* 1. 

Hands-On Lab

Taskbar with Visual C++ and MFC

* 1. Lab version: 1.0.0
  2. Last updated: 12/22/2010

Contents

[Overview 3](#_Toc280793654)

[Exercise 1: Experiment with the New Windows 7 Taskbar Features 5](#_Toc280793655)

[Task 1—Using Taskbar Overlay Icons 6](#_Toc280793656)

[Task 2—Using Taskbar Progress Bars 8](#_Toc280793657)

[Task 3—Using Taskbar Jump Lists 10](#_Toc280793658)

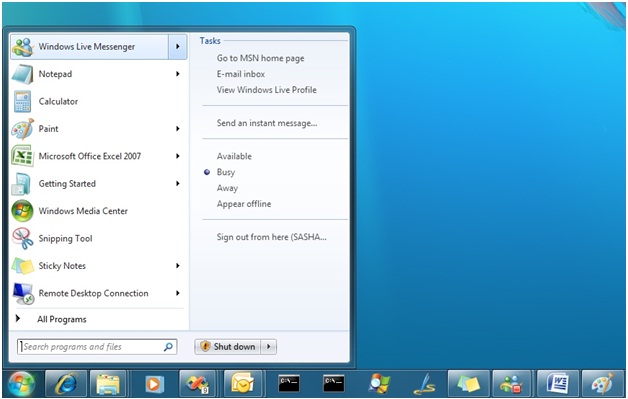
[Exercise 2: Supporting Taskbar Thumbnails 16](#_Toc280793659)

[Task 1—Supporting Taskbar Thumbnails 16](#_Toc280793660)

[Summary 18](#_Toc280793661)

Overview

* 1. The new Windows 7 Taskbar represents the culmination of many years of Windows launch-surface evolution. The new taskbar streamlines many end-user scenarios including:
  + Launching applications
  + Switching between running applications and windows within a single application
  + Managing recent/frequent user destinations
  + Accessing common application tasks
  + Reporting progress and status notifications through the taskbar button
  + Controlling the application without leaving the taskbar thumbnail



* 1. Figure 1
  2. *The new Windows 7 Taskbar and Start Menu*
  3. The new taskbar is a differentiating opportunity that allows applications to shine on the Windows 7 platform. It is the end user’s primary point-of-contact for initiating and managing activities. Thus, the integration of new taskbar features into modern Windows 7 applications is a critically important goal.
  4. The consolidation of features into the new taskbar user interface is designed to provide a cleaner look to Windows, reduce the number of concepts users must learn in order to interact with the system, ease discoverability of key features, tasks and destinations, and make common end-user scenarios more accessible.

# Objectives

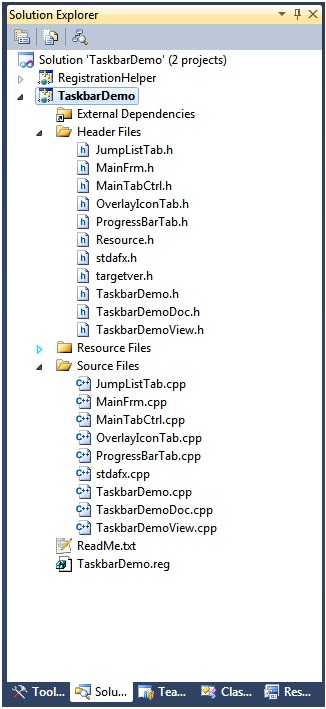
* 1. In this Hands-On Lab, you will learn how to integrate your MFC application with the Windows 7 Taskbar, including how to:
  + Provide visual progress and status indicators using taskbar progress bars and overlay icons
  + Quickly access common tasks and frequent destinations using taskbar jump lists, system categories and custom categories
  + Customize the taskbar thumbnail with live previews of MDI views

# System Requirements

* 1. You must have the following items to complete this lab:
  + Microsoft Visual Studio 2010
  + Windows 7
  + Windows 7 SDK

Exercise 1: Experiment with the New Windows 7 Taskbar Features

* 1. In this exercise, you will experiment with the new Windows 7 taskbar features. You will extend a showcase application that demonstrates the use of the new taskbar functionality to provide a taskbar overlay icon, progress bar, jump list, and thumbnail previews. Most of the application’s user interface is already implemented; you will need to fill in only the missing parts that interact with the Windows 7 taskbar using Visual C++ MFC (Microsoft Foundation Class Library).
  2. To begin this exercise, open the *TaskbarDemo\_Starter\TaskbarDemo.sln* solution (under the Source folder) using Visual Studio 2010.



