Computing listview infotips in the background

devblogs.microsoft.com/oldnewthing/20061214-02

December 14, 2006

Raymond Chen

When the listview control asks you for an infotip, it sends you then `LVN_GETINFOTIP` notification, and when you return, the result is displayed as the infotip. But what if computing the infotip takes a long time? You don't want to stall the UI thread on a long operation, after all. This is where `LVM_SETINFOTIP` comes in.

If you want to say, "Um, I'm not ready with that infotip yet," you do two things: First, you return a blank infotip to the listview in response to `LVN_GETINFOTIP`; this tells the listview not to display anything. Then, when you have the infotip, you send the `LVM_SETINFOTIP` message to say, "Oh, here's that infotip you asked for. If you were still wondering." If the user is still hovering over the item that the infotip was requested for, the infotip will be displayed. Otherwise, the infotip will be thrown away (since the user doesn't want to see it any more).

Here's a quick and dirty (and not very good) implementation of this algorithm, just to illustrate the point. Start with the program from last time and make the following changes:

```
// add to top of file
#define UNICODE
#define _UNICODE
class BackgroundInfoTip {
public:
 BackgroundInfoTip() { ZeroMemory(&m_sit, sizeof(m_sit)); }
 BOOL Start(HWND hwnd, NMLVGETINFOTIP *pit)
 {
  m_hwnd = hwnd;
  m_sit.cbSize = sizeof(m_sit);
  m_sit.dwFlags = 0;
  m_sit.iItem = pit->iItem;
  m_sit.iSubItem = pit->iSubItem;
  if ((pit->dwFlags & LVGIT_UNFOLDED) ||
     (m_pszPrefix = StrDup(pit->pszText)) != NULL) {
   return QueueUserWorkItem(s_Work, this, WT_EXECUTELONGFUNCTION);
  }
  return FALSE;
 }
 ~BackgroundInfoTip() {
  LocalFree(m_pszPrefix);
  CoTaskMemFree(m_sit.pszText);
 }
 static DWORD CALLBACK s_Work(void *lpParameter);
 void Work();
 HWND m_hwnd;
 LVSETINFOTIP m_sit;
 LPTSTR m_pszPrefix;
};
void BackgroundInfoTip::Work()
{
 Sleep(3000); // artificial delay
 TCHAR szInfotip[INFOTIPSIZE];
 if (m_pszPrefix) {
  StringCchCopy(szInfotip, INFOTIPSIZE, m_pszPrefix);
  StringCchCat(szInfotip, INFOTIPSIZE, TEXT("\r\n"));
 } else {
  szInfotip[0] = TEXT('\0');
 }
 StringCchCat(szInfotip, INFOTIPSIZE, TEXT("Here is an infotip"));
 if (SUCCEEDED(SHStrDup(szInfotip, &m_sit.pszText)) &&
     PostMessage(m_hwnd, WM_APP, 0, (LPARAM)this)) {
  // ownership transferred to main window
 } else {
  delete this;
 }
}
DWORD BackgroundInfoTip::s_Work(void *lpParameter)
{
 BackgroundInfoTip *self =
    reinterpret_cast<BackgroundInfoTip*>(lpParameter);
 self->Work();
```

```
 return 0;
}
void OnGetInfoTip(HWND hwnd, NMLVGETINFOTIP *pit)
{
 if (!pit->cchTextMax) return;
 // note: uses no-throwing "new"
 BackgroundInfoTip *pbit = new BackgroundInfoTip();
 if (pbit && pbit->Start(hwnd, pit)) {
  pit->pszText[0] = TEXT('\0'); // no tip yet
 } else {
  delete pbit;
 }
}
void FinishInfoTip(BackgroundInfoTip *pbit)
{
 SendMessage(g_hwndChild, LVM_SETINFOTIP, 0, (LPARAM)&pbit->m_sit);
 delete pbit;
}
    case WM_APP: FinishInfoTip((BackgroundInfoTip *)lParam); return 0;
```

We start by defining `UNICODE` and `_UNICODE` because we're using the Windows XP common controls (version 6), and that version of the common controls supports only Unicode. (Version 5 of the common controls doesn't support the `LVM_SETINFOTIP` message.)

Next, let's skip ahead to the `OnGetInfoTip` function. When we are asked to produce an infotip, we create an instance of our helper class and get it started. Once we're convinced that the infotip computation is under way, we return a blank infotip to the listview to tell it, "Don't display anything yet."

The helper class `BackgroundInfoTip` starts by capturing the parameters of the `NMLVGETINFOTIP`. Again, we pay close attention to the `LVGIT_UNFOLDED` flag: If it is not set, then we save the text currently in the infotip so we can prepend it to the infotip text. We then toss the item onto the thread pool and wait for the work item to fire.

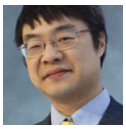
As before, our infotip computation is artificially simple: It's just a hard-coded string. In real life you presumably would actually sit down and compute something. I stuck in a `Sleep(3000)` to create an artificial delay in order to simulate this "computation time". Once we have our answer, remembering to prefix the original infotip text if the item was folded, we save it in the `LVSETINFOTIP` structure and post a message back to our main thread to say, "Okay, the infotip is ready."

On receipt of the `WM_APP` message (in a proper program, it would have a more meaningful name like `WM_INFOTIPREADY`), we tell the listview that we have our infotip, in case it was still interested. And since this completes the background infotip calculation, we can delete the helper object.

This is not very good code because it fails to handle some obvious cases: If the user moves to a new listview item, the listview will ask for a new infotip. Our code doesn't attempt to cancel the previous background infotip; as a result, if the user waves the mouse over the listview, we may end up with a large number of background infotip computations, all but one of which will be discarded. Even worse, all the discarded ones will be ahead of the important one in the work item queue: You're spending all your time doing something whose result is going to be thrown away, and not executing the work item whose result is actually useful.

The code also doesn't handle the case where the window is closed while the background work items are still running. Closing the window should cancel the work items or at least tell them that they don't have a main window to talk to any more.

Adding code to handle all these edge cases would have distracted from the point of this article, so I leave you to make this code more solid as an exercise.



Raymond Chen

Follow