Extending Windows Embedded Compact 7 Media Library with a Media Parser Plug-In

Windows Embedded Compact 7 Technical Article

Writers: Frankie Anderson, Olaf Trytten

Technical Reviewer: Michael Byrd

Published: March 2011

Applies To: Windows Embedded Compact 7

Abstract

Windows Embedded Compact 7 Media Library is, at its core, a database that contains information about media files on embedded devices. The information that Media Library contains, also called metadata, describes and supplements the content of the media files themselves. Media Library uses metadata to browse, organize, and manage media files, and to sync the media files on the embedded device with the desktop. You can extend Media Library to recognize, catalog, and manage a new file type by implementing a media parser plug-in for that file type. This article describes how to create a media parser to support wave sound (.wav) and MPEG-4 (.mp4) files, and steps through instructions, with sample code, to create the plug-in module.

Introduction

Windows Embedded Compact 7 Media Library is a tool that helps you organize and manage media files. Media Library contains all relevant information about your media files, commonly known as metadata, except for the actual content of the files. Windows Media Player can then process, or play, file contents. Using the Media Library database, media consumers can sort, search, filter, and view metadata as required. The functionality in Windows Embedded Compact 7 Media Library is incorporated into the user interface in Windows Media Player (http://go.microsoft.com/fwlink/?LinkID=197310).

Media Library obtains and maintains metadata for files in specific configured locations, called watch locations. Watch locations are folders on the desktop or on the embedded device. Media Library monitors watch locations and whenever a file at a watch location is created or modified, Media Library calls a media parser module to refresh the metadata for that file. You can extend Media Library to recognize and manage a new file type by implementing a media parser plug-in that will obtain the relevant metadata for the new type of file.

This article describes how to create a media parser plug-in with which Media Library can manage and maintain metadata for wave sound (.wav) and MPEG-4 (.mp4) media files, and steps through instructions, with sample code, to create the plug-in module.

Media Library Plug-In Overview

Media Library includes a Component Object Model (COM) application programming interface (API) that you can use to manage media from your applications. This API includes interfaces to add, modify, remove, browse, and search media files in the library. Windows Embedded Compact 7 Documentation describes the API in detail.

Media Library has an extensible architecture so you can add the ability to interact with custom data sources on embedded devices, extend the database schema, and add support for other file types. To add capabilities to Media Library, you can create two types of plug-in modules: metadata parser modules that implement custom metadata handling for additional file types, and data source modules that implement interactions with custom data sources. This article addresses creating the first type of plug-in, a custom metadata parser.

You must register each of your custom plug-ins as a COM server, and add it to the registry using Name-Value pairs under the Media Library application key. The Name to use for your plug-in is “Plugin#”, where “#” is a unique sequential integer; the Value is the GUID of your module. Media Library keeps the list of available media parser plug-ins in the registry key **`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MLib\MetadataParserPlugins`**. Media Library keeps the list of available data source plug-ins in the registry key **`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MLib\DataSourcePlugins`**.

Prerequisites for Developing Media Library Plug-Ins

The following are required for your plug-in development environment:

- Windows Embedded Compact 7 installed on your computer.

- Microsoft Visual Studio 2008 installed on your computer.
- A Platform Builder OS Design project based on Windows Embedded Compact 7, based on the Consumer Media Device - Portable Media Player design template, and which includes Media Formats and Media Library multimedia technologies.

Media Parser Plug-In

Media Library maintains metadata for some common media file types such as Windows Media Audio (.wma) files and Windows Media Video (.wmv) files; see Microsoft Support Technical Article ID: 316992 for information about supported file types. You extend Media Library to other file types by adding one or more media parser plug-in modules.

A media parser plug-in module, at minimum, must be:

- A registered COM server.
- Co-creatable.
- A Windows Embedded Compact 7 Dynamic-Link Library subproject of an appropriate Portable Media Player OS design.

The plug-in module also must implement the methods of the Media Library **IMLMediaParser** interface.

Media Library service calls the **IMLMediaParser::Init** method on all plug-in modules on start up. The Media Library media parser manager determines which media parser plug-in to call for each request, and then internally calls the **IMLMediaParser::GetKnownFileExtensions** method to get the file extensions supported by a particular media parser plug-in. Media Library calls **IMLMediaParser::GetEntityType** to determine the entity type of a particular media file, and then calls **IMLMediaParser::ExtractMetadata** to obtain the metadata for the media file.

Sample Media Parser Functional Design

This article describes the sample plug-in, SampleMediaParser, which identifies a file as a music or video entity based on its file name extension. Taking advantage of registry search efficiency, this design uses registry entries to map file name extensions to entity types. The design uses name/data entries stored in the registry key

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MLib\EntityExtensions. The sample plug-in implements **GetEntityType** to identify a file with extension .wav as a music entity and a file with extension .mp4 as a video entity. It implements **ExtractMetadata** to collect metadata for the file and return it to Media Library as one of these entities. Refer to the listings for SampleMediaParser.h and SampleMediaParser.cpp for details on this implementation.

SampleMediaParser supports only file system–related metadata. It collects this metadata without inspecting the contents of the file.

In the **ExtractMetadata** method, a registry query uses the file extension as a value name and returns the entity type as value data of type REG_SZ. This string matches one of the enumerated Media Library **EntityId** values; see mlibdll_id.h for a complete list of the **EntityId** values. SampleMediaParser uses the entity to compose a metadata collection with the correct **PropertyId** values; see mlibdll.id.h for a complete list of the **PropertyId** values.

If an entity extension is not found among the **EntityExtensions** values in the registry, the sample plug-in returns **S_FALSE**. This result indicates that the media parser does not support the media file, and Media Library moves to the next media parser until it finds support for the media file (or all media parsers have been tried). If Media Library does not find a media parser for a specific media file, Media Library does not add the information for the file to the database and does not manage the file.

A custom media parser that identifies file types that are not supported by all other parsers has a number of advantages. One of these is that Media Library clients, such as the Windows Media Player user interface, have a better view of a watched folder and can better determine how to present media. Also, you can make an entity assignment for new file types by adding a registry value instead of adding code. Registering unsupported file types improves performance because Media Library can immediately identify unsupported media in the entity tables without executing additional parsing logic.

Build and Run the Sample Media Parser

To build and run the sample media parser, complete the following steps.

