



---

## **PictureTel LiveDTK V2.1 Sample Application**

### **Visual C++ 5.0**

**May 1998**

---

[Enhancements to the LiveDTK 2.0 Sample Application](#)

[Overview of Enhancements](#)

[The NetMeeting Wrapper OCX Integration](#)

[Additional GUI Modifications](#)

[Using the NetMeeting Wrapper OCX](#)

---

## **Enhancements to the LiveDTK 2.0 Sample Application**

### **Overview of Enhancements**

There are two primary goals for this release of the sample application. The first goal replaces the LiveShare Plus OLE Controls (LSP OCX) with the equivalent functionality in Microsoft NetMeeting 2.1, made available through the DTK's NetMeeting Wrapper OLE Control (NMW OCX). This particular OCX is not to be confused with the "NetMeeting OCX". The NMW OCX is designed to present the NetMeeting API in way that current users of the LiveDTK would feel comfortable with.

The second goal enhances the user interface of the sample application to both improve appearance and provide more sample code to illustrate the usage of the LiveDTK OCXs in VC++.

---

## The NetMeeting Wrapper OCX Integration

The following steps integrate the new NMW OCX:

1. The LSP OCXs were removed from the application. These appear as pushbuttons that provide access to the whiteboard, message, application sharing and unsharing, and file transfer functionality of LSP.
  2. A toolbar was created with bitmap buttons that look identical to the LSP OCXs but invoke the methods of the NMW OCX.
  3. The Call Interface OCX is still used to make a call but no longer connects to an LSP conference. In the previous version, this was a pushbutton that you could click to make a call and start an LSP conference. In the current sample, the Call Interface OCX was left in the form but made invisible so that LSP could not be invoked. It was replaced with a toolbar button hooked up to a new method that collects the necessary call information via a new dialog box, then invokes the Connect() method of Call Interface OCX. To make the change as transparent as possible, the new dialog box was given a dialpad that has the same appearance as the Call Interface dialog box. The state is stored in the Registry, and the last saved call information displays when the box is invoked.
  4. The NMW OCX was made as an invisible control and its methods are hooked up to the new toolbar buttons, and handlers for its relevant events were put in. The details of how to do this are discussed later in this document; the following is a summary. First, the control is set up by having its Init() method called in the View classes OnInitialUpdate(). Second, the NetMeeting conference is started using the NMW OCX Connect() method after the CallInterface Connect() succeeds (signaled by the arrival of that control's CallConnected event). It can take several seconds for NetMeeting to start; a new event handler was put in to catch the ConferenceCreated event that NMW OCX emits after everything is ready. The NM conference must be in progress for any of the subsequent calls to the NMW OCX to succeed.
  5. The NMW OCX replacements for the LSP controls are:
    - o In the case of the whiteboard and message, each involves just a single call to the OCX (LaunchWhiteBoard() and LaunchMessage(), respectively).
    - o The file transfer is not a straightforward replacement; the corresponding call in the NMW OCX does not pop up an elaborate dialog box to get the file information (as the old LSP control did); it just takes the pathname of the file to transfer. A new class called CFileSend (derived from CFileDialog) was created to provide the familiar 'File open' dialog box except with a title of 'Send files' and a 'Send' button rather than the standard 'Ok' button. The CFileSend constructor was passed the OFN\_ALLOWMULTISELECT flag to enable the user to select multiple files. The NMW OCX file transfer method only allows one file to be sent at a time, additional code is needed to make multiple method calls (one for each file) until all files selected in the dialog box have been sent.
    - o The share and unshare integration required additional coding. First, a dialog box is created that uses the NMW OCX call, GetShareableApps(), to present a list of applications. Make a selection and press OK, the sample application calls ShareApp() with the window handle of the selected item. The sample stores the application handle and name in a CMap data member. A different dialog box is used for the Unshare side of things; when the Unshare button is pressed, this dialog box displays the items that are tucked away in the CMap object. After an application is selected to unshare and the OK button is pressed, the sample calls the NMW OCX UnShare() method, and removes the unshared item from the CMap.
  6. The NMW OCX automatically disconnects from the NetMeeting conference when the call is disconnected.
-

## Additional GUI Modifications

The following changes were made to the user interface to make the application look similar to the LSP controls:

- The Video OCX and Audio OCX, which previously provided picture and sound, are replaced with the A/V Interface OCX. This requires additional coding to use but provides greater flexibility for presenting and managing A/V capabilities. The purpose for this change was to provide examples of how to incorporate this control. A new dialog box called the "Audio/Video Control Panel" was created for controlling the A/V characteristics. The methods used are the following: `Avmute()`, `DestroyLocalVideoWindows()`, `EnableDTKMode()`, `EnableLocalVideo()`, `EnableRemoteVideo()`, `GetCallHandle()`, `Mute()`, `SetAudioInputMode()`, `SetAutomaticGainControl()`, `SetCallHandle()`, `SetupDTKMode()`, `SetBrightness()`, `SetContrast()`, `SetLipSync()`, `SetNoiseReduction()`, `SetSaturation()`, `SetVolume()`, and the Get methods corresponding to these sets.
- As part of the A/V interface changes, the main view now shows the local and remote video windows, and has some controls to manage these: volume, brightness, and the ability to make the windows movable and resizable. There is a pushbutton for invoking the new A/V Control Panel.
- The state of the UI controls is saved in the Registry. When an application is run, the controls are initialized with these values.
- The bitmap buttons at the top of the view have been replaced with a proper toolbar. This enables the display of tooltips, the fly-by help, and improves the overall appearance.
- The existing Address Book dialog box, that showcases the three Address Book OCXs, had some cosmetic changes, a number of bug fixes. Some memory leaks still occur if there is a fair amount of open, closing, creating and modifying of address lists.

---

## Using the NetMeeting Wrapper OCX

This section discusses how to use the NM Wrapper OCX in more detail, incorporating code fragments from the Visual C++ 5.0-based sample application. It is intended to be a supplement to the existing official NM Wrapper OCX document.

### Step 1: Create an instance of the NM wrapper OCX

After registering `Manager.dll` with Windows NT, import the NMW OCX into the project, then either place it on a form and set it to invisible, or call its `Create()` method. The default class name is `CManage`. Create a data member of this type in one of your classes, such as your derived `View` class if your application has a document/view architecture. In the sample, this data member is named `m_ctrlManage`.

### Step 2: Initialize the NM wrapper OCX

If your application is document/view, call the initialization routine in the `OnInitialUpdate()` method of your view class. Your code for this should look very similar to the following:

```
BOOL bSuccess = m_ctrlManage.Init();
if ( FALSE == bSuccess )
{
    // Do something such as inform the user of this failure, and that they will be
    // unable to connect to NM from this application because of this.
}
```

### Step 3: Start a Live200 call with the Call Interface OCXs Connect() method

The Call Interface OCX is in your application, but not made visible, or the user launches LSP, not NM, connections. Instead, have a second control in your UI that is visible to the user, such as a plain old pushbutton or menu item, with a handler that lets the user set up and make a call. Provide something that lets the user enter their call information easily, for example, something like the CMakeCallDlg dialog box in the sample application. After the user fills in the box and presses OK, the call is launched. Your code for this should look very similar to the following (with m\_ctrlCallInterface being the Call Interface OCX object):

```
// This dialog box simply collects call information from the user:
CMakeCallDlg makeCallDlg(&m_ctrlCallInterface);
if ( makeCallDlg.DoModal() == IDOK )
{
    // Establish a Live200 call connection:
    long lStatus = -1;
    long lConnection = 0;
    lStatus = m_ctrlCallInterface.Connect(
        makeCallDlg.GetNetType(),
        makeCallDlg.GetCallType(),
        makeCallDlg.GetChannelRate(),
        *(makeCallDlg.GetAddress()),
        &lConnection
    );
    if ( 0 != lStatus )
        // Handle the fact that the connection attempt failed.
}
}
```

**Note:** If your application allows data-only calls such as the C++ sample app, then one approach to take would be the following:

```
CMakeCallDlg makeCallDlg(&m_ctrlCallInterface);
if ( makeCallDlg.DoModal() == IDOK )
{
    // Establish a Live200 call connection:
    long lStatus = -1;
    long lConnection = 0;
    // Handle data only call differently
    if ( 3 == makeCallDlg.GetCallType() ) // Data only call
    {
        // The CALLSTATE typedef is defined in Manager.h

        CALLSTATE csState = CALL_NOT_STARTED;
        csState =

(CALLSTATE)m_ctrlManage.NMConnect(1/*CALL_T120*/,1/*ADDR_IP*/,*(makeCallDlg.GetIPAddress
()));

        switch( csState )
        {
        case CALL_NOT_STARTED:
            AfxMessageBox( IDS_NM_CALL_NOT_STARTED );
            break;
        case MANAGER_NO_INIT_CALL_NOT_STARTED:
            AfxMessageBox( IDS_NM_NO_INIT );
            break;
        case CREATECALL_FAILED_CALL_NOT_STARTED:
            AfxMessageBox( IDS_NM_CREATECALL_FAILED );
            break;
        case CALL_STARTED:
        default:
            m_eCallType = DATA_ONLY;
            TRACE( "NetMeeting call started." );
            // Note: the call is started at this point, but we need to
```

```

        // wait for the NM wrapper's ConferenceCreated event to know
        // when the NM conference is fully established and ready to go.
        // See Step 5.
        break;
    }
}
else
{
    lStatus = m_ctrlCallInterface.Connect(
        makeCallDlg.GetNetType(),
        makeCallDlg.GetCallType(),
        makeCallDlg.GetChannelRate(),
        *(makeCallDlg.GetAddress()),
        &lConnection);

    if ( 0 != lStatus )
        AfxMessageBox( IDS_CONNECT_FAILED );
}
}
}

