MFC アプリケーションへの WiseMo Guest ActiveX の統合（Visual Studio 2022 版）

0.0 目次一覧

表の項目をクリックすると該当箇所を直接表示できます。

1.0 はじめに

1.1 概要

本ガイドでは、**推奨されるランタイムのみのホスティング方式**（Visual Studio のデザイン時 ActiveX サポート[1]に依存しない方法）について解説します。このソリューションは以下の原則に基づいています。

- ダイアログリソース内に軽量なプレースホルダーコントロールを使用
- ATL ActiveX ホスティング技術を用いたランタイム時の ActiveX コントロールの動的生成
- すべての COM、ATL、およびイベント処理ロジックを再利用可能なヘルパークラスにカプセル化

このアプローチの利点：

- Visual Studio 2022 以降のバージョンで確実に動作します。
- 設計時における互換性問題や制限を回避できます。
- WiseMo Guest ActiveX コントロールの完全な機能を維持できます。
- アプリケーションコードを簡潔で堅牢、かつメンテナンス性の高い状態に保てます。

1.2 前提条件

本ガイドでは、**Microsoft Visual Studio 2022** を使用して **WiseMo Guest ActiveX** を **MFC ベースのアプリケーション**に組み込むための、一つの対応手法について説明します。

読者は以下の前提条件を満たしている必要があります。

- WiseMo 製品全般、および WiseMo リモートデスクトップのゲスト/ホストコンポーネント、および関連する専門用語についての基本的な知識があること

---

[1] Visual Studio 2022 以降、Visual Studio IDE（devenv.exe）は **64 ビットアプリケーション**になりました。そのため、Visual Studio のデザイナーおよびリソースエディターは、**デザイン時に 32 ビットの ActiveX コントロール**を読み込んだりホストしたりできなくなりました。この変更は、ダイアログのビジュアル編集やデザイナーを使用した ActiveX の挿入など、いくつかの一般的な開発ワークフローに影響します。

この制限は**デザイン時に限られる点**に注意してください。WiseMo Guest ActiveX コントロールは、Visual Studio 2022 でビルドされたアプリケーションを含め、**32 ビット（Win32/x86）アプリケーション**でホストされた場合、**実行時**には引き続き正常に動作します。

- C++および MFC アプリケーション開発に関する実務レベルの知識（ダイアログベースアプリケーションの開発および基本的なメッセージ処理を含む）
- COM および ActiveX の基本概念についての理解（インターフェース、登録処理、インプロセスコントロールなど。ATL に関する詳細な知識は不要）
- Visual Studio 2022 の使用経験（プロジェクト設定、プラットフォーム選択（Win32/x86）、デバッグ作業を含む）
- 開発マシンおよびテストマシンに WiseMo Guest ActiveX コントロール（WGuestX.ocx）が登録済みでインストールされていること
- 32 ビットビルド構成（Win32/x86）であること。WiseMo Guest ActiveX コントロールは 32 ビットのインプロセスコンポーネントであるため
- 開発マシンで ActiveX の登録とデバッグを行う場合に必要な管理者権限

このガイドでは、ATL ActiveX ホスティングや COM 接続ポイント実装に関する事前知識は必要ありません。これらの詳細な処理はすべて、提供されているヘルパークラスによって完全にカプセル化されているためです。

2.0 統合ガイド

本セクションでは、提供されている C++ヘルパークラスを使用して、WiseMo Guest ActiveX コントロールを MFC ベースのアプリケーションに統合する方法について説明します。

2.1 クラス概要

WiseMo では、MFC アプリケーションに WiseMo Guest ActiveX コントロールを簡単に組み込み、使用するための 2 つの C++ラッパークラス、**CIWGuestX** と **CIWGuestSink** を用意しています。そのため、**お客様側で実装する必要のある機能は一切ありません**。

- ATL ウィンドウホスティング（CAxWindow）
- COM インターフェースの取得（QueryInterface）
- 接続ポイントイベントのサブスクリプション（Advise / Unadvise）
- イベント転送とコールバック用グルーコード