Step 1: Create the Dynamic-Link Library Subproject in Your OS Design

Your OS design uses the Consumer Media Device - Portable Media Player design template and includes Media Formats and Media Library multimedia technologies. Open your OS design in Visual Studio 2008 (in the example in this article, the OS design is named OSDesignMediaPlugin) to create a subproject for the sample media parser.

1. In Solution Explorer, right-click **Subprojects**, and then click **Add New Subproject**.
2. Click the **WCE Dynamic-Link Library** template, then type the sample media parser name (this article uses the name MediaParserPlugin) and location (for example, C:\WINCE700\OSDesigns\OSDesignMediaPlugin\OSDesignMediaPlugin\MediaParserPlugin), and then click **Next**.
3. Click **A simple DLL subproject**, and then click **Finish** to create the subproject.

Step 2: Configure the Subproject Properties for Use with COM and Media Library

You must configure your subproject to support adding COM objects and code, and to support Media Library use.

1. In Solution Explorer, right-click the subproject name (MediaParserPlugin), and then click **Properties** to open the properties editor for the subproject.
2. On the **General** tab, click **Release Type**, and then click **OAK** in the drop-down list of release types.
3. On the **General** tab, click **Custom Variables**, and then click **...** to open the **Custom Variables** dialog box. Add the five custom variables shown in the following table to the subproject.

Variable name	Variable value
WINCEATL	1
_DCOMUUID	$(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\uuid.lib
_OLEAUT32	$(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\oleaut32 .lib
_OLE32	$(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\ole32.lib
_COMMCTRL	$(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\commctr l.lib

4. On the **C/C++** tab, click **Include Directories**, and then type the two semicolon-separated include directory locations: **`$(_PUBLICROOT)\ie7\sdk\inc;$(_PUBLICROOT)\mediaapps\sdk\inc.`**
5. On the **Link** tab, click **Additional Libraries**, which already contain `$(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\coredll.lib`, and type the space-separated list of additional libraries, using four of the custom variable names defined in step 3: **`$(_OLE32) $(_COMMCTRL) $(_DCOMUUID) $(_OLEAUT32)`**.

Step 3: Modify Subproject Parameter and Source Files for Use with COM and Media Library

You must modify several of the automatically generated subproject files to support Active Template Library (ATL) and COM objects and the Media Library.

1. To properly register your plug-in, using your plug-in name and CLSID/GUID instead of “Plugin5” and “{FD61F5AC-4745-4994-81A1-13E3479138A3}”, replace the contents of the subproject registration entries file (MediaParserPlugin.reg) with the following code.

```
; @XIPREGION IF PLATFORM_FILES_MISC


[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MLib\MetadataParserPlugins]
"Plugin5"="{FD61F5AC-4745-4994-81A1-13E3479138A3}"

[HKEY_CLASSES_ROOT\CLSID\{FD61F5AC-4745-4994-81A1-13E3479138A3}]
@="IMLMediaParser Media Parser Plugin Class"

[HKEY_CLASSES_ROOT\CLSID\{FD61F5AC-4745-4994-81A1-
13E3479138A3}\InprocServer32]
@="MediaParserPlugin.dll"
"ThreadingModel"="Both"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MLib\EntityExtensions]
    "WAV"="ml_Entity_music"
    "MP4"="ml_Entity_video"
; @XIPREGION ENDIF PLATFORM_FILES_MISC
```

2. You need to export the functions **DllCanUnloadNow** and **DllGetClassObject** to make your DLL a functioning COM server. Replace the contents of the module-definition file (MediaParserPlugin.def) with the following code.

Important The following code example may not contain sufficient error handling for your device. Do not include the following code in a shipping device without extensive scenario testing.

```
;
; Copyright (c) Microsoft Corporation.  All rights reserved.
;


; Use of this source code is subject to the terms of the Microsoft
; premium shared source license agreement under which you licensed
; this source code. If you did not accept the terms of the license
; agreement, you are not authorized to use this source code.
; For the terms of the license, see the license agreement
; signed by you and Microsoft.
; THE SOURCE CODE IS PROVIDED "AS IS", WITH NO WARRANTIES OR
INDEMNITIES.
;
; KnownExtDll.def : Declares the module parameters.

LIBRARY      "MediaParserPlugin.DLL"

EXPORTS
        DllCanUnloadNow        PRIVATE
        DllGetClassObject      PRIVATE
```

3. Replace the contents of the subproject standard include file (StdAfx.h) with the following contents.

Important The following code example may not contain sufficient error handling for your device. Do not include the following code in a shipping device without extensive scenario testing.

```
//
// Copyright (c) Microsoft Corporation.  All rights reserved.
//
//
// Use of this sample source code is subject to the terms of the
Microsoft
// license agreement under which you licensed this sample source
code. If
// you did not accept the terms of the license agreement, you are
not
// authorized to use this sample source code. For the terms of the
license,
// please see the license agreement between you and Microsoft or, if
applicable,
// see the LICENSE.RTF on your install media or the root of your
tools installation.
// THE SAMPLE SOURCE CODE IS PROVIDED "AS IS", WITH NO WARRANTIES OR
```

```
INDEMNITIES.
//
/***********************************************************************
******


  Disclaimer:

    This code and information is provided "as is" without warranty
of
    any kind, either expressed or implied, including but not limited
to
    the implied warranties of merchantability and/or fitness for a
    particular purpose.

*/
//
// stdafx.h : include file for standard system include files,
//      or project specific include files that are used frequently,
//      but are changed infrequently

#pragma once

// Insert your headers here
#define WIN32_LEAN_AND_MEAN
// Exclude rarely-used stuff from Windows headers

#include <windows.h>

#ifndef STRICT
#define STRICT
#endif

#define _ATL_FREE_THREADED

#define _ATL_STATIC_REGISTRY

// including crtdbg.h here because atlbase.h defines _ASSERTE and
// dmoimpl.h will later include crtdbg.h which also defines
// _ASSERTE without using an ifndef wrapper.
//#include <crtdbg.h>

#include <atlbase.h>
#include <atlcom.h>

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#define Chk(val) { hr = (val); if (FAILED(hr)) goto Cleanup; }
#define ChkBool(val, x) { if (!(val)) { hr = x; goto Cleanup; } }

#define ENTITYLABELMUSIC    L"ml_Entity_music"
#define ENTITYLABELVIDEO    L"ml_Entity_video"
#define ENTITYLABELPHOTO    L"ml_Entity_photo"
```

```
#define ENTITYLABELGENERIC  L"ml_Entity_generic"
#define ENTITYLABELPLAYLIST L"ml_Entity_playlist"

#include "mlibdll.h"
#include "mlibdll_id.h"
#include "mlibdll_plugin.h"
```

