Config System Sample

Collapse All

The Config System sample showcases a configuration system using a database of hardware device models. The sample detects the hardware on the system, and then looks up the database to find custom behavior defined for those existing devices.

Path

Source	*SDK root*\Samples\C++\Direct3D\ConfigSystem
Executable	*SDK root*\Samples\C++\Direct3D\Bin*x86 or x64*\ConfigSystem.exe

Sample Overview

The Config System sample is a demonstration of a database-driven application configuration system. Today, several shipping games, such as Halo PC, Microsoft Flight Simulator, Microsoft Combat Flight Simulator, and Rise of Nations are already taking advantage of this versatile and powerful system. 3D graphic cards have a set of capabilities that they support to make better-looking images. Realistically, not every card supports the exact same set of capabilities; therefore, applications need to check each card for the capabilities it supports. In Direct3D, device capabilities are reported in the capability structure, or cap bits for short, that can be obtained by calling IDirect3DDevice9::GetDeviceCaps. The cap bits structure details what the device can and cannot support. Usually, the cap bits provide a fast and easy way to know what rendering features an application can utilize.

However, the cap bits are not always accurate or complete. Devices may mistakenly report cap bits which they actually do not support, or which are broken. Furthermore, a capability may be enabled at the expense of severe performance penalty. In this case, many developers prefer to have a steady performance over a particular feature. To overcome these issues associated with the cap bits, an application developer could test all supported hardware models during development, then record the list of hardware that the application has trouble with (as well as the specific problems each device has) in a database. This allows an application to detect the host system hardware at run time. If the hardware model is found, the database provides the detail as to what special actions the application must take to take advantage of the hardware.

How the Sample Works

The sample demonstrates a database-driven configuration system with three code components:

- The configuration system client makes use of the system as well as handling necessary sample code, such as rendering and animation routines. This is located in Main.cpp.
- The configuration database implements the configuration database as a Component Object Model (COM) object. The database object handles parsing of the configuration file. This is located in ConfigDatabase.cpp.
- The configuration manager encapsulates access to the database and provides efficient access to the properties. This is located in ConfigManager.cpp.

The sample represents a simple game. When the sample is run, the user is in a closed room with several balls on the ground. The user can shoot additional balls and watch them bounce around. At the bottom of the screen, a list of properties is listed. These properties are parsed and loaded from the configuration database file based on the display and audio devices that the sample detects. The existence of these properties affects how the sample runs (for instance, how it renders or handles sound).

Graphics Card Capabilities

A collection of the capabilities exposed by current drivers of graphics hardware available on the market today are available as separate files in the DirectX SDK.

See the **Graphics Card Capabilities** sample in the DirectX Sample Browser for more details.

The Database

In the sample, the configuration database file is a text file. Each line is either a statement or keyword that affects parsing, a property definition, or a comment that starts with a double slash. When parsing the database, the text letter case is ignored. The file contains three sections:

- Section 1: Hardware Requirements
- Section 2: Named Property Sets
- Section 3: Hardware Devices in the Database

Section 1: Hardware Requirements

The first section lists the hardware requirement of the sample. It starts with the Requirements statement followed by one or more properties, or property-value assignments. The section ends with a break statement as shown below:

```
Requirements
    OS=Win98
    CpuSpeed=733
    Memory=128
    VideoMemory=64
    DirectX=4.9.0.902
    DiskSpace=100
    break
```

Here are the values for each of the requirements:

Requirement Property	Description
OS	Operating system version. Supported values, in ascending order, are: Win95, Win98, WinME, Win2k, WinXP, and Win2003.
CpuSpeed	Processor speed in megahertz.
Memory	Amount of system memory in megabytes.
VideoMemory	Amount of video memory in megabytes.
DirectX	DirectX version on the system in this format: #.#.#.#, where # denotes a decimal digit
DiskSpace	Amount of available disk space in megabytes. This requirement is here for illustrative purposes only and is not checked by the sample.