CIWGuestSink は、機能の範囲を明確に分離するため、意図的に CIWGuestX から分離されています。

- **CIWGuestX**
  ホスティング、ライフサイクル管理、およびアプリケーションに公開されるパブリック統合 API
- **CIWGuestSink**
  ActiveX イベントインターフェースの実装とコールバックの転送処理

この分離により：

- イベントシンクの実装が安定し、再利用可能になります。
- お客様が接続ポイントの配管処理を記述・保守する必要がなくなります。
- 異なるプロジェクト間でコールバック動作の一貫性を確保します。

要約すると：**アプリケーションが UI とレイアウトを管理し、CIWGuestX が ActiveX コントロールと CIWGuestSink イベント処理を管理します。**

通常の場合、CIWGuestSink に直接アクセスする必要はありません。これは、現在 CIWGuestX で公開されていない使用頻度の低い ActiveX イベントをイベント転送に追加する必要がある場合にのみ関係します。

2.2 環境とセットアップ

2.2.1 WiseMo ActiveX コントロールのインストールと登録

WiseMo Guest ActiveX コントロール（WGuestX.ocx）は、開発マシンおよびテストマシンにインストールされ、登録されている必要があります。これは通常、WiseMo インストーラーによって自動的に処理されます。

WiseMo Guest ActiveX コントロールが未インストールの場合は、以下のサイトから試用版をリクエストできます：

https://its.tos.co.jp/products/wisemo/trial/

WiseMo Guest ActiveX インストーラーの機能：

- ActiveX コントロールのインストールと登録を行います。
- ソースコード付きのデモアプリケーションをインストールします。
- ActiveX のドキュメントをインストールします。

インストール完了後、Windows のスタートメニューから「**WiseMo Guest Component API Reference**」を開いてください。
ドキュメント内で、**IWGuestCore20**（ActiveX がインストールされている場合のみリンクが有効になります）をクリックすると、利用可能な関数とプロパティを確認できます。

2.2.2 アプリを Win32（x86）としてビルド

WiseMo OCX は 32 ビット版であるため、ホストプロセスも 32 ビットである必要があります。

2.2.3 サンプルプログラムについて

サンプルプログラムは、WiseMo Guest ActiveX コントロールのインストール時に、インストールフォルダーに自動で展開されます。

2.2.4 完成したアプリの配布方法

アプリが完成したら、WiseMo ActiveX をアプリケーションにインストールして登録する必要があります。ActiveX の登録に代わる方法として、サイドロードしてアプリケーションから直接参照することも可能です（登録不要）。詳細は 4.0「登録なしで WiseMo Guest ActiveX をサイドロードする方法」のセクションを参照してください。

2.3 統合アーキテクチャの概要

WiseMo Guest ActiveX の統合は、**アプリケーションレベルのコード**と**再利用可能な統合コード**とに意図的に分割されています。この分離により、お客様のアプリケーションの複雑さを最小限に抑えるとともに、COM および ATL 固有のロジックをすべて分離しています。

役割と機能：

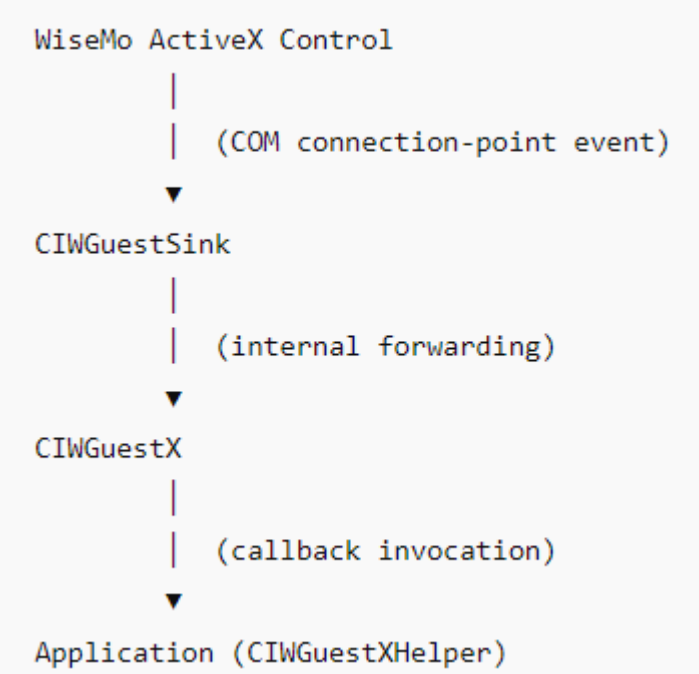
コンポーネント	機能
アプリケーション（ダイアログ/ビュー）	UI、レイアウト、ユーザー操作
CIWGuestX	ActiveX のホスティング、ライフサイクル、パブリック統合 API
CIWGuestSink	COM イベントシンクおよびイベント転送
WiseMo ActiveX	リモートデスクトップ機能

