

# Windows Color System

Article • 12/30/2021

## Purpose

Color management schemes are implemented in Microsoft Windows operating systems to improve the rendering of color content in all forms of digital communication.

The color management scheme that's used starting with Windows 2000, Windows XP, and Windows Server 2003 is called Image Color Management (ICM) 2.0. The color management scheme that's used starting with Windows Vista is called Windows Color System (WCS) 1.0. The WCS 1.0 color management scheme is a superset of ICM 2.0 APIs and functionality.

## Where applicable

ICM can be used in all applications on Windows 2000 and later operating systems. WCS can be used in all applications on Windows Vista and later operating systems.

## Developer audience

The WCS API is designed for use by C/C++ programmers. Familiarity with the Windows graphical user interface, message-driven architecture, and a working knowledge of color management concepts are required.

## Run-time requirements

Applications that use the ICM API require Windows 2000 or later. Applications that use the WCS API require Windows Vista or later. For specific run-time information on a function, see the Requirements section of the reference page for that function.

## In this section

- [Security Considerations: Windows Color System](#)
- [Basic color management concepts](#)
- [Windows Color System Schemas and Algorithms](#)
- [About Windows Color System Version 1.0](#)
- [Using WCS 1.0](#)

- Reference
- WCS 1.0 Glossary

## Related topics

[OpenGL](#)

[Windows Multimedia](#)

[Windows GDI](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Security Considerations: Windows Color System

Article • 12/30/2021

This document provides information about security considerations related to image color management. This document doesn't provide all you need to know about security issues—instead, use it as a starting point and reference for this technology area.

## Non-Microsoft CMMs can run in system context

Non-Microsoft Color Management Modules (CMMs) should be treated like non-Microsoft printer drivers because they have the potential to run in a system context for printing operations.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Basic Color Management Concepts

To understand how Windows Color System (WCS) version 1.0 is used, a working knowledge of color management concepts is required. This overview presents enough color management background to provide a definition of the color management terms and concepts that appear in the WCS Programmer's Reference in the Platform Software Development Kit (SDK) documentation. It is not exhaustive in its coverage. The topics in this section cover the major areas of color management, such as:

- [Color in Imaging](#)
- [Color Spaces](#)
- [Color Conversion and Color Matching](#)
- [Windows Color System Schemas and Algorithms](#)
- [Further Information](#)

---

Last updated on 05/19/2025

# Color In Imaging

Article • 12/30/2021

Color management is not a new topic. It has been in common use for nearly as long as there have been commercial printing and publishing companies. A well-developed body of theory and experience has been applied in recent years to computerized imaging and publishing systems. The concepts embodied in these theories and practices are based on human color perception.

It is not necessary for application developers to possess a high level of expertise in the physiology of the eye and nervous system. Nor is it necessary to be an expert in the physics of light. However, an understanding of the fundamentals of vision and imaging will aid developers in using Windows Color System 1.0. These fundamentals include:

- [Human Color Perception](#)
- [Additive Primary Colors](#)
- [Subtractive Primary Colors](#)
- [Describing Color](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Human Color Perception

Article • 12/30/2021

Human vision is made possible by the presence of light sensitive cells in the eye called *rods* and *cones*. Research has shown that the rods do not seem to be involved in the perception of color in human beings. There are three different types of cones in the retina. Each type of cone detects either red, green, or blue. All other colors humans see are mixtures of these three colors. For instance, white is perceived when approximately equal amounts of red, green, and blue are seen. Black is seen when no red, green, or blue (very little or no light at all) is detected by the eye. The amount of color humans see also depends on the strength, concentration, and position of the light source. Lighting conditions can have a profound effect on the perception of color.

In imaging systems, colors can be mixed in different ways to produce a desired result for the eye. The mixing methods most commonly used are based on the *additive primary colors* and the *subtractive primary colors*. All colors can be reproduced using either method.

---

## Feedback

Was this page helpful?

 Yes

 No

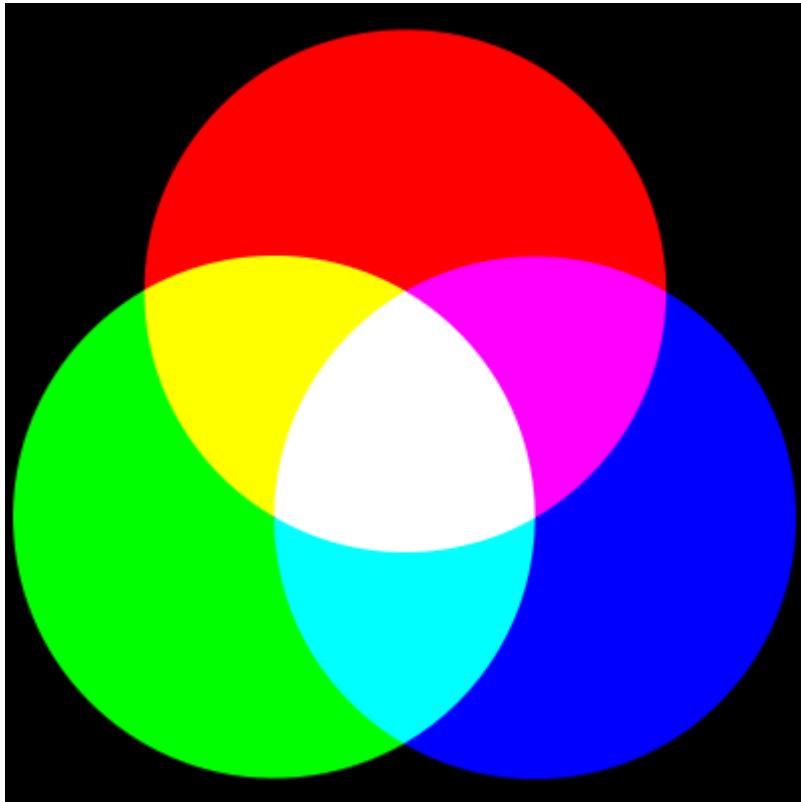
[Get help at Microsoft Q&A](#)

# Additive Primary Colors

Article • 12/30/2021

The additive method of color mixing is based on the assumption that you start with black. That is, if there are no other colors present (the image is black) and you add red, the image appears red. If you then add blue, the image appears magenta.

The three additive [primary colors](#) are red, green, and blue. The following figure illustrates additive primary color mixing.



---

## Feedback

Was this page helpful?

Yes

No

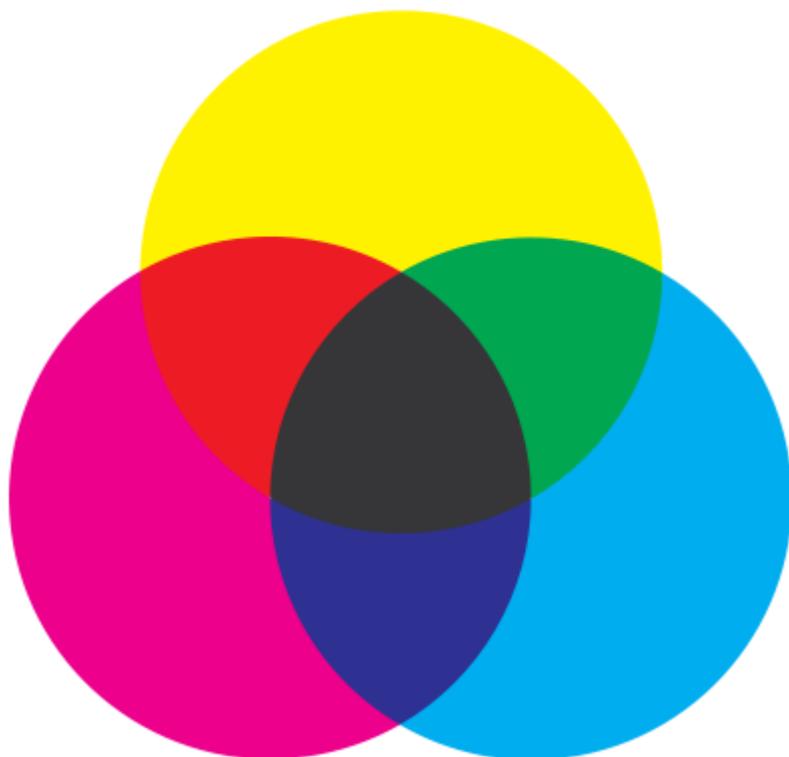
[Get help at Microsoft Q&A](#)

# Subtractive Primary Colors

Article • 12/30/2021

The subtractive method of color mixing is based on the assumption that you start with white. All colors are present in equal amounts. If you subtract cyan and yellow from white, the resulting image is magenta. In a green image, subtracting cyan will result in changing the image to yellow.

The subtractive [primary colors](#) are cyan, yellow, and magenta. The following figure demonstrates subtractive primary color mixing.



---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Describing Color

Article • 12/30/2021

Color can be described in terms of components. The components most often used by imaging professionals are *hue*, *chroma*, and *lightness*. A hue is what is normally thought of as a color. A color's hue fixes its place in the visible spectrum of light. The chroma of a color is how "pure" or "strong" a color is. Neutral gray is said to have zero saturation. The lightness of a color refers to the intensity of light that is reflected or transmitted by an image.

The terms *tint*, *tone*, and *shade* are also well-used in color imaging literature. A tint of a color is obtained by mixing its hue with white. A tone of a color is created by mixing a hue with gray. A shade of a color is made by adding black to its hue.

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Color Spaces

Article • 12/30/2021

The human eye is often able to detect many more colors than digital devices can reproduce. For instance, if you look at a blank, white page of paper, your eye is probably detecting at least 100 distinct shades of white. A white wall can easily have 1500 shades of white.

High-quality digital cameras, scanners, and other image acquisition devices can also detect hundreds of thousands or even millions of colors. Because of the presence of so many detectable colors, imaging professionals have invented models for specifying colors. These models are called *color spaces*.

The reason these models are referred to as color spaces is that most of them can be mapped into a 2-D, 3-D, or 4-D coordinate system similar to a Cartesian coordinate system. Hence colors can be said to be composed of coordinates in a 2-D, 3-D, or 4-D space. The color components in a color space are also referred to as *color channels*.

Some color spaces are intended to be independent of any device that is used to produce color images. Some are very device dependent. Both device-dependent and device-independent color spaces are discussed in the following sections:

- [RGB Color Spaces](#)
- [HSV Color Spaces](#)
- [HLS Color Spaces](#)
- [CMY and CMYK Color Spaces](#)
- [Device-Dependent Color Spaces](#)
- [Device-Independent Color Spaces](#)

---

## Feedback

Was this page helpful?

 Yes

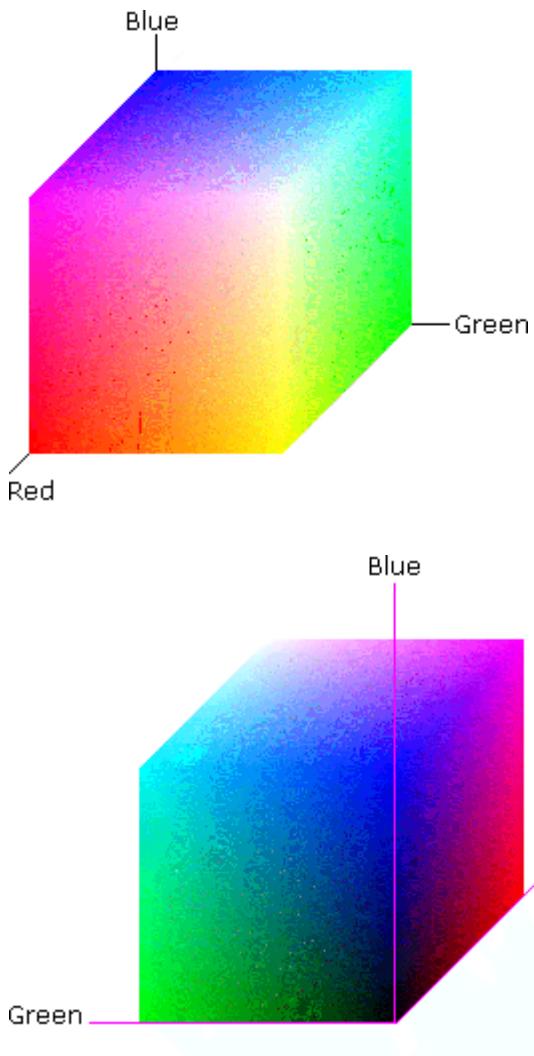
 No

[Get help at Microsoft Q&A](#)

# RGB Color Spaces

Article • 12/30/2021

An RGB [color space](#) is created by mapping the colors red, green, and blue onto a 3-D Cartesian coordinate system. This results in a 3-D cube like the ones shown in the following figure. This figure displays the same RGB cube from two different angles. Notice that the origin of the coordinate system is black. This is where the red, green, and blue (RGB) color components are all 0.0. The diagonally opposite corner of the cube is white, where the RGB color components are at their maximum value.



Like most [color spaces](#), the RGB color space is normalized. That is, all color values are restricted to the range of zero to one inclusive. So black is (0.0, 0.0, 0.0), and white is (1.0, 1.0, 1.0).

In the RGB color space, the [primary colors](#) are red, green, and blue. The secondary colors are cyan, yellow, and magenta.

RGB color spaces can be device dependent or device independent.

---

# Feedback

Was this page helpful?

 Yes

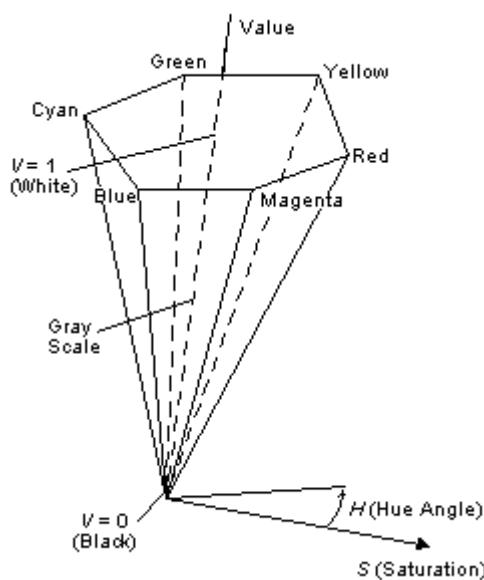
 No

[Get help at Microsoft Q&A](#)

# HSV Color Spaces

Article • 12/30/2021

Hue, saturation, and value (HSV) [color spaces](#) are often used by artists. "Hue" is what we normally think of as color. It is the attribute of a color by which we give it a name such as "red" or "blue". "Value" is another word for "lightness," the attribute of a color that makes it seem equivalent to some shade of gray between black and white. Saturation is a measure of how different a color appears from a gray of the same [lightness](#). Zero saturation indicates no hue, just gray scale. The HSV color space is normalized.



The preceding figure shows a line drawing of HSV space in the form of a hexcone. Each of its cross sections is a hexagon. At the vertices of each cross section are the colors red, yellow, green, cyan, blue, and magenta. A color in HSV space is specified by stating a hue angle, the chroma level, and the lightness level. A hue angle of zero is red. The hue angle increases in a counterclockwise direction. Complementary colors are 180° apart.

HSV color spaces can be device dependent or device independent.

---

## Feedback

Was this page helpful?

Yes

No

[Get help at Microsoft Q&A](#)

# HLS Color Spaces

Article • 12/30/2021

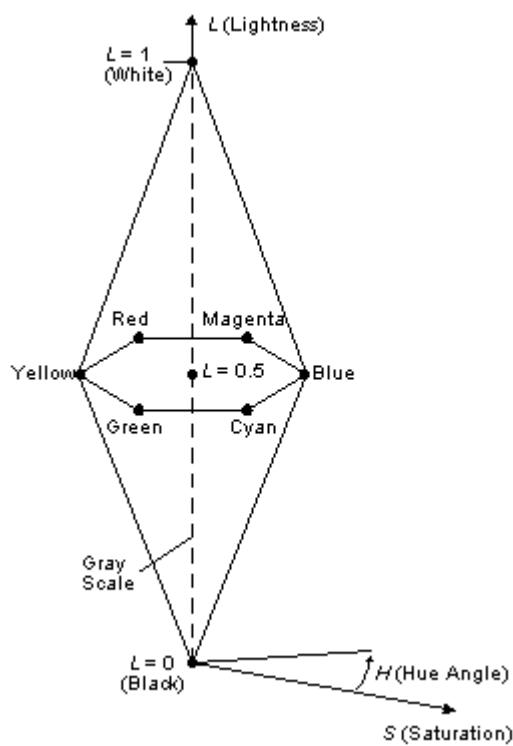
HLS [color spaces](#) are also widely used by artists. Its color components are hue, lightness, and saturation (chroma).

Hue has the same meaning as the HSV model, except that a hue angle of 0 corresponds to blue in this model. Magenta is at 60, red is at 120. As with the HSV model, complementary colors are 180 apart.

Lightness is the amount of black or white in a color. Increasing lightness adds white to the hue. Decreasing lightness adds black to the hue.

[Saturation](#) in the HLS model is a measure of the "purity" of a hue. As saturation is decreased, the hue becomes more gray. A saturation value of zero results in a gray-scale value.

The following figure is a line drawing of HLS space, which is a double hexcone. Any horizontal cross section of the HLS color space is a hexagon. HLS is a normalized color space. That is, the lightness and saturation values range from 0.0 to 1.0 inclusive. Hue varies from 0 to 360 inclusive.



HLS color spaces can be device dependent or device independent.

---

# Feedback

Was this page helpful?