4. Replace the contents of the subproject source file (MediaParserPlugin.cpp) with the following listing.

Important The following code example may not contain sufficient error handling for your device. Do not include the following code in a shipping device without extensive scenario testing.

```
//
// Copyright (c) Microsoft Corporation.  All rights reserved.
//
//
// Use of this sample source code is subject to the terms of the
Microsoft
// license agreement under which you licensed this sample source
code. If
// you did not accept the terms of the license agreement, you are
not
// authorized to use this sample source code. For the terms of the
license,
// please see the license agreement between you and Microsoft or, if
applicable,
// see the LICENSE.RTF on your install media or the root of your
tools installation.
// THE SAMPLE SOURCE CODE IS PROVIDED "AS IS", WITH NO WARRANTIES OR
INDEMNITIES.
//
//====================================================================
========;
//
//  THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY
OF ANY
//  KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO
THE
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
PARTICULAR
//  PURPOSE.
//
//
//--------------------------------------------------------------------
--------;
//
//                          Sample Media Parser Plug-In
//
// MediaParserPlugin.cpp - Implementation of DLL exports.
//
//--------------------------------------------------------------------
------------
```

```
#include "stdafx.h"

class CSampleMediaParserAtlModule : public CAtlDllModuleT<
CSampleMediaParserAtlModule > { };

CSampleMediaParserAtlModule _AtlModule;


BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    return _AtlModule.DllMain(ul_reason_for_call, lpReserved);
}

// Used to determine whether the DLL can be unloaded by OLE
STDAPI DllCanUnloadNow(void)
{
    return _AtlModule.DllCanUnloadNow();
}

// Returns a class factory to create an object of the requested type
STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, LPVOID* ppv)
{
    return _AtlModule.DllGetClassObject(rclsid, riid, ppv);

}
```

Step 4: Copy the Sample Implementation Source Code to Files on Your Computer

Using Windows Explorer on your computer, browse to the subproject directory (in our example, C:\WINCE700\OSDesigns\OSDesignMediaPlugin\OSDesignMediaPlugin\MediaParserPlugin) and then create the file SampleMediaParser.h with the following content.

Important The following code example may not contain sufficient error handling for your device. Do not include the following code in a shipping device without extensive scenario testing.

```
//// Copyright (c) Microsoft Corporation.  All rights reserved.
//
//
// Use of this sample source code is subject to the terms of the Microsoft
// license agreement under which you licensed this sample source code. If
// you did not accept the terms of the license agreement, you are not
// authorized to use this sample source code. For the terms of the
license,
// please see the license agreement between you and Microsoft or, if
applicable,
// see the LICENSE.RTF on your install media or the root of your tools
installation.
// THE SAMPLE SOURCE CODE IS PROVIDED "AS IS", WITH NO WARRANTIES OR
INDEMNITIES.
//
//===========================================================================
==;
//
//  THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
//  KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR
//  PURPOSE.
//
//
//---------------------------------------------------------------------------
--;
//
//                          Sample Media Parser Plug-In
//
// SampleMediaParser.h - Declaration of ATL CoCreate'able class
CSampleMediaParser.
//
//---------------------------------------------------------------------------
------

DEFINE_GUID(CLSID_SampleMediaParser, 0x65C0781B, 0xDA05, 0x465B, 0xB5,
0xC4, 0xE9, 0x0B, 0x43, 0xAF, 0xDD, 0x35);

/////////////////////////////////////////////////////////////////////////////
///
// CSampleMediaParser
class ATL_NO_VTABLE CSampleMediaParser :
    public CComObjectRoot,
    public CComCoClass<CSampleMediaParser, &CLSID_SampleMediaParser>,
    public IMLMediaParser
{
public:
    CSampleMediaParser(void);
    virtual ~CSampleMediaParser(void);

DECLARE_PROTECT_FINAL_CONSTRUCT()

DECLARE_NOT_AGGREGATABLE(CSampleMediaParser)

DECLARE_NO_REGISTRY()

BEGIN_COM_MAP(CSampleMediaParser)
    COM_INTERFACE_ENTRY(IMLMediaParser)
END_COM_MAP()

    // IMLMediaParser interface
    STDMETHOD(Init)();
    STDMETHOD(GetKnownFileExtensions)(ULONG *pcFileExtensions, BSTR
**prgFileExtensions);
    STDMETHOD(GetEntityType)(MLDSItem file, ULONG *pEntityType);
    STDMETHOD(ExtractMetadata)(MLDSItem file, IMLPropertySet
*pPropertySetWanted, IMLPropertySet *pFileMetadata);

private:
    HRESULT ConvertLabelToEntityId(LPCWSTR szLabel, DWORD dwLabelCC,
EntityId *pEntityId);
    HRESULT GetFileExtension(BSTR bstrURL, LPCWSTR wcsExtension, size_t
cchExtension);
    HRESULT GetURLExtension(BSTR bFilename, LPWSTR wcsExtension, const
USHORT uExtensionMaxChars);
    HRESULT GetKnownEntity(LPCWSTR wcsExtension, EntityId *pKnownEntity);
    HRESULT LoadEntityExtensions(void);
    HRESULT SetFolderFileNames(BSTR fileURL, PropertyId propidFileURL,
PropertyId propidFileName, PropertyId propidFolderName, IMLPropertySet
*pPropertySetWanted, IMLPropertySet *pFileMetadata);
    HRESULT SetFileSizeAndTime(ULONGLONG llFileSize, const FILETIME
&fileTime, PropertyId mlidFileSize, PropertyId mlidFileTime,
IMLPropertySet *pPropertySetWanted, IMLPropertySet *pFileMetadata);
    HRESULT ConvertFileTimeToMLFileTime(FILETIME const *pftValue,
MLDateTime *pmlftDest);

    // Member data
    BSTR *m_rgEntityExtensions;
    DWORD m_dwEntityExtensionCount;
};

OBJECT_ENTRY_AUTO(CLSID_SampleMediaParser, CSampleMediaParser)
```

In the same directory location (in our example, C:\WINCE700\OSDesigns\OSDesignMediaPlugin\OSDesignMediaPlugin\MediaParserPlugin), create the file SampleMediaParser.cpp with the following content.

