Математические структуры и моделирование
2016. № 3(39). С. 127–141

УДК 004.42

ПОШАГОВАЯ ИНСТРУКЦИЯ НАПИСАНИЯ SHELL NAMESPACE EXTENSION

Д.С. Сиберт
студент, e-mail: dimaaasik.s@gmail.com

Омский государственный университет им. Ф.М. Достоевского

Аннотация. В статье приведена развёрнутая инструкция написания Shell Namespace Extensions, включающая создание пустого приложения и набора необходимых функций файловой системы.

Ключевые слова: пошаговая инструкция, windows, shell, namespace, extension.

Введение

Существуют популярные приложения Dropbox, Google Drive, Yandex.Диск и т.д., которые предоставляют возможность работать с файлами и директориями, расположенными на удалённых серверах, одному или нескольким пользователям.

Некоторые из этих инструментов встраиваются в операционную систему и позволяют работать со своим содержимым через её графическую оболочку. В семействе операционных систем Windows организовать взаимодействие пользователя с данными через GUI можно, написав расширение пространства имён оболочки (Shell Namespace Extension, далее NSE). Оно состоит из двух компонент:

- Менеджер данных.
- Интерфейс между менеджером данных и оболочкой Windows.

Первая делается полностью по усмотрению разработчика. Чтобы сделать вторую, необходимо реализовать набор интерфейсов: IShellFolder2, IPersistFolder2, IShellView, IEnumIDList, IDropTarget, IExplorerCommandProvider, IEnumExplorerCommand, IContextMenu, IShellExtInit, IObjectWithSite. К ним будет обращаться оболочка Windows за обработкой запросов пользователя. Кроме того, необходимо реализовать функции, стандартные для DLL: DllMain, DllGetClassObject, DllCanUnloadNow.

Информацию о написании NSE можно найти в MSDN [1], книге Visual C++ Windows Shell Programming [2], цикле статей [3]. Некоторые шаги могут быть упрощены:

- Графическое представление директории.
- Перечисление содержимого виртуальной директории.

Некоторые шаги реализации не представлены в этих источниках, а именно:

- Переименование директории\файла.

- Удаление директории\файла.
- Создание новой директории.
- Drag and drop элементов файловой системы в NSE и элементов внутри NSE.

В тексте статьи освещены все эти пункты.

Написание пустого расширения

В целях упрощения задачи можно воспользоваться готовым примером расширения из примеров Windows SDK версии 7.1 [4]. После установки SDK пример будет находиться по пути C:\ProgramFiles\Microsoft SDKs\Windows\v7.1\Samples\winui\shell\shellextensibility\explorerdataprovider. В примере используются актуальные, на момент написания статьи, версии классов, подходящие как для 32, так и для 64 разрядных систем. В расширении частично реализованы дополнительные функции, такие как контекстное меню (IContextMenu), панель инструментов (IExplorerCommandProvider), перечисление элементов (IEnumIDList), категоризация элементов (ICategoryProvider). Для пустого расширения в них нету необходимости. Рассмотрим только базовый класс и регистрацию в системе.

Основной класс NSE CFolderViewImplFolder (CFolder в листингах кода) находится в файле ExplorerdataProvider.cpp. Он реализует интерфейсы IShellFolder2, IPersistFolder2. Реализация отвечает за внутреннюю работу корневой и вложенных директорий, инициализации обработки панели инструментов, контекстного меню, графического представления и др. Каждой директории соответствует свой экземпляр такого класса.

