i>daExport</i> flag and <i>EDocumentExportFlags</i> enum flags</li> </ul>
9.7.145	14.02.2008	IConnectIDE::Document

		<ul style="list-style-type: none"> <li>Added <i>daExportV</i> flag</li> </ul>
9.7.154	26.03.2008	IConnectProfiler::GetStatus <ul style="list-style-type: none"> <li>Added <i>stexLoading</i> flag to <i>SStatus</i></li> </ul>
9.9.5	15.04.2008	IConnectProfiler::Activate <ul style="list-style-type: none"> <li><i>S_FALSE</i> return implemented</li> </ul>
9.9.12	22.05.2008	IConnectIDE::Application <ul style="list-style-type: none"> <li>Added MessageBox hook manipulation functions</li> </ul>
9.9.17	16.06.2008	IConnectDebug::RunControl <ul style="list-style-type: none"> <li>Added polling control</li> </ul>
9.9.52	25.11.2008	IConnectProfiler <ul style="list-style-type: none"> <li>Return handle from <i>AddArea</i> can no longer be used to access results</li> <li>Added <i>IconnectProfiler2</i> interface to provide access to results</li> </ul> IConnectIDE::Document <ul style="list-style-type: none"> <li>Added <i>dmMarkPos</i> flag</li> <li>Added <i>daSetExportFormat</i> action</li> </ul>
9.9.53	8.12.2008	IConnectIDE::Document <ul style="list-style-type: none"> <li>Added <i>daDeactivate</i> action</li> </ul>
9.9.57	22.12.2008	IConnectIDE::SApplication <ul style="list-style-type: none"> <li>Window positions type changed from DWORD to LONG</li> </ul>
9.9.61	02.02.2009	IConnectDebug::GetAddress <ul style="list-style-type: none"> <li>Added extended type info capability</li> </ul>
9.9.67	05.03.2009	IConnectProfiler::GetStatus <ul style="list-style-type: none"> <li>Added <i>gafNumXXXX</i> flags to extract number of recorded/analyzed samples</li> </ul>
9.10.5	14.05.2009	IConnectIDE::Document <ul style="list-style-type: none"> <li>Added <i>ICONNECT_E_ALREADY_EXISTS</i> return</li> </ul>
9.10.7	28.05.2009	IConnectIDE::Document Added <i>ffSaveAsOverwrite</i> flag