 Yes

 No

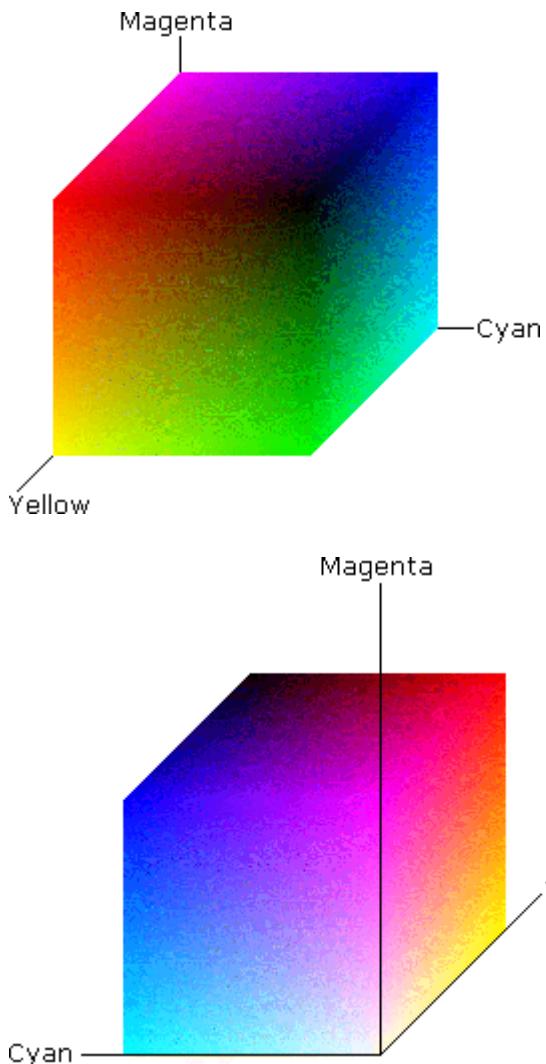
[Get help at Microsoft Q&A](#)

# CMY and CMYK Color Spaces

Article • 12/30/2021

The CMY and CMYK color spaces are often used in color printing. A CMY color space uses cyan, magenta, and yellow (CMY) as its [primary colors](#). Red, green, and blue are the secondary colors.

The following figures are color representations of the CMY color space. The CMY color space is normalized.



The CMY color space is subtractive. Therefore, white is at  $(0.0, 0.0, 0.0)$  and black is at  $(1.0, 1.0, 1.0)$ . If you start with white and subtract no colors, you get white. If you start with white and subtract all colors equally, you get black.

The CMYK color space is a variation on the CMY model. It adds black (Cyan, Magenta, Yellow, and black). The CMYK color space closes the gap between theory and practice. In theory, the extra black component is not needed. However, experience with various types of inks and papers has shown that when equal components of cyan, magenta, and

yellow inks are mixed, the result is usually a dark brown, not black. Adding black ink to the mix solves this problem.

The CMY and CMYK colors spaces can be device independent, but most often they are used in reference to a specific device.

---

## Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

# Device-Dependent Color Spaces

Article • 12/30/2021

Most [color spaces](#) are device dependent. Even though a particular device, such as a printer, may use the CMYK color space, the colors it renders for specific CMYK values are often slightly different than all other types of printers.. It is precisely for this reason that the WCS 1.0 color management system is so useful.

All color spaces have *a white point*, which is defined as the whitest white that can be produced in that color space. Since devices can differ from one another, their white points can also differ. All colors that a device can produce are relative to its white point. Therefore, a color management system must be able to map the white point of one color space into another and preserve the relative locations of all colors. It must also be able to map a color in one color space to its closest match in another color space regardless of the differences in the white points. WCS 1.0 is able to accomplish both of these tasks.

The RGB color space is often used on computer monitors. As such, it is device dependent. Printers typically use CMYK [colorants](#). Each printer implements its own version of the CMYK color space. Digital images may not actually store colors in them. They may store index numbers into a palette of colors. The result is that it is very hard for individual application developers to use or provide a standardized method of moving color images from one device to another. Imaging professionals commonly experience the frustration of creating a graphic image on a computer screen and having it turn out very differently when it is printed. WCS 1.0 addresses these imaging needs.

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Device-Independent Color Spaces

Article • 12/30/2021

Recognizing the need for standard, device-independent color measurements, the Commission International de l'Eclairage (International Commission on Illumination), or CIE, created a [color space](#) based on "imaginary" [primary colors](#). No actual device is expected to produce colors in this color space. It is used as a means of converting colors from one color space to another. The primary colors in this color space are the abstract colors X, Y, and Z.

The 1931 CIEXYZ color space is widely used as the basis for color space conversion. With the rise of the Internet, however, bandwidth considerations have made the XYZ color space unwieldy. The exchange of images over the limited bandwidth of the Internet necessitates a more compact [color model](#).

As a result, Hewlett-Packard and Microsoft have proposed the adoption of a standard predefined RGB color space known as sRGB, so as to allow accurate color management with very little data overhead. This standard has been approved as a formal international standard by the International Electrotechnical Commission (IEC) as IEC 61966-2-1: Multimedia Systems and EquipmentColour Measurement and ManagementPart 2-1: Colour ManagementDefault RGB Colour Space. It is available directly from the IEC at <https://www.iec.ch/>. Information discussing the technical issues involved in sRGB is available on the Internet at:

## [A Standard Default Color Space for the Internet - sRGB](#)

A help-file version of a white paper discussing the technical issues involved in sRGB, sRGB.HLP, is available in the \Help folder of the WCS 1.0 Programmer's Reference in the Platform SDK.

Different file formats may use or add a flag to specify that the image is in sRGB color space. In the Windows device-independent bitmap (DIB) format, setting the **bV5CSType** member of the [BITMAPV5HEADER](#) structure to LCS\_sRGB specifies that the DIB colors are in the sRGB color space.

---

## Feedback

---

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Color Conversion and Color Matching

Article • 12/30/2021

The process of converting colors from one [color space](#) to another is called *color conversion*. All colors in a color space are fixed relative to the color space's [white point](#). Since the white point of a color space varies from device to device, a converted color must then be matched to its visually closest color in the destination color space. This process is called *color mapping*.

For example, if you have a digital image that you created on your display, it may be in a device-dependent RGB color space. If you want to print it on a printer, it must be converted to the printer's color space. Since the printer probably uses a device-dependent CMYK color space, all RGB values must be converted to CMYK. This is [color conversion](#). Once the values are specified in terms of the CMYK space, they need to be matched to the closest color that the printer can produce. This is called color mapping or [color matching](#).

Both color conversion and color mapping must take into account a number of device-specific factors. For instance, the building blocks of screen images are pixels. Each pixel has a set number of bits to store its color or color index value. The number of bits per pixel depends on the type of display and display adapter being used, and the mode to which that the adapter is set. Most every printer type uses different [colorants](#) and prints at particular resolutions.

In addition, the physical tone characteristics of a device are often different on different devices. For instance, when colors are produced by computer monitors, they can appear different than they would if they were produced on a printing press. A correction factor, called *gamma correction*, is frequently used to compensate for such differences.

The result of these device-dependent factors is that each color device has a particular set of colors that it can produce. This color set is known as its *gamut*. To perform color conversion and color mapping, the colors in an image must be converted from the color space and gamut of the source device into the color space of the destination device. They are then matched into the gamut of the destination device.

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Further Information

Article • 12/30/2021

Further explanation of the terms and concepts discussed in this color management overview can be found in the following sources:

- R.W.G. Hunt, *Measuring Color*, Prentice Hall 1991 (2nd edition).
- James D. Foley, *Computer Graphics: Principles and Practice*, Addison-Wesley 1990 (2nd edition).
- Roy Hall, *Illumination and Color in Computer Generated Imagery*, Springer Verlag 1988.
- Fred W. Billmeyer, Jr., Max Saltzmann, and Roy Berns, *Principles of Color Technology*, Wiley 1999 (3rd edition).

Another good source of information the Web page of the International Color Consortium (ICC) ([color.org](http://color.org)). The membership of the ICC includes leaders and standard-setting bodies of the computer and imaging industries.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Windows Color System Schemas and Algorithms

Article • 12/30/2021

This section contains extended descriptions of the schemas and algorithms that are used by the Windows Color System.

- [WCS Common Profile Types Schema, Versioning, and Localization Strategies](#)
- [Color Appearance Model Profile Schema and Algorithms](#)
- [Color Device Model Profile Schema and Algorithms](#)
- [Calibration Schema](#)
- [Gamut Map Model Profile Schema and Algorithms](#)
- [WCS Transform Creation Algorithms](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Windows Color System Common Profile Types schema, Versioning and Localization Strategies

Article • 01/26/2022

- WCS Common Profile Types Schema V1
- WCS Common Profile Types Schema V2 Additions
- WCS Profile Schema Versioning
- WCS Profile Localization
  - Expressing localizable elements
  - Schema Support
  - Selecting the Language
  - Built-in profiles
- Related topics

## WCS Common Profile Types Schema V1

The following provides the v1.0 schema definition for the WCS Common Profile Types.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xss:schema
  xmlns:xss="http://www.w3.org/2001/XMLSchema"

  xmlns:wcs="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfileTypes"
  targetNamespace="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfileTypes"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  blockDefault="#all"
  version="1.0">

  <xss:import namespace="http://www.w3.org/XML/1998/namespace" />

  <xss:annotation>
    <xss:documentation xml:lang="en">
      Common types used in WCS profile schemas.
      Copyright (C) Microsoft. All rights reserved.
    </xss:documentation>
  </xss:annotation>
```

```

<xs:simpleType name="NonNegativeFloatType">
  <xs:restriction base="xs:float">
    <xs:minInclusive value="0"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="NonNegativeCIEXYZType">
  <xs:attribute name="X" type="wcs:NonNegativeFloatType" use="required"/>
  <xs:attribute name="Y" type="wcs:NonNegativeFloatType" use="required"/>
  <xs:attribute name="Z" type="wcs:NonNegativeFloatType" use="required"/>
</xs:complexType>

<xs:simpleType name="GUIDType">
  <xs:restriction base="xs:string">
    <xs:pattern value="\{[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}\}"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="LocalizedTextType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute ref="xml:lang" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="MultiLocalizedTextType">
  <xs:sequence>
    <xs:element name="Text" type="wcs:LocalizedTextType" minOccurs="1" />
  </xs:sequence>
</xs:complexType>

</xs:schema>

```

Only UTF-8 or UTF-16 encodings are supported. All other XML encodings are strongly discouraged in order to preserve optimum interoperability.

## WCS Common Profile Types Schema V2 Additions

In Windows 7, the WCS Common Profile Types schema has been updated to include types to support the [WCS Calibration schema](#).

XML

```

<xs:complexType name="ParameterizedCurvesType">
  <xs:sequence>
    <xs:element name="RedTRC" type="wcs:ParameterizedCurveType"/>

```

```

<xs:element name="GreenTRC" type="wcs:ParameterizedCurveType"/>
<xs:element name="BlueTRC" type="wcs:ParameterizedCurveType"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="ParameterizedCurveType">
  <xs:attribute name="Gamma" type="wcs:NonNegativeFloatType"
use="required"/>
  <xs:attribute name="Offset1" type="xs:float" use="optional"/>
  <xs:attribute name="Gain" type="wcs:NonNegativeFloatType"
use="optional"/>
  <xs:attribute name="Offset2" type="xs:float " use="optional"/>
  <xs:attribute name="TransitionPoint" type="wcs:NonNegativeFloatType"
use="optional"/>
  <xs:attribute name="Offset3" type="xs:float" use="optional"/>
</xs:complexType>

<xs:simpleType name="FloatList">
  <xs:list itemType="xs:float"/>
</xs:simpleType>

<xs:complexType name="OneDimensionLutType">
  <xs:sequence>
    <xs:element name="Input" type="wcs:FloatList"/>
    <xs:element name="Output" type="wcs:FloatList"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="HDRToneResponseCurvesType">
  <xs:sequence>
    <xs:element name="RedTRC" type="wcs:OneDimensionLutType"/>
    <xs:element name="GreenTRC" type="wcs:OneDimensionLutType"/>
    <xs:element name="BlueTRC" type="wcs:OneDimensionLutType"/>
  </xs:sequence>
  <xs:attribute name="TRCLength" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:int">
        <xs:minInclusive value="2" />
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>

```

## WCS Profile Schema Versioning

In this Appendix, the term "profile consumer" refers to the WCS software, or to a third-party software component that consumes WCS profiles.

Newer versions of profile consumers will be able to consume WCS profiles written according to older versions of the schemas. To take full advantage of features defined in the newest version of the schemas, the newest version of a profile consumer will

generally be required. However, older profile consumers can consume profiles written according to newer versions of the schemas. The profile consumer can ignore XML elements and attributes that it does not understand. The results will be correct, but performance or fidelity may degraded as compared to the results achieved with the newest version of the profile consumer.

The following are versioning guidelines for profile consumers:

- A given version of a profile consumer must recognize either all of the elements and attributes from a version namespace, or none of them.
- If a profile consumer encounters an element or attribute from a namespace that it does not understand, it must treat this as an error condition, unless the profile contains explicit guidance on fallback processing.
- The [Open Packaging Specification](#) defines an additional namespace URI, the markup compatibility namespace, which defines elements and attributes that provide this fallback processing guidance.
- All versions of all profile consumers must recognize and respect all the elements and attributes in the markup compatibility namespace.
- Profile consumers based on a new version of the profile schemas must support all previously defined version namespaces.

In the v1.0 release, WCS recognizes elements and attributes from the following namespaces:

- `https://schemas.microsoft.com/windows/2005/02/color/ColorDeviceModel`
- `https://schemas.microsoft.com/windows/2005/02/color/GamutMapModel`
- `https://schemas.microsoft.com/windows/2005/02/color/ColorAppearanceModel`
- `https://schemas.microsoft.com/winfo/2005/06/markup-compatibility`

The following demonstrates an example of markup compatibility.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<ColorAppearanceModel

  xmlns=http://schemas.microsoft.com/windows/2005/02/color/ColorAppearanceMode
  l

  xmlns:cam2=http://schemas.microsoft.com/windows/2007/08/color/ColorAppeari
  eModel
  xmlns:mc=http://schemas.microsoft.com/winfo/2005/02/markup-compatibility

  xs:schemaLocation="http://schemas.microsoft.com/windows/2005/02/color/ColorA
  ppearanceModel ColorAppearanceModel.xsd"
  xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
```

```

mc:Ignorable="cam2">

<ProfileName>Default profile for ICC viewing conditions</ProfileName>
<mc:AlternateContent>
  <mc:Choice Requires="cam2">
    <cam2:AudioDescription source="ProfileExplanation.wav"/>
  </mc:Choice>
  <mc:Fallback>
    <Description>Appropriate for a print in a D50 viewing
booth</Description>
  </mc:Fallback>
</mc:AlternateContent>
<Author>Microsoft</Author>

<ViewingConditions>
  <WhitePointName>D50</WhitePointName>
  <Background X="19.3" Y="20.0" Z="16.5" />
  <Surround>Average</Surround>
  <LuminanceOfAdaptingField>31.8</LuminanceOfAdaptingField>
  <DegreeOfAdaptation>-1</DegreeOfAdaptation>
  <cam2:Humidity>30%</cam2:Humidity>
</ViewingConditions>

</ColorAppearanceModel>

```

# WCS Profile Localization

## Requirements

WCS profiles contain certain elements such as ProfileName and Description which contain human-readable text. This text is localizable.

The following are versioning guidelines for profile creators:

- For profiles created by 3rd parties such as printer manufacturers, the profile author should incorporate descriptive text in multiple languages.
- The following elements should be localizable:

Element	CDMP	GMMP	CAMP
ProfileName	x	x	x
Description	x	x	x
Author	x	x	x

# Expressing localizable elements

Instead of directly containing text, each localizable element contains one or more wcs:Text sub-elements, each of which must specify the xml:lang attribute. As described in the [XML specification](#) <sup>↗</sup> Section 2.12 ("Language Identification"), the value of the xml:lang attribute must be an IETF RFC 3066 language identifier such as "en-US", "en", or "fr-FR". In WCS schemas, the value of the xml:lang attribute must not be the empty string "", even though XML allows this in the general case. For example:

XML

```
<cdm:ColorDeviceModel ... />
  <cdm:Description>
    <wcs:Text xml:lang="en-US">Hello</wcs:Text>
    <wcs:Text xml:lang="fr-FR">Bonjour</wcs:Text>
  </cdm:Description>
  ...
</cdm:ColorDeviceModel>
```

No two wcs:Text elements can have the same xml:lang attribute. Each WCS profile schema must enforce this constraint separately for each localizable element. Only UTF-8 and UTF-16 encodings are supported.

## Schema Support

The design makes use of the XSD types wcs:LocalizedText and wcs:MultiLocalizedType defined in the WCS Common Profile Type schema:

XML

```
<xss:complexType name="LocalizedText">
  <xss:simpleContent>
    <xss:extension base="xss:string">
      <xss:attribute name="xml:lang" type="xss:string"
use="required"/>
      <xss:extension/>
    <xss:simpleContent/>
  </xss:complexType>

  <xss:complexType name="MultiLocalizedType">
    <xss:element name="Text" type="cam:LocalizedText" minOccurs="1"/>
  </xss:complexType>
```

Each WCS profile schema declares each localizable element to be of type MultiLocalizedType, for example:

## XML

```
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"

    xmlns:cdm="http://schemas.microsoft.com/windows/2005/02/color/ColorDeviceMod
    el"
    xmlns:wcs="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfi
    leTypes"
    targetNamespace=>

    "http://schemas.microsoft.com/windows/2005/02/color/ColorDeviceModel"
    version="1.0">
    <xs:import namespace=>

    "http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfileTypes"/>

        <xs:element name="cdm:Description" type="wcs:MultiLocalizableType"
minOccurs="0"/>

        <xs:unique name="uniqueDescriptionLanguage">
            <xs:selector xpath="cdm:ColorDeviceModel/cdm:Description/wcs:Text"/>
            <xs:field xpath="@xml:lang"/>
        </xs:unique>

        ...
    </xs:schema>
```

The CDMP schema enforces the constraint that each wcs:Text child of the cdm:Description element must have a unique xml:lang attribute. It enforces the same constraint for the other localizable elements. The CAMP and GMMP schemas do the same.