Для работы пустого расширения необходимо реализовать только часть методов:

1. Методы интерфейса IUnknown (является базовым классом для IShellFolder2): QueryInterface, AddRef, Release. Они служат для того, чтобы получать определённый интерфейс объекта, увеличивать счётчик ссылок, уменьшать счётчик ссылок и уничтожать объект.

```
HRESULT CFolder::QueryInterface(REFIID riid, void **ppv)
{
  static const QITAB qit[] =
  {
    QITABENT(CFolder, IShellFolder),
    QITABENT(CFolder, IShellFolder2),
    QITABENT(CFolder, IPersist),
    QITABENT(CFolder, IPersistFolder),
    QITABENT(CFolder, IPersistFolder2),
    { 0 },
  };
  return QISearch(this, qit, riid, ppv);
}
```

```
ULONG CFolder::AddRef()
{
  return InterlockedIncrement(&m_cRef);
}

ULONG CFolder::Release()
{
  long cRef = InterlockedDecrement(&m_cRef);
  if (0 == cRef)
  {
    delete this;
  }
  return cRef;
}
```

2. Метод IShellFolder::BindToObject. Создает новый экземпляр класса директории для вложенной директории. Конструирует её PIDL и передаёт ссылку системе. Напомню, что в пространстве имён оболочки Windows каждый элемент уникально идентифицируется объектом PIDL. Это указатель на список структур, которые содержат полный путь до элемента в пространстве имён. При написании своего расширения разработчику обязательно надо будет осуществлять действия над этими структурами, такие как: создание, удаление, конкатенация, клонирование и другие. Во всех источниках, которые мне попадались, авторы реализовывали специальный класс – менеджер структур PIDL, который выполнял вышеописанные действия. Вместо этого можно воспользоваться функциями, которые предоставляет API Windows. Эти функции имеют префикс IL* и находятся в библиотеке Shell32.lib. Например, ILClone, которая создаёт полную копию некоторой структуры PIDL. Для корневой директории PIDL создаётся системой на основе заполненных при регистрации расширения для регистра данных и передаётся экземпляру класса директории с помощью метода IPersistFolder::Initialize.

```
HRESULT CFolder::BindToObject(
  PCUIDLIST_RELATIVE pidl, IBindCtx *pbc,
  REFIID riid, void **ppv)
{
  *ppv = NULL;
  HRESULT hr = S_OK;
  if (SUCCEEDED(hr))
  {
    CFolder* pCFolder = new (std::nothrow) CFolder();
    hr = pCFolder ? S_OK : E_OUTOFMEMORY;
    if (SUCCEEDED(hr))
    {
      PITEMID_CHILD pidlFirst = ILCloneFirst(pidl);
      hr = pidlFirst ? S_OK : E_OUTOFMEMORY;
      if (SUCCEEDED(hr))
```

```
      {
        PIDLIST_ABSOLUTE pidlBind =
          ILCombine(m_pidl, pidlFirst);
        hr = pidlBind ? S_OK : E_OUTOFMEMORY;
        if (SUCCEEDED(hr))
        {
          hr = pCFolder->Initialize(pidlBind);
          if (SUCCEEDED(hr))
          {
            PCUIDLIST_RELATIVE pidlNext = ILNext(pidl);
            if (ILIsEmpty(pidlNext))
            {
              hr = pCFolder->QueryInterface(riid, ppv);
            }
            else
            {
              hr = pCFolder->BindToObject(
                pidlNext, pbc, riid, ppv);
            }
          }
          CoTaskMemFree(pidlBind);
        }
        ILFree(pidlFirst);
      }
      pCFolder->Release();
    }
  }
  return hr;
}
```

3. Метод IPersistFolder::Initialize. Для корневой директории вызывается системой, а для вложенных внутри обработчика BindToObject, для инициализации экземпляра класса директории её уникальным идентификатором PIDL.

```
HRESULT CFolder::Initialize(PCIDLIST_ABSOLUTE pidl)
{
  m_pidl = ILCloneFull(pidl);
  return m_pidl ? S_OK : E_FAIL;
}
```

4. Метод IShellFolder::CreateViewObject. Вызывается системой для получения объекта, который затем будет использован для взаимодействия с директорией.

```
HRESULT CFolder::CreateViewObject(
  HWND hwnd, REFIID riid, void **ppv)
```

```
{
  *ppv = NULL;
  HRESULT hr = E_NOINTERFACE;
  if (riid == IID_IShellView)
  {
    SFV_CREATE csfv = { sizeof(csfv), 0 };
    hr = QueryInterface(IID_PPV_ARGS(&csfv.pshf));
    hr = SHCreateShellFolderView(
      &csfv, (IShellView**)ppv);
    csfv.pshf->Release();
  }
  return hr;
}
```

Для отображения директории в GUI используется реализация интерфейса IShellView.