アプリケーションは CIWGuestX とのみ相互作用します。
すべての ActiveX のホスティング、イベントサブスクリプション、および COM ライフタイム管理は内部で処理されます。

2.4 イベントフローとコールバック処理

WiseMo Guest ActiveX コントロールは、標準的な COM 接続ポイントを使用してイベントを公開します。これらのイベントは、CIWGuestX および CIWGuestSink によって実装された制御された転送チェーンを通じてアプリケーションに配信されます。

イベントフローの順序



2.5 アプリケーションが実装する必要のない機能（実装不要）

CIWGuestX を使用する場合、アプリケーションは以下の機能を実装したり理解したりする必要は**ありません**。

- ATL ウィンドウホスティング（CAxWindow）
- ActiveX コントロールの作成（CreateControl）
- COM インターフェースの発見処理（QueryInterface）
- 接続ポイントのサブスクリプション登録／解除（Advise / Unadvise）
- イベントシンククラスまたは IDispatch 実装クラス
- ActiveX のライフタイムとクリーンアップ
- Visual Studio における設計時の ActiveX 埋め込み

上記のすべての懸念事項は、CIWGuestX および CIWGuestSink によって完全にカプセル化されています。

2.6 必要な初期化処理（アプリケーション起動時の一度きりの設定）

本統合では **ATL ActiveX ホスティング**を使用しているため、アプリケーション起動時に少量の一度きりの初期化処理を実行する必要があります。

2.6.1 OLE/COM の初期化処理

ActiveX コントロールを作成する前に、アプリケーション側で OLE/COM の初期化を行う必要があります：

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

ほとんどの MFC アプリケーションテンプレートでは、すでにこの呼び出しが組み込まれています。

2.6.2 ATL ActiveX ホスティングの初期化処理

アプリケーション起動時に一度だけ、ATL ActiveX ホスティングの初期化を行う必要があります。

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

AtlAxWinInit();
```

これにより、CAxWindow で必要とされる ATL ウィンドウクラスが登録されます。

2.6.3 ATL モジュールの定義

MFC アプリケーション内で ATL を使用する場合、最小限の ATL モジュールを提供する必要があります。これは、プロジェクトに AtlModule.cpp ファイルを含めることで実現します。

このファイルは、ATL の内部ランタイムインフラストラクチャが正しく初期化されることを保証し、ActiveX コントロールの作成やホスティング時に発生するランタイムエラーを防止します。

2.6.4 まとめ

CIWGuestX および CIWGuestSink を使用することで、アプリケーションは WiseMo Guest ActiveX コントロールを現代的で堅牢な方法で統合できます。具体的には以下の利点があります。

- Visual Studio 2022 以降のバージョンと互換性があります。
- 設計時における ActiveX の制限を回避できます。
- アプリケーションコードを簡潔に保ち、UI ロジックに集中できるようにします。
- COM、ATL、ActiveX に関するすべての複雑な処理をカプセル化します。
- 将来のメンテナンス作業のための安定した基盤を提供します。

3.0 最小限の統合事例

このセクションでは、WiseMo Guest ActiveX コントロールを MFC ダイアログベースアプリケーションに**統合する最小限のエンドツーエンド**実装例を紹介しています。**CIWGuestX** を使用した例を通じて、推奨される**ランタイムのみのホスティング方式**の利点を説明し、必要なアプリケーションコード量の少なさを強調しています。

このサンプルの目的は、**明確で実用的な統合例**を提示することです。同時に、ATL、COM、ActiveX に関連するすべての複雑な処理は、提供されているヘルパークラス内に完全にカプセル化されています。この例は意図的に簡潔にまとめられていますが、可能な限り最小のプロジェクトというわけではありません。追加のガイダンスやベストプラクティスに関するヒントも含まれているためです。