## Selecting the Language

When deciding which language version of a localizable element to display, application code should select the "closest version" to the "current language". What "closest version" and "current language" actually mean is platform-dependent; see below. If the profile does not contain a language version that matches the current language, the application should select the first version in the profile (the first wcs:Text child element of the localizable element).

On Windows Vista, the language selection is done as follows: Each thread has an associated list of "preferred UI languages", which can be obtained by calling [GetThreadPreferredUILanguages](#). The list is returned in order of user preference. For instance, on a system set up for US English, the list returned by

`GetThreadPreferredUILanguages` is { "en-US", "en" }. This means: 1) US English if present. 2) Otherwise, use any regional variant of English, such as "en-GB" or just plain "en".

For each language in that list, in list order, the UI code looks for a `wcs:Text` element whose `xml:lang` attribute is an exact match. The first matching version is used. If no version matches, the first `wcs:Text` element is used.

## Built-in profiles

The standard WCS profiles shipped with Windows Vista, such as `sRGB.cdm`, have the same localizable properties as other profiles.

The standard Windows solution would be to put the localized strings in a MUI DLL, and refer to them like this:

XML

```
<cdm:Description resourceId="mscms.dll,-101">
    <wcs:Text xml:lang="en-US">Hello, world!</wcs:Text>
<cdm:Description>
```

On a Windows system, the description text would be extracted from resource ID 101 in the appropriate localized version of the MUI resources for `mscms.dll`. Non-Windows systems would use the `wcs:Text` sub-elements as described above.

We introduce an optional, globally unique ID attribute on the root element of each WCS profile schema. The standard profile `wcsRGB.cdm` might look like this:

XML

```
<cdm:ColorDeviceModel
    ID="http://schemas.microsoft.com/2005/02/color/profiles/wcsRGB.cdm"
    ... >
    <cdm:Description>
        <wcs:Text xml:lang="en-US">Hello.</wcs:Text>
        <wcs:Text xml:lang="fr-FR">Bonjour.</wcs:Text>
    </cdm:Description>
    ...
</cdm:ColorDeviceModel>
```

For the WCS color profiles shipped with windows, the Color CPL displays descriptive information from the resources, rather than from the `wcs:Text` elements in the profiles. This allows Windows to display localized information for these profiles in all supported

languages, without requiring the profiles themselves to contain descriptive information in all languages.

## Related topics

[Basic color management concepts](#)

[Windows Color System Schemas and Algorithms](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WCS Color Appearance Model Profile Schema and Algorithm

Article • 12/30/2021

## [Overview](#)

### [Color Appearance Model Profile \(CAMP\) Architecture](#)

### [The CAMP Schema](#)

#### [The CAMP Schema Elements](#)

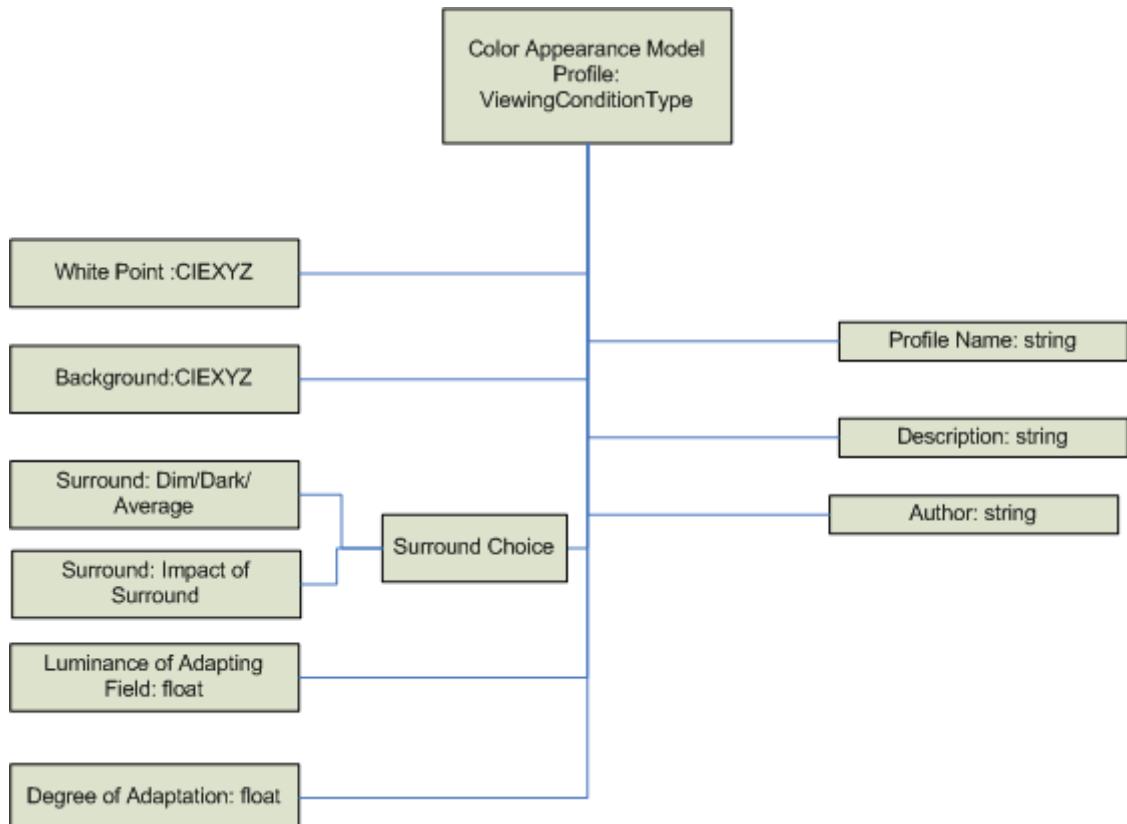
#### [The CAMP Algorithm](#)

## Overview

This schema is used to specify the content of a color appearance model profile (CAMP). The associated baseline algorithms are described in the following sections.

The CAMP is composed of XML tags that provide parametric values to the CIECAM02 baseline color appearance model variables. Details on the ranges for parameters are provided in the baseline color appearance model specification and CIECAM02 recommendation.

## Color Appearance Model Profile Architecture



## The CAMP Schema

C++

```

<?xml version="1.0" encoding="UTF-8"?>
<xss:schema

  xmlns:cam="http://schemas.microsoft.com/windows/2005/02/color/ColorAppearenceModel"

  xmlns:wcs="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfileTypes"

  targetNamespace="http://schemas.microsoft.com/windows/2005/02/color/ColorAppearenceModel"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  blockDefault="#all"
  version="1.0">

  <xss:annotation>
    <xss:documentation>
      Color Appearance Model profile schema.
      Copyright (C) Microsoft. All rights reserved.
    </xss:documentation>
  </xss:annotation>

  <xss:import
    namespace="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfile"

```

```

    leTypes" />

    <xs:annotation>
        <xs:documentation>
            ColorAppearanceModel element contains viewing conditions
            parameters based on CIECAM02.
        </xs:documentation>
    </xs:annotation>
    <xs:element name="ColorAppearanceModel">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ProfileName" type="wcs:MultiLocalizedTextType"/>
                <xs:element name="Description" type="wcs:MultiLocalizedTextType"
minOccurs="0"/>
                <xs:element name="Author" type="wcs:MultiLocalizedTextType"
minOccurs="0"/>
                <xs:element name="ViewingConditions">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:choice>
                                <xs:element name="WhitePoint"
type="wcs:NonNegativeCIEXYZType"/>
                                    <xs:element name="WhitePointName">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:string">
                                                <xs:enumeration value="D50"/>
                                                <xs:enumeration value="D65"/>
                                                <xs:enumeration value="A"/>
                                                <xs:enumeration value="F2"/>
                                            </xs:restriction>
                                        </xs:simpleType>
                                    </xs:element>
                            </xs:choice>
                            <xs:element name="Background"
type="wcs:NonNegativeCIEXYZType"/>
                                <xs:choice>
                                    <xs:element name="ImpactOfSurround" type="xs:float"/>
                                    <xs:element name="Surround">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:string">
                                                <xs:enumeration value="Average"/>
                                                <xs:enumeration value="Dim"/>
                                                <xs:enumeration value="Dark"/>
                                            </xs:restriction>
                                        </xs:simpleType>
                                    </xs:element>
                                </xs:choice>
                                <xs:element name="LuminanceOfAdaptingField" type="xs:float"/>
                                <xs:element name="DegreeOfAdaptation" type="xs:float"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="NormalizeToMediaWhitePoint" minOccurs="0">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">

```

```
<xs:enumeration value="True"/>
<xs:enumeration value="False"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
</xs:sequence>
<xs:attribute name="ID" type="xs:string" use="optional" />
</xs:complexType>
</xs:element>
</xs:schema>
```

## The CAMP Schema Elements

### ColorAppearanceModel

This element is a sequence of:

1. ProfileName string,
2. optional Description string,
3. optional Author string,
4. ViewingConditions element.

**Validation conditions:** Each sub-element is validated by its own type. String lengths are limited to 10,000 characters.

### Namespace

xmlns:cam="http://schemas.microsoft.com/windows/2005/02/color/ColorAppearanceModel"

targetNamespace="http://schemas.microsoft.com/windows/2005/02/color/ColorAppearanceModel"

### Version

Version >0.1 or <= "1.0" with the first release of Windows Vista.

**Validation conditions:** Any version value <=2.0 is also valid to support non-breaking changes to the format.

### Documentation

Color Appearance Model Profile Schema.

Copyright (C) Microsoft. All rights reserved.

**Validation conditions:** Each sub-element is validated by its own type.

## SurroundType

This element is either an enumeration of "Average," "Dim," or "Dark" CIECAM02 parameters or the actual quantitative parameters from the CIECAM02 recommendation c, impact of the surround.

**Validation conditions:** The c parameter can range from 0.525 to 0.69.

## ViewingConditions

This element consists of the following sub-elements:

Element	Type
WhitePoint	WhitePointType
Background	CIEXYZ
Surround	SurroundType
LuminanceOfAdaptingField	float
DegreeOfAdaptation	float
NormalizeToMediaWhitePoint	Boolean

**Validation conditions:** CIEXYZ sub-elements are validated by NonNegativeXYZType. The LuminanceOfAdaptingField is a maximum of 10,000cd/m<sup>2</sup>. The DegreeOfAdaptation can range from 0.0 to 1.0. The NormalizeToMediaWhitePoint value can be either "true" or "false." If the NormalizeToMediaWhitePoint sub-element is absent, it effectively defaults to "true." See the following CAMP algorithm section.

## WhitePointType

This element is either an enumeration of CIE light source value ("D50," "D65," "A," or "F2") or a CIEXYZ sub-element.

**Validation conditions:** Each sub-element is validated by its own type.

## CIEXYZType

The CIEXYZType element is composed of three NonNegativeFloatType single-precision IEEE floating-point elements, named "X," "Y," and "Z." These measurements can be either absolute (not relative) CIEXYZ 1931 reflective values or absolute (not relative) CIEXYZ 1931 direct (transmissive) values in candelas per meter squared units.

**Validation conditions:** This means that only real-world values are valid, and negative CIEXYZ measurement values are invalid. Since these are absolute values, values can range well beyond 1.0f. A reasonable limit for any X, Y, or Z value will be arbitrarily set to 10000.0f.

## The CAMP Algorithm

The color appearance model (CAM) is based on the CIE CIECAM02 color appearance model equations.

This class implements the color appearance modeling. Note that the WCS CAM is *not* replaceable, for example, using a plug-in. It is a design goal to have only one color appearance model. The CAM is based on CIECAM02 recommendations.

CIECAM02 can be used in two ways. In the colorimetric-to-appearance direction, it provides a mapping from CIE XYZ space to color appearance space. In the appearance-to-colorimetric direction, it maps from color appearance space back to XYZ space. The color appearance correlates lightness, J, chroma, C, and hue, h. These three values form a cylindrical coordinate system. Frequently, it turns out to be more convenient to work in a rectangular coordinate system, so compute  $a = C \cos h$  and  $b = C \sin h$ , to give CIECAM02 Jab.

You can use CAM lightness values greater than 100. The CIE committee that formulated CIECAM02 did not address the behavior of the lightness axis for input values with a luminance greater than the adopted white point; that is, for input Y values greater than the adopted white point Y value. Experimentation has shown that the luminance equations in CIECAM02 behave reasonably for such values. The lightness increases exponentially and follows the same exponent (roughly 1/3).

Users sometimes want to change the way that the degree of adaptation (D) is calculated. The WCS design enables users to control this calculation by changing the degreeOfadaptation value in the viewing conditions parameters.

To provide a more consistent match to users' ICC-influenced expectations, the degreeOfAdaptation in the default CAMPS is 1.0. This produces better results in all cases

other than MinCD Absolute, where one might want to let WCS compute the degreeOfAdaptation (via degreeOfAdaptation = -1).

Instead of using a surround value of "Average," "Dim," and "Dark," a continuous surround value, computed from the value c, is provided. The value of c must be a float between 0.525 and 0.69.

From c, Nc and F can be computed, using piecewise linear interpolation between the values already provided for "Average," "Dim," and "Dark." This models what is shown in Figure 1 of CIE 159:2004, the CIECAM02 specification.

degreeOfAdaption	Behavior
-1.0	$D = F \left[ 1 - \frac{1}{3.6} \right] e^{\left( \frac{-(L_4+42)}{92} \right)}$ This is the default CIECAM02 behavior.
0.0 <= degreeOfAdaption <= 1.0	$D = \text{degreeOfAdaptation}$ (the value supplied by the user)

A certain amount of error checking has also been added to the implementation. The following equation numbers are those used in the CIE 159:2004 definition of CIECAM02.

### ColorimetricToAppearanceColors

The input values are checked for reasonableness: If X or Z < 0.0, or if Y < -1.0, then the HRESULT is E\_INVALIDARG. If -1.0 <= Y < 0.0, then J, C, and h are all set to 0.0.

There are certain internal conditions that can produce error results. Instead of producing such results, the internal results are clipped to produce in-range values. This happens for specifications of colors that would be dark and impossibly chromatic: In equation 7.23, if A < 0, A = 0. In equation 7.26, if t < 0, t = 0.

### AppearanceToColorimetricColors

The input values are checked for reasonableness. If C < 0 , C > 300, or J > 500, then the HRESULT is E\_INVALIDARG.

$R'_{a,i}$ ,  $G'_{a,i}$ , and  $B'_{a,i}$  (equations 8.19 - 8.21) are clipped to the range 399.9 .

For all Color Appearance Model Profiles (CAMPs), the WCS engine will examine the adopted white point. If Y is not 100.0, then the adopted white point will be scaled so that Y does equal 100.0. The same scaling will be applied to the background value. The scaling factor is 100.0/adoptedWhitePoint.Y. The same scaling factor is applied to each of X, Y, and Z. If the NormalizeToMediaWhitePoint field is set to "True," or if it is absent

from the CAMP, the engine also scales all device colors input to DeviceToColorimetric so that the Y value of the device media white point equals 100.0. Device colors coming from ColorimetricToDevice will be scaled by the multiplicative inverse of that scaling factor. If the NormalizeToMediaWhitePoint flag is set to "False," then the colorimetric data is not scaled.

For some tasks, it makes sense to scale the colorimetric values coming from DeviceToColorimetric. The hyperbolic lightness equations in the CAM are really designed for a white point luminance of 100.0. The only place where a difference in the absolute luminance (or illuminance) comes into play is in the luminance of the adapting field. So the CAM must be initialized with a white point Y of 100.0. But if the device model's medium white point is being used as the adopted white point, then all colors coming from the device must be scaled accordingly, or device white will not come out with a J value of 100.0. So the Y values have to be scaled in the measurements. The measurement values could be scaled before initializing the device model. Then results would already be in the proper range. But that would make testing the device model more difficult, because the values coming out would require scaling. For tasks in which the device medium white point is perceived to be a true white, normalizing by the device media white point is desirable.

The CAM is initialized directly from the CAMP. This allows developers some flexibility in initializing the CAM, based upon the task they want to perform. In some tasks, observers will ignore any chroma in the media white points, because they cognitively "know" that the source and destination media are "white." In such cases, developers will want to initialize the forward and inverse CAMs with their respective media white points. In some cases, observers may be comparing the color of the media backgrounds. In these cases, it is advisable to use one CAM for both devices, and it may be desirable not to scale each device's colorimetric values by that device's medium white point. Then the different tristimulus values of the media will lead to different appearance values in CIECAM02.

## Related topics

[Basic color management concepts](#)

[Windows Color System Schemas and Algorithms](#)

---

## Feedback

Was this page helpful?

 Yes

 No

---

---

Get help at Microsoft Q&A

# WCS Calibration Schema

Article • 12/30/2021

This topic describes the WCS Calibration schema that expands the [WCS color device model profile](#).

## The WCS Calibration Schema

The following schema definition is used to specify the new Windows 7 definitions that support [WCS color device model profile](#) calibration.