Section 2: Named Property Sets

The second section is a list of named property sets. A property set groups related custom properties under one name. Each property set is a set of custom properties followed by a break statement. It starts with the PropertySet statement followed by one or more properties, or property-value assignments. The section ends with a break statement as shown below:

```
PropertySet = "MyPropSet"
    CustomPropertyName1          // Name
    CustomPropertyName2 = Value  // Name/Value
    break
```

Each property set begins with the name of the property set enclosed in quotes; this example contains two custom properties. A custom property is a name/value pair, or simply a name by itself. It is defined and applied to devices by the application and it serves as a flag to signal the application that a special action should be taken when dealing with the devices to which this custom property is applied.

Note

As you can see in the previous example, comments can be included by using a double forward slash notation (//).

A property set can refer to other property sets as shown here:

```
PropertySet = "SmallPropSet"
    CustomPropertyName1          // Name
    CustomPropertyName2 = Value  // Name/Value
    break

PropertySet = "BigPropSet"
    PropertySet = "SmallPropSet" // BigPropSet includes SmallPropSet's properties
    CustomPropertyName3          // Name
    CustomPropertyName4 = Value  // Name/Value
    break
```

The PropertySet statement is used in place of a custom property name, and its value is the name of the property set to include, in quotes. It is as if BigPropSet were defined like this:

```
PropertySet = "BigPropSet"
    CustomPropertyName1          // Name
    CustomPropertyName2 = Value  // Name/Value
    CustomPropertyName3          // Name
    CustomPropertyName4 = Value  // Name/Value
    break
```

Section 3: Hardware Devices in the Database

The third section lists the hardware device entries included in the database. In the sample, the database contains audio and display devices. The device entries are organized by vendors and device ids. Each entry must follow one of the following four formats:

Format 1: Vendor Definition

The vendor definition looks like this:

```
AudioVendor = 0x1102 "Audio card vendor name"       // For audio vendor
DisplayVendor = 0x10b4  "Display card vendor name"  // For display vendor
```

A vendor string follows the hexadecimal number vendor id with the vendor's name.

Format 2: Device Definition

The device definition looks like this:

```
0x1015 = "Device Name"
```

The device hexadecimal number is followed by a string with the device name. You can use "unknown" for the device string to represent all other devices from the current vendor that are not found in the database.

Format 3: Custom Property Definition

A custom property or reference to a named property must be defined like this:

```
PropertyName1             // Name
PropertyName2 = Value     // Name/Value
PropertySet = "MyPropSet" // Apply all properties included in MyPropSet
```

The first example (without a value pair) is useful for enabling or disabling a particular feature, such as specular lighting. The second property (with a value pair) is useful when a parameter needs to be set to a different value for some devices. The third form is a reference to the previously named property set MyPropSet, as if all properties in the previously named property set were listed.

Format 4: Conditional Property Definition

A conditional property is a custom property (and/or named property set references) embedded in an if block, and are only applied when the specified condition is true. Additionally, you may nest if blocks. Here is an example:

```
if identifier operator value
    PropertyName3 = Value
    PropertySet = "MyPropSet"
```

```
endif
```

Where identifier can be one of the following:

Identifier	Value	Description
ram	Number in decimal or hexadecimal (begins with 0x)	Amount of system memory.
videoram	Number in decimal or hexadecimal (begins with 0x)	Amount of video memory.
SubSysID	Number in decimal or hexadecimal (begins with 0x)	The subsystem id of the hardware device.
Revision	Number in decimal or hexadecimal (begins with 0x)	The revision number of the hardware device.
driver	Full version format, defined as #.#.#.#, where # denotes a number in decimal or hexadecimal	The device's driver version.
guid	A globally unique identifier (GUID) in the format of ########-####-####-####-############ where each # denotes a hexadecimal digit	The device's GUID.
OS	One of the following: Win95, Win98, Win98SE, WinME, Win2k, WinXP	The computer's operating system.
Any DWORD members of **D3DCAPS9**	Number in decimal or hexadecimal (begins with 0x)	Values such as Caps, Caps2, and so on.

And where operator can be one of the following:

= or ==	True if the two operands are equal in value.
<> or !=	True if the two operands are not equal in value.
>=	True if the left operand is greater than or equal to the right one in value.
<=	True if the left operand is less than or equal to the right one in value.
>	True if the left operand is greater than the right one in value.
<	True if the left operand is less than the right one in value.
&	Performs a bit-wise AND operation with the two operands and returns the result.

Do not use quotes around the identifier or its value.

