Integrating the WiseMo Guest ActiveX in MFC Applications (Visual Studio 2022)

0.0 Table of Contents

Click on the table to skip directly to the paragraph.

1.0 Introduction

1.1 Overview

This guide demonstrates a **recommended runtime-only hosting approach** that avoids reliance on Visual Studio's design-time ActiveX support[1]. The solution is based on the following principles:

- Use of a lightweight placeholder control in the dialog resource
- Runtime creation of the ActiveX control using ATL ActiveX hosting
- Encapsulation of all COM, ATL, and event-handling logic in reusable helper classes

This approach:

- Works reliably with Visual Studio 2022 and later
- Avoids design-time incompatibilities and limitations
- Preserves full functionality of the WiseMo Guest ActiveX control
- Keeps application code simple, robust, and maintainable

1.2 Prerequisites

This guide describes one supported approach for encapsulating the **WiseMo Guest ActiveX** in an **MFC-based application** using **Microsoft Visual Studio 2022**.

The reader is expected to meet the following prerequisites:

- General familiarity with WiseMo products, including the WiseMo Remote Desktop Guest and Host components, and associated terminology
- Working knowledge of C++ and MFC application development, including dialog-based applications and basic message handling
- Basic understanding of COM and ActiveX concepts, such as interfaces, registration, and in-process controls (detailed ATL knowledge is not required)

[1] Starting with Visual Studio 2022, the Visual Studio IDE (devenv.exe) is a **64-bit application**. As a result, the Visual Studio designer and resource editors can no longer load or host **32-bit ActiveX controls at design time**. This change affects several common development workflows, including visual dialog editing and ActiveX insertion using the designer.

It is important to note that this limitation is **specific to design time**. The WiseMo Guest ActiveX control continues to work correctly at **runtime** when hosted by a **32-bit (Win32/x86) application**, including applications built with Visual Studio 2022.

- Experience with Visual Studio 2022, including project configuration, platform selection (Win32/x86), and debugging
- A registered installation of the WiseMo Guest ActiveX control (WGuestX.ocx) on the development and test machines
- A 32-bit (Win32/x86) build configuration, as the WiseMo Guest ActiveX control is a 32-bit in-process component
- Administrator rights on the development machine, if required for ActiveX registration and debugging

This guide does **not** require prior experience with ATL ActiveX hosting or COM connection-point implementation, as those details are fully encapsulated in the provided helper classes.

2.0 Integration guide

This section describes how to integrate the WiseMo Guest ActiveX control into an MFC-based application using the provided C++ helper classes.

2.1 Class overview

WiseMo provides two C++ wrapper classes, **CIWGuestX** and **CIWGuestSink**, that make it straightforward to embed and use the WiseMo Guest ActiveX control in an MFC application **without requiring customers to implement**:

- ATL window hosting (CAxWindow)
- COM interface acquisition (QueryInterface)
- Connection-point event subscription (Advise / Unadvise)
- Event forwarding and callback glue code

CIWGuestSink is deliberately separated from CIWGuestX to keep responsibilities clearly defined:

- **CIWGuestX**
  Hosting, lifecycle management, and the public integration API exposed to the application
- **CIWGuestSink**
  Implementation of the ActiveX event interface and forwarding of callbacks

This separation:

- Keeps the event sink implementation stable and reusable
- Prevents customers from having to write or maintain connection-point plumbing

- Ensures consistent callback behavior across different projects

In short: **your application owns the UI and layout; CIWGuestX owns the ActiveX control and the CIWGuestSink event handling**.

In most cases, you do not need to interact with CIWGuestSink directly. It is only relevant if you need to extend event forwarding to include rarely used ActiveX events that are not currently exposed by CIWGuestX.

2.2 Environment and Setup

2.2.1 Install and register the WiseMo ActiveX control

The WiseMo Guest ActiveX control (WGuestX.ocx) must be installed and registered on the development and test machines. This is normally handled automatically by the WiseMo installer.

If you do not already have the WiseMo Guest ActiveX control installed, you can request a trial version from:

https://shop.wisemo.com/purchase/RSM/DownloadTrial.aspx

The WiseMo Guest ActiveX installer:

- Installs and registers the ActiveX control
- Installs demo applications with source code
- Installs the ActiveX documentation

After installation, open **"WiseMo Guest Component API Reference"** from the Windows Start menu.
Within the documentation, click **IWGuestCore20** (the link is active only when the ActiveX is installed) to view the available functions and properties.

2.2.2 Build the app as Win32 (x86)

The WiseMo OCX is 32-bit, so the hosting process must be 32-bit.

2.2.3 Download demo sample

The demo sample can be downloaded here.

2.2.4 Distribution of the finished app

When your app is finished, the WiseMo ActiveX must be installed and registered with your application. As an alternative to registering the ActiveX, it can be sideloaded and referenced directly by your application without registration – please refer to the section *4.0 Sideloading the WiseMo Guest ActiveX without registration*.