C++

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (C) Microsoft. All rights reserved. The Windows Color System
    color profile schemas may be used according to the terms of the license
    agreement
    available at
    https://www.microsoft.com/whdc/device/display/color/wcs_license.mspx.
    -->
<xss:schema
    xmlns:cal="http://schemas.microsoft.com/windows/2007/11/color/Calibration"
    xmlns:wcs="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfi
    leTypes"

    targetNamespace="http://schemas.microsoft.com/windows/2007/11/color/Calibrat
    ion"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    blockDefault="#all"
    version="1.0">

    <xss:annotation>
        <xss:documentation>
            Color Device Model Calibration profile schema.
            Copyright (C) Microsoft. All rights reserved.
        </xss:documentation>
    </xss:annotation>

    <xss:import
        namespace="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfi
        leTypes" />

    <xss:complexType name="AdapterGammaConfiguration">
        <xss:choice>
            <xss:element name="ParameterizedCurves">
```

```
type="wcs:ParameterizedCurvesType"/>
    <xss:element name="HDRToneResponseCurves"
type="wcs:HDRToneResponseCurvesType"/>
</xss:choice>
</xss:complexType>

<xss:complexType name="Calibration">
    <xss:sequence>
        <xss:element name="AdapterGammaConfiguration"
type="cal:AdapterGammaConfiguration" minOccurs="0"/>
    </xss:sequence>
</xss:complexType>

</xss:schema>
```

For compatibility with Windows Vista, profiles containing calibration tags should include the attribute `mc:Ignoreable="cdm_calibration"`.

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# WCS Color Device Model Profile Schema and Algorithms

Article • 12/30/2021

This topic provides information about the WCS Color Device Model Profile Schema and its associated algorithms.

This topic contains the following sections:

- [Overview](#)
- [Color Device Model Profile Architecture](#)
- [The CDMP Schema](#)
- [WCS CDMP v2.0 Calibration Addition](#)
- [The CDMP Schema Elements](#)
  - [ColorDeviceModelProfile](#)
  - [ColorDeviceModel](#)
  - [NamespaceVersion](#)
  - [Version](#)
  - [Documentation](#)
  - [CRTDevice element](#)
  - [LCDDevice element](#)
  - [ProjectorDevice element](#)
  - [ScannerDevice element](#)
  - [CameraDevice element](#)
  - [RGBPrinterDevice element](#)
  - [CMYKPrinterDevice element](#)
  - [RGBVirtualDevice element](#)
  - [PlugInDeviceType](#)
  - [RGBVirtualMeasurementType](#)
  - [GammaType](#)
  - [GammaOffsetGainType](#)
  - [GammaOffsetGainLinearGainType](#)
  - [ToneResponseCurvesType](#)
  - [GamutBoundarySamplesType](#)
  - [FloatPairList](#)
  - [CMYKPrinterMeasurementType](#)
  - [RGBPrinterMeasurementType](#)
  - [RGBCaptureMeasurementType](#)
  - [OneBasedIndex](#)
  - [RGBProjectorMeasurementType](#)

- [DisplayMeasurementType](#)
- [MeasurementConditionsType](#)
- [GeometryType](#)
- [RGBPrimariesGroup](#)
- [NonNegativeCMYKSampleType](#)
- [NonNegativeRGBSampleType](#)
- [NonNegativeCMYKType](#)
- [NonNegativeRGBType](#)
- [ExtensionType](#)
- [NonNegativeXYZType](#)
- [XYZType](#)
- The CDMP Baseline Algorithms
  - [CRT Device Model Baseline](#)
  - [LCD Device Model Baseline](#)
  - [RGB Printer Device Model Baseline](#)
  - [RGB Virtual Device Model Baseline](#)
  - [CMYK Printer Device Model Baseline](#)
  - [RGB Projector Device Model Baseline](#)
  - [ICC Device Model Baseline](#)
- Related topics

## Overview

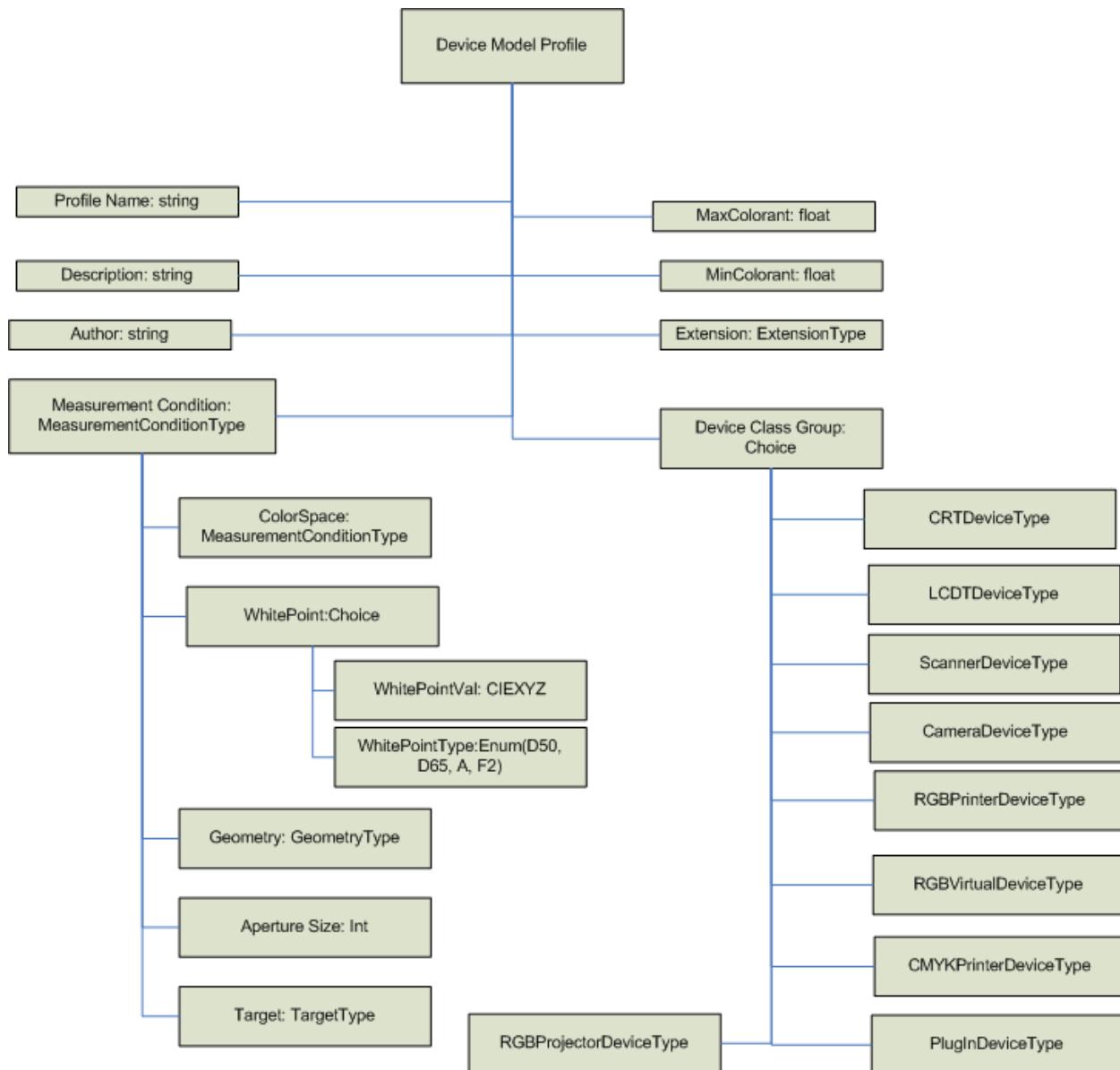
This schema is used to specify the content of a color device model profile(CDMP). The associated baseline algorithms are described below.

The basic device model profile (DMP) schema consists of the sampling measurement data.

The sampling component of the DMP XML schema provides support for basic color measurement targets, focusing on common standard targets and targets optimized for the baseline device models.

In addition, the device profile provides specific information on the targeted device model and provides a policy that the baseline fallback device model can use if the targeted model is unavailable. The profile instances can include private extensions using standard XML extension mechanisms.

## Color Device Model Profile Architecture



# The CDMP Schema

C++

```

<?xml version="1.0" encoding="UTF-8"?>
<xss:schema

xmlns:cdm="http://schemas.microsoft.com/windows/2005/02/color/ColorDeviceMod
el"

```

**xmlns:wcs="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfi
leTypes"**

**targetNamespace="http://schemas.microsoft.com/windows/2005/02/color/ColorDev
iceModel"**

**xmlns:xs="http://www.w3.org/2001/XMLSchema"**

**elementFormDefault="qualified"**

**attributeFormDefault="unqualified"**

**blockDefault="#all"**

**version="1.0">**

```

<xs:annotation>
  <xs:documentation>
    Color Device Model profile schema.
    Copyright (C) Microsoft. All rights reserved.
  </xs:documentation>
</xs:annotation>

<xs:import
namespace="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfileTypes" />

<xs:complexType name="RGBType">
  <xs:attribute name="R" type="xs:float" use="required"/>
  <xs:attribute name="G" type="xs:float" use="required"/>
  <xs:attribute name="B" type="xs:float" use="required"/>
</xs:complexType>

<xs:complexType name="NonNegativeRGBType">
  <xs:attribute name="R" type="wcs:NonNegativeFloatType" use="required"/>
  <xs:attribute name="G" type="wcs:NonNegativeFloatType" use="required"/>
  <xs:attribute name="B" type="wcs:NonNegativeFloatType" use="required"/>
</xs:complexType>

<xs:complexType name="NonNegativeCMYKType">
  <xs:attribute name="C" type="wcs:NonNegativeFloatType" use="required"/>
  <xs:attribute name="M" type="wcs:NonNegativeFloatType" use="required"/>
  <xs:attribute name="Y" type="wcs:NonNegativeFloatType" use="required"/>
  <xs:attribute name="K" type="wcs:NonNegativeFloatType" use="required"/>
</xs:complexType>

<xs:complexType name="NonNegativeRGBSampleType">
  <xs:sequence>
    <xs:element name="RGB" type="cdm:NonNegativeRGBType"/>
    <xs:element name="CIEXYZ" type="wcs:NonNegativeCIEXYZType"/>
  </xs:sequence>
  <xs:attribute name="Tag" type="xs:string" use="optional"/>
</xs:complexType>

<xs:complexType name="NonNegativeCMYKSampleType">
  <xs:sequence>
    <xs:element name="CMYK" type="cdm:NonNegativeCMYKType"/>
    <xs:element name="CIEXYZ" type="wcs:NonNegativeCIEXYZType"/>
  </xs:sequence>
  <xs:attribute name="Tag" type="xs:string" use="optional"/>
</xs:complexType>

<xs:group name="RGBPrimariesGroup">
  <xs:sequence>
    <xs:element name="WhitePrimary" type="wcs:NonNegativeCIEXYZType"/>
    <xs:element name="RedPrimary" type="wcs:NonNegativeCIEXYZType"/>
    <xs:element name="GreenPrimary" type="wcs:NonNegativeCIEXYZType"/>
    <xs:element name="BluePrimary" type="wcs:NonNegativeCIEXYZType"/>
    <xs:element name="BlackPrimary" type="wcs:NonNegativeCIEXYZType"/>
  </xs:sequence>

```

```

</xs:group>

<xs:complexType name="MeasurementConditionsType">
  <xs:annotation>
    <xs:documentation>
      Optional measurement conditions.

      We only support CIEXYZ for measurement color space in this version.
      If the white point value from the measurement conditions is not
      available,
      the default processing will use
      - "D50" for printer and scanners
      - "D65" for camera and displays.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="ColorSpace" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="CIEXYZ"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:choice minOccurs="0">
      <xs:element name="WhitePoint" type="wcs:NonNegativeCIEXYZType"/>
      <xs:element name="WhitePointName">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="D50"/>
            <xs:enumeration value="D65"/>
            <xs:enumeration value="A"/>
            <xs:enumeration value="F2"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:choice>
    <xs:element name="Geometry" minOccurs="0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="0/45"/>
          <xs:enumeration value="0/diffuse"/>
          <xs:enumeration value="diffuse/0"/>
          <xs:enumeration value="direct"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="ApertureSize" type="xs:int" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="DisplayMeasurementType">
  <xs:sequence>
    <xs:group ref="cdm:RGBPrimariesGroup"/>
    <xs:element name="GrayRamp">
      <xs:complexType>

```

```

<xs:sequence>
    <xs:element name="Sample" type="cdm:NonNegativeRGBSampleType"
maxOccurs="4096"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="RedRamp">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Sample" type="cdm:NonNegativeRGBSampleType"
maxOccurs="4096"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="GreenRamp">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Sample" type="cdm:NonNegativeRGBSampleType"
maxOccurs="4096"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="BlueRamp">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Sample" type="cdm:NonNegativeRGBSampleType"
maxOccurs="4096"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="TimeStamp" type="xs:dateTime"/>
</xs:complexType>

<xs:complexType name="RGBProjectorMeasurementType">
<xs:sequence>
    <xs:group ref="cdm:RGBPrimariesGroup"/>
    <xs:element name="ColorSamples">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Sample" type="cdm:NonNegativeRGBSampleType"
maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:sequence>
<xs:attribute name="TimeStamp" type="xs:dateTime"/>
</xs:complexType>

<xs:simpleType name="OneBasedIndex">
    <xs:restriction base="xs:int">
        <xs:minInclusive value="1"/>
    </xs:restriction>
</xs:simpleType>
```

```

<xs:complexType name="RGBCaptureMeasurementType">
  <xs:sequence>
    <xs:element name="PrimaryIndex">
      <xs:complexType>
        <xs:all>
          <xs:element name="White" type="cdm:OneBasedIndex"/>
          <xs:element name="Black" type="cdm:OneBasedIndex"
minOccurs="0"/>
          <xs:element name="Red" type="cdm:OneBasedIndex" minOccurs="0"/>
          <xs:element name="Green" type="cdm:OneBasedIndex"
minOccurs="0"/>
          <xs:element name="Blue" type="cdm:OneBasedIndex" minOccurs="0"/>
          <xs:element name="Cyan" type="cdm:OneBasedIndex" minOccurs="0"/>
          <xs:element name="Magenta" type="cdm:OneBasedIndex"
minOccurs="0"/>
          <xs:element name="Yellow" type="cdm:OneBasedIndex"
minOccurs="0"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
    <xs:element name="NeutralIndices">
      <xs:simpleType>
        <xs:list itemType="cdm:OneBasedIndex"/>
      </xs:simpleType>
    </xs:element>
    <xs:element name="ColorSamples">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Sample" type="cdm:NonNegativeRGBSampleType"
maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="TimeStamp" type="xs:dateTime"/>
</xs:complexType>

<xs:complexType name="RGBPrinterMeasurementType">
  <xs:sequence>
    <xs:element name="ColorCube">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Sample" type="cdm:NonNegativeRGBSampleType"
maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="TimeStamp" type="xs:dateTime"/>
</xs:complexType>

<xs:complexType name="CMYKPrinterMeasurementType">
  <xs:sequence>
    <xs:element name="ColorCube">
      <xs:complexType>

```

```

<xs:sequence>
    <xs:element name="Sample" type="cdm:NonNegativeCMYKSampleType"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="TimeStamp" type="xs:dateTime"/>
</xs:complexType>

<xs:complexType name="GammaType">
    <xs:attribute name="value" type="wcs:NonNegativeFloatType"
use="required"/>
</xs:complexType>

<xs:complexType name="GammaOffsetGainType">
    <xs:attribute name="Gamma" type="wcs:NonNegativeFloatType"
use="required"/>
    <xs:attribute name="Offset" type="wcs:NonNegativeFloatType"
use="required"/>
    <xs:attribute name="Gain" type="wcs:NonNegativeFloatType"
use="required"/>
</xs:complexType>

<xs:complexType name="GammaOffsetGainLinearGainType">
    <xs:attribute name="Gamma" type="wcs:NonNegativeFloatType"
use="required"/>
    <xs:attribute name="Offset" type="wcs:NonNegativeFloatType"
use="required"/>
    <xs:attribute name="Gain" type="wcs:NonNegativeFloatType"
use="required"/>
    <xs:attribute name="LinearGain" type="wcs:NonNegativeFloatType"
use="required"/>
    <xs:attribute name="TransitionPoint" type="wcs:NonNegativeFloatType"
use="required"/>
</xs:complexType>

<xs:simpleType name="FloatList">
    <xs:list itemType="xs:float"/>
</xs:simpleType>

<xs:complexType name="OneDimensionLutType">
    <xs:sequence>
        <xs:element name="Input" type="cdm:FloatList"/>
        <xs:element name="Output" type="cdm:FloatList"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="HDRToneResponseCurvesType">
    <xs:sequence>
        <xs:element name="RedTRC" type="cdm:OneDimensionLutType"/>
        <xs:element name="GreenTRC" type="cdm:OneDimensionLutType"/>
        <xs:element name="BlueTRC" type="cdm:OneDimensionLutType"/>
    </xs:sequence>
    <xs:attribute name="TRCLength" use="required">

```

```

<xs:simpleType>
  <xs:restriction base="xs:int">
    <xs:minInclusive value="0" />
  </xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>

<xs:complexType name="GamutBoundarySamplesType">
  <xs:sequence>
    <xs:element name="RGB" type="cdm:RGBType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="RGBVirtualMeasurementType">
  <xs:sequence>
    <xs:element name="MaxColorantUsed" type="xs:float"/>
    <xs:element name="MinColorantUsed" type="xs:float"/>
    <xs:group ref="cdm:RGBPrimariesGroup"/>
    <xs:choice>
      <xs:element name="Gamma" type="cdm:GammaType"/>
      <xs:element name="GammaOffsetGain" type="cdm:GammaOffsetGainType"/>
      <xs:element name="GammaOffsetGainLinearGain"
type="cdm:GammaOffsetGainLinearGainType"/>
      <xs:element name="HDR Tone Response Curves"
type="cdm:HDR Tone Response CurvesType"/>
      </xs:choice>
      <xs:element name="GamutBoundarySamples"
type="cdm:GamutBoundarySamplesType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="TimeStamp" type="xs:dateTime"/>
  </xs:complexType>