5. Метод IShellFolder2:: GetDetailsOf. Он вызывается системой для получения более детальной информации о конкретном элементе директории. Даже для пустого расширения, для корректной работы, требуется верно обработать запрос с параметром iColumn равным 0.

```
HRESULT CFolder::GetDetailsOf(PCUITEMID_CHILD pidl,
  UINT iColumn, SHELLDETAILS *pDetails)
{
  PROPERTYKEY key;
  HRESULT hr = S_OK;
  pDetails->cxChar = 24;
  WCHAR szRet[MAX_PATH];
  if (!pidl)
  {
    switch (iColumn)
    {
      case 0:
      {
        key = PKEY_ItemNameDisplay;
        pDetails->fmt = LVCFMT_LEFT;
        hr = StringCchCopy(
        szRet, ARRAYSIZE(szRet), L"Name");
        break;
      }
      default:
      {
        hr = E_FAIL;
        break;
      }
    }
  }
```

```
  else if (SUCCEEDED(hr))
  {
    hr = S_OK;
  }
  if (SUCCEEDED(hr))
  {
    hr = StringToStrRet(szRet, &pDetails->str);
  }
  return hr;
}
```

6. Метод IPersist::GetClassID. Он вызывается системой, чтобы запросить GUID, сгенерированный для всего расширения.

```
DEFINE_GUID(CLSID_FolderViewImpl,
  0xba16ce0e, 0x728c, 0x4fc9, 0x98, 0xe5,
  0xd0, 0xb3, 0x5b, 0x38, 0x45, 0x97);

HRESULT Folder::GetClassID(CLSID *pClassID)
{
  *pClassID = CLSID_FolderViewImpl;
  return S_OK;
}
```

7. Необходима функция, которая будет создавать экземпляр класса корневой директории.

```
HRESULT CFolder_CreateInstance(REFIID riid, void **ppv)
{
  *ppv = NULL;
  CFolder* pFolder = new (std::nothrow) CFolder();
  HRESULT hr = pFolder ? S_OK : E_OUTOFMEMORY;
  if (SUCCEEDED(hr))
  {
    hr = pFolder->QueryInterface(riid, ppv);
    pFolder->Release();
  }
  return hr;
}
```

8. Метод IShellFolder::EnumObjects должен возвращать значение S_FALSE. Все остальные, присутствующие в интерфейсах методы, должны возвращать значение E_NOTIMPL.

Графическое представление директории

Для отображения директории необходимо реализовать интерфейс IShellView. Система запрашивает его при помощи вызова метода

CFolder::CreateViewObject(HWND hwnd, REFIID riid, void **ppv) с riid равным IID_IShellView. В зависимости от требований, можно сделать какой угодно внешний вид, в большинстве материалов по написанию расширений делают свою реализацию этого интерфейса. Если необходимо эмулировать стандартную работу с файловой системой, лучше всего воспользоваться функцией SHCreateShellFolderView(const SFV_CREATE *pcsfv, IShellView **ppsv). На выходе получаем объект, который отобразит директорию в обычной, для Windows форме (этот метод используется для пустого расширения из пункта выше). Непосредственно для отображения элементов в директории реализации IShellView необходимы данные о списке элементов, их типах, порядке и другие. Для этого системой будут вызываться методы интерфейсов IShellFolder и IShellFolder2: EnumObjects, CompareIDs, GetAttributesOf, GetDefaultColumn, GetUIObjectOf, GetDefaultColumnState, GetDetailsEx, GetDetailsOf, MapColumnToSCID.