2.3 Integration Architecture Overview

The WiseMo Guest ActiveX integration is intentionally split into **application-level code** and **reusable integration code**. This separation minimizes complexity in customer applications and isolates all COM- and ATL-specific logic.

Roles and responsibilities:

Component	Responsibility
Application (Dialog/View)	UI, layout, user interaction
CIWGuestX	ActiveX hosting, lifecycle, public integration API
CIWGuestSink	COM event sink and event forwarding
WiseMo ActiveX	Remote desktop functionality

The application interacts only with `CIWGuestX`.
All ActiveX hosting, event subscription, and COM lifetime management is handled internally.

2.4 Event Flow and Callback Handling

The WiseMo Guest ActiveX control exposes events using standard COM connection points. These events are delivered to the application through a controlled forwarding chain implemented by CIWGuestX and CIWGuestSink.

Event flow sequence

```
WiseMo ActiveX Control
        │
        │  (COM connection-point event)
        ▼
CIWGuestSink
        │
        │  (internal forwarding)
        ▼
CIWGuestX
        │
        │  (callback invocation)
        ▼
Application (CIWGuestXHelper)
```

2.5 What the Application Does *Not* Need to Implement

When using CIWGuestX, the application does **not** need to implement or understand any of the following:

- ATL window hosting (CAxWindow)
- ActiveX control creation (CreateControl)
- COM interface discovery (QueryInterface)
- Connection-point subscription (Advise / Unadvise)
- Event sink classes or IDispatch implementations
- ActiveX lifetime and cleanup logic
- Design-time ActiveX embedding in Visual Studio

All of the above concerns are fully encapsulated in CIWGuestX and CIWGuestSink.

2.6 Required Initialization (One-Time Application Setup)

Because the integration uses **ATL ActiveX hosting**, the application must perform a small amount of one-time initialization during startup.

2.6.1 OLE / COM initialization

The application must initialize OLE/COM before creating the ActiveX control:

```
if (!AfxOleInit())
   return FALSE;
```

Most MFC application templates already include this call.

2.6.2 ATL ActiveX hosting initialization

The application must initialize ATL ActiveX hosting once during startup:

```
#include <atlbase.h>
#include <atlhost.h>

AtlAxWinInit();
```

This registers the ATL window classes required by CAxWindow.

2.6.3 ATL module definition

When using ATL inside an MFC application, a minimal ATL module must be provided. This is done by including AtlModule.cpp in the project.

This file ensures that ATL's internal runtime infrastructure is initialized correctly and prevents runtime failures when creating or hosting ActiveX controls.

2.6.4 Summary

By using CIWGuestX and CIWGuestSink, applications can integrate the WiseMo Guest ActiveX control in a modern, robust way that:

- Is compatible with Visual Studio 2022 and later
- Avoids design-time ActiveX limitations
- Keeps application code clean and focused on UI logic
- Encapsulates all COM, ATL, and ActiveX complexity
- Provides a stable foundation for future maintenance

3.0 Minimal Integration Example

This section walks through a **minimal, end-to-end integration** of the WiseMo Guest ActiveX control into an MFC dialog-based application using **`CIWGuestX`**. The example demonstrates the recommended **runtime-only hosting approach** and highlights the small amount of application code required.

The goal is to present a **clear and working integration example** while keeping all ATL, COM, and ActiveX-related complexity fully encapsulated in the provided helper classes. While the example is intentionally kept compact, it is not the absolute smallest possible project, as it also includes additional guidance and best-practice hints.

The accompanying sample project implements all of the steps described in this section. As such, the steps serve both as **documentation** and as a **practical reference** when integrating the WiseMo Guest ActiveX control into your own application.

3.1 Visual Studio project setup

This section describes how to add the WiseMo Guest ActiveX integration package (WGuestX.zip) to an existing MFC application project in Visual Studio 2022.

Extract the integration package

1. Unzip WGuestX.zip.
2. The archive will extract to a folder named **WGuestX** containing the following files:
    - AtlModule.cpp
    - IWGuestX.cpp
    - IWGuestX.h
    - IWGuestSink.cpp
    - IWGuestSink.h
    - WGuestX.ocx
    - wguestx.tlh
    - wguestx.tli
3. Recommended: Place the WGuestX folder under your solution directory, for example: $(SolutionDir)\WGuestX\

Add the files to your Visual Studio project

In **Solution Explorer**:

1. Right-click **Source Files** → **Add** → **New Filter** → name it `WGuestX`
2. Browse to the WGuestX folder.
3. Drag all .cpp and .h files into the filter folder
4. If you prefer, you can put the *.h files in their own folder

This ensures the files are compiled as part of your project and are available for include/import.