<xs:element name="ColorDeviceModel">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ProfileName" type="wcs:MultiLocalizedTextType"/>
      <xs:element name="Description" type="wcs:MultiLocalizedTextType"
minOccurs="0"/>
      <xs:element name="Author" type="wcs:MultiLocalizedTextType"
minOccurs="0"/>
      <xs:element name="MeasurementConditions"
type="cdm:MeasurementConditionsType" minOccurs="0"/>
      <xs:element name="SelfLuminous" type="xs:boolean" />
      <xs:element name="MaxColorant" type="xs:float"/>
      <xs:element name="MinColorant" type="xs:float"/>
      <xs:choice>
        <xs:element name="CRTDevice">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="MeasurementData"
type="cdm:DisplayMeasurementType"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```
<xs:element name="LCDDevice">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MeasurementData"
type="cdm:DisplayMeasurementType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="RGBProjectorDevice">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MeasurementData"
type="cdm:RGBProjectorMeasurementType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ScannerDevice">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MeasurementData"
type="cdm:RGBCaptureMeasurementType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="CameraDevice">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MeasurementData"
type="cdm:RGBCaptureMeasurementType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="RGBPrinterDevice">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MeasurementData"
type="cdm:RGBPrinterMeasurementType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="CMYKPrinterDevice">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MeasurementData"
type="cdm:CMYKPrinterMeasurementType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="RGBVirtualDevice">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MeasurementData"
type="cdm:RGBVirtualMeasurementType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

</xs:element>
</xs:choice>
<xs:element name="PlugInDevice" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace="#other" processContents="skip"
        minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="GUID" type="wcs:GUIDType" use="required"/>
  </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="ID" type="xs:string" use="optional" />
</xs:complexType>
</xs:element>
</xs:schema>

```

## WCS CDMP v2.0 Calibration Addition

The **ColorDeviceModel** element of the CDMP schema has been updated in Windows 7 to include the new calibration element. The following shows the change to the CDMP schema.

C++

```

...
<xs:element name="ColorDeviceModel">
  <xs:complexType>
    <xs:sequence>
      ...
      <xs:element name="PlugInDevice" minOccurs="0">
        ...
      </xs:element>
      <xs:element name="Calibration" type="cal:Calibration"
minOccurs="0"/>
        ...
      <xs:sequence>
        ...
      <xs:complexType>
      ...

```

## The CDMP Schema Elements

 Note

Primaries are primary samples of red, green, blue, black, and white. A primary ramp is a tonal ramp from least luminance to full primary value. The maximum number of entries in a tone ramp is 4096.

 **Note**

DMPs are required to have measurement data.

## ColorDeviceModelProfile

This element is of type ColorDeviceModel.

**Validation conditions:** Each sub-element is validated by its own type.

## ColorDeviceModel

This element is a sequence of the following sub-elements

1. ProfileName string,
2. optional Description string,
3. optional Author string,
4. optional MeasurementConditions MeasurementConditionsType,
5. Self-Luminous Boolean,
6. MaxColorant float,
7. MinColorant float,
8. Choice of elements
  - a. CRTDevice,
  - b. LCDDevice,
  - c. RGBProjectorDevice,
  - d. ScannerDevice,
  - e. CameraDevice,
  - f. RGBPrinterDevice,
  - g. CMYKPrinterDevice,
  - h. RGBVirtualDevice,
9. PlugInDevice,
10. optional Extension ExtensionType

**Validation conditions:** Each sub-element is validated by its own type. String sub-elements have a maximum of 10,000 characters. The MaxColorant sub-element must be

greater than or equal to zero (0) and greater than the MinColorant sub-element. The MinColorant can be negative.

## NamespaceVersion