Important The following code example may not contain sufficient error handling for your device. Do not include the following code in a shipping device without extensive scenario testing.

```
//
// Copyright (c) Microsoft Corporation.  All rights reserved.
//
//
// Use of this source code is subject to the terms of the Microsoft
// premium shared source license agreement under which you licensed
// this source code. If you did not accept the terms of the license
// agreement, you are not authorized to use this source code.
// For the terms of the license, please see the license agreement
// signed by you and Microsoft.
// THE SOURCE CODE IS PROVIDED "AS IS", WITH NO WARRANTIES OR INDEMNITIES.
//
//-----------------------------------------------------------------------
------
//
//                          Sample Media Parser Plug-In
//
// SampleMediaParser.cpp  - Implementation of ATL CoCreate'able class
```

```
CSampleMediaParser.
//                 Complies with IMLMediaParser.
//
//-------------------------------------------------------------------------
------

#include "stdafx.h"
#include "SampleMediaParser.h"

//=========================================================================
====
//                              CSampleMediaParser
//=========================================================================
====

//=========================================================================
====
//  Initialize static constant members
//=========================================================================
====
static LPCWSTR      wcsEntityExtensionsRegKey =
L"Software\\Microsoft\\MLib\\EntityExtensions";
static HKEY         hkEntityExtensionsRootKey = HKEY_LOCAL_MACHINE;
static const USHORT cchMaxExtension = MAX_PATH;  // Support long
filenames.

//=========================================================================
====
//
// CSampleMediaParser constructor
//
//=========================================================================
====
CSampleMediaParser::CSampleMediaParser()
: m_rgEntityExtensions(NULL),
  m_dwEntityExtensionCount(0)
{
}

//=========================================================================
====
//
// CSampleMediaParser destructor
//
// Description:
//     Clean up dynamically allocated memory.
//
//=========================================================================
====
CSampleMediaParser::~CSampleMediaParser(void)
{
    if (NULL != m_rgEntityExtensions)
    {
        for (ULONG iExtension = 0; iExtension < m_dwEntityExtensionCount;
iExtension++)
```

```
        {
            if (m_rgEntityExtensions[iExtension] != NULL)
            {
                SysFreeString(m_rgEntityExtensions[iExtension]);
            }
        }
        delete [] m_rgEntityExtensions;
    }
}

//===========================================================================
====
//
// Init
//
// Expected Result Code:
//     S_OK : success
//
// Description:
//     Media Library service calls this method upon start up.
//
//===========================================================================
====
HRESULT
CSampleMediaParser::Init(void)
{
    return LoadEntityExtensions();
}

//===========================================================================
====
//
// LoadEntityExtensions
//
// Possible Result Codes:
//     S_OK      : call succeeded.
//
// Description:
//     Gathers an array of extensions from the Media Library application
//     registry key.  Only the labels successfully interpreted as
belonging
//     to a Media Library entity id will be supported.
//
//===========================================================================
====
HRESULT
CSampleMediaParser::LoadEntityExtensions(void)
{
    HRESULT hr = S_OK;
    CRegKey reg;
    LONG    lRet = 0L;
    HKEY    hKey = NULL;
    DWORD   dwIndex = 0;
    DWORD   cValues = 0;
    DWORD   dwLocalExtCount = 0;
    BSTR   *prgLocalExtensions = NULL;
    ASSERT(m_rgEntityExtensions == NULL);

    lRet = reg.Open(hkEntityExtensionsRootKey, wcsEntityExtensionsRegKey,
KEY_READ);
    ChkBool((lRet == ERROR_SUCCESS), S_OK);  // TODO: Explain

    hKey = (HKEY)reg;  // Use a copy of the HANDLE for Reg functions not
implemented in CRegKey.
    ChkBool(hKey != NULL, E_FAIL);

    // Get a count of values.
    lRet = RegQueryInfoKey(
               hKey,        // key handle
               NULL,
               NULL,
               NULL,
               NULL,
               NULL,
               NULL,
               &cValues,    // number of values for this key
               NULL,
               NULL,
               NULL,
               NULL);

    ChkBool((lRet == ERROR_SUCCESS), E_FAIL);
    ChkBool((cValues > 0), S_OK);   // Finding no supported extensions is
OK, but don't continue.

    prgLocalExtensions = new BSTR[cValues];
    memset((void *)prgLocalExtensions, 0, sizeof(BSTR)*cValues);

    for (dwIndex = 0; dwIndex < cValues && lRet != ERROR_NO_MORE_ITEMS;
dwIndex++)
    {
        DWORD dwType = 0;
        WCHAR szValueName[cchMaxExtension];
        DWORD cchValueName = 0;

        szValueName[0] = L'\0';
        cchValueName = cchMaxExtension;

        //Capacity going in must include NULL termination character,
        // but count coming out excludes NULL termination character.
        lRet = RegEnumValue(hKey, dwIndex, szValueName, &cchValueName,
NULL, &dwType, NULL, NULL);
        switch (lRet)
        {
        case ERROR_SUCCESS:
            if (dwType == REG_SZ && cchValueName <= cchMaxExtension)
            {
                DWORD dwEntity = 0;
                WCHAR szValueData[cchMaxExtension];
                ULONG nCch = _countof(szValueData);
                EntityId entityId = ml_Entity_NONE;

                if (ERROR_SUCCESS == reg.QueryStringValue(szValueName,
szValueData, &nCch))
                {
                    // Ignore bad labels.  They will simply be
unsupported.
                    if (S_OK == ConvertLabelToEntityId(szValueData, nCch,
&entityId))
                    {
                        prgLocalExtensions[dwLocalExtCount] =
SysAllocString((LPWSTR)szValueName);
                        dwLocalExtCount++;
                    }
                }
            }
            break;
        case ERROR_NO_MORE_ITEMS:
            // Done.  Let the "for" condition stop the loop.
            break;
        case ERROR_MORE_DATA:
            // The buffer recieving the value name is too small.
            // Overly long extensions are disallowed but are not a
blocking problem.
            break;
        default:
            ChkBool(FALSE, HRESULT_FROM_WIN32(lRet));
            break;
        }
    }

    hKey = NULL;  // Remove temptation to close this copy of the key
handle.

    // Transfer valid extensions from local to member storage.
    if (dwLocalExtCount == cValues)
    {
        // All elements are valid.  Just use the array as-is.
        m_rgEntityExtensions = prgLocalExtensions;
    }
    else
    {
        // Allocate member array for only valid elements.
        m_rgEntityExtensions = new BSTR[dwLocalExtCount];
        for (dwIndex = 0; dwIndex < dwLocalExtCount; dwIndex++)