Parsing the Device Database

The sample separately parses the device's audio and display properties from the database. The following discussion describes parsing of display device properties, though the process also applies to audio device properties.

To match for a specific hardware device, the parser first searches for a display vendor statement with a matching vendor id. Once a matching vendor definition is found, the parser skips vendor definitions after the matching definition. This is done because sometimes a vendor has several vendor ids, and they are defined with several vendor definition lines. Next, the parser looks for any and all custom properties after the vendor definition but before a device definition. The properties found here will be applied to all devices from that particular vendor. Therefore, properties that apply to all devices from a particular vendor should be placed immediately after the vendor definition when constructing the database.

After parsing the vendor-wide properties, the parser continues to search for a device definition line. The parser then searches all the properties. The parser stops when it finds a break statement. It does not stop when it finds another device definition. This behavior is useful as it becomes easy to add properties

that apply to multiple devices.

The following is an example database that defines properties for devices from a particular vendor:

```
DisplayVendor = 0x1101 "Display Vendor Name"
DisplayVendor = 0x1102 "Display Vendor Name"
    DisableSpecular         // Disable specular lighting
0x1001 = "Card Model 1"
    ForceShader=14          // Restrict shader to 1.4 or lower
0x1002 = "Card Model 2"
    MaximumResolution=1024 // Max resolution 1024x768
    break
0x1003 = "Card Model 3"
Unknown = "Unknown"
    UseAnisotropicFilter    // Use anisotropic filtering
    break
```

In the above example, three devices from a display vendor are defined. The vendor has two known vendor ids: 0x1101 and 0x1102. DisableSpecular appears before any device definition for this vendor; therefore, it applies to all devices from this vendor.

Specifying a Default for All Devices

The database can specify a default set of properties to apply to all devices from all vendors, whether or not they are included in the database. To do this, the database can include two ApplyToAll property sets. One appears before the hardware vendor definitions, and the other appears after. The syntax of an ApplyToAll property set is as follows:

```
ApplyToAll
    Property1
    Property2
    break
```

ApplyToAll property sets, like any other property set, should end with a break statement. The first ApplyToAll block is parsed before the hardware vendor section is; therefore, the first block defines default properties that can be overridden by vendor-specific or device-specific properties and the second ApplyToAll is parsed after parsing the hardware vendor section so the second block defines default properties that cannot be overridden.

Custom Properties Implemented by the Sample

Custom properties are designed and supported by the application. The sample implements a list of common custom properties to demonstrate typical uses and the affect that properties can have on application behavior. The sample also lists a number of additional properties not implemented in the sample, but which may be valuable to other previously-released applications. These serve the purpose of indicating the types of problems hardware and drivers can have.

The custom properties which are implemented by the sample are detailed below:

Property	Value	Description
MaximumResolution	640, 800, 1024, etc.	This limits the maximum resolution used by the sample by specifying the width of the resolution. For instance, to limit resolution to 800 x 600, use MaximumResolution=800.
UseFixedFunction	None	Use the fixed function pipeline. When this property is not present, the renderer uses programmable shaders. In the sample, fixed-function uses per-vertex lighting, while pixel shader 2.0 or higher uses per-pixel lighting. The visual difference is dramatic.
ForceShader	0, 11, 12, 13, 14, 20, A2, B2, 30	Use a particular pixel shader version, even if the graphic device supports a newer version. For instance, to force the renderer to use ps_1_4, use ForceShader=14. In the sample, pixel shader 2.0 or higher uses per-pixel lighting. The visual difference is dramatic.
UseAnisotropicFilter	None	Use anisotropic filtering for textures.
DisableSpecular	None	Warn the user about the presence of known prototype cards. When

		this is present, the sample brings up a message box to inform the user.
UnsupportedCard	None	This property is used to disallow running the sample with cards below specification. When this is present, the sample brings up a message box to inform the user, then terminates.
OldDriver	None	This property is used to warn the application user that the display driver is older than the version tested by the developer. The sample brings up a message box to inform the user.
InvalidDriver	None	This property is used to disallow running the sample with display drivers that are known to cause severe problems with the application. When this is present, the sample brings up a message box to inform the user, then terminates.
OldSoundDriver	None	This property is used to warn the application user that the sound driver is older than the version tested by the developer. The sample brings up a message box to inform the user.
InvalidSoundDriver	None	This property is used to disallow running the sample with sound drivers that are known to cause severe problems with the application. When this is present, the sample brings up a message box to inform the user, then terminates.
UMA	None	This property is used to alert the application that the display card on the system is a Unified Memory Architecture (UMA) card. A UMA display card uses part of the system memory as its video memory. When this property is present, the amount of available video memory on the system is calculated from the system memory size.