При запросе интерфейса IShellView во входном параметре hwnd находится корректный уникальный идентификатор окна расширения. Его необходимо сохранить во внутреннюю переменную класса и в дальнейшем использовать для инициирования обновления окна, при помощи посылки сообщения.

```
SendMessage(m_hWnd, 0x111, 0x7103, 0);
```

Перечисление содержимого виртуальной директории

Чтобы отобразить список всех элементов в корневой и дочерних директориях, система вызывает метод IShellFolder:: EnumObjects.

```
HRESULT CFolder::EnumObjects(HWND hwnd, DWORD grfFlags,
  IEnumIDList **ppenumIDList)
{
  HRESULT hr;
  CEnumIDList *penum = new (std::nothrow) CEnumIDList();
  hr = penum ? S_OK : E_OUTOFMEMORY;
  if (SUCCEEDED(hr))
  {
    hr = penum->Initialize();
    if (SUCCEEDED(hr))
    {
      hr = penum->QueryInterface(
        IID_PPV_ARGS(ppenumIDList));
    }
    penum->Release();
  }
  return hr;
}
```

Ей необходимо передать указатель на реализацию интерфейса IEnumIDList, которая является итератором по всем элементам конкретной директории. Для каждого элемента система запрашивает информацию об атрибутах вызовом

метода IShellFolder::GetAttributesOf. Атрибутами определяется тип элемента (файл или папка), возможность переименования, возможность удаления и другие.

```
HRESULT CFolder::GetAttributesOf(UINT cidl,
  PCUITEMID_CHILD_ARRAY apidl, ULONG *rgfInOut)
{
  HRESULT hr = S_OK;
  //Случай, когда система запрашивает
  //информацию об одном объекте
  if (1 == cidl)
  {
    //Прежде, чем выставлять флаги,
    //необходимо проанализировать apidl[0]
    //для того чтобы понять, какие свойства имеет объект
    //эти данные можно либо запросить у сервера,
    //либо хранить прямо в PIDL
    DWORD dwAttribs = 0;
    dwAttribs |= SFGAO_FOLDER;
    dwAttribs |= SFGAO_DROPTARGET;
    dwAttribs |= SFGAO_CANRENAME;
    dwAttribs |= SFGAO_CANDELETE;
    dwAttribs |= SFGAO_HASSUBFOLDER;
    *rgfInOut &= dwAttribs;
  }
  return hr;
}
```

В момент инициализации реализации IEnumIDList может понадобиться строковое представление пути к текущей директории. Его можно получить по PIDL с помощью функции SHGetNameFromIDList(PCIDLIST_ABSOLUTE, SIGDN, PWSTR).

Переименование директории\файла

При использовании стандартной реализации IShellView, полученной с помощью функции IShellFolder::SHCreateShellView, переименование элементов обрабатывается классом директории. Каждому объекту, который можно будет переименовывать, необходимо выставить атрибут SFGAO_CANRENAME в методе IShellFolder::GetAttributesOf. В контекстном меню для этого элемента, появится пункт «переименовать». Также инициировать процесс переименования можно будет повторным нажатием на выделенный элемент. Если имя элемента будет изменено, то при подтверждении будет вызвана функция IShellFolder::SetNameOf, одними из параметров которой будут PIDL элемента и его новое имя. В теле этой функции необходимо обработать переименование.

Удаление директории\файла

Элементам директории можно выставлять атрибут SFGAO_CANDELETE в методе IShellFolder::GetAttributesOf, но даже при том, что в контекстном меню появляется активируемый пункт удаления, точки входа для обработки удаления, как в случае с переименованием, я не нашёл. Пришлось реализовывать не совсем стандартный способ.

У NSE существует два типа контекстных меню: для директории и для элементов директории. Контекстное меню для элементов директории является общим для всего расширения. Оно реализует интерфейсы IContextMenu, IShellExtInit. Подобно основному классу расширения, создание экземпляра класса меню оборачивается вызовом IClassFactory::CreateInstance. Для меню создаются соответствующие записи в регистре. Помимо основных команд, таких как переименование и открытие, реализуемых системой, контекстное меню можно расширять собственными командами. Удаление элемента будет одной из таких команд.