        {
            m_rgEntityExtensions[dwIndex] = prgLocalExtensions[dwIndex];
        }

        // Free local container array.
        delete [] prgLocalExtensions;
    }

    m_dwEntityExtensionCount = dwLocalExtCount;
```

```
Cleanup:
    return hr;
}

//==========================================================================
====
//
// GetKnownFileExtensions
//
// Parameters:
// out ULONG *pcFileExtensions   : Number of file extensions
// out BSTR **pprgFileExtensions : Supported file extensions (caller must
free)
//
// Possible Result Codes:
//     S_OK          : Success
//     E_POINTER     : One or more parameters are NULL.
//     E_OUTOFMEMORY : An out of memory error occurred.
//     E_FAIL        : The method was not successful
//
// Description:
//     The media parser manager
IMLMediaParserManager::GetKnownFileExtensions
//     method determines which media parser object to call for each
request
//     and internally calls this method to retrieve the file extensions
//     supported by this media parser.
//
//     The caller frees elements of this array using SysFreeString and
frees
//     the array itself with CoTaskMemFree.
//
//==========================================================================
====
HRESULT
CSampleMediaParser::GetKnownFileExtensions(ULONG *pcFileExtensions, BSTR
**pprgFileExtensions)
{
    HRESULT hr = S_OK;
    BSTR   *prgLocalExt = NULL;  // Dynamic memory held local until ready
to be returned with S_OK.
    DWORD   dwIndex = 0;

    ChkBool((pcFileExtensions != NULL), E_POINTER);
    ChkBool((pprgFileExtensions != NULL), E_POINTER);

    // Allocate containing array using memory shared with this COM
object's client.
    prgLocalExt = (BSTR*)CoTaskMemAlloc(sizeof(BSTR) *
m_dwEntityExtensionCount);
    ChkBool((prgLocalExt != NULL), E_OUTOFMEMORY);

    // Initialize allocated memory.  Use NULL to detect allocated elements
during emergency cleanup.
    memset((void *)prgLocalExt, 0, sizeof(BSTR) *
m_dwEntityExtensionCount);

    // Fill allocated array with copies of supported extensions.
    for (dwIndex = 0; dwIndex < m_dwEntityExtensionCount; dwIndex++)
    {
        ASSERT(m_rgEntityExtensions[dwIndex] != NULL);

        // Give caller a copy of each element.
        prgLocalExt[dwIndex] =
SysAllocString(m_rgEntityExtensions[dwIndex]);
        ChkBool((prgLocalExt[dwIndex] != NULL), E_OUTOFMEMORY);
    }

    // Safe to make assignments to output arguments.
    *pcFileExtensions = m_dwEntityExtensionCount;
    *pprgFileExtensions = prgLocalExt;

    // Don't clean up memory passed to caller.
    prgLocalExt = NULL;

Cleanup:
    if (prgLocalExt)
    {
        ASSERT(FAILED(hr)); // If allocated memory is still local, there
will be an error.

        // Free local allocated memory.
        for (dwIndex = 0; dwIndex < m_dwEntityExtensionCount; dwIndex++)
        {
            if (prgLocalExt[dwIndex] != NULL)
            {
                SysFreeString(prgLocalExt[dwIndex]);
            }
        }
        CoTaskMemFree(prgLocalExt);
    }
    return hr;
}

//==========================================================================
====
//
// GetEntityType
//
// Parameters:
//     MLDSItem file    : File from which to determine the entity type.
// out ULONG *pEntityId : EntityId belonging to the file.
//
// Possible Result Codes:
//     S_OK      : success
//     S_FALSE   : unknown file extension
//     E_POINTER : parameters invalid
//
// Description:
//     Searches for the extension found in file.URL in the known entity
//     list for a match. Does not read the file, just checks the
extension.
//
//==========================================================================
====

HRESULT
CSampleMediaParser::GetEntityType(MLDSItem file, ULONG *pEntityType)
{
    HRESULT hr = S_OK;
    WCHAR   wsExtension[cchMaxExtension];

    ChkBool((pEntityType != NULL), E_POINTER);

    Chk(GetURLExtension(file.URL, wsExtension, cchMaxExtension));
    if (hr == S_OK)
    {
        Chk(GetKnownEntity(wsExtension, (EntityId *)pEntityType));
    }

Cleanup:
    return hr;
}

//==========================================================================
====
//
// ExtractMetadata
//
// Parameters:
//     MLDSItem file                      : File to parse.
//     IMLPropertySet *pPropertySetWanted : Metadata requested by caller.
//     IMLPropertySet *pFileMetadata      : Metadata collection to return
to caller.
//
// Possible Result Codes:
//     S_OK      : call succeeded.
//     E_POINTER : a property collection is invalid.
//     E_FAIL    : file system or other unknown failure
//
// Description:
//     Satisfies IMLMediaParser interface requirements.
//     Collects basic file system metadata from the file if it is of a
known type.
//     Passes metadata on to StoreMetadata consumer, if provided, before
returning.
//
//==========================================================================
====
HRESULT
CSampleMediaParser::ExtractMetadata(MLDSItem file, IMLPropertySet
*pPropertySetWanted, IMLPropertySet *pFileMetadata)
{
    HRESULT hr = S_OK;
    PropertyId propidFileURL = mlid_NONE;
    PropertyId propidFileName = mlid_NONE;
    PropertyId propidFolderName = mlid_NONE;
    PropertyId propidFileSize = mlid_NONE;
    PropertyId propidFileTime = mlid_NONE;
    ULONG      ulEntityType = (ULONG)ml_Entity_NONE;

    ChkBool((pFileMetadata != NULL), E_POINTER);

    Chk(GetEntityType(file, &ulEntityType));

    switch (ulEntityType)
    {
        case ml_Entity_video:
            propidFileURL = mlid_video_fileURL;
            propidFileName = mlid_video_fileName;
            propidFolderName = mlid_video_folderName;
            propidFileSize = mlid_video_fileSize;
            propidFileTime = mlid_video_fileTime;
            break;

        case ml_Entity_music:
            propidFileURL = mlid_music_fileURL;
            propidFileName = mlid_music_fileName;
            propidFolderName = mlid_music_folderName;
            propidFileSize = mlid_music_fileSize;
            propidFileTime = mlid_music_fileTime;
            break;

        case ml_Entity_photo:
            propidFileURL = mlid_photo_fileURL;
            propidFileName = mlid_photo_fileName;
            propidFolderName = mlid_photo_folderName;
            propidFileSize = mlid_photo_fileSize;
            propidFileTime = mlid_photo_fileTime;
            break;

        case ml_Entity_playlist:
            propidFileURL = mlid_playlist_fileURL;
            propidFileName = mlid_playlist_fileName;
            propidFolderName = mlid_playlist_folderName;
            propidFileSize = mlid_playlist_fileSize;
            propidFileTime = mlid_playlist_fileTime;
            break;

        case ml_Entity_generic:
            propidFileURL = mlid_generic_fileURL;
            propidFileName = mlid_generic_fileName;
            propidFolderName = mlid_generic_folderName;
            propidFileSize = mlid_generic_fileSize;
            propidFileTime = mlid_generic_fileTime;
            break;

        default:
            ChkBool(FALSE, E_FAIL);
            break;
```