The following properties are included in and parsed from the database, but not implemented by the sample.

Property	Value	Description
LinearTextureAddressing	None	When this is defined, non-square textures use texture coordinates that have not been normalized.
DoNotUseMinMaxBlendOp	None	When this is defined, the display device does not handle minimum or maximum blending mode properly, so these should be avoided.
DisableDriverManagement	None	With this property, the device should be created with **D3DCREATE_DISABLE_DRIVER_MANAGEMENT** so that resource management is done by Direct3D, not the driver.
DisableAlphaRenderTargets	None	When this is defined, the application should not allow render targets with an alpha channel.
DisableRenderTargets	None	All render targets should be disabled.
EnableStopStart	None	The sound card supports calling a IDirectSoundCaptureBuffer8::Start or IDirectSoundCaptureBuffer8::Stop method at a high frequency.

Loading and Accessing the Database

When an application needs to access the content of the database, it calls the static method IConfigDatabase::Create, which initializes and returns an IConfigDatabase object. The application should then call the IConfigDatabase::Load method, passing in the following parameters:

- the configuration database file path
- a SOUND_DEVICE structure with information about the sound device on the system

- a **D3DADAPTER_IDENTIFIER9** structure with similar information about the display device on the system
- the display cap bits as returned by Direct3D
- the amount of system and video memory
- the processor speed

IConfigDatabase::Load parses the configuration database file and saves the requirements and relevant custom properties based on the detected display and sound devices. Internally, IConfigDatabase saves this information as a property name/value pair. Once the operation is complete, the application can query for available requirements or custom properties and obtain the string pairs.

Because the properties control how the application renders, the application will very likely need to check the value of properties in a rendering loop. Because comparing strings is not as efficient as comparing two numbers, it is generally not a good idea for the rendering loop to query properties directly from IConfigDatabase. The CConfigManager helper class acts as a code layer on top of IConfigDatabase. This application-customized class provides detection for hardware on the system. It also queries its IConfigDatabase for properties, then stores the property values in its member variables. During rendering, the application can check the value of the member variables to determine the appropriate behavior instead of querying string pairs from IConfigDatabase.

Running in Safe Mode

The sample also demonstrates a simple implementation of safe mode. Safe mode is a mode in which the application runs everything in the lowest possible settings, such as the lowest resolution or disabling any feature that is not supported by every video card. In safe mode, the sample saves the state of the application in a file in case the application unexpectedly ends (allowing the status data to persist outside of the application lifetime). To do this, the sample creates an empty file at startup (called Launch.sta) and deletes this file before it exits. The sample writes this file to the following path:

```
Drive:\Document and Settings\username\Local Settings\Application Data\ConfigSystem
```

Note that username as used in the previous path is the log-in name of the current user. The following folder is used by applications local to the system to store information for the current user.

```
Drive\Document and Settings\username\Local Settings\Application Data
```

This path can be obtained by calling SHGetFolderPath with CSIDL_LOCAL_APPDATA. The sample uses this path instead of the application executable directory because it is the best practice to write user-specific data to a location that is specific to the user. Not every user will have access to the executable directory all the time. Therefore, writing user-specific data to the executable directory should be avoided.

When the sample is launched and before it writes Launch.sta, it determines whether Launch.sta already exists or not. If it does exist, then the implication is that the previous launch of the application encountered an unrecoverable problem and could not exit normally. This is a good indication to turn on safe mode so that the application can minimize the chance of a recurring problem.

Summary

A configuration system using a database allows applications to precisely define the behavior to employ for a specific hardware device. It is worth noting that the properties in the database are defined by the client application, and the application is free to extend the database by adding more properties as it requires, or adding more hardware vendors or devices as newer hardware becomes available. For applications that get patched or updated, the most up-to-date configuration database file can accompany the patch to ensure that customers with newer hardware can still benefit from the system.