

## Overview

The Audio Effect DMO Wizard simplifies the task of creating a Microsoft® DirectX® Media Object (DMO) that implements an audio effect. The audio effect DMO can be used in Microsoft DirectSound® applications, Microsoft DirectShow® applications, or in other custom application scenarios.

Microsoft Visual C++® 6.0 is required to run the wizard. The wizard creates a Visual Studio® project and several C++ code and header files, which implement the framework for the DMO. After the project is created, you must add code that implements the audio effect. For more information, see [Completing the DMO](#).

**Quick links:** [Step 1](#), [Step 2](#), [Generated Files](#), [Completing the DMO](#).

## Wizard Dialogs: Step 1 of 2

Under **DMO Information**, enter the following information.

Entry	Description
<b>DMO Name</b>	The name of the DMO.
<b>Class Name</b>	The name of the implementation class for the DMO. This name is derived from <b>DMO Name</b> and cannot be changed independently.
<b>Interface Name</b>	The name of the custom COM interface for the DMO. This name is derived from <b>DMO Name</b> and cannot be changed independently.
<b>DLL Name</b>	The name of the DLL that will contain the DMO. The .dll extension is added automatically.
<b>Author</b>	Your name.
<b>Description</b>	A short description of the DMO.

Under **DMO Option**, click one of the following.

Option	Description
<b>Empty DMO</b>	An empty DMO framework. If you choose this option, the second step of the wizard is enabled.
<b>Panner</b>	A sample DMO that creates a panner effect. If you choose this option, the wizard generates a complete DMO. Values from the <b>DMO Information</b> area are ignored, and no more steps are available in the wizard.

**Quick links:** [Overview](#), [Step 2](#), [Generated Files](#), [Completing the DMO](#).

## Wizard Dialogs: Step 2 of 2

In this step, you define which audio formats the DMO will support and other functionality of the DMO.

Under **Media Types**, select one or more audio media types.

Option	Description
<b>8-bit PCM</b>	The DMO will support 8-bit PCM audio.
<b>16-bit PCM</b>	The DMO will support 16-bit PCM audio.
<b>Float (32-bit)</b>	The DMO will support IEEE 32-bit floating point audio data.

Under **Sample Rates**, select one or more sample rates. The following rates are supported:

- 96,000 Hz
- 48,000 Hz
- 44,100 Hz
- 32,000 Hz
- 22,050 Hz
- 16,000 Hz
- 11,025 Hz
- 8,000 Hz

Under **Channels**, select one or more of the following:

- **Stereo**
- **Mono**

Under **Compatibility**, select one of the following options.

Option	Description
<b>Minimum Functionality</b>	The DMO will support the minimum functionality to be compatible with DirectSound and DirectShow.
<b>Real Time Parameter Control</b>	The DMO will have all of the functionality of the previous option. In addition, it will support parameters that can change in real time through the <b>IMediaParameters</b> interface.
<b>Authoring (Parameter Control + GUI)</b>	The DMO will have all of the functionality of the previous option. In addition, it will support a property page for setting the parameters.

**Quick links:** [Overview](#), [Step 1](#), [Generated Files](#), [Completing the DMO](#).

## Generated Files

The wizard creates several files, depending on which option you selected for **Compatibility** in the previous step. Some of the file names are determined by the name that you selected for the DMO. For example, if you named the DMO "MyDMO," the following files are always created.

File	Description
StdAfx.h	Used to create the precompiled header file.
StdAfx.cpp	Used to create the precompiled header file.
Resource.h	Resource header file.
MyDMO.rc	Resource file.
MyDMO.h	Header file for the IMyDMO interface.
MyDMO.cpp	Declares the DLL entry-point function and module registration functions.
MyDMO.def	Exports the public DLL functions.
CMyDMO.h	Header file for the DMO implementation class, CMyDMO.
CMyDMO.cpp	Contains the implementation of CMyDMO.
CMyDMO.rgs	Contains an ATL registry script.

If you selected the **Real Time Parameter Control** option, the wizard creates additional files.

File	Description
Alist.h	Used by the CParamsManager class, which implements the parameter control functionality.
Alist.cpp	Used by the CParamsManager class.
ControlHelp.h	Used by the CParamsManager class.
ControlHelp.cpp	Used by the CParamsManager class.
Param.h	Declares the CParamsManager class.
Param.cpp	Implements the CParamsManager class.
Validate.h	Used by the CParamsManager class.

If you selected the **Authoring** option, the wizard creates all of the files listed previously, plus the following. (Again, the actual file names are determined by the name that you selected for the DMO.)

File	Description
CMyDMOProp.h	Declares the implementation class for the property page, CMyDMOProp.
CMyDMOProp.cpp	Contains the implementation of CMyDMOProp.
CMyDMOProp.rgs	Contains an ATL registry script.