Add the include directory for the WGuestX folder

To allow source files to include the WiseMo wrapper headers without using long relative paths, add the WGuestX folder to the project's include directories.

1. Right-click the project → **Properties**
2. Go to **C/C++ → General**
3. Under **Additional Include Directories**, add:

```
$( SolutionDir)WGuestX\
```

If you placed the folder under the project directory instead, use:

```
$( ProjectDir)WGuestX
```

Add a Post-Build Event to deploy WGuestX.ocx

The demo application expects WGuestX.ocx to be available alongside the executable at runtime. To automate this, add a **Post-Build Event** that copies the OCX into the output folder.

1. Right-click the project → **Properties**
2. Go to **Build Events** → **Post-Build Event**
3. Add the following command:

```
copy WGuestX\WGuestX.ocx $(SolutionDir)Bin\$(Configuration)_$(Platform)\
```

3.2 Add a placeholder control to the dialog resource

In the dialog resource (.rc file), add a simple placeholder control where the WiseMo Guest ActiveX control should appear.

Important notes:

- The placeholder is not an ActiveX control.
- Any simple control type can be used (e.g. Static).
- Use a real control ID (not IDC_STATIC).

Example:

```
CONTROL "", IDC_WGUESTX1, "Static", WS_CHILD | WS_VISIBLE, 0, 0, 320, 240
```

This placeholder defines the initial position and size. It will be replaced at runtime by the ActiveX host window.

3.3 Add CIWGuestX to the dialog class

Include the wrapper header and add a CIWGuestX member to the dialog class.

The dialog also implements CIWGuestXHelper to receive callbacks.

```
#include "IWGuestX.h"

class CWGuestXDemoDlg :
   public CDialogEx,
   public CIWGuestXHelper
{
public:
   CWGuestXDemoDlg();

protected:
   CIWGuestX m_wguest;
};
```

3.4 Create the ActiveX control in OnInitDialog

In OnInitDialog, initialize and create the ActiveX control using the placeholder.

The simplest option is to dock the control to the dialog’s client area:

```
BOOL CWGuestXDemoDlg::OnInitDialog()
{
   CDialogEx::OnInitDialog();
```

```
        m_wguest.SetOwner(this);
        m_wguest.CreateAndDock(this, IDC_WGUESTX1);
        return TRUE;
    }
```

What happens internally:

- The placeholder control is located and removed
- An ATL CAxWindow is created at the same location
- The WiseMo Guest ActiveX control is created inside the host
- Event callbacks are automatically wired

No ATL, COM, or ActiveX code is required in the dialog.

3.5 Handle resizing (one-liner)

If CreateAndDock(...) is used, resizing the control is handled with a single call in OnSize:

```
    void CWGuestXDemoDlg::OnSize(UINT nType, int cx, int cy)
    {
        CDialogEx::OnSize(nType, cx, cy);
        m_wguest.UpdateDocking();
    }
```

This keeps the ActiveX control sized to the dialog’s client area.

If you prefer custom layout logic, compute the target rectangle and call:

```
    m_wguest.SetPos(x, y, width, height);
```

3.6 Implement callback methods (optional but typical)

To receive events from the WiseMo Guest ActiveX control, implement the relevant virtual methods from CIWGuestXHelper.

Example:

```
void CWGuestXDemoDlg::OnWGuestXConnectPost ()
{
   SetDlgItemText(IDC_STATIC_STATUS, _T("Connected"));
}

void CWGuestXDemoDlg: OnWGuestXClosePost ()
{
   SetDlgItemText(IDC_STATIC_STATUS, _T("Disconnected"));
}
```

Callbacks are delivered automatically via CIWGuestSink and forwarded through CIWGuestX.

3.7 Calling WiseMo Guest ActiveX methods

CIWGuestX manages the underlying COM interface. Depending on your wrapper's public API, you either:

- Call convenience methods exposed by CIWGuestX, or
- Access the underlying interface via an accessor (for example, GetInterface())

Example pattern:

```
auto spGuest = m_wguest.GetInterface();
if (spGuest) {
   // Call WiseMo Guest ActiveX methods here
   spGuest->SendCtrlAltDel();
}
```

Refer to the **WiseMo Guest Component API Reference** for available methods and properties.

3.8 Releasing the ActiveX control

The WiseMo Guest ActiveX control is hosted at runtime using an ATL window (CAxWindow) managed internally by CIWGuestX. As with all window-hosted ActiveX controls, it is important to **explicitly destroy the host window** when the dialog is closed or the control is no longer needed.

Although MFC will destroy child windows automatically when a dialog is destroyed, relying solely on implicit cleanup can lead to:

- Late or undefined release order of COM objects

- ActiveX callbacks arriving during shutdown
- Access to already-destroyed UI objects
- Resource leaks during repeated create/destroy cycles

For these reasons, the demo and recommended integration explicitly release the ActiveX control at a well-defined point.