```
xmlns:cdm="http://schemas.microsoft.com/windows/2005/02/color/ColorDeviceModel"  
targetNamespace="http://schemas.microsoft.com/windows/2005/02/color/ColorDevice  
Model"
```

## Version

Version = "1.0" with Windows Vista.

**Validation conditions:** Any version value >0.1 or <=2.0 is valid to support non-breaking changes to the format.

## Documentation

Device Model Profile schema.

Copyright (C) Microsoft. All rights reserved.

## CRTDevice element

This element is a sequence of sub-elements of a MeasurementData DisplayMeasurementType.

**Validation conditions:** Each sub-element is validated by its own type.

## LCDDevice element

This element is a sequence of sub-elements of a MeasurementData DisplayMeasurementType.

**Validation conditions:** Each sub-element is validated by its own type.

## ProjectorDevice element

This element is a sequence of sub-elements of a MeasurementData RGBProjectorMeasurementType.

**Validation conditions:** Each sub-element is validated by its own type.

## ScannerDevice element

This element is a sequence of sub-elements of a MeasurementData  
RGBCaptureMeasurementType

**Validation conditions:** Each sub-element is validated by its own type.

## CameraDevice element

This element is a sequence of sub-elements of a MeasurementData  
RGBCaptureMeasurementType

**Validation conditions:** Each sub-element is validated by its own type.

## RGBPrinterDevice element

This element is a sequence of sub-elements of a MeasurementData  
RGBPrinterMeasurementType.

**Validation conditions:** Each sub-element is validated by its own type.

## CMYKPrinterDevice element

This element is a sequence of sub-elements of a MeasurementData  
CMYKPrinterMeasurementType.

**Validation conditions:** Each sub-element is validated by its own type.

## RGBVirtualDevice element

This element is a sequence of sub-elements of a RGBVirtualMeasurementDataType.

**Validation conditions:** Each sub-element is validated by its own type.

## PlugInDeviceType

This element is a sequence of a GUID GUIDType and any sub-elements.

**Validation conditions:** The GUID is used to match the DM PlugIn Dll GUID. There are a maximum of 100,000 custom sub-elements.

## RGBVirtualMeasurementType

This element is a sequence consisting of

1. RGBPrimariesGroup group
2. A choice of
3.
  - Gamma
  - GammaOffsetGain
  - GammaOffsetGainLinearGam
  - ToneResponseCurves elements
4. optional GamutBoundarySamples GamutBoundarySamplesType
5. TimeStamp dateTime

**Validation conditions:** Each sub-element of these types has its own validation conditions.

## GammaType

This element is a complex type consisting of the attribute

- Gamma NonNegativeFloatType

**Validation conditions:** To be determined from industry feedback.

## GammaOffsetGainType

This element is a complex type consisting of the attributes

- Gamma NonNegativeFloatType
- Offset NonNegativeFloatType
- Gain NonNegativeFloatType

**Validation conditions:** To be determined from industry feedback.

## GammaOffsetGainLinearGainType

This element is a complex type consisting of the attributes

- Gamma NonNegativeFloatType
- Offset NonNegativeFloatType

- Gain NonNegativeFloatType
- LinearGain NonNegativeFloatType
- TransitionPoint NonNegativeFloatType.

**Validation conditions:** To be determined from industry feedback.

## ToneResponseCurvesType

This element is a sequence of

1. RedTRC FloatPairList
2. GreenTRC FloatPairList
3. BlueTRC FloatPairList

The element also has an attribute TRCLength of unsignedint type.

**Validation conditions:** To be determined from industry feedback.

## GamutBoundarySamplesType

This element is a sequence of RGB RGBTypes.

**Validation conditions:** Currently unbounded maximum occurrences, to be capped based on industry feedback.

## FloatPairList

This element is a simple type of list of pairs of floats.

**Validation conditions:** To be determined from industry feedback.

## CMYKPrinterMeasurementType

This element is a

1. sequence of ColorCube element consisting of a sequence of Sample NonNegativeCMYKSampleType
2. TimeStamp dateTime attribute.

**Validation conditions:** To be determined from industry feedback.

## RGBPrinterMeasurementType

This element is a

1. sequence of ColorCube element consisting of a sequence of Sample NonNegativeRGBSampleType
2. TimeStamp dateTime attribute.

**Validation conditions:** To be determined from industry feedback.

## RGBCaptureMeasurementType

This element is a sequence of

1. PrimaryIndex complexType of
  2. a. White OneBasedIndex
  - b. Optional Black OneBasedIndex
  - c. Optional Red OneBasedIndex
  - d. Optional Green OneBasedIndex
  - e. Optional Blue OneBasedIndex
  - f. Optional Cyan OneBasedIndex
  - g. Optional Magenta OneBasedIndex
  - h. Optional Yellow OneBasedIndex
3. NeutralIndices of lines of OneBasedIndex
  4. ColorSamples sequence of Sample NonNegativeRGBSampleType

The element also has a TimeStamp dateTime attribute.

**Validation conditions:** To be determined from industry feedback.

## OneBasedIndex

This element is a simple type of restriction base unsigned int with minInclusive value = "1."

**Validation conditions:** To be determined from industry feedback.

## RGBProjectorMeasurementType

This element is a sequence of

1. RGBPrimariesGroup group

2. element ColorSamples consisting of sequence of Sample NonNegativeRGBSampleType

The element also has a TimeStamp dateTime attribute.

**Validation conditions:** To be determined from industry feedback.

## DisplayMeasurementType

This element is a sequence of

1. group RGBPrimariesGroup
2. GrayRamp of sequence of Sample NonNegativeRGBType
3. RedRamp of sequence of Sample NonNegativeRGBType
4. GreenRamp of sequence of Sample NonNegativeRGBType
5. BlueRamp of sequence of Sample NonNegativeRGBType

The DisplayMeasurementType element also has a TimeStamp dateTime attribute.

**Validation conditions:** To be determined from industry feedback.

## MeasurementConditionsType

The MeasurementConditionsType is a sequence of sub-elements that contains:

1. ColorSpace restricted string enumeration value of "CIEXYZ"
2. optional choice of WhitePoint NonNegativeXYZType or WhitePointName string enumeration of values D50, D65, A, or F2
3. Geometry GeometryType
4. ApertureSize integer in millimeters

Defaults are:

1. RGB and CMYK Printers:
  - a. CIEXYZ MeasurementSpaceType
  - b. D50 WhitePointValue
  - c. 0/45 GeometryType
  - d. 10mm ApertureSize
2. Scanners:
  - a. CIEXYZ MeasurementSpaceType
  - b. D50 WhitePointValue
  - c. 0/45 GeometryType
  - d. 10mm ApertureSize

3. Displays and RGB Virtual Device:

- a. CIEXYZ MeasurementSpaceType
- b. D65 WhitePointValue
- c. 0/45 GeometryType
- d. 10mm ApertureSize

4. Cameras:

- a. CIEXYZ MeasurementSpaceType
- b. D65 WhitePointValue
- c. Direct GeometryType
- d. 10mm ApertureSize

**Validation conditions:** Validation of each sub-element is determined by validation conditions for those sub-elements. If any sub-element is missing, the device model type specific default is used.

## GeometryType

String

Enumeration values:

- "0/45"
- "0/diffuse"
- "diffuse/0"
- "Direct"

**Validation conditions:** Any value except the enumeration values listed is invalid. This information will not change baseline processing behavior.

## RGBPrimariesGroup

This element is a sequence of

1. WhitePrimary NonNegativeXYZType
2. RedPrimary NonNegativeXYZType
3. GreenPrimary NonNegativeXYZType
4. BluePrimary NonNegativeXYZType
5. BlackPrimary NonNegativeXYZType

**Validation conditions:** To be determined from industry feedback.

## NonNegativeCMYKSampleType

This element is a sequence of

1. CMYK NonNegativeCMYKType
2. CIEXYZ NonNegativeXYZType

The element also has an optional attribute Tag string

**Validation conditions:** To be determined from industry feedback.

## NonNegativeRGBSampleType

This element is a sequence of

1. RGB NonNegativeRGBType
2. CIEXYZ NonNegativeXYZType

The element also has an optional attribute Tag string

**Validation conditions:** To be determined from industry feedback.

## NonNegativeCMYKType

This element consisting of attributes

1. C float
2. M float
3. Y float
4. K float

**Validation conditions:** To be determined from industry feedback.

## NonNegativeRGBType

This element consisting of attributes

1. R float
2. G float
3. B float

**Validation conditions:** To be determined from industry feedback.

## ExtensionType

The ExtensionType element is a sequence of any sub-element type and is used for proprietary information from non-Microsoft applications.

**Validation conditions:** This element is optional. There can be a maximum of 1000 extension sub-elements.

## NonNegativeXYZType

The NonNegativeXYZType element is composed of NonNegativeFloatType three single-precision IEEE floating-point elements named "X," "Y," and "Z." These values are limited to the DMP profiles measurement values. These measurements can be either absolute (not relative) CIEXYZ 1931 reflective values or absolute (not relative) CIEXYZ 1931 direct (transmissive) values in candelas per meter squared units.

**Validation conditions:** Only real-world values are valid, and negative CIEXYZ measurement values are invalid. Because these are absolute values, values can be greater than 1.0f. A reasonable limit for any "X," "Y," or "Z." value is arbitrarily set to 10000.0f.

## XYZType

The XYZType element is composed of three single-precision IEEE floating-point values: "X," "Y," and "Z."

# The CDMP Baseline Algorithms

## CRT Device Model Baseline

To understand this model, you must consider both the characterization process and the device modeling. In the characterization process, XYZ measurements are first performed on the colors obtained by filling the display buffer of a CRT display device. The following example values will generate good data for the baseline CRT device model:

Red: R = 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, G = B = 0

Green: G = 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, R = B = 0

Blue: B = 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, R = G = 0

Neutrals: R = G = B = 0, 8, 16, 32, 64, 128, 192, 255

Increments other than 15 and nonlinear increments can also be used. Each red, green, blue, and neutral ramp must contain at least three samples, but providing more samples is recommended. You must provide samples for pure red, green, blue, black, and white. The samples do not have to be uniformly spaced.

The process of building the tristimulus matrix consists of two steps. First, estimate the black point XYZ value, or the flare. This step is based largely on work of Berns[3] with a slightly modified objective function for the nonlinear optimization. Second, calculate tristimulus matrix based on the result from step one and also from an averaging calculation on all of the per-channel measurements, not just the one for maximum digital count.

Each of these steps contains detailed procedures. The starting point is the ramps (17 steps in our example) for each of R, G, and B channels. When the XYZ measurements are plotted on the chromaticity  $xy$ -plane, a typical situation is shown in Figure 1. Step one consists of solving a nonlinear optimization problem to find the "best fit" black point that will minimize the drift in chromaticity as one traverses along the R, G, and B channels. Based on Berns[3], we seek  $(X_K, Y_K, Z_K)$  that minimizes the following objective function:

$$\Phi(X_K, Y_K, Z_K) = \Psi(X_K, Y_K, Z_K; S_R) + \Psi(X_K, Y_K, Z_K; S_G) + \Psi(X_K, Y_K, Z_K; S_B),$$

where  $S_R, S_G$ , and  $S_B$  are the set of data points corresponding to the points on the R, G, and B channels. For any set  $S$ , define:

$$\Psi(X_K, Y_K, Z_K; S) = \sum_{i \in S} \left\| f(X_i, Y_i, Z_i; X_K, Y_K, Z_K) - \bar{f}(X_K, Y_K, Z_K; S) \right\|^2,$$

$$f(X, Y, Z; X_K, Y_K, Z_K) = \begin{pmatrix} X - X_K \\ X - X_K + Y - Y_K + Z - Z_K \\ X - X_K + Y - Y_K + Z - Z_K \end{pmatrix},$$

$$\bar{f}(X_K, Y_K, Z_K; S) = \frac{1}{|S|} \sum_{i \in S} f(X_i, Y_i, Z_i; X_K, Y_K, Z_K).$$

In the preceding,  $|S|$  is the cardinality of  $S$ , i.e., the number of points in the set  $S$ .

$f(X, Y, Z; X_K, Y_K, Z_K)$  is the chromaticity coordinates of the point

$(X - X_K, Y - Y_K, Z - Z_K)$ , so  $\bar{f}(X_K, Y_K, Z_K; S)$ , is the average, or center of mass, of all the points in the set  $S$  in the chromaticity plane. Thus,  $\Psi(X_K, Y_K, Z_K; S)$  is the sum of second moments of the points about the center of mass and is a measure of how

spread out the points are about it. Finally,  $\Phi(X_K, Y_K, Z_K)$  is a total measure of how spread out the three clusters of points are about their respective centers of mass.

In the calculation of  $f(X, Y, Z; X_K, Y_K, Z_K)$ , if  $X = X_K, Y = Y_K, Z = Z_K$ , then the calculation is skipped, and the cardinality of  $S$  is adjusted accordingly.

Despite the apparent complexity of the objective function, it is a sum of the squares of many differentiable functions in  $X_K, Y_K, Z_K$  (17 points 2xy-components 3 channels = 102, in the example), and, therefore, is amenable to standard nonlinear least squares techniques, such as the Levenberg-Marquardt algorithm, which is the algorithm used in WCS. Note that the preceding objective function is different from the one suggested in Berns[3] in that the latter function measures the variance of the distances from the center of mass, so that the variance is zero when the points are equidistant from the center of mass, even though they may spread out quite a bit about it. In the example, the dispersion of points is controlled directly using the second moments.

As with any iterative algorithm for the nonlinear least squares problem, Levenberg-Marquardt requires an initial guess. There are two obvious candidates. One is (0, 0, 0); the other is the measured black point. For the CTE, the measured black point is first used as the initial guess. If a maximum of 100 iterations is exceeded without achieving a threshold of an average distance of 0.001 of each point from its center of mass (which corresponds to a threshold value of  $(0.001)^2 \cdot 17 \cdot 3 = 0.000051$  for the objective function), then another round of iterations with the initial guess of (0, 0, 0) is performed. The resulting estimate of the black point is XYZ compared with the best estimate from the previous round of iterations (with the measured black point as the initial guess). Use the estimate that gives the smallest value for the objective function. The choice of 100 iterations and the error distance of 0.001 were each selected empirically. In future versions, it might be reasonable to parameterize the error distance.

The result of step one is the estimated black point ( $X_K, Y_K, Z_K$ ). Step two consists of determining the tristimulus matrix by averaging the chromaticity of the points in the three clusters obtained in step one. For CRTs, this is done primarily to minimize the effects of measurement errors. The points used in averaging the chromaticity must be the same points used in the optimization in step one. In other words, if the first point (digital count 15, in the example) in each ramp is discarded in the optimization step, then the same must be done in the averaging. If  $(\bar{x}_r, \bar{y}_r)$ ,  $(\bar{x}_g, \bar{y}_g)$ , and  $(\bar{x}_b, \bar{y}_b)$  are the averaged chromaticity coordinates of the red, green, and blue channels, then the following procedure determines the tristimulus matrix. First, solve the 3?3 linear system:

$$\begin{pmatrix} \bar{x}_r & \bar{x}_g & \bar{x}_b \\ \bar{y}_r & \bar{y}_g & \bar{y}_b \\ 1 - \bar{x}_r - \bar{y}_r & 1 - \bar{x}_g - \bar{y}_g & 1 - \bar{x}_b - \bar{y}_b \end{pmatrix} \begin{pmatrix} t_r \\ t_g \\ t_b \end{pmatrix} = \begin{pmatrix} X_W \\ Y_W \\ Z_W \end{pmatrix},$$

$t_r, t_g, t_b$

$X_W, Y_W, Z_W$

$$\begin{pmatrix} t_r \bar{x}_r & t_g \bar{x}_g & t_b \bar{x}_b \\ t_r \bar{y}_r & t_g \bar{y}_g & t_b \bar{y}_b \\ t_r(1 - \bar{x}_r - \bar{y}_r) & t_g(1 - \bar{x}_g - \bar{y}_g) & t_b(1 - \bar{x}_b - \bar{y}_b) \end{pmatrix}.$$

After the tristimulus matrix is determined, the determination of tone curves follows the standard approach. For CRT displays, the individual channels are assumed to follow the "GOG" model:

$$f(x) = ((1 - k_g) + k_g x)^{\gamma},$$

where  $k_g$  is the "gain",  $1 - k_g$  is the "offset", and  $\gamma$  is the "gamma." The inverse matrix of the tristimulus matrix is applied to the XYZ data of the neutrals to obtain the linear RGB data, which is then correlated with the digital RGB values using nonlinear regression on the GOG model. These characteristics do not have to be the same for the R, G, and B channels, and generally are not the same.

Berns[1]: Berns, Billmeyer and Saltzman's *Principles of Color Technology*, 3<sup>rd</sup> Ed. John Wiley & Sons (2000).

Berns[2]: Berns and Katoh, The digital to radiometric transfer function for computer controlled CRT displays, *CIE Expert Symposium '97 Colour Standards for Imaging Technology*, Nov. 1997.

Berns[3]: Berns, Fernandez and Taplin, Estimating Black-Level Emissions of Computer-Controlled Displays, *Color Research and Application*, 28: 379-383 Wiley Periodicals, Inc. (2003)

Kang[1]: Kang, *Color Technology for Electronic Imaging Devices*, SPIE (1997)

Katoh[1]: Katoh, Deguchi and Berns, An accurate characterization of CRT monitor (II) proposal for an extension to CIE method and its verification, *Opt. Rev.* 8: 397-408 (2001)

## LCD Device Model Baseline

The LCD Device Model Baseline is similar to the CRT Device Model Baseline. This section will explain the ways in which LCD modeling differs from CRT modeling.

One difference is that you cannot assume that LCD displays follow the GOG model used for CRTs, and the tone curves are obtained by interpolation of measured data. Because of that, the device neutral axis should be sampled more frequently.

Here are some typical example values that can generate good data for the LCD device model baseline:

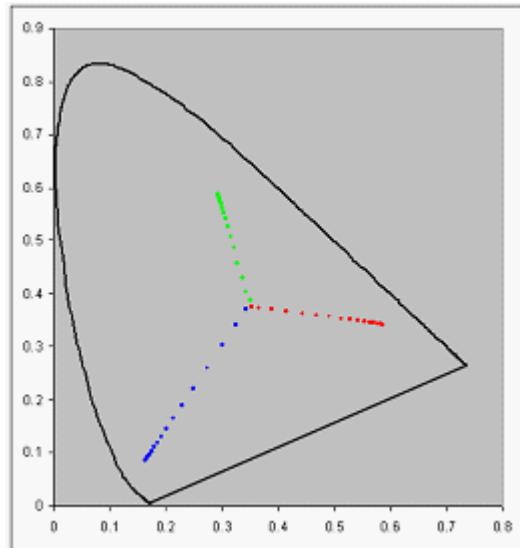
Red: R = 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, G = B = 0

Green: G = 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, R = B = 0

Blue: B = 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, R = G = 0

Neutrals: R = G = B = 0, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255.