©2011 Microsoft

```
    }

    Chk(SetFolderFileNames(file.URL, propidFileURL, propidFileName,
propidFolderName, pPropertySetWanted, pFileMetadata));
    Chk(SetFileSizeAndTime(file.cbFileSize, file.ftTimeStamp,
propidFileSize, propidFileTime, pPropertySetWanted, pFileMetadata));

Cleanup:
    return hr;
}

//==========================================================================
====
//
// GetKnownEntity
//
// Parameters:
//     LPCWSTR wcsExtension   : The file name extension to query.
// out EntityId *pKnownEntity : The entity of the extension.
//
// Possible Result Codes:
//     S_OK   : call succeeded.
//     E_FAIL : file system or other unknown failure
//
// Description:
//     References the registry for the entity id of a known extension.
//
//==========================================================================
====
HRESULT
CSampleMediaParser::GetKnownEntity(LPCWSTR wcsExtension, EntityId
*pKnownEntity)
{
    HRESULT hr = S_OK;
    CRegKey reg;
    LONG    lRet = 0L;
    WCHAR   szValueData[MAX_PATH];
    DWORD   dwValueDataCC = _countof(szValueData);

    lRet = reg.Open(HKEY_LOCAL_MACHINE, wcsEntityExtensionsRegKey,
KEY_READ);
    ChkBool((lRet == ERROR_SUCCESS), E_FAIL);

    lRet = reg.QueryStringValue(wcsExtension, szValueData,
&dwValueDataCC);
    ChkBool((lRet != ERROR_FILE_NOT_FOUND), S_FALSE);
    ChkBool((lRet == ERROR_SUCCESS), E_FAIL);

    // Interpret the entity label.  The character count includes z-term.
Subtract it for string comparison.
    hr = ConvertLabelToEntityId(szValueData, dwValueDataCC, pKnownEntity);

Cleanup:
    return hr;
}
```

```
//======================================================================
====
//
// GetURLExtension
//
// Parameters:
//       BSTR bFilename            : media library detected file name.
//   out LPWSTR wcsExtension       : In: pointer to string buffer. Out:
extension of bFilename
// const USHORT uExtensionMaxChars : Capacity of wcsExtension in
characters.
//
// Possible Result Codes:
//     S_OK         : call succeeded.
//     E_POINTER    : bFilename or wcsExtension is NULL
//     E_UNEXPECTED : Output Buffer has no capacity.
//     E_FAIL       : file system or other unknown failure
//
// Description:
//     Search the file name for the extension part.
//     Presumes name is not that of a directory because Media Library does
//     not pass directories.
//
//======================================================================
====
HRESULT
CSampleMediaParser::GetURLExtension(BSTR bFilename, LPWSTR wcsExtension,
const USHORT uExtensionMaxChars)
{
    HRESULT hr = S_OK;
    UINT    cchb = 0;
    LPCWSTR pFilenamePart = NULL;
    LPCWSTR pExtensionPart = NULL;

    ChkBool((bFilename != NULL), E_POINTER);
    ChkBool((wcsExtension != NULL), E_POINTER);
    ChkBool((uExtensionMaxChars > 0), E_UNEXPECTED);

    // Find filename part after path.
    pFilenamePart = wcsrchr(bFilename, L'\\');
    if (pFilenamePart == NULL)
    {
        // No path delimiter found; file name has no path part.
        pFilenamePart = bFilename;
    }

    // Find extension part.
    pExtensionPart = wcsrchr(pFilenamePart, L'.');
    if (pExtensionPart == NULL)
    {
        // No filename extension was found, which is OK.  Return empty
string.
        ChkBool((uExtensionMaxChars > 0), E_FAIL);
        wcsExtension[0] = L'\0';
    }
    else
    {
        // Pointer now indicates L'.' and will need to indicate extension.
        pExtensionPart++;
        Chk(StringCchCopy(wcsExtension, uExtensionMaxChars,
pExtensionPart));
    }

Cleanup:
    return hr;
}

//=========================================================================
====
//
// ConvertLabelToEntityId
//
// Parameters:
//     LPCWSTR   szLabel   : Entity label which begins with ml_Entity_
//     DWORD     dwLabelCC : Character count of label (including z-term).
// out EntityId *pEntityId : Schema defined entity id.
//
// Possible Result Codes:
//     S_OK      : call succeeded.
//     E_POINTER : Bad argument
//     E_FAIL    : Couldn't interpret the label.
//
// Description:
//     Interprets label as entity id.
//     Since string lengths have already been gathered, this method takes
//     advantage of these lengths to optimize the label search.
//
//=========================================================================
====
HRESULT
CSampleMediaParser::ConvertLabelToEntityId(LPCWSTR szLabel, DWORD
dwLabelCC, EntityId *pEntityId)
{
    HRESULT hr = S_OK;

    ChkBool((szLabel != NULL), E_POINTER);
    ChkBool((pEntityId != NULL), E_POINTER);

    if (0 == wcsncmp(szLabel, ENTITYLABELMUSIC, dwLabelCC))
    {
        *pEntityId = ml_Entity_music;
    } else if (0 == wcsncmp(szLabel, ENTITYLABELVIDEO, dwLabelCC))
    {
        *pEntityId = ml_Entity_video;
    } else if (0 == wcsncmp(szLabel, ENTITYLABELPHOTO, dwLabelCC))
    {
        *pEntityId = ml_Entity_photo;
    } else if (0 == wcsncmp(szLabel, ENTITYLABELGENERIC, dwLabelCC))
    {
        *pEntityId = ml_Entity_generic;
    } else if (0 == wcsncmp(szLabel, ENTITYLABELPLAYLIST, dwLabelCC))
    {
        *pEntityId = ml_Entity_playlist;
    } else
    {
        hr = S_FALSE;
    }

Cleanup:
    return hr;
}

//==========================================================================
====
//
// SetFolderFileNames
//
// Parameters:
//     BSTR            fileURL            : file URL with the parts to
extract.
//     PropertyId      propidFileURL      : File URL property id specific
to the entity.
//     PropertyId      propidFileName     : File name property id specific
to the entity.
//     PropertyId      propidFolderName   : Folder name property id
specific to the entity.
//     IMLPropertySet *pPropertySetWanted : Requested properties.  If
NULL, store all.
// out IMLPropertySet *pFileMetadata      : PropertySet to receive
properties.
//
// Possible Result Codes:
//     S_OK          : call succeeded.
//     E_FAIL        : Couldn't interpret the URL.
//     E_INVALIDARG  : An invalid propid value.
//     E_OUTOFMEMORY : Unable to dynamically allocate memory.
//
// Description:
//     Extracts the parts of the fileURL and stores them in the property
set as requested.
//
//==========================================================================
====
HRESULT
CSampleMediaParser::SetFolderFileNames(
    BSTR fileURL,
    PropertyId propidFileURL,
    PropertyId propidFileName,
    PropertyId propidFolderName,
    IMLPropertySet *pPropertySetWanted,
    IMLPropertySet *pFileMetadata)
{
    HRESULT hr = S_OK;
    BSTR   bstrFilename(NULL);
    BSTR   bstrFoldername(NULL);
    WCHAR  wcsParts[MAX_PATH];
    LPWSTR lpFilePart = NULL;
    LPWSTR lpFolderPart = NULL;
    DWORD  cchParts = 0L;
    DWORD  cchFilename = 0L;
    DWORD  cchPath = 0L;
    DWORD  dwAttributes = GetFileAttributes(fileURL);

    Chk(((propidFileURL > mlid_NONE && propidFileURL < mlid_MAX) ? S_OK :
E_INVALIDARG));
    Chk(((propidFileName > mlid_NONE && propidFileName < mlid_MAX) ? S_OK
: E_INVALIDARG));
    Chk(((propidFolderName > mlid_NONE && propidFolderName < mlid_MAX) ?
S_OK : E_INVALIDARG));

    ChkBool((dwAttributes != INVALID_FILE_ATTRIBUTES), E_FAIL);

    // Can't accept hidden files or directories.
    ChkBool((dwAttributes & (FILE_ATTRIBUTE_HIDDEN |
FILE_ATTRIBUTE_DIRECTORY)) == 0, E_FAIL);

    cchParts = GetFullPathName(fileURL, MAX_PATH, wcsParts, &lpFilePart);
    ChkBool((cchParts != 0), E_FAIL);
    cchFilename = cchParts - (lpFilePart - wcsParts);
    cchPath = lpFilePart - wcsParts;

    bstrFilename = SysAllocStringLen(lpFilePart, cchFilename);
    ChkBool((bstrFilename != NULL), E_OUTOFMEMORY);

    lpFolderPart = &(wcsParts[cchPath - 1]);

    if (*lpFolderPart == L'\\' || *lpFolderPart == L'/')
    {
        UINT cchFolderName = 0;

        *lpFolderPart = L'\0';
        lpFolderPart--;

        while(lpFolderPart > wcsParts)
        {
            if (*lpFolderPart == L'\\' || *lpFolderPart == L'/')
            {
                lpFolderPart++;
                break;
            }
            else
            {
                lpFolderPart--;
            }
        }

        cchFolderName = cchPath - (lpFolderPart - wcsParts) - 1;
        bstrFoldername = SysAllocStringLen(lpFolderPart, cchFolderName);
    }

    if (pPropertySetWanted == NULL)
    {
        Chk(pFileMetadata->SetBSTR(propidFileURL, fileURL));
        Chk(pFileMetadata->SetBSTR(propidFileName, bstrFilename));
        Chk(pFileMetadata->SetBSTR(propidFolderName, bstrFoldername));
    }
    else
    {
        ULONG cProperties = 0;
        ULONG iProperty = 0;

        Chk(pPropertySetWanted->Count(&cProperties));

        for (iProperty = 0; iProperty < cProperties; iProperty++)
        {
            ULONG propertyId;
            BSTR bstr;

            Chk(pPropertySetWanted->GetIDByIndex(iProperty, &propertyId));

            if (propertyId == propidFileURL)
            {
                bstr = fileURL;
            }
            else if (propertyId == propidFileName)
            {
                bstr = bstrFilename;
            }
            else if (propertyId == propidFolderName)
            {
                bstr = bstrFoldername;
            }
            else
            {
                continue;
            }

            Chk(pFileMetadata->SetBSTR(propertyId, bstr));
        }
    }
Cleanup:

    // Free remaining dynamic memory.
    if (bstrFilename != NULL)
    {
        SysFreeString(bstrFilename);
    }
    if (bstrFoldername != NULL)
    {
        SysFreeString(bstrFoldername);
    }

    return hr;
```