Для добавления и обработки своих команд необходимо реализовать функции IContextMenu::QueryContextMenu, IContextMenu::InvokeCommand, IShellExtInit::Initialize. QueryContextMenu - добавляет пользовательские пункты в меню.

```
HRESULT CContextMenu::QueryContextMenu(HMENU hmenu,
  UINT indexMenu, UINT idCmdFirst,
  UINT /*idCmdLast*/, UINT /*uFlags*/)
{
  WCHAR szMenuItem[80];
  //g_hInst - глобальная переменная,
  //которая инициализируется в DllMain
  LoadString(g_hInst, IDS_DELETE,
    szMenuItem, ARRAYSIZE(szMenuItem));
  //MENUVERB_DELETE - числовой идентификатор пункта меню,
  //для последующих пунктов можно брать на единицу больше
  InsertMenu(hmenu, indexMenu++, MF_BYPOSITION,
    idCmdFirst + MENUVERB_DELETE, szMenuItem);
  //тут добавляем следующие пункты...

  return MAKE_HRESULT(SEVERITY_SUCCESS, 0, (USHORT)(2));
}
```

Initialize - система вызывает функцию и передаёт через неё объект, содержащий информацию об элементе, для которого вызвано контекстное меню.

```
HRESULT CContextMenu::Initialize(
  PCIDLIST_ABSOLUTE /* pidlFolder */,
  IDataObject *pdtobj, HKEY /* hkeyProgID */)
{
  //Освобождаем старый объект
  if (m_pdtobj)
```

```
  {
    m_pdtobj->Release();
    m_pdtobj = NULL;
  }
  //Запоминаем новый объект
  m_pdtobj = pdtobj;
  if (pdtobj)
  {
    pdtobj->AddRef();
  }
  return S_OK;
}
```

InvokeCommand - выполнение команды над объектом, полученным при вызове функции IShellExtInit::Initialize.

```
HRESULT CContextMenu::InvokeCommand(
  LPCMINVOKECOMMANDINFO pici)
{
  HRESULT hr = E_INVALIDARG;
  UINT uID;
  //Получаем числовой идентификатор пункта меню,
  //способ получения зависит от реализации,
  //можно анализировать название команды
  if (SUCCEEDED(_GetCommandId(pici, &uID)) && _pdtobj)
  {
    switch (uID)
    {
      case MENUVERB_DELETE:
      {
        IShellItemArray *psia;
        hr = SHCreateShellItemArrayFromDataObject(
          _pdtobj,IID_PPV_ARGS(&psia));
        IShellItem *psi;
        //Получаем объект с данными об элементе директории
        hr = psia->GetItemAt(0, &psi);
        if (SUCCEEDED(hr))
        {
          int nDlgRes = MessageBox(
            NULL, L"Sure?", L"Deleting!", MB_OKCANCEL);
          if (nDlgRes == IDOK) {
            //Обрабатываем удаление
            //Посылаем сообщение перерисовки окну NSE
            SendMessage(pici->hwnd, 0x111, 0x7103, 0);
          }
          psi->Release();
          psia->Release();
        }
        break;
```

```
      }
    }
  }
  return hr;
}
```

Создание новой директории

Создавать новые директории можно с помощью контекстного меню для директории. Для этого необходимо создать класс, реализующий интерфейс IContextMenu. Добавление пунктов в это меню происходит в функции IContextMenu::QueryContextMenu, точно так же, как и в предыдущем пункте. Обработка команды в функции IContextMenu::InvokeCommand выглядит иначе.

```
HRESULT FolderContextMenu::InvokeCommand(
  LPCMINVOKECOMMANDINFO pici)
{
  HRESULT hr = E_INVALIDARG;
  UINT uID;
  //Получаем числовой идентификатор пункта меню,
  //способ получения зависит от реализации,
  //можно анализировать название команды
  if (SUCCEEDED(_GetCommandId(pici, &uID)) && _pdtobj)
  {
    switch (uID)
    {
      case MENUVERB_CREATE:
      {
        m_pFolder->_OnCreateFolder();
        break;
      }
    }
  }
  return hr;
}
```