**Quick links:** [Overview](#), [Step 1](#), [Step 2](#), [Completing the DMO](#).

## Completing the DMO

The wizard creates a partial implementation of the DMO. At a minimum, you must add code to the **DoProcess** method to process the audio data. You might have to write additional code as well, depending on the functionality of the DMO and the compatibility option that you chose in the previous step.

The following tables list the items that you might need to modify.

### Minimum Functionality

Wizard-generated code	Remarks
Class declaration	Add any additional member variables that are needed.
Constructor	Initialize member variables.
Destructor	You might need to free resources in the destructor. However, most resources will be released inside the <b>InternalFreeStreamingResources</b> method, listed later.
Custom COM Interface	The wizard declares a custom COM interface for the DMO. The interface name is derived from the name of the DMO. For example, if the DMO is named "MyDMO," the interface is named IMyDMO. You can add methods to the interface from the Class View window in Visual C++ 6.0. The DMO must implement any methods that are declared in the interface.
<b>UpdateStatesInternal</b>	Use this method to update the internal state of the DMO. For example, if the DMO has a member variable whose value depends on other member variables, update the value here.
<b>GetLatency</b>	This method returns an estimate of the latency that the DMO introduces into the stream. The default implementation returns a value of 500 nanoseconds. Modify this value if needed.
<b>InternalAllocateStreamingResources</b>	Use this method to allocate any resources that the DMO needs when it processes audio data.
<b>InternalDiscontinuity</b>	This method is called when the client signals a <i>discontinuity</i> , which represents a break in the input. Use this method to clear any internal states that should not carry over after a discontinuity.
<b>InternalFlush</b>	Discard any data the DMO has buffered, and restore the DMO to its original internal state.
<b>InternalFreeStreamingResources</b>	Use this method to free any resources that were allocated by the <b>InternalAllocateStreamingResources</b> method.
<b>InternalGetInputMaxLatency</b>	This method returns the maximum offset between input time stamps and output time stamps. The default implementation returns a value of 3,000 nanoseconds. You can modify this value if needed.
<b>DoProcess</b>	Use this method to process input data and write it to the specified output buffer.

**InternalAcceptingInput**

This method checks whether the DMO can accept more input. The default implementation does not accept input if the DMO already holds an input buffer. You might change the implementation if your DMO can hold multiple input buffers; otherwise, there is no reason to change this method.

**InternalProcessInput**

This method is called when the client delivers an input buffer to the DMO.

**InternalProcessOutput**

This method is called when the client delivers an output buffer for the DMO to process the input data.

## Real Time Parameter Control

If you clicked the **Real Time Parameter Control** option, the wizard creates a class named `CParamsManager` that manages the parameter curves. You must add the following items to the code.

**Wizard-generated code****Remarks**

DMO Parameters Enumeration

The wizard declares an enumeration type to identify the parameters in your DMO. The name of the enumeration is derived from the name of the DMO. For example, if your DMO is named "MyDMO," the enumeration is named `MyDMOFilterParams`. For each parameter, add an enumeration value that identifies that parameter. For example, if your DMO supports a delay parameter, you might add an enumeration value named `MYDMO_DELAY`. However, the names are arbitrary.

**ParamInfo** Structure

The wizard declares a global **ParamInfo** variable. Use this to provide information about each parameter, including the minimum and maximum values and the data type. The generated code contains an example of how to declare this information.

**SetParamInternal**

This method sets the value of a parameter, specified by enumeration value. Add case statements for each parameter. The generated code contains an example.

**SetAllParameters**

This method sets values for all of the parameters. Use the **SET\_PARAM** macro to set each parameter. The generated code shows an example of the macro call.

**GetAllParameters**

This method copies the value of every parameter into a structure provided by the caller. Use the **GET\_PARAM** macro to retrieve each parameter value. The generated code shows an example of the macro call.

## Authoring Compatibility

If you selected the **Authoring** option, the wizard creates a class that implements a property page for the DMO.

**Wizard-generated code**

String Resources

Dialog Resource

Property Page Class

**OnInitDialog**

**Apply**

**Remarks**

The wizard creates several string resources which are used by the property page. Set the string values as appropriate for your DMO.

The wizard creates a resource for the property page dialog box. Add whatever controls are needed for the user to set the parameters.

Add member variables for the dialog box controls.

This method is called when the property dialog box receives a WM\_INITDIALOG message. Use this method to initialize the dialog box controls. The generated code shows an example.

This method is called when the user clicks **OK** or **Apply**. Use this method to set the parameter values, based on contents of the dialog box controls. The generated code shows an example.

**Quick links:** [Overview](#), [Step 1](#), [Step 2](#), [Generated Files](#).