Recommended cleanup sequence

When the dialog is closing (for example in OnDestroy() or the dialog destructor), the application should:

1. Disconnect from the WiseMo Host (if connected)
2. Unsubscribe from ActiveX events
3. Destroy the ActiveX host window
4. Allow normal MFC destruction to continue

All of these steps are encapsulated by CIWGuestX.

Example: releasing the control in OnDestroy

```
BOOL CWGuestXDemoDlg::DestroyWindow()
{
        m_wguest.DestroyWindow();

        return super::DestroyWindow();
}
```

Notes and best practices:

- DestroyWindow() is safe to call even if the control was never created.
- Do not attempt to reuse a CIWGuestX instance after calling DestroyWindow().
- If the control is recreated later, call CreateInPlaceholder() or CreateAndDock() again.
- Avoid destroying the ActiveX from inside an ActiveX callback; instead, post a message or defer cleanup.

3.9 Summary of minimal integration steps

To integrate the WiseMo Guest ActiveX control:

1. Add a placeholder control to the dialog resource
2. Add a CIWGuestX member to the dialog class
3. Call CreateAndDock(...) (or CreateInPlaceholder(...)) in OnInitDialog
4. Call UpdateDocking() from OnSize
5. Implement callbacks in CIWGuestXHelper as needed
6. Destroy the ActiveX window

That is all that is required for a working integration.

4.0 Sideloading the WiseMo Guest ActiveX without registration

In some scenarios it is desirable to **sideload** the WiseMo Guest ActiveX control rather than relying on a system-wide registration. This can be useful for:

- Running the demo application without installing the full WiseMo package
- Testing different versions of the ActiveX side-by-side
- Distributing a self-contained application
- Simpler installation/uninstallation

This section describes how to sideload the WiseMo Guest ActiveX control for the demo application (also called *manifest-based COM activation*). The manifest is **embedded in the application executable** as a resource, so deployment is a single EXE (and any dependencies) plus the WiseMo ActiveX control.

4.1 Deployment layout

Place the OCX next to the executable:

- WGuestXDemo.exe
- WGuestX.ocx

No .manifest file is required on disk, because the manifest is embedded.

4.2 Create the manifest file (source file)

Create a file in your project, for example:

- WGuestXDemo.manifest

Recommended minimal manifest (adjust only if you have a reason to add more):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1"
xmlns:asmv3="urn:schemas-microsoft-com:asm.v3">
  <description> WiseMo WGuestX Demo Application </description>

  <assemblyIdentity
        type="win32"
        name="WGuestXDemo.WiseMo.com"
        version="10.10.0.0"
        processorArchitecture="x86"
  />

  <file name="WGuestX.ocx">
    <comClass
       clsid="{CFE5C49D-202E-4029-8F5A-6A9BE6E51570}"
       threadingModel = "Apartment" />
    <typelib tlbid="{B883CB37-D5BB-4AA3-A012-E9BF7EF945A9}"
         version="1.0" helpdir=""/>
  </file>

  <comInterfaceExternalProxyStub
     name="_IWGuestX20"
     iid="{18FA91C1-F883-47D7-99BF-2B2C61B9F6F3}"
     proxyStubClsid32="{00020424-0000-0000-C000-000000000046}"
     baseInterface="{00000000-0000-0000-C000-000000000046}"
     tlbid = "{B883CB37-D5BB-4AA3-A012-E9BF7EF945A9}"
  />

</assembly>
```

4.3 Embed the manifest into the EXE as RT_MANIFEST (resource ID 1)

Add `WGuestXDemo.manifest` to your Visual Studio project (as an existing item).

In your application's .rc file (or a dedicated .rc2 file), add:

```
IDR_RT_MANIFEST RT_MANIFEST "WGuestXDemo.manifest"
```

In the resource header file (e.g. resource.h), add:

```
#define IDR_RT_MANIFEST 1
```

It’s a critical requirement that the RT_MANIFEST resource ID is **1**.

4.4 Ensure the linker does not override your manifest

In **Project Properties → Linker → Manifest File**, use one of these approaches:

Recommended for demos (simple and deterministic):

- **Generate Manifest** = No

This ensures the embedded RT_MANIFEST you provide is the only manifest.

If you keep linker manifest generation enabled, ensure you are not accidentally embedding a different manifest as resource ID 1.

4.5 Build and run

1. Build the demo as **Win32 (x86)**.
2. Copy WGuestX.ocx next to WGuestXDemo.exe.
3. Run WGuestXDemo.exe.

If successful:

- The ActiveX is activated from the embedded manifest.
- No COM registration is required.
- No registry entries are created.

When you deploy your application, simply install WGuestX.ocx to the same folder as your executable using it.