©2011 Microsoft

```
}

//==========================================================================
====
//
// SetFileSizeAndTime
//
// Parameters:
//     ULONGLONG llFileSize               : FileSize property value to
store if requested.
//     FILETIME &fileTime                 : FileTime property value to
store if requested.
//     PropertyId propidFileSize          : ML id representing file size.
//     PropertyId propidFileTime          : ML id representing file time.
//     IMLPropertySet *pPropertySetWanted : Requested properties.  If
NULL, store all.
// out IMLPropertySet *pFileMetadata      : PropertySet receiving fileSize
and/or fileTime
//
// Possible Result Codes:
//     S_OK         : Successfully stored one or both properties.
//     E_FAIL       : File time conversion failure.
//     E_INVALIDARG : An mlid arg is invalid
//     E_POINTER    : pFileMetadata argument is NULL
//
// Description:
//     Stores the file size and time properties, as requested, in the
receiving property set.
//
//==========================================================================
====

HRESULT
CSampleMediaParser::SetFileSizeAndTime(
    ULONGLONG llFileSize,
    const FILETIME &fileTime,
    PropertyId propidFileSize,
    PropertyId propidFileTime,
    IMLPropertySet *pPropertySetWanted,
    IMLPropertySet *pFileMetadata)
{
    HRESULT hr = S_OK;
    BOOL  fCollectAllProperties = (BOOL)(pPropertySetWanted == NULL);
    Chk(((propidFileSize > mlid_NONE && propidFileSize < mlid_MAX) ? S_OK
: E_INVALIDARG));
    Chk(((propidFileTime > mlid_NONE && propidFileTime < mlid_MAX) ? S_OK
: E_INVALIDARG));
    Chk((pFileMetadata != NULL ? S_OK : E_POINTER));

    if (fCollectAllProperties || S_OK == pPropertySetWanted-
>HasID(propidFileTime))
    {
        MLDateTime mldt;
        Chk(ConvertFileTimeToMLFileTime(&fileTime, &mldt));
        Chk(pFileMetadata->SetDATE(propidFileTime, &mldt));
    }

    if (fCollectAllProperties || S_OK == pPropertySetWanted-
>HasID(propidFileSize))
    {
        Chk(pFileMetadata->SetLONGLONG(propidFileSize, llFileSize));
    }

Cleanup:
    return hr;
}

//===========================================================================
====
//
// ConvertFileTimeToMLFileTime
//
// Parameters:
//     FILETIME   *pftValue  : FILETIME value
// out MLDateTime *pmldtDest : MLDateTime value
//
// Possible Result Codes:
//     S_OK : success
//     E_FAIL : could not convert
//     E_POINTER : parameter validation
//
// Description:
//     Converts from the CE FILETIME structure to a MLDateTime.
//     No range checking is done on the output values.
//
//===========================================================================
====

HRESULT
CSampleMediaParser::ConvertFileTimeToMLFileTime(FILETIME const *pftValue,
MLDateTime *pmldtDest)
{
    HRESULT hr = S_OK;
    SYSTEMTIME stValue;

    ChkBool((pftValue != NULL), E_POINTER);
    ChkBool((pmldtDest != NULL), E_POINTER);

    ChkBool(FileTimeToSystemTime(pftValue, &stValue), E_FAIL);

    pmldtDest->year     = stValue.wYear;
    pmldtDest->month    = stValue.wMonth;
    pmldtDest->day      = stValue.wDay;
    pmldtDest->hour     = stValue.wHour;
    pmldtDest->minute   = stValue.wMinute;
    pmldtDest->second   = stValue.wSecond;
    pmldtDest->fraction = stValue.wMilliseconds * (1000L * 1000L); //
fraction is "billionths" of a second

Cleanup:
    return hr;
}
```