付属のサンプルプロジェクトでは、このセクションで説明されているすべての手順を実装しています。したがって、これらの手順は**ドキュメント**としての役割を果たすとともに、実際の統合作業における**実用的な参考資料**としても機能します。WiseMo Guest ActiveX コントロールをご自身のアプリケーションに統合する際にご活用ください。

3.1 Visual Studio プロジェクトの設定

このセクションでは、Visual Studio 2022 で既存の MFC アプリケーションプロジェクトに WiseMo Guest ActiveX 統合パッケージ（WGuestX.zip）を追加する方法について説明します。

統合パッケージの展開

1. WGuestX.zip ファイルを解凍してください。
2. アーカイブは以下のファイルを含む「**WGuestX**」という名前のフォルダーに展開されます。
    - AtlModule.cpp
    - IWGuestX.cpp
    - IWGuestX.h
    - IWGuestSink.cpp

- IWGuestSink.h
- WGuestX.ocx
- wguestx.tlh
- wguestx.tli

3. 推奨設定：「WGuestX」フォルダーをソリューションディレクトリ以下に配置してください。
   例：
   $(SolutionDir)\WGuestX\

Visual Studio プロジェクトにこれらのファイルを追加

ソリューションエクスプローラー：

1. 「**ソースファイル**」を右クリック → **追加** → **新しいフィルター** → 名前を「WGuestX」として設定します。
2. 「WGuestX」フォルダーの場所を参照します。
3. すべての .cpp および .h ファイルをフィルターフォルダー内にドラッグします。
4. お好みで、 *.h ファイルを別のフォルダーに配置することも可能です。

これにより、これらのファイルがプロジェクトの一部としてコンパイルされ、インクルード/インポート可能になります。

「WGuestX」フォルダーのインクルードディレクトリを追加

ソースファイルが長い相対パスを使用せずに WiseMo ラッパーヘッダーをインクルードできるようにするため、プロジェクトのインクルードディレクトリに「WGuestX」フォルダーを追加してください。

1. プロジェクトを右クリック → 「**プロパティ**」を選択します。
2. 「**C/C++**」 → 「**全般**」の設定に移動します。
3. 「**追加のインクルードディレクトリ**」セクションに以下を追加します。

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

もし「WGuestX」フォルダーをプロジェクトディレクトリ直下に配置した場合は、代わりに以下を使用してください。

```
$(ProjectDir)WGuestX
```

WGuestX.ocx を配置するビルド後のイベントを追加

デモアプリケーションは、実行時に WGuestX.ocx が実行ファイルと同じフォルダーに存在することを前提としています。これを自動化するために、OCX を出力フォルダーにコピーするビルド後のイベントを追加します。

1. プロジェクトを右クリック→「プロパティ」を選択します。
2. 次に「ビルドイベント」→「ビルド後のイベント」に移動します。
3. 以下のコマンドを追加します。

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

3.2 ダイアログリソースにプレースホルダーコントロールを追加

ダイアログリソースファイル（.rc ファイル）内で、WiseMo Guest ActiveX コントロールを配置する位置に、シンプルなプレースホルダーコントロールを追加します。

重要な注意事項：

- プレースホルダーは ActiveX コントロールではありません。
- 任意のシンプルなコントロールタイプを使用できます（例：Static コントロール）。
- 実際のコントロール ID を使用してください（IDC_STATIC ではありません）。

例：

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

このプレースホルダーは初期位置とサイズを定義します。実行時に ActiveX ホストウィンドウによって置き換えられます。

3.3 ダイアログクラスに CIWGuestX を追加

ラッパーヘッダーをインクルードし、ダイアログクラスに CIWGuestX メンバーを追加します。

このダイアログはまた、コールバックを受信するために CIWGuestXHelper を実装しています。

```
#include "IWGuestX.h"
```

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

protected:
   CIWGuestX m_wguest;
};
```