The process of averaging measured color chromaticities to obtain the chromaticities for the device primaries is more critical for LCDs than it is for CRTs. When XYZ measurements are plotted on the chromaticity  $xy$ -plane, a typical situation is shown in Figure 1. Notice how the chromaticity drifts toward the black point. This is because all LCDs have a certain amount of light leakage.



**Figure 1 :** The chromaticity diagram using raw data with no correction

When this is subtracted from the raw XYZ measurements, a typical situation is depicted in Figure 2. The points are now clustered about three centers, although they don't fall identically on them. The averaging process described for CRTs greatly improves the results for LCDs.

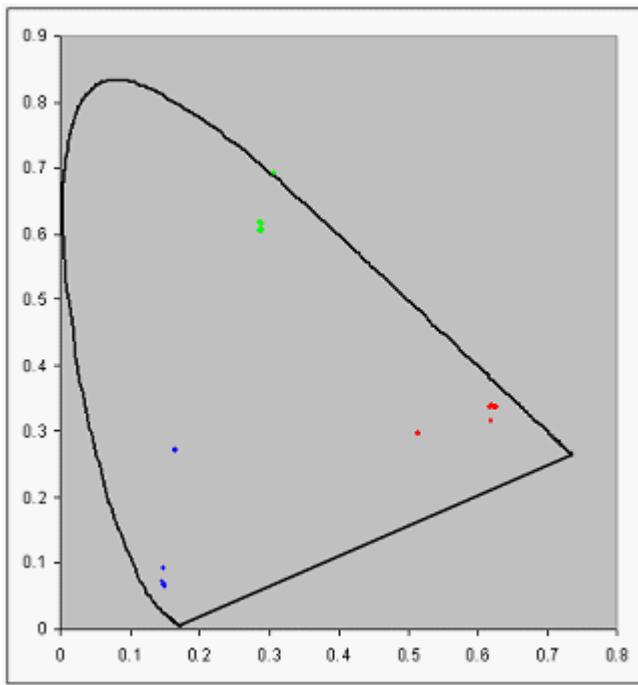


Figure 2 : The chromaticity diagram using data with adjusted black point

## RGB Capture Device Model Baseline

The baseline RGB capture device model is a subclass of the `IDeviceModel` class. In the colorimetric characterization of color capture devices, such as scanners and digital cameras, the following approach is used. A target consisting of color patches with known CIEXYZ values is captured using the capture device. The result of the capture is an RGB bitmap image in which the color of each patch is encoded in an RGB value. These device RGB values are specific to a particular capture device. The goal of colorimetric characterization is to establish an empirical relationship between the device RGB values and CIEXYZ values, or a mathematical transformation from RGB to XYZ that models as accurately as possible the behavior of the capture device.

Such a mathematical transformation can be modeled reasonably by polynomials of low degrees. This procedure is detailed in the literature, for example Kang[92], Kang[97]. In Kang[97], an approach is reported that uses a set of three polynomials with 3, 6, 8, 9, 11, 14 or 20 terms in the R, G, and B variables, while the three polynomials regress respectively into the X, Y, Z components of the CIEXYZ space. For the 20-term polynomial, the form is:

$$X = a_1 + a_2R + a_3G + a_4B + a_5R^2 + a_6RG + a_7RB + a_8G^2 + a_9GB + a_{10}B^2 + a_{11}R^3 + a_{12}R^2G + a_{13}R^2B + a_{14}RG^2 + a_{15}RGB + a_{16}RB^2 + a_{17}G^3 + a_{18}G^2B + a_{19}GB^2 + a_{20}B^3$$

There are similar expressions for Y and Z. The mathematical technique for fitting the polynomials falls within "Multivariate Linear Regression" and is described in any elementary text in Statistics.

This method of linear regression suffers from not minimizing the "right" objective function. By design, linear regression finds the least squares solution, which implies that the coefficients obtained will minimize the total sum of squares of errors in the underlying space, or equivalently, the sum of squares of the Euclidean distances. In practice, you want to minimize the sum of squares of  $\Delta E$ s, where  $\Delta E$  is one of the accepted standards such as CIE94. Minimizing this objective function is a nonlinear regression problem.

In the new engine, Lab to XYZ is the CIE color space that is regressed into, and the 20-term cubic polynomial is used as the model for the capture device, or coefficients  $\lambda_{1,2,3,4}$ ,  $\alpha_{1,2,3,4}$ ,  $\beta_{1,2,3,4}$  such that the following polynomials minimize the sum of squares of  $\Delta E_{CIE94}$ s.

$$\begin{aligned}\hat{L}(R, G, B) &= \lambda_1 + \lambda_2 R + \lambda_3 G + \lambda_4 B \\ &\quad + \lambda_5 R^2 + \lambda_6 RG + \lambda_7 RB + \lambda_8 G^2 + \lambda_9 GB + \lambda_{10} B^2 \\ &\quad + \lambda_{11} R^3 + \lambda_{12} R^2 G + \lambda_{13} R^2 B + \lambda_{14} RG^2 + \lambda_{15} RGB \\ &\quad + \lambda_{16} RB^2 + \lambda_{17} G^3 + \lambda_{18} G^2 B + \lambda_{19} GB^2 + \lambda_{20} B^3 \\ \hat{u}(R, G, B) &= \alpha_1 + \alpha_2 R + \alpha_3 G + \alpha_4 B \\ &\quad + \alpha_5 R^2 + \alpha_6 RG + \alpha_7 RB + \alpha_8 G^2 + \alpha_9 GB + \alpha_{10} B^2 \\ &\quad + \alpha_{11} R^3 + \alpha_{12} R^2 G + \alpha_{13} R^2 B + \alpha_{14} RG^2 + \alpha_{15} RGB \\ &\quad + \alpha_{16} RB^2 + \alpha_{17} G^3 + \alpha_{18} G^2 B + \alpha_{19} GB^2 + \alpha_{20} B^3 \\ \hat{v}(R, G, B) &= \beta_1 + \beta_2 R + \beta_3 G + \beta_4 B \\ &\quad + \beta_5 R^2 + \beta_6 RG + \beta_7 RB + \beta_8 G^2 + \beta_9 GB + \beta_{10} B^2 \\ &\quad + \beta_{11} R^3 + \beta_{12} R^2 G + \beta_{13} R^2 B + \beta_{14} RG^2 + \beta_{15} RGB \\ &\quad + \beta_{16} RB^2 + \beta_{17} G^3 + \beta_{18} G^2 B + \beta_{19} GB^2 + \beta_{20} B^3\end{aligned}$$

The solution  $(l_i, a_i, b_i)$  in the 60-dimensional real numeric space  $\mathbb{R}^{203}$  must be such that the following total error is minimized:

$$\Psi(\lambda_i, \alpha_i, \beta_i) = \sum_i \Delta E_{CIE94}^2(\hat{L}(R_i, G_i, B_i), \hat{u}(R_i, G_i, B_i), \hat{v}(R_i, G_i, B_i); L_i, u_i, v_i)$$

where the summation is through all the data point pairs  $(R_i, G_i, B_i; L_i, u_i, v_i)$  in the sampled data set plus additional control points to be detailed in the following. This is a nonlinear regression problem because the parameters  $\lambda_i, \alpha_i, \beta_i$  enter into the objective function in a nonlinear way (not quadratically).

Because the objective function  $\Psi$  is a nonlinear (and nonquadratic) function of the parameters  $\lambda_i, \alpha_i$  and  $\beta_i$ , you must resort to iterative techniques to solve the optimization problem. Because the form of the objective function is a sum of squares, a standard optimization technique called the Levenberg-Marquardt algorithm is used. It is considered the method of choice for nonlinear least squares problems. For iterative algorithms such as Levenberg-Marquardt, you must supply an initial guess. A good

initial guess is usually critical in finding the correct minimum value. In this case, one good candidate for the initial guess is the solution of the linear regression problem. First, minimize the sum of the square of Euclidean distances in Lab space, by defining a quadratic objective function:

$$\Theta(\lambda_i, \alpha_i, \beta_i) = \sum_i \text{dist}_{CIELUV}^2(\hat{L}(R_i, G_i, B_i), \hat{u}(R_i, G_i, B_i), \hat{v}(R_i, G_i, B_i); L_i, u_i, v_i)$$

$$\text{dist}_{CIELUV}^2(L, u, v, L_0, u_0, v_0) = (L - L_0)^2 + (u - u_0)^2 + (v - v_0)^2$$

The mathematical solution to such "linear least squares" problem is well known. Because  $\lambda_i$  only appears in the  $L$  modeling,  $\alpha_i$  only appears in the  $u$  modeling, and  $\beta_i$  only appears in the  $v$  modeling; the optimization problem can be decomposed into three subproblems: one for  $L$ , one for  $u$  and one for  $v$ . Consider the  $L$  equations. (The  $u$  equations and the  $v$  equations follow exactly the same argument.) The problem of minimizing the sum of squares of errors in  $L$  can be stated as solving the following matrix equation in the least squares sense:

$$\begin{pmatrix} 1 & R_1 & G_1 & B_1 & \dots & G_1 B_1^2 & B_1^3 \\ 1 & R_2 & G_2 & B_2 & \dots & G_2 B_2^2 & B_2^3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & R_N & G_N & B_N & \dots & G_N B_N^2 & B_N^3 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{20} \end{pmatrix} = \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_N \end{pmatrix} \text{ or } \mathbf{R}\boldsymbol{\lambda} = \mathbf{L}$$

where  $N$  is the total number of data points (original sampled points plus control points created in a manner described below). Typically,  $N$  is much larger than 20, so the preceding equation is over-determined, requiring a least squares solution. A closed form solution for  $\lambda$  is available:

$$\boldsymbol{\lambda} = (\mathbf{R}^T \mathbf{R})^{-1} (\mathbf{R}^T \mathbf{L})$$

In practice, direct evaluation using the closed form solution is not used because it has poor numerical properties. Instead, some kind of matrix factorization algorithm is applied to the coefficient matrix which reduces the system of equations to a canonical form. In the current implementation, Singular Value Decomposition (SVD) is applied to the matrix  $\mathbf{R}$  and then the resulting decomposed system is solved.

The solution to the linear regression problem, denoted by  $\tilde{\lambda}_i, \tilde{\alpha}_i, \tilde{\beta}_i$ , is used as the starting point of the Levenberg-Marquardt algorithm. In this algorithm, a trial step is computed that should move the point closer to the optimal solution. The trial step satisfies a set of linear equations dependent on the functional value and values of the derivatives at the current point. For this reason, the derivatives of the objective function  $\Theta$  with respect to the parameters  $\lambda_i, \alpha_i, \beta_i$  are required inputs to the Levenberg-Marquardt algorithm. Although there are 60 parameters, there is a shortcut that allows you to compute a lot less. By the Chain Rule of Calculus,

$$\frac{\partial \Psi}{\partial \lambda_j} = \sum_{i=1}^N \frac{\partial \Delta E_{CIE94}^2(L, u, v, L_i, u_i, v_i)}{\partial L} \Bigg|_{\hat{L}, \hat{u}, \hat{v}} R_{ij}$$

$$\frac{\partial \Psi}{\partial \alpha_j} = \sum_{i=1}^N \frac{\partial \Delta E_{CIE94}^2(L, u, v, L_i, u_i, v_i)}{\partial u} \Bigg|_{\hat{L}, \hat{u}, \hat{v}} R_{ij}$$

$$\frac{\partial \Psi}{\partial \beta_j} = \sum_{i=1}^N \frac{\partial \Delta E_{CIE94}^2(L, u, v, L_i, u_i, v_i)}{\partial v} \Bigg|_{\hat{L}, \hat{u}, \hat{v}} R_{ij}$$

where  $j = 1, 2, \dots, 20$ ,  $L_i, u_i, v_i$  are the CIELAB value of the  $i$  th sample point, and  $R_{ij}$  is the  $(i, j)$ th entry of the matrix  $\mathbf{R}$  defined above. So instead of computing derivatives for 60 parameters, you can compute derivatives for  $L, a$ , and  $b$  using numerical forward differencing.

It is also necessary to set up a stopping criterion for iterative algorithms. In the current implementation, the iterations are terminated if the mean square DECIE94 is less than 1, or the number of iterations performed has exceeded 10. The number 10 comes from the practical experience that if the first few iterations do not reduce the error significantly, further iterations would not help much other than moving the point in an oscillatory manner, i.e., the algorithm may not converge. Even in the case that the algorithm diverges, we can be sure that the DECIE94 is no worse than what we started, i.e. with the parameters obtained from linear regression.

Even with the preceding method of nonlinear regression, there are several problems with the fitting. One problem is that the polynomials tend to overshoot or undershoot beyond the data points. Artificial local maxima and minima may be introduced in the fitting process. This can be counteracted by using polynomials of low degree, which is the reason you should not use higher than three degrees. A more serious aspect of overshooting or undershooting is that, while a polynomial can take any real value theoretically, the space it tries to model typically has physical constraints and practical constraints. CIEXYZ must have all of X, Y, Z non-negative, which is a physical constraint. In practice, they only take values in the magnitude of hundreds, not thousands or higher. Similarly, CIELAB or CIELUV has its own physical and practical constraints. When the RGB space is filled sufficiently with sample points, the problem of overshooting or undershooting is not serious. However, the captured RGB points from the capture device do not usually fill up the RGB space uniformly. The points may fill up only inner the 80% of the RGB cube, or worse, they may reside in a lower dimensional manifold. When this happens, the regressed polynomials may do a poor job at extrapolating the values beyond the data points, sometimes returning unrealistic predictions. You want a model that always returns reasonably realistic values. This requires a method that can effectively control the boundary behavior of regression polynomials by imposing additional cost to those polynomials that behave erratically near the boundary of the

RGB cube. A further measure to ensure that the polynomials always return realistic values is applied by clipping the output to within the CIE spectral locus.

It is at this point that the scanner modeling and camera modeling diverge from each other. The camera model is expected to extrapolate to regions beyond the sampled points including the "specular highlights," the same extrapolation is not required for the scanner model. The scanner target is expected to cover a characterization that is similar to the printed materials to be scanned. The scanner model is still required to be robust in the sense that it should not return unrealistic values, but extrapolation far beyond the characterization target is not required. To ensure robustness, the L-polynomial above is clipped at 100, that is, the polynomial model is forced not to extrapolate beyond "Dmin" of the scanner target.

The camera model is expected to extrapolate to specular highlights, so clipping at 100 is undesirable. Instead, the following algorithm is used.

Artificial control points are introduced to control the behavior of the polynomials in regions with insufficient sampling. By strategically placing these control points with appropriate values, they serve to "pull" the polynomials in the required direction. In the current implementation, eight control points corresponding to the corners of the RGB cube are used. If the device values are normalized to unity, then these points are:

$$R = 0, G = 0, B = 0$$

$$R = 0, G = 0, B = 1$$

$$R = 0, G = 1, B = 0$$

$$R = 0, G = 1, B = 1$$

$$R = 1, G = 0, B = 0$$

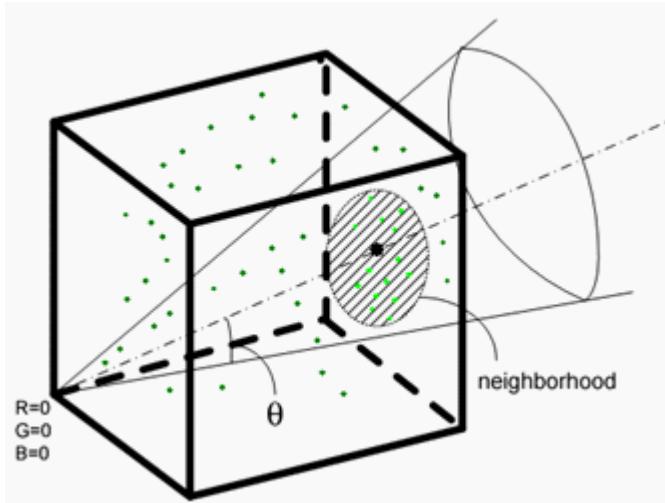
$$R = 1, G = 0, B = 1$$

$$R = 1, G = 1, B = 0$$

$$R = 1, G = 1, B = 1$$

Except for the white  $R = G = B = 1$ , which is associated with a CIELAB value of  $L = 100, u = v = 0$ , the following extrapolation algorithm is used to determine the appropriate CIELAB value to be associated with. Generally, for a given  $(R, G, B)$ , a weight is associated with each of the  $(R_i, G_i, B_i)$  in the sampled data set. There are two goals to assigning the weight. First, the weight is inversely proportional to the distance between  $(R, G, B)$  and  $(R_i, G_i, B_i)$ . Second, you want to discard, or assign weight 0 to, points that have a different hue than the given point  $(R, G, B)$ . To take the hue into account, consider points that lie

within a cone whose vertex is at  $(0, 0, 0)$ , whose axis coincides with the line joining  $(0, 0, 0)$  to  $(R, G, B)$ , and whose semi-vertical angle  $\theta$  satisfies  $\cos \theta = 0.9$ . See Figure 3 for an illustration of this cone.

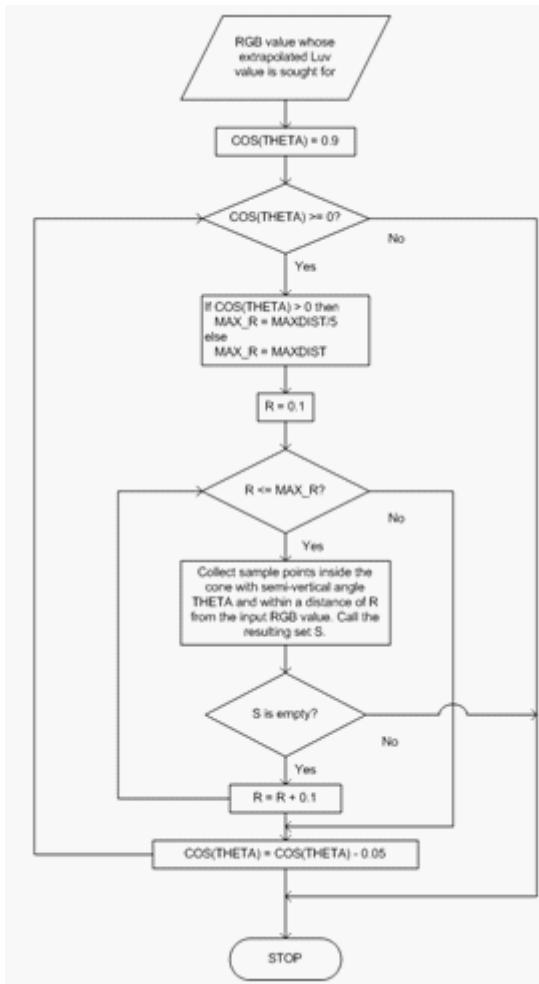


**Figure 3 :** Filtering the sample points by angle and distance. The shape of the neighborhood depicted is for illustration purpose only. The actual shape depends on the distance used; it is a diamond-shaped neighborhood if the 1-norm is used.

Within this cone, a second filtering is performed that is based on the RGB distance, which uses the 1-norm, defined by

$$dist(R, G, B; R_i, G_i, B_i) = |R - R_i| + |G - G_i| + |B - B_i|$$

With the current cone, the initial search is for points that are within a distance of 0.1 from  $(R, G, B)$ . If no point is found within this radius, the radius is increased by 0.1, and the search is restarted. If the next round nets no point either, the radius is increased by 0.1. This process continues until the radius exceeds  $\text{MaxDist}/5$ , where  $\text{MaxDist} = 3$ , in the case of 1-norm. If no point is found, the cone is enlarged by decreasing the  $\cos \theta$  by 0.05, that is, increasing the angle  $\theta$  and restarting the whole process with an increasing radius. This process continues until a non-empty set of points is found, or  $\cos \theta$  reaches 0, that is, the cone has opened up to become a plane. At this point, the search is restarted by increasing the radius, except that the search continues until the radius reaches  $\text{MaxDist}$ . This guarantees that in the worst-case scenario, a non-empty set of points will be found. The algorithm is summarized in the flow diagram in Figure 4.



**Figure 4 :** Flow diagram for determining the set  $S$  of sample points used in the extrapolation for an input RGB value

Assuming that the preceding process yields a non-empty set  $S$  of points  $(R_i, G_i, B_i)$  and corresponding  $(L_i, a_i, b_i)$ , then for each such point, a weight  $w_i$  is assigned, given by

$$h_i = \left( \frac{\max(\text{MaxDist} - d_i, 0)}{d_i} \right)^2 \text{ where } d_i = |R - R_i| + |G - G_i| + |B - B_i|$$

$$w_i = h_i / \sum_{j \in S} h_j$$

Finally, the extrapolant is defined by

$$L = \sum_{i \in S} w_i L_i$$

$$u = \sum_{i \in S} w_i u_i$$

$$v = \sum_{i \in S} w_i v_i$$

The preceding equations constitute an instance of the "inverse-distance weighted methods," commonly called the Shepard methods. By running each of the eight points from eq (6) through the algorithm, eight control points are obtained, each with  $R, G, B$  and  $L, a, b$  values, which are put into the pool with the original sample data.

To ensure that the model always produces valid color values and for system integrity and stability down the whole color processing pipeline, you must perform a final clipping to the output of the polynomial model. The CIE visual gamut is described by the achromatic component ( $Y$  or  $L'$ ) and the chromatic component ( $xy$  or  $a'b'$ , which are related to the XYZ space by a projective transformation). In the current implementation, the  $a'b'$  chromaticity is used because it is directly related to the CIELUV space. For any  $CIELAB$  value, first clip  $L$  to a non-negative value:

$$L \leftarrow \max(0, L)$$

To allow extrapolation for specular highlights,  $L$  is not clipped at 100, the "conventional" upper bound for  $L$  in Lab space.

Next, if  $L = 0$ , then  $a$  and  $b$  are clipped such that  $a^* = b^* = 0$ . If  $L > 0$ , calculate

$$u'_{rel} = u / 13L$$

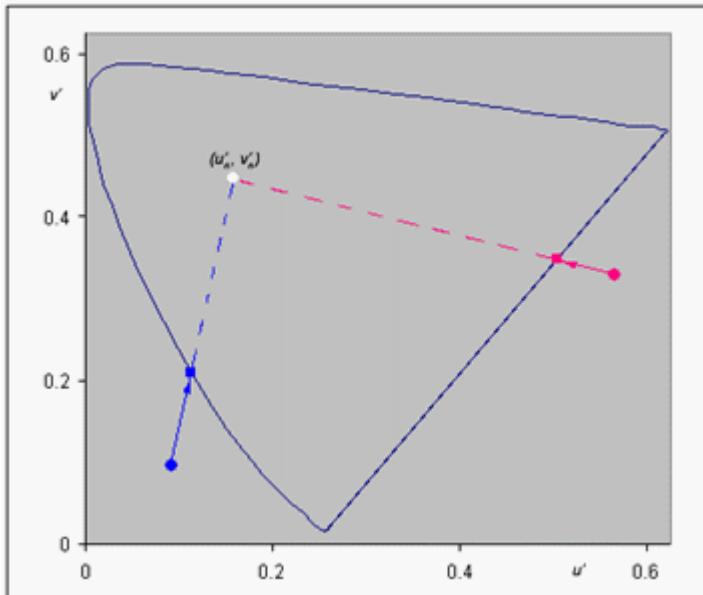
$$v'_{rel} = v / 13L$$

These are the components of a vector in the  $a'b'$  diagram from the white point ( $u'_w, v'_w$ ) to the color in question. Define the CIE spectral locus as the convex hull of all the points  $(a'_s, b'_s)$ , parameterized by the wavelength  $\lambda$ :