Инициализация экземпляра класса меню происходит в методе IShellFolder::CreateViewObject.

```
HRESULT CFolder::CreateViewObject(
  HWND hwnd, REFIID riid, void **ppv)
{
  *ppv = NULL;
  HRESULT hr = E_NOINTERFACE;
  //...
  if (riid == IID_IContextMenu)
  {
    FolderContextMenu* p_folderContextMenu =
      new (std::nothrow) FolderContextMenu(this);
```

```
    hr = p_folderContextMenu ? S_OK : E_OUTOFMEMORY;
    if (SUCCEEDED(hr))
    {
      hr = p_folderContextMenu->QueryInterface(riid, ppv);
      p_folderContextMenu->Release();
    }
  }
  //....
  return hr;
}
```

В конструктор класса меню передаётся ссылка на экземпляр класса директории. Это сделано, чтобы вызывать метод_OnCreateFolder(), который будет непосредственно обрабатывать создание новой директории.

Drag and drop элементов файловой системы в NSE и элементов внутри NSE

Для того чтобы NSE начало поддерживать эти операции, необходимо будет реализовать интерфейс IDropTarget. Он содержит методы, в которых будут обрабатываться перетаскивание файлов и отображение нужных эффектов.

Необходимо обработать два случая перетаскивания файлов. Если «бросить» элементы на пустое место текущей открытой директории, то будет вызван метод IShellFolder::CreateVIewObject с REFIID равным IID_IDropTarget.

```
HRESULT CFolder::CreateViewObject(
  HWND hwnd, REFIID riid, void **ppv)
{
  *ppv = NULL;
  HRESULT hr = E_NOINTERFACE;
  //...
  if (riid == IID_IDropTarget)
  {
    IUnknown *pFolder;
    this->QueryInterface(IID_IUnknown, (void**) &pFolder);
    MyDropTarget *pMyDropTarget =
      new MyDropTarget(pFolder, NULL);
    pMyDropTarget->AddRef();
    hr = pMyDropTarget->QueryInterface(
      IID_IDropTarget, ppv);
    pMyDropTarget->Release();
  }
  //...
  return hr;
}
```

Если «бросить» элементы на элемент-директорию в текущей открытой директории, то будет вызван метод IShellFolder::GetUIObjectOf(HWND

hwndOwner, UINT cidl, PCUITEMID_CHILD_ARRAY apidl, REFIID riid, UINT *rgfReserved,void **ppv), при этом riid будет равен IID_IDropTarget, а первым и единственным элементом массива apidl будет PIDL целевой директории.

```
HRESULT CFolder::GetUIObjectOf(HWND hwnd, UINT cidl,
  PCUITEMID_CHILD_ARRAY apidl, REFIID riid,
  UINT * /* prgfInOut */, void **ppv)
{
  *ppv = NULL;
  HRESULT hr = E_NOINTERFACE;
  //...
  if (riid == IID_IDropTarget)
  {
    IUnknown *pFolder;
    this->QueryInterface(IID_IUnknown, (void**)&pFolder);
    MyDropTarget *pMyDropTarget =
      new MyDropTarget(pFolder, apidl[0]);
    pMyDropTarget->AddRef();
    hr = pMyDropTarget->QueryInterface(
      IID_IDropTarget, ppv);
    pMyDropTarget->Release();
  }
  //...
  return hr;
}
```

Для простоты методы IDropTarget::DragEnter, IDropTarget::DragOver, IDropTarget::DragLeave могут в любых случаях возвращать S_OK, при этом DragEnter и DragOver в выходной параметр DWORD *pdwEffect записывают значение DROPEFFECT_MOVE.