3.4 OnInitDialog での ActiveX コントロールの作成

OnInitDialog では、プレースホルダーを使用して ActiveX コントロールを初期化し、作成します。

最も簡単な方法は、コントロールをダイアログのクライアント領域にドッキングさせることです。

```
BOOL CWGuestXDemoDlg::OnInitDialog()
{
   CDialogEx::OnInitDialog();
   m_wguest.SetOwner(this);
   m_wguest.CreateAndDock(this, IDC_WGUESTX1);
   return TRUE;
}
```

内部では以下の処理が行われます：

- プレースホルダーコントロールが位置を特定され、削除されます。
- 同じ位置に ATL の CAxWindow が作成されます。
- そのホスト内に WiseMo Guest ActiveX コントロールが作成されます。
- イベントコールバックが自動的に接続されます。

ダイアログ内で ATL、COM、または ActiveX 関連のコードを記述する必要はありません。

3.5 リサイズ処理の処理方法（一行コード）

CreateAndDock(...)を使用する場合、コントロールのリサイズは OnSize イベント内で単一の呼び出しで処理されます。

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

これにより、ActiveX コントロールがダイアログのクライアント領域に合わせて適切なサイズに保たれます。

独自のレイアウトロジックを実装したい場合は、ターゲットとなる矩形を計算した上で以下のメソッドを呼び出してください。

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

3.6 コールバックメソッドの実装（任意だが一般的な手法）

WiseMo Guest ActiveX コントロールからのイベントを受信するには、CIWGuestXHelper の該当する仮想メソッドを実装してください。

使用例：

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

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

コールバック処理は CIWGuestSink を介して自動的に配信され、 CIWGuestX を通じて転送されます。

3.7 WiseMo Guest ActiveX メソッドの呼び出し方法

CIWGuestX が基盤となる COM インターフェースを管理します。使用するラッパーの公開 API に応じて、以下のいずれかの方法を選択します。

- CIWGuestX で公開されている便利メソッドを呼び出す、あるいは、
- アクセッサー経由で基盤となるインターフェースに直接アクセスします（例： GetInterface()）

使用例パターン：

```
auto spGuest = m_wguest.GetInterface();
if (spGuest) {
   // ここで WiseMo Guest ActiveX メソッドを呼び出します
   spGuest->SendCtrlAltDel();
}
```

利用可能なメソッドとプロパティについては、**WiseMo Guest Component API Reference** を参照してください。

3.8 ActiveX コントロールの解放

WiseMo Guest ActiveX コントロールは、実行時に ATL ウィンドウ（CAxWindow）としてホストされ、CIWGuestX によって内部的に管理されます。ウィンドウにホストされるすべての ActiveX コントロールと同様、ダイアログが**明示的にホストウィンドウを破棄**する必要があります（ダイアログが閉じられた場合、またはコントロールが不要になった場合）。

MFC ではダイアログが破棄されると自動的に子ウィンドウも解放されますが、暗黙的なクリーンアップに依存するだけでは以下のような問題が生じる可能性があります。

- COM オブジェクトのリリース順序の遅延、または不確定
- シャットダウン中に受信する ActiveX コールバック
- 既に破棄された UI オブジェクトへのアクセスが発生
- 作成/破棄サイクルを繰り返すことにより、リソースリークが発生

これらの理由から、デモアプリケーションおよび推奨される統合方法では、明確に定義された時点で ActiveX コントロールを確実に解放します。

推奨されるクリーンアップ手順

ダイアログが閉じられる際（例えば OnDestroy() メソッドまたはダイアログのデストラクタ内で）には、アプリケーションは以下の処理を行う必要があります。

1. WiseMo ホストからの接続を切断します（接続されている場合）。
2. ActiveX イベントのサブスクライブを解除します。
3. ActiveX ホストウィンドウを破棄します。
4. 通常の MFC オブジェクトの破棄処理を継続させます。