$$u'(\lambda) = \frac{4\bar{x}(\lambda)}{\bar{x}(\lambda) + 15\bar{y}(\lambda) + 3\bar{z}(\lambda)}$$

$$v'(\lambda) = \frac{9\bar{y}(\lambda)}{\bar{x}(\lambda) + 15\bar{y}(\lambda) + 3\bar{z}(\lambda)}$$

where  $\bar{x}(\lambda)$ ,  $\bar{y}(\lambda)$ ,  $\bar{z}(\lambda)$  are the CIE color-matching functions for the 2-degree observer. If the vector lies outside the CIE locus, the color is clipped to the point on the CIE locus that is the intersection of the locus and the line defined by the vector. See Figure 5. If clipping has occurred, the  $a$  and  $b$  value is reconstructed by first subtracting  $a'_s$  and  $b'_s$  from the clipped  $a'$  and  $b'$ , and then multiplying by  $13L$ .



**Figure 5 :** Clipping algorithm for Lab values that are outside the CIE visual gamut

In the current implementation, the CIE spectral locus in the  $a'b'$  plane is represented by a piecewise linear curve with 35 segments (corresponding to a wavelength from 360 nm to 700 nm inclusive). By ordering the line segments so that their subtended angles at the white point are ascending, which is equivalent to descending wavelengths, the line segment that intersects with the ray formed by the above vector can be found by a simple binary search.

## RGB Printer Device Model Baseline

A device characterization of a RGB printer consists of constructing an empirical model of the device that predicts the device-independent CIELUV color for any given RGB value

There are two ways to construct the empirical model. One way is to use the device data for a RGB printer, and the other is to use analytical parameter data. In the first one, measurement data provided by a user for a RGB printer device is used to construct 3-D lookup table (LUT). The measurement data consists of XYZ values for uniformly sampled RGB patches. Typical sampling sizes are 9 or 17 for each component. Each patch is measured with a colorimeter or spectrophotometer in CIEXYZ space. The XYZ value for a patch is then converted into CIELUV value, forming a 3-D LUT. In the device model, the Sakamoto's tetrahedral interpolation method is applied to the 3-D LUT in order to predict the CIELUV data for a given RGB data. (Confer US Patent 4275413 (Sakamoto et al.), US Patent 4511989 (Sakamoto), Kang [1]. The two patents mentioned have expired.). The analytical parameter data passed in the second method is simply a LUT that was built previously. Typically, that LUT was built using the first method, although it could be hand-built.

In the current color management, the source map is defined as the map that goes from RGB space to a device-independent CIEXYZ color space. The destination map is defined as the map that goes from the device-independent CIEXYZ color space to RGB space. It is the inverse of the source map.

The empirical model is directly used in the source map. It first maps a given RGB data into a CIELUV data, which is converted into XYZ data. In the destination map, device-independent CIEXYZ data is first converted into CIELUV data. Then, the empirical model and the classical Newton-Raphson method are used to predict the best RGB data for the CIELUV data. The details about conversion from CielUV to RGB data are as follows:

After generating a 3-D LUT from RGB to CieLUV, the map from RGB to LUV is built using tetrahedral interpolation on RGB. This map is denoted by the following equations:

$$L = L(R, G, B)$$

$$u = u(R, G, B)$$

$$v = v(R, G, B)$$

Inversion of the map consists of solving, for any color  $L^*, u^*, v^*$ , the following system of nonlinear equations:

$$L^* = L(R, G, B)$$

$$u^* = u(R, G, B)$$

$$v^* = v(R, G, B)$$

A nonlinear equation that is based on the classical Newton-Raphson method is used in the new CTE. An initial guess, or *a priori* see,  $s_{\text{prior}}(R_0, G_0, B_0)$  is obtained by searching through a "seed matrix" consisting of a uniform 8x8x8 grid of pre-computed (RGB,Luv) pairs. The RGB corresponding Luv that is closest to the  $L^*u^*v^*$  is chosen. Each point in the seed matrix corresponds to the center of a cell so that the iterations don't start with a point on the boundary face of the RGB cube. In other words, the RGB of the seeds is defined by:  $\text{STEP} = 1/8$   $s_{ijk} = (\text{STEP}/2 + (i-1)\text{STEP}, \text{STEP}/2+(j-1)\text{STEP}, \text{STEP}/2+(k-1)\text{STEP})$  with  $i,j,k = 1\dots8$  At the  $i$  th step of Newton-Raphson, the next estimate  $R_{i+1}, G_{i+1}, B_{i+1}$  is obtained by the formula:

$$\begin{pmatrix} R_{i+1} \\ G_{i+1} \\ B_{i+1} \end{pmatrix} = \begin{pmatrix} R_i \\ G_i \\ B_i \end{pmatrix} + \begin{pmatrix} \partial L / \partial R & \partial L / \partial G & \partial L / \partial B \\ \partial u / \partial R & \partial u / \partial G & \partial u / \partial B \\ \partial v / \partial R & \partial v / \partial G & \partial v / \partial B \end{pmatrix}_{R_i, G_i, B_i}^{-1} \begin{pmatrix} L^* - L(R_i, G_i, B_i) \\ u^* - u(R_i, G_i, B_i) \\ v^* - v(R_i, G_i, B_i) \end{pmatrix}$$

Iteration stops when the error (distance in the CIELUV space) is less than a pre-set tolerance level (0.1 in the CTE), or when the number of iterations has exceeded the maximum allowed number of iterations (10 in the CTE). The values for the tolerance and

the number of iterations were empirically determined to be effective. In future versions, the tolerance value may be changed.

The Jacobian matrix is calculated using forward difference, except at a boundary point (one or more of the R, G, B is 1) where backward difference is used instead. Instead of calculating the inverse of the Jacobian matrix, the linear system is solved directly using Gauss-Jordan elimination with full pivoting.

At the end of the iterations, convergence still might not be achieved because Newton-Raphson is a "local" algorithm, that is, it only works well if you start with an initial guess that is close to the true solution. If, at the end of the Newton-Raphson iterations, convergence within the pre-defined error tolerance has not been achieved, the iterations are restarted with a new set of seeds, defined as follows.

For example, the best solution obtained so far is (r, g, b). From this solution, N a posteriori seeds are derived, where N = 4. Intuitively, the solution is moved "toward the center" in a step size that depends on N. See Figure 6.

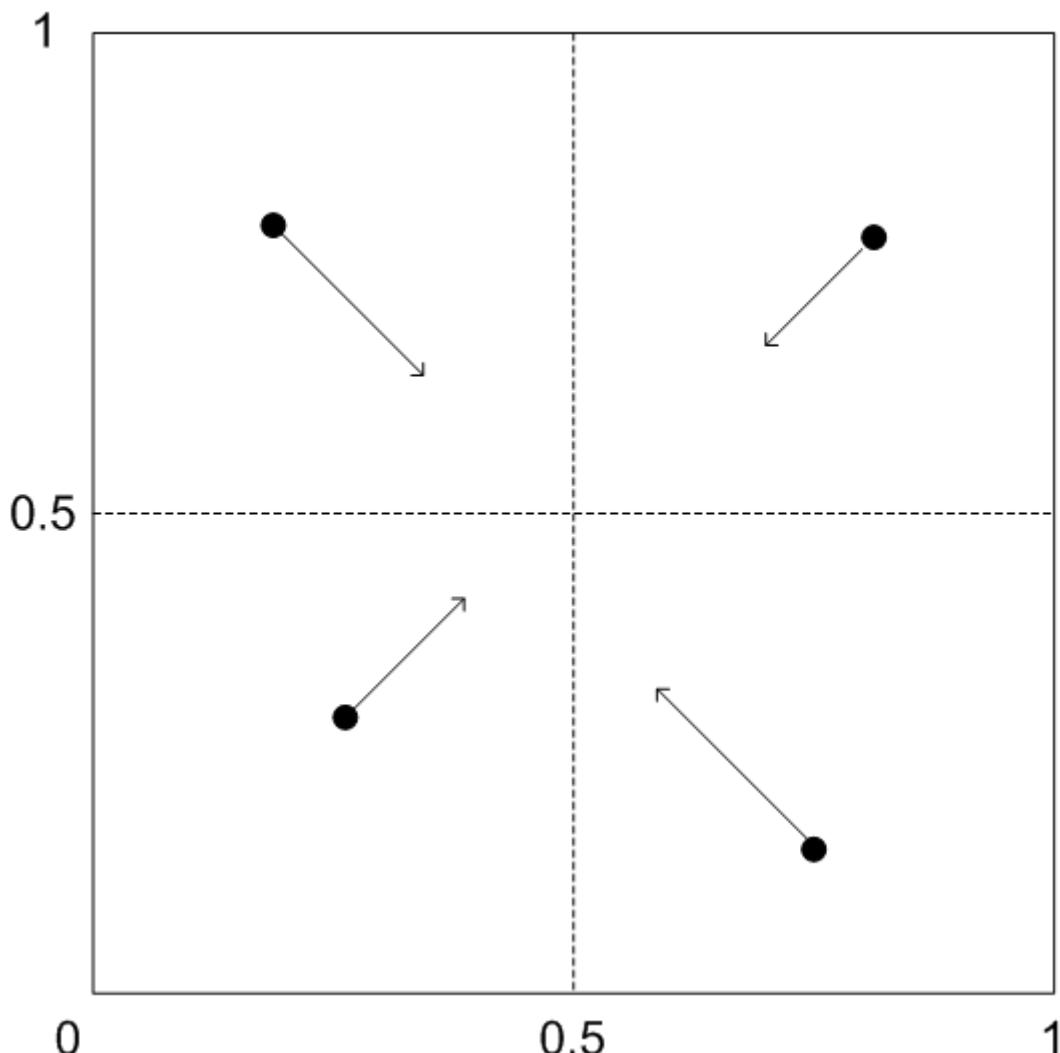


Figure 6 : Perturbation direction of the solution depends on which octant it is in.

In other words, if  $r > 0.5$ , the value on the R channel is decreased, otherwise the value is increased. There is similar logic for the G and B channels. The precise definitions are:

$$\text{PERTURBATION} = 0.5/(N+1)$$

$$\text{Dir}(r) = -1 \text{ if } r > 0.5; +1 \text{ otherwise. Similarly for Dir(g) and Dir(b)}$$

The  $j$ th a posteriori seed,  $s_{????}$ , is  $(r + \text{Dir}(r) * j * \text{PERTURBATION}, g + \text{Dir}(g) * j * \text{PERTURBATION}, b + \text{Dir}(b) * j * \text{PERTURBATION})$

Try the first  $s_{????}$  and if it gives a new solution within error tolerance, you can stop. Otherwise, try the second  $s_{????}$  and so on until the  $N$ th  $s_{????}$ .

The schematics of the whole algorithm is shown in Figure 7.

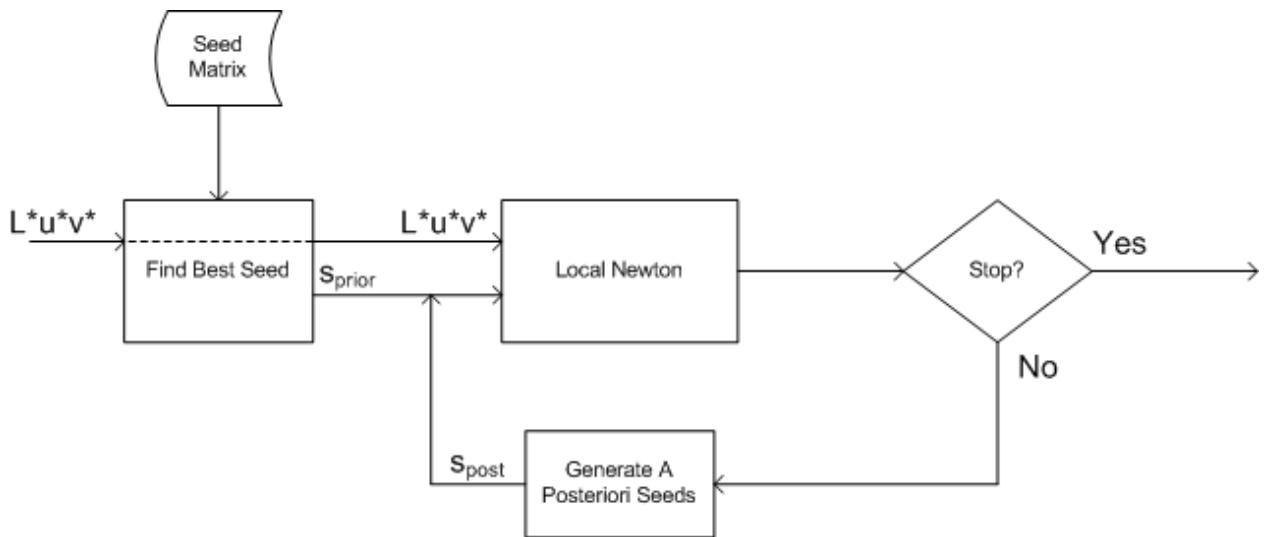


Figure 7 : Schematics of inverting the device model

## RGB Virtual Device Model Baseline

This device model(DM) is a simple matrix/tone reproduction algorithm. The matrix is derived from the white point and primaries using standard color science algorithms. The tone reproduction curve is derived from the measurement parameters according to the ICC descriptions of CurveType and ParametricType (or inverted as required). Details of the internal algorithms will be provided after additional validation of high dynamic range issues.

The RGB virtual device model is an idealized matrix/tone reproduction curve RGB similar to the ICC three-component matrix-based profile design. The "virtual measurement" parameters of the DM include a white point value (absolute CIEXYZ), RGB primary values (absolute CIEXYZ), and a tone reproduction curve that is based on the ICC ParametricCurveType and CurveType in XML formatting consistent with the DMP schemas.

ICC parametricCurveType function type encoding and corresponding support in IRGBVirtualDeviceModelBase are shown in the following table.

Function type	Parameters	Type	Note
$Y = X^y$	g	GammaType	Common implementation
$Y = (aX + b)^y \quad (X \geq -b/a)$ $Y = 0 \quad (X < -b/a)$	ga b	GammaOffsetGainType	CIE 122-1966
$Y = (aX + b)^y + c \quad (X \geq -b/a)$ $Y = c \quad (X < -b/a)$	ga b c	GammaOffsetGainOffsetType	IEC 61966-3
$Y = (aX + b)^y \quad (X \geq d)$ $Y = cX \quad (X < d)$	ga b c d	GammaOffsetGainGainType	IEC 61966-2.1 (sRGB)
$Y = (aX + b)^y + e \quad (X \geq d)$ $Y = (cX + f) \quad (X < d)$	ga b c d e f	N/A	Not supported in WCS

The tone curve for RGB virtual devices is applied in DeviceToColorimetric between the input data, pDeviceColors, and the matrix multiply. For ColorimetricToDevice, a method must be used to invert the tone curve. In the baseline implementation, this is done by direct interpolation in the same tone curve used for DeviceToColorimetric.

The curves should be specified in the profiles as pairs of numbers in float space. The first number represents values in pDeviceColors. The second number represents the output of the tone curve. All values in the tone curve must be between minColorantUsed and maxColorantUsed. Tone curves must contain at least two entries: one for minColorantUsed and one for maxColorantUsed. The maximum number of entries in the ToneCurve is 2048. In general, the more entries in the table, the more accurately you can model curvature. A piecewise linear interpolation is performed between the entries.

You might consider alternative interpolation methods. If you know something about the underlying behavior of the device, you can use fewer samples and model more accurately with a higher order curve. But if you use the wrong curve type, you will be very inaccurate. Without more information, you cannot guess the curve type. So, use linear interpolation and provide many data points.

## CMYK Printer Device Model Baseline

A device characterization of a CMYK printer consists of constructing an empirical model of the device that predicts the printed color for any given CMYK value. The characterization also includes the inversion of this model so that a prescription of the

CMYK value to us for a given color to be printed can be given. This is typically defined in terms of CIEXYZ or CIELAB value.

Typically, an IT8.7/3 target containing CMYK patches is used. The patches consist of sampling of the CMYK space in a well-defined manner so that a rectangular grid (with non-uniform spacing in C, M, Y and K) is formed. Each patch is then measured with a colorimeter or spectrophotometer. The measurements in CIEXYZ values then form a lookup table (LUT), from which an empirical model of the printer is built using Sakamoto's interpolation method. US Patent 4275413 (Sakamoto et al.), US Patent 4511989 (Sakamoto), Kang [1]. The two patents mentioned have expired.

Specific requirements for the CMYK measurement samples necessary for a device model profile to be accepted as valid by the CMYK printer baseline device model are as follows. (The sample set is most clearly described as a set of CMY sample cubes, each associated with a specific K level.)

- At minimum, valid CMY cubes must be provided for the K = 0 and K = 100 levels.
- Intermediate K levels may be non-uniformly spaced.
- Any intermediate K level without a valid CMY cube will be ignored.
- The CMY cubes may use non-uniform sample intervals (grid spacing), but the same set of sample intervals must be used in all of the C,M, and Y dimensions in the CMY cube for a given K level.
- Each K level CMY cube can use a different number and spacing of sample intervals.
- All CMY cubes must contain the "corners" of the CMY cube, that is, CMY samples at [0,0,0], [0,0,100], [0,100,0], [100,0,0], [0,100,100], [100,0,100], [100,100, 0], [100,100,100].
- Any intermediate CMY grid levels must be fully sampled in each channel. In other words, a sample must exist at each 3-D grid intersection within the CMY cube.
- For the K = 0 and K = 100 CMY cubes, 2x2x2 "corners-only" cubes are the minimum accepted as valid.

[NOTE: for K=0 and K=100 levels, a 3x3x3 CMY cube will be processed as a 2x2x2 "corners-only" cube; the intermediate sample level is ignored. 4x4x4 and larger cubes will have all on-grid samples used.]

- For intermediate K levels, 4x4x4 CMY cubes are the minimum accepted as valid.

A high-quality profile will use finer sampling grids than the minimum required for validity, usually 9x9x9x9 or higher. The samples from the IT8.7/3, IT8.7/4, and ECI targets produce valid device model profiles (DMPs) for the CMYK printer baseline device model. While this device model is able to ignore the extraneous (off-grid) samples in these targets, it is not guaranteed to be able to do so for other targets, and so, it is recommended that extraneous samples be removed from measurement sets going into profiles for this device model.

The inversion of the printer model presents more difficulties. Given an input color in CIECAM, there is a question of whether this color is within the printer gamut. There is also the issue regarding the arrangement of points in the color appearance space. While we can arrange the CMYK values to fall on a rectangular grid, as is done in the IT8.7/3 target, the same cannot be said of the resulting printed colors as they are situated in the color appearance space. In general, they are scattered in the color appearance space with no particular pattern.

There are generally two approaches to the inversion problem of scattered points. One approach is to use a geometric subdivision of the printer gamut using elementary 3-dimensional solids, such as tetrahedra. A subdivision of the printer gamut in the color appearance space can be induced from the corresponding subdivision of the CMY(K) space, see Hung[93], Kang[97]. This approach has the advantage of computational simplicity. In the case of a tetrahedron, only four points are used in an interpolation. On the other hand, the result depends heavily on a few points, which means that a measurement error will have a significant effect on the result. The interpolant also tends to be not as smooth. The second approach does not assume any subdivision, and is based on the technique of scattered data interpolation. A classical example is the technique of Shepard interpolation, or inverse weighted method (See Shepard [68]). Here, several points surrounding the input point are chosen in some manner, each assigned a weight, usually inversely proportional to the distance, and the interpolant is taken as the weighted average of the neighboring points. In this approach, the choice of neighboring points is paramount to performance. While too few points can render the interpolant inaccurate and non-smooth, too many points impose a high computational cost, as the weights are typically non-linear functions and costly to compute.

The two approaches described above both have problems. The subdivision approach depends critically on the data being reasonably void of noise, and generally the interpolant is not very smooth. The scattered data interpolation is more tolerant to data noise, and generally gives a smoother interpolant, but it is computationally more expensive.

The new CTE takes an alternative approach. The CMYK device is treated as a collection of several CMY devices, each of which has a specific value of black (K). Using a selection

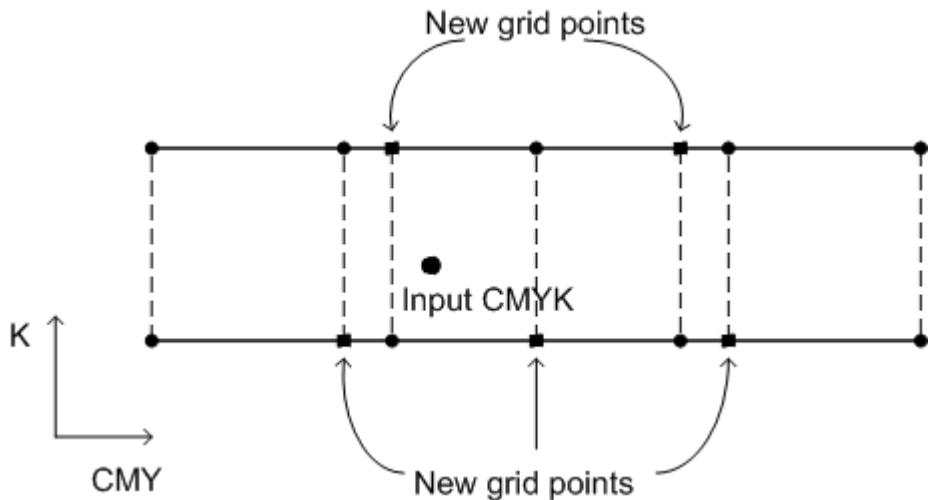
algorithm that takes as parameters lightness and chroma, a level of black is chosen. The CMY values are obtained by inversion of the corresponding CMY to Luv table using the Newton methods employed elsewhere by the RGB printer algorithm.

Use the following steps.

1. Print the characterization target, which is either the IT8.7/3 target, or a target containing sampling of the CMYK space at regularly or irregularly spaced intervals.
2. Measure the target using a spectrophotometer, and convert the measurements to CIELUV space.
3. Construct the forward map from CMYK to Luv.
4. Use the forward map to construct a set of CMY to Luv maps for a range of K values.
5. For any input Luv value, the corresponding CMYK value is obtained by selecting one of the maps constructed in step 4 above and inverting using Newton's method to obtain a CMY colorant set to accompany the K value selected.

Steps 1 and 2, which are standard procedures, are performed by a profiling program that is not part of the new CTE. The IT8.7/3 target contains a reasonably detailed sampling of all the CMYK values at various levels of C, M, Y, and K. Alternatively, a custom target with uniform sampling of the C, M, Y, and K channels can be used. After the target is printed, a spectrophotometer or colorimeter can be used to measure the XYZ value of each patch, and the XYZ value can be converted to the Luv value using the CIELUV model.

Step 3, construction of the forward map from CMYK to Luv, can be achieved by applying any known interpolation technique, such as tetrahedral or multilinear method, on the rectangular grid in CMYK space. In the new CTE, a 4-dimensional tetrahedral interpolation is used. Because the CMY sampling grids are generally different on each level of K, however, we use a technique of super-sampling, as detailed below. For a given CMYK point, the sandwiching K levels are first determined based on the K value. Then introduce a "super-grid" on each K level that is a union of the CMY grids on each of the two K levels. On each K level, the Luv value of any newly introduced grid point is obtained by a 3-dimensional tetrahedral interpolation within that K level. Finally, a 4-dimensional tetrahedral interpolation for the specific CMYK point is performed on this new grid.



**Figure 8 : Supersampling**

Step 4 constructs a set of CMY-to-Luv lookup tables (LUTs). The forward map constructed in Step 3 is called repeatedly to resample the CMYK space. The CMYK color space is sampled using an even spacing of K and a different, but still evenly spaced, sampling of CMY.