Основная обработка происходит в методе IDropTarget::Drop.

```
STDMETHOD(Drop)(LPDATAOBJECT pDataObj, DWORD dwKeyState,
  POINTL /*pt*/, LPDWORD pdwEffect)
{
  *pdwEffect = DROPEFFECT_NONE;
  IShellItemArray *psia;
  HRESULT hr = SHCreateShellItemArrayFromDataObject(
    pDataObj, IID_PPV_ARGS(&psia));
  IShellItem *psi;
  DWORD nFiles;
  std::list<TCHAR*> files;
  if (SUCCEEDED(hr))
  {
    hr = psia->GetCount(&nFiles);
    if (SUCCEEDED(hr))
    {
      for (DWORD i = 0; i < nFiles; ++i)
      {
```

```
        hr = psia->GetItemAt(i, &psi);
        if (SUCCEEDED(hr))
        {
          PIDLIST_ABSOLUTE pidl;
          hr = SHGetIDListFromObject(psi, &pidl);
          wchar_t *szFileDropped = new wchar_t[MAX_PATH];
          SHGetNameFromIDList(pidl,
            SIGDN_DESKTOPABSOLUTEEDITING, &szFileDropped);
          files.push_back(szFileDropped);
        }
      }
    }
  }
  IDropHandler *pDropHandler;
  this->m_pFolder->QueryInterface(IID_IDropHandler,
    (void**) &pDropHandler);
  pDropHandler->DoDrop(files, this->m_subfolder);
  while(!files.empty())
  {
    TCHAR *file = files.back();
    CoTaskMemFree(file);
    files.pop_back();
  }
  this->m_pFolder->Release();
  return S_OK;
}
```

Экземпляр класса реализации IDropTarget на вход конструктора принимает указатель на экземпляр класса директории типа IUnknown, и PIDL элемента-директории (если процесс перетаскивания завершился на нём). Это сделано, чтобы вызывать метод IDropHandler::DoDrop, где IDropHandler - это специальный интерфейс, который реализовывает класс директории.

```
#pragma once
#include <list>
#include <vector>
//{052979AD-F172-4318-A828-9A6AC20FC403}
static const GUID IID_IDropHandler = { 0x52979ad, 0xf172,
  0x4318, { 0xa8, 0x28, 0x9a, 0x6a, 0xc2, 0xf, 0xc4, 0x3 }};

class IDropHandler
{
public:
    virtual void DoDrop(std::list<TCHAR*> files,
      LPCITEMIDLIST subfolder) = 0;
};
```

Реализацию NSE со всеми вышеописанными элементами можно найти по ссылке https://github.com/DmitrySibert/virtual-folder.

Литература

1. Creating Shell Data Source Objects and Extending the Shell Namespace // msdn.microsoft.com : информ.-справочный портал. [Электронный ресурс]. URL: `https://msdn.microsoft.com/en-us/library/windows/desktop/ff521656(v=vs.85).aspx` (дата обращения: 04.03.2016).
2. Esposito D. Visual C++ Windows Shell Programming. Wrox Press Ltd., 1998.
3. Dunn M. The Complete Idiot's Guide to Writing Shell Extensions. Part I // www.CodeProject.com: информ.-справочный портал. 2006. 15 марта. [Электронный ресурс]. URL: `http://www.codeproject.com/Articles/441/The-Complete-Idiot-s-Guide-to-Writing-Shell-Extens` (дата обращения: 04.03.2016).
4. Microsoft Windows SDK for Windows 7 and .NET Framework 4 // msdn.microsoft.com : информ.-справочный портал. [Электронный ресурс]. URL: `https://www.microsoft.com/en-us/download/confirmation.aspx?id=8279` (дата обращения: 04.03.2016).

STEP BY STEP GUIDE TO WRITING SHELL NAMESPACE EXTENSION

D.S. Sibert
Student, e-mail: dimaaasik.s@gmail.com

Dostoevsky Omsk State University

Abstract. The article provides detailed guide for writing Shell Namespace Extension, including creating a blank application and a set of necessary file system functions.

Keywords: guide, windows, shell, namespace, extension.