```

---

#### Step 4: Catch the Call Interface OCXs CallConnected event and attempt the connection to NetMeeting in the event handler

When making an Audio, Video and Data call, wait until the Live200 call is established before connecting to a NetMeeting conference. When you receive the CallConnected event, you can begin. In the handler for this event, call the NMW OCX Connect() method to establish the connection. The sample application has the following code to accomplish this:

```

CALLSTATE csState = CALL_NOT_STARTED;
csState = (CALLSTATE)m_ctrlManage.Connect();

switch( csState )
{
case CALL_NOT_STARTED:
    // error handling
    break;
case MANAGER_NO_INIT_CALL_NOT_STARTED:
    // error handling
    break;
case CREATECALL_FAILED_CALL_NOT_STARTED:
    // error handling
    break;
case CALL_STARTED:
default:
    TRACE( "NetMeeting call started." );
    break;
}

```

The call states are defined in a header file, or include the header file for the NM wrapper OCX. It is named Manager.h by default. Either way, add this enum somewhere in the code to make the above fragment both build and interpret return codes correctly:

```

typedef enum CallState
{
    CALL_NOT_STARTED          = 0,
    MANAGER_NO_INIT_CALL_NOT_STARTED = 1,
    CREATECALL_FAILED_CALL_NOT_STARTED = 2,
    CALL_STARTED              = 3
} CALLSTATE;

```

---

## Step 5: Wait for the NM wrapper OCXs ConferenceCreated event before launching any NM applications

It can take several seconds for the NM conference to be fully set up, and until it is, you will not be allowed to use any NetMeeting functionality. You should therefore always check that the NM setup is complete, as signaled by the wrapper sending out the ConferenceCreated event, before attempting to do your data conferencing. The sample application has a boolean member in the View class that acts as a flag to indicate whether the ConferenceCreated event was caught or not. Because of the delay necessary for the conference to get going, there is a method in the sample that tests the flag and gives it 5 seconds to change to TRUE, in case the user tries to do something like launch Message prematurely. Control handlers in the sample that invoke the NM applications always call this test method before attempting to do anything further. The following is the sample code for the test method:

```
BOOL CDTKLive200View::NetMeetingIsReady()
{
    if ( m_bInConference )
        return TRUE;

    // The m_bInConference data member is set to TRUE when the
    // NM wrapper OCX fires the ConferenceCreated event. It can
    // take several seconds before the NM conference is established,
    // and meanwhile this test method can be called while this is happening,
    // so we wait a few seconds here.
    for ( int j = 0; ((FALSE==m_bInConference) && (j<5)); j++ )
        Sleep( 1000 );
    if ( !m_bInConference )
        AfxMessageBox( "NetMeeting conference is not established." );
    return m_bInConference;
}
```

---

## Step 6: Create Windows controls for users to invoke the NetMeeting applications

Create menus, pushbuttons, toolbars or anything that has handlers to test if NetMeeting is ready, and invoke its applications through the NMW OCX. The code below, taken from the sample, illustrates ways to do this.

---

### Whiteboard

```
void CDTKLive200View::OnNMwhiteboard()
{
    if ( FALSE == NetMeetingIsReady() )
    {
        AfxMessageBox( "Cannot launch Whiteboard: NetMeeting conference not
started." );
        return;
    }
    (void)m_ctrlManage.LaunchWhiteBoard();
}
```

---

### Message (also known as Chat)

```
void CDTKLive200View::OnToolsChat()
{
    if ( FALSE == NetMeetingIsReady() )
```

```

    {
        AfxMessageBox( "Cannot launch message: NetMeeting conference not started.");
        return;
    }
    m_ctrlManage.SetAutoShowChat(true); // Enable the internal chat dialog box
    (void)m_ctrlManage.LaunchMessage();
}