Step 5: Add the SampleMediaParser Source Code to the Subproject

To use the sample code, you must add it to your subproject.

1. In Visual Studio 2008 Solution Explorer, under your subproject name (MediaParserPlugin), right-click **Include files**, click **Add**, and then click **Existing item** to open the **Add Existing Item** dialog box.
2. Browse to the SampleMediaParser.h file you created in step 4, and then click **Add** to add it to your subproject.
3. Under your subproject name (MediaParserPlugin), right-click **Source files**, then click **Add**, and then click **Existing item** to open the **Add Existing Item** dialog box.
4. Browse to the SampleMediaParser.cpp file you created in step 4, and then click **Add** to add it to your subproject.
5. Modify the CLSID/GUID in the DEFINE_GUID statement in SampleMediaParser.h to be the actual GUID for your subproject.

Step 6: Build the OS Design and Subproject

To run the sample media parser, you must build your OS design and the media parser plug-in module so it can be downloaded to your device.

1. To build the entire OS design, including all subprojects, in Solution Explorer, right-click the OS design name (OSDesignMediaPlugin), and then click **Build OSDesignMediaPlugin**.
2. If you have already built your OS design, you can right-click on the subproject name (MediaParserPlugin), and then click **Build** to build the subproject.
3. In the **Output** window, verify that the build succeeded.

Step 7: (Optional) Debug the Media Parser Plug-In Subproject

To use the debugger to run your plug-in module, you need to add its symbols to your solution.

1. In Visual Studio, select the **Debug** menu item, and then select **Symbol Search Path** to display the **Symbol Search Path** dialog box.
2. Using the **Symbol Search Path** dialog box, add the full path to your subproject DLL file (in our example, MediaParserPlugin.dll). In our example, if you use an OS design that includes a CEPC and your compile settings specify debug, type C:\WINCE700\OSDesigns\OSDesignMediaPlugin\OSDesignMediaPlugin\Wince700\CEPC_x86_Debug\cesysgen\oak\target\x86\debug.
3. Using the **Target** menu item, select **Attach Device** to attach Visual Studio to your connected device.

Conclusion

This article describes how to extend Windows Embedded Compact 7 Media Library to support file types in addition to those supported by default. This article illustrates how to create a media parser plug-in module to manage and maintain metadata for two file types: audio media files with extension .wav and video media files with extension .mp4. This paper includes instructions on how to use Microsoft Visual Studio and Platform Builder to configure an OS design solution and subproject for plug-in development.

Additional Resources

To learn more about building projects in Windows Embedded Compact 7, see the following resources:

- WEC7 IDE Build Overview (http://go.microsoft.com/fwlink/?LinkId=208639)
- WEC7 Build Process White Paper (http://go.microsoft.com/fwlink/?LinkId=208641)
- Practical WEC7 Build Questions (http://go.microsoft.com/fwlink/?LinkId=208643)

To learn more about Windows Embedded, see the following resources:

- Windows Embedded Website (http://go.microsoft.com/fwlink/?LinkId=192020)

Copyright

This document is provided “as-is.” Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2011 Microsoft. All rights reserved.