これらの処理はすべて CIWGuestX クラスによってカプセル化されています。

例： OnDestroy メソッド内でコントロールを解放する場合

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

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

注意事項とベストプラクティス：

- DestroyWindow() メソッドは、コントロールが一度も作成されていない場合でも安全に呼び出すことができます。
- CIWGuestX インスタンスを DestroyWindow() メソッド呼び出し後に再利用しようとしないでください。
- コントロールを後で再作成する場合は、再度 CreateInPlaceholder() メソッドまたは CreateAndDock() メソッドを呼び出してください。
- ActiveX コールバック内で ActiveX オブジェクトを破棄することは避けてください。代わりにメッセージを送るか、クリーンアップ処理を遅延実行してください。

3.9 最小限の統合手順のまとめ

WiseMo Guest ActiveX コントロールを組み込むには：

1. ダイアログリソースにプレースホルダーコントロールを追加します。
2. ダイアログクラスに CIWGuestX メンバーを追加します。
3. CreateAndDock(...)または CreateInPlaceholder(...)を OnInitDialog メソッド内で呼び出します。
4. UpdateDocking()を OnSize メソッドから呼び出します。
5. 必要に応じて CIWGuestXHelper でコールバックを実装します。
6. ActiveX ウィンドウを破棄します。

以上が、正常に動作する統合を実現するために必要なすべての手順です。

4.0 登録なしで WiseMo Guest ActiveX をサイドロードする方法

特定のシナリオでは、システム全体への登録に依存する代わりに WiseMo Guest ActiveX コントロールを個別に**サイドロード**することが望まれる場合があります。これは以下のような場合に有用です。

- 完全な WiseMo パッケージをインストールせずにデモアプリケーションを実行する場合
- ActiveX の異なるバージョンを並行してテストする場合
- 自己完結型アプリケーションを配布する場合
- インストール/アンインストールプロセスを簡素化する場合

本セクションでは、デモアプリケーション向けに WiseMo Guest ActiveX コントロールをサイドロードする方法について説明します（マニフェストベースの COM アクティベーションとも呼ばれます）。マニフェストは**アプリケーション実行ファイルに**リソースとして**埋め込まれる**ため、展開は単一の EXE ファイル（および必要な依存ファイル）と WiseMo ActiveX コントロールのみで済みます。

4.1 展開構成

実行ファイルの隣に OCX ファイルを配置します：

- WGuestXDemo.exe
- WGuestX.ocx

ディスク上に.manifest ファイルを用意する必要はありません。マニフェストが埋め込まれているためです。

4.2 マニフェストファイル（ソースファイル）を作成

プロジェクト内にファイルを作成します。

例：WGuestXDemo.manifest

推奨される最小限のマニフェスト内容（追加する必要がある場合のみ調整してください）：

```
<?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 マニフェストを EXE ファイル内に RT_MANIFEST リソース ID（リソース ID 1）として埋め込み

Visual Studio プロジェクトに WGuestXDemo.manifest を追加します（既存の項目として）。

アプリケーションの.rc ファイル（または専用の.rc2 ファイル）に以下を追加してください。

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

リソースヘッダーファイル（例：resource.h）に以下を追加します。

```
#define IDR_RT_MANIFEST 1
```

RT_MANIFEST リソース ID を **1** に設定することは必須要件です。

4.4 リンカーによるマニフェストの上書き防止

プロジェクトの「**プロパティ**」→「**リンカー**」→「**マニフェストファイル**」にて、以下のいずれかの方法を使用してください：

デモ用途（シンプルで決定的な動作が求められる場合）に推奨：

- **マニフェストの生成** = いいえ

これにより、提供された埋め込み RT_MANIFEST のみがマニフェストとして扱われることが保証されます。

リンカーによるマニフェスト生成を有効にしたままにする場合は、誤って別のマニフェストをリソース ID 1 として埋め込まないように注意してください。

4.5 ビルドと実行

1. デモを **Win32 (x86)** としてビルドします。
2. WGuestX.ocx を WGuestXDemo.exe の隣にコピーします。
3. WGuestXDemo.exe を実行します。

正常に動作した場合：

- 組み込みマニフェストから ActiveX が有効化されます。
- COM 登録は不要です。
- レジストリへのエントリは作成されません。

アプリケーションを展開する際には、実行ファイルと同じフォルダーに WGuestX.ocx をインストールしてください。