```

---

## File Transfer

```

void CDTKLive200View::OnToolsFiletransfer()
{
    if ( FALSE == NetMeetingIsReady() )
    {
        AfxMessageBox( "Cannot transfer file: NetMeeting conference not started.");
        return;
    }
    CFileSend dlg( TRUE, NULL, NULL,
    OFN_ALLOWMULTISELECT|OFN_PATHMUSTEXIST|OFN_NOCHANGEDIR );
    if ( dlg.DoModal() == IDOK )
    {
        // Obtain starting position for 1st file selected and keep
        // sending until the end of the list is reached.
        POSITION pos = dlg.GetStartPosition();

        while ( pos != NULL )
        {
            (void)m_ctrlManage.SendFile( dlg.GetNextPathName(pos) );
        }
    }
}

```

---

## Application Sharing

```

void CDTKLive200View::OnNMshare()
{
    if ( FALSE == NetMeetingIsReady() )
    {
        AfxMessageBox( "Cannot share apps: NetMeeting conference not started.");
        return;
    }

    CExeListDlg dlg;
    if ( IDOK == dlg.DoModal() )
    {
        if ( m_ctrlManage.ShareApp(dlg.GetSelectedHandle() ) )
        {
            // m_mapSharedApps is a CMap for storing info about the shared apps:
            m_mapSharedApps.SetAt( dlg.GetSelectedHandle(),
            *(dlg.GetWindowName() ) );
        }
        else
        {
            AfxMessageBox( IDS_APP_SHR_ERR );
        }
    }
}

```

The sample application's CExeListDlg dialog box, used in the fragment above, presents the user with a listbox of sharable applications. The key method from that class uses the NMW OCXs ShareableApps() method to fill up this listbox:

```

void CExeListDlg::FillListBox ()
{
    CString sWinText;
    long lWnd = -1;
    const int nBufLen = 128;
    CDTKLive200View* pView = (CDTKLive200View*)GetParentFrame()->GetActiveView();
    // First check that NetMeeting conference is established.
    if ( FALSE == pView->NetMeetingIsReady() )
        return;
for ( long i = 0; 0 != lWnd; i++ )
{
    lWnd = pView->m_ctrlManage.GetSharableApps(i); // NM wrapper OCX
    if ( lWnd > 0 )
    {
        LPTSTR szBuf = sWinText.GetBuffer( nBufLen );
        ::GetWindowText( (HWND)lWnd, szBuf, nBufLen );
        sWinText.ReleaseBuffer();
        // Add the application name to the listbox:
        int nIndex = SendDlgItemMessage(
            IDC_EXELIST, LB_ADDSTRING, 0,
            (LPARAM)(LPCTSTR)sWinText );

        // Store its window handle in that same listbox item:
        (void)SendDlgItemMessage(
            IDC_EXELIST, LB_SETITEMDATA,
            (LPARAM)nIndex, (LPARAM)(HWND)lWnd );
    }
}
}
}

```

---

## Application Unsharing

```

void CDTKLive200View::OnNMshare()
{
    if ( FALSE == NetMeetingIsReady() )
    {
        AfxMessageBox( "Cannot share apps: NetMeeting conference not started." );
        return;
    }

    CExeListDlg dlg;
    if ( IDOK == dlg.DoModal() )
    {
        if ( m_ctrlManage.ShareApp(dlg.GetSelectedHandle() ) )
        {
            // m_mapSharedApps is a CMap for storing info about the shared apps:
            m_mapSharedApps.SetAt( dlg.GetSelectedHandle(),
                *(dlg.GetWindowName() ) );
        }
        else
        {
            AfxMessageBox( IDS_APP_SHR_ERR );
        }
    }
}
}

```

The CShrListDlg box used above takes the items in the CMap data member, i.e., the stored handles and names of currently shared applications, and presents them to the user in a listbox:

```
void CShrListDlg::FillListBox ()
{
    CDTKLive200View* pView = (CDTKLive200View*)GetParentFrame()->GetActiveView();
    if ( 0 == pView->m_mapSharedApps.GetCount() )
        return;
    POSITION pos = pView->m_mapSharedApps.GetStartPosition();
    while ( pos != NULL )
    {
        long hWnd;
        CString winName;
        pView->m_mapSharedApps.GetNextAssoc( pos, hWnd, winName );
        int index = m_xShareList.AddString( winName );
        m_xShareList.SetItemData( index, hWnd );
    }
}
```



*Copyright © 1998, PictureTel Corporation. All rights reserved.*