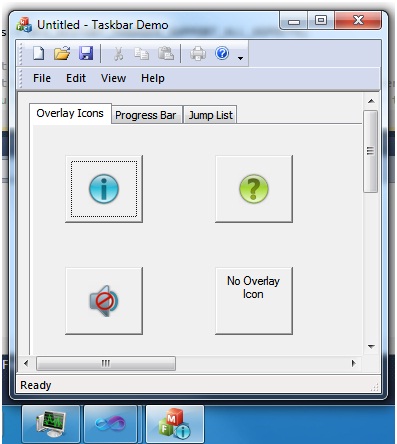
* 1. Figure 2
  2. Taskbar Concepts solution structure in Visual Studio 2010
  3. Spend a minute or two exploring the header, source and resource files. Take a look at the various dialogs and main form. Take a few moments to build and run the application as well.

1. Task 1—Using Taskbar Overlay Icons

In this task, you will toggle an overlay icon on the application’s taskbar button when a user clicks an icon button.

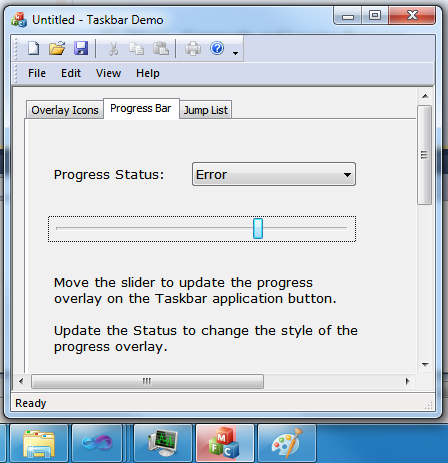
* 1. Note that in MFC, you can set an overlay icon by using CFrameWnd::SetTaskbarOverlayIcon(). To clear the Taskbar overlay icon, pass NULL (0) to SetTaskbarOverlayIcon().
  2. In the TaskbarDemo\SourceFiles sub tree, open OverlayIconTab.cpp and look for the message handler for the Info overlay icon, COverlayIconTab::OnOverlayIconInfo(). We’ll use this message handler to add the code necessary to set the same icon in the button as a Taskbar overlay icon.
  3. Add the following code to COverlayIconTab::OnOverlayIconInfo():
     1. C++
     2. CMainFrame\* mainFrm = dynamic\_cast<CMainFrame\*>(AfxGetApp()->GetMainWnd());
     3. if (mainFrm)

mainFrm->SetTaskbarOverlayIcon(IDI\_ICON\_INFO,L"Info");

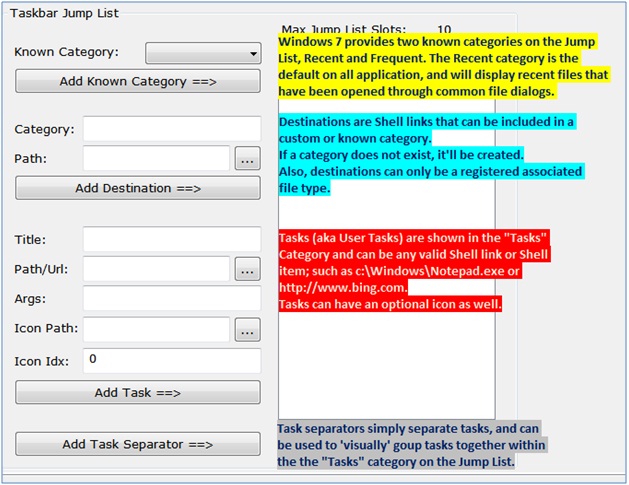
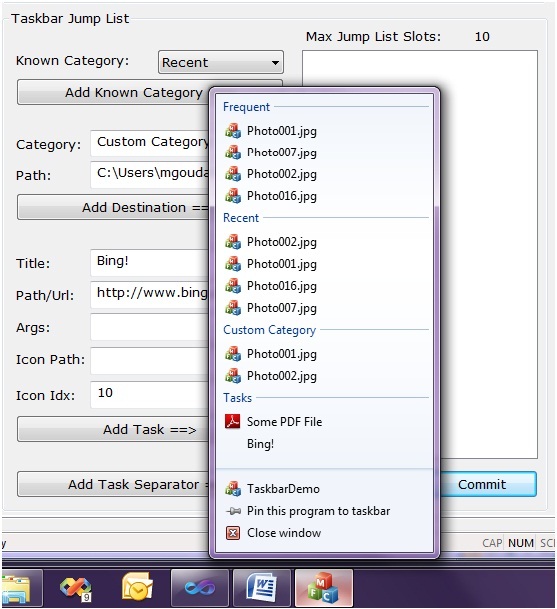
* 1. Compile and run the application.
  2. Press the information button ().
  3. Notice that the display on the taskbar has changed, from the original (image5.png), and now contains an overlay of the information button () superimposed on it (see Figure 3 below).
     1. 
     2. Figure 3
     3. Taskbar overlay icon shown
  4. Continue adding the necessary code for the rest of the icon overlay message handlers:
     1. C++
     2. void COverlayIconTab::OnOverlayIconQuestion()
     3. {
     4. // TODO: Add code to set question overlay icon
     5. CMainFrame\* mainFrm = dynamic\_cast<CMainFrame\*>(AfxGetApp()->GetMainWnd());
     6. if (mainFrm)
     7. mainFrm->SetTaskbarOverlayIcon(IDI\_ICON\_QUESTION, L"Question");
     8. }
     9. void COverlayIconTab::OnOverlayIconNoVolume()
     10. {
     11. // TODO: Add code to set no-volume overlay icon
     12. CMainFrame\* mainFrm = dynamic\_cast<CMainFrame\*>(AfxGetApp()->GetMainWnd());
     13. if (mainFrm)
     14. mainFrm->SetTaskbarOverlayIcon(IDI\_ICON\_NO\_VOLUME,L"No Volume");
     15. }
     16. void COverlayIconTab::OnOverlayIconNone()
     17. {
     18. // TODO: Add code to clear overlay icon
     19. CMainFrame\* mainFrm = dynamic\_cast<CMainFrame\*>(AfxGetApp()->GetMainWnd());
     20. if (mainFrm)
     21. mainFrm->SetTaskbarOverlayIcon((HICON)0, L"");
     22. }
  5. Build and run the demo to test the remaining icon overlays.

1. Task 2—Using Taskbar Progress Bars

In this task, you will set the state and value of the application’s taskbar progress bar when the user selects the progress state from a combo box or changes the value by using a slider. MFC provides a number of methods to update the Taskbar progress bar’s value and status:

* + CFrameWnd::SetProgressBarRange()
  + CFrameWnd::SetProgressBarPosition()
  + CFrameWnd::SetProgressBarState()
  1. In the TaskbarDemo\SourceFiles sub tree:
     1. Open the file ProgressBarTab.cpp, and
     2. Look for the slider control’s message handler, CProgressBarTab::OnHScroll()
     3. Add the following code to update the Taskbar’s progress bar after the comment to add code to handle the slide bar change:
     4. C++
     5. UpdateData();
     6. CMainFrame\* mainFrm = dynamic\_cast<CMainFrame\*>(AfxGetApp()->GetMainWnd());
     7. mainFrm->SetProgressBarPosition(m\_ProgressSliderValue);
  2. Build and run the application, and then:
     1. Go to the Progress Bar tab.
     2. Move the slider, and notice the movement on the application’s Taskbar button.   
        You should see an output similar to Figure 4 below:
     3. 
     4. Figure 4
     5. Slider bar and taskbar button progress indicators
  3. Now, we want to update the status of the progress bar whenever the relevant combo box option is updated, so we add the necessary code to the CProgressBarTab::OnCbnSelchangeProgressStatusCombo() event handler:
     1. C++
     2. CMainFrame\* mainFrm = dynamic\_cast<CMainFrame\*>(AfxGetApp()->GetMainWnd());
     3. switch (m\_ProgrssStatusCombo.GetCurSel())
     4. {
     5. case 0 :
     6. mainFrm->SetProgressBarState(TBPF\_NORMAL);
     7. break;
     8. case 1 :
     9. mainFrm->SetProgressBarState(TBPF\_ERROR);
     10. break;
     11. case 2 :
     12. mainFrm->SetProgressBarState(TBPF\_PAUSED);
     13. break;
     14. case 3 :
     15. mainFrm->SetProgressBarState(TBPF\_INDETERMINATE);
     16. break;
     17. case 4 :
     18. mainFrm->SetProgressBarState(TBPF\_NOPROGRESS);
     19. break;
     20. default:
     21. mainFrm->SetProgressBarState(TBPF\_NORMAL);
     22. }
  4. Build and run the application, and then:
     1. Navigate to the Progress Bar tab, and
     2. Select a different style from the Progress Status Combo and then
     3. Move the progress slider. The progress bar state will be changed to the appropriate style; namely, the taskbar’s progress bar is red when the Progress Status is set to Error, the taskbar’s progress bar is green when the Progress Status is set to Normal (check out the other styles too):
     4. image8.jpeg
     5. Figure 5
     6. Red progress bar indicating error
  5. Finally, this last step is optional, but you might want to reset the status combo to ‘Normal’ whenever the progress is updated from No Progress or Intermediate.   
     To fix this, add the following at the end of CProgressBarTab::OnHScroll():
     1. C++
     2. if (m\_ProgressSliderValue > 0 && m\_ProgrssStatusCombo.GetCurSel() >= 3) // No progress or intermediate
     3. {
     4. // Set to something else other than no progress
     5. m\_ProgrssStatusCombo.SetCurSel(0); // Normal
     6. }

Task 3—Using Taskbar Jump Lists

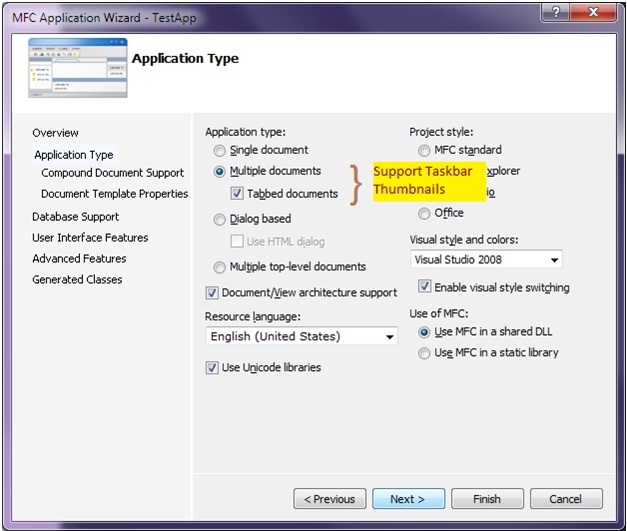
* 1. In this task, you will add the functionality to support adding known categories, custom categories, destinations, and tasks to the application’s jump list. To support this, MFC provides a new class, CJumpList, that allows manipulating the categories and entries on the Taskbar application’s JumpList.
  2. In the TaskbarDemo\SourceFiles sub tree, open the file JumpListTab.cpp and look for the implementation of CJumpListTab::UpdateRegistration(). This method executes another process to register needed file associations. The file association registration does not need to be in another process; however it was placed in another process to keep this tutorial application simple and focused.
     1. **Note:** We need to register the file associations because an application must be a registered file type handler for an item of that type to appear in its Jump List. It does not, however, need to be the default handler for that file type.
  3. Add the necessary code to update the registry and unregister jpg file associations in the corresponding message handlers:
     1. C++
     2. void CJumpListTab::OnRegisterFileType()
     3. {
     4. // TODO: Register file associations
     5. UpdateRegistration(TRUE);
     6. }
     7. void CJumpListTab::OnUnregisterFileType()
     8. {
     9. // TODO: Unregister file associations
     10. UpdateRegistration(FALSE);
     11. }
  4. The next steps will be used to add the necessary code to add the various Jump List items. Here’s a description for each:
     1. 
     2. Figure 6
     3. Jump List item descriptions
  5. To append a known category to the Jump List, add this code to the CJumpListTab::OnAddKnownCategory() event handler:
     1. C++
     2. int curSel = m\_knownCategoryCombo.GetCurSel();
     3. if (curSel < 0)
     4. {
     5. AfxMessageBox(L"Must select a known category above");
     6. return;
     7. }
     8. if (curSel == 0)
     9. {
     10. if (m\_jumpList.AddKnownCategory(KDC\_FREQUENT))
     11. m\_JumpList\_Display\_Items.AddString(L"Known Category: Frequent");
     12. else
     13. AfxMessageBox(L"Failed to add Frequent Known Category");
     14. }
     15. else if (curSel == 1)
     16. {
     17. if (m\_jumpList.AddKnownCategory(KDC\_RECENT))
     18. m\_JumpList\_Display\_Items.AddString(L"Known Category: Recent");
     19. else
     20. AfxMessageBox(L"Failed to add Recent Known Category");
     21. }
  6. To add a Destination in the Jump List, add this code in the CJumpListTab::OnAddDestination() event handler:
     1. C++
     2. UpdateData(); // Sync variables to UI controls
     3. if (m\_destinationPath.IsEmpty() || m\_destinationCategory.IsEmpty())
     4. {
     5. AfxMessageBox(L"Both Destination categeory and path must be added.");
     6. return;
     7. }
     8. if (!m\_jumpList.AddDestination(m\_destinationCategory, m\_destinationPath))
     9. {
     10. AfxMessageBox(L"Failed to add destination.");
     11. }
     12. else
     13. {
     14. CString text = L"Destination: " + m\_destinationPath;
     15. m\_JumpList\_Display\_Items.AddString(text);
     16. }
  7. To add a new task in the Jump List, add this code in the CJumpListTab::OnAddTask() event handler:
     1. C++
     2. UpdateData(); // Sync variables to UI controls
     3. if (m\_taskPath.IsEmpty() || m\_taskTitle.IsEmpty())
     4. {
     5. AfxMessageBox(L"Both Task Path and Title must be added.");
     6. return;
     7. }
     8. if (!m\_jumpList.AddTask(m\_taskPath, m\_taskArgs, m\_taskTitle, m\_taskIconPath, m\_taskIconIndex))
     9. {
     10. AfxMessageBox(L"Failed to add Task.");
     11. }
     12. else
     13. {
     14. CString text = L"Task: " + m\_taskTitle ;
     15. m\_JumpList\_Display\_Items.AddString(text);
     16. }
  8. To allow the removal of all entries from the Jump List, add the following code to the CJumpListTab::OnClearAll() event handler:
     1. C++
     2. m\_jumpList.ClearAll();
     3. m\_JumpList\_Display\_Items.ResetContent();
  9. To commit changes made to the Jump List, add the following code to the CJumpListTab::OnCommit() event handler:
     1. C++
     2. if (m\_jumpList.CommitList())
     3. {
     4. m\_JumpList\_Display\_Items.ResetContent();
     5. // Start another list
     6. m\_jumpList.InitializeList();
     7. }
     8. else
     9. {
     10. AfxMessageBox(L"Failed to commit jump list.");
     11. }
     12. **Note:** All updates to a JumpList must take place in between calls to CJumpList::InitializeList() and CJumpList::CommitList(), otherwise, additions will fail.
  10. Build and run the application.
  11. In the Jump List tab, register the jpeg file association by pressing the **Register .jpg file association** button.
      1. **Note:** This registration process will need to run elevated, so UAC will prompt you for elevated privileges.
  12. In the Jump List tab, start adding items into the Jump List, making sure to click the “Commit” button at the end so added items can appear on the Jump List. Examples of items to be added:
      1. Add both Recent and Frequent Known Categories (select *Frequent* from the Known Category drop-down and then click **Add Known Category ==>**)
      2. Add a Destination by filling in Category = “Custom Category” (no quotes) and Path = C:\Users\Public\Pictures\Sample Pictures\Desert.jpg, and then clicking **Add Destination ==>**.
      3. Add a Task by filling in Title=”Bing” (no quotes) and Path/Url=http://www.bing.com (Icon is optional), and then clicking **Add task ==>**.
      4. Click **Commit**.
      5. Right-click on the application’s taskbar button and you’ll observe output that will look similar to the following:
      6. 
      7. Figure 7
      8. Sample Jump List

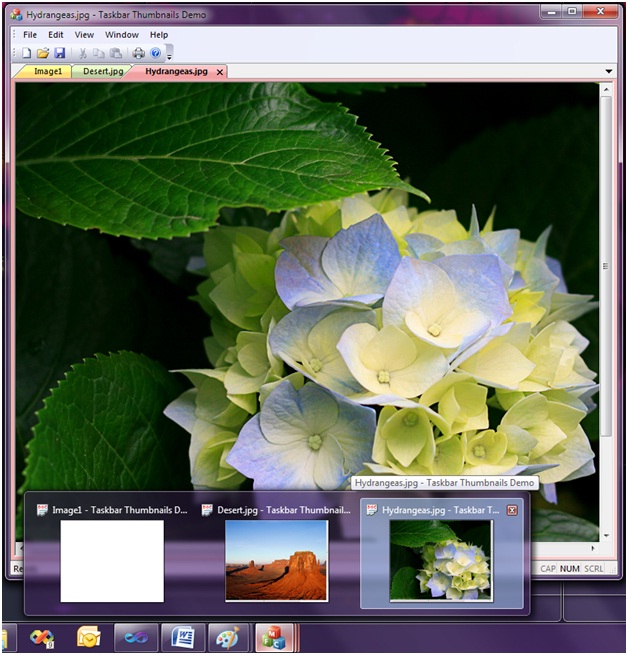
Exercise 2: Supporting Taskbar Thumbnails

* 1. MFC has added support for Taskbar Thumbnails in Visual Studio 2010. The support built into MFC applications is based on the tabbed Multiple Document Interface (MDI).
  2. **Note:** In MFC, the support for Taskbar thumbnail previews is built in, so the Taskbar thumbnails will show any rendering within the views. Therefore, other than implementing your own View drawing, you need not provide any explicit code to update those Thumbnails.

Task 1—Supporting Taskbar Thumbnails

To enable Taskbar Thumbnails in an MFC application while using the MFC application wizard, all the user needs to do is select the “Multiple documents” application type with the option “Tabbed documents” enabled (see Figure 8). When the application runs, MFC will take a snapshot of each view and send it to the Taskbar APIs to display as thumbnails.

* 1. 
  2. Figure 8
  3. MFC Application Wizard - Application Type

1. In Visual Studio 2010, open the *TaskbarThumbnails\_Solution\TaskbarthumbnailsDemo.sln* solution (under the Source folder).
2. Build and Run the application.
3. Open a few .jpeg image files using the application.
4. Hover over the application Demo button on the taskbar. You should see an output similar to Figure 9 below.
   * 1. 
     2. Figure 9
     3. Taskbar Thumbnails for MDI applications

Summary

* 1. In this lab, you have experimented with Windows 7 Taskbar functionality from within a simple MFC application by using the new MFC support for the Windows 7 Taskbar. You have seen how easy it is to provide relevant progress and status information from the taskbar button using progress bars and overlay icons, how to quickly access common tasks and destinations using jump lists, and how to take advantage of MFC’s built-in support for thumbnail previews.
  2. Integrating a production-quality application with the Windows 7 Taskbar might require slightly more work than you have done in this lab, but you are now sufficiently prepared to begin this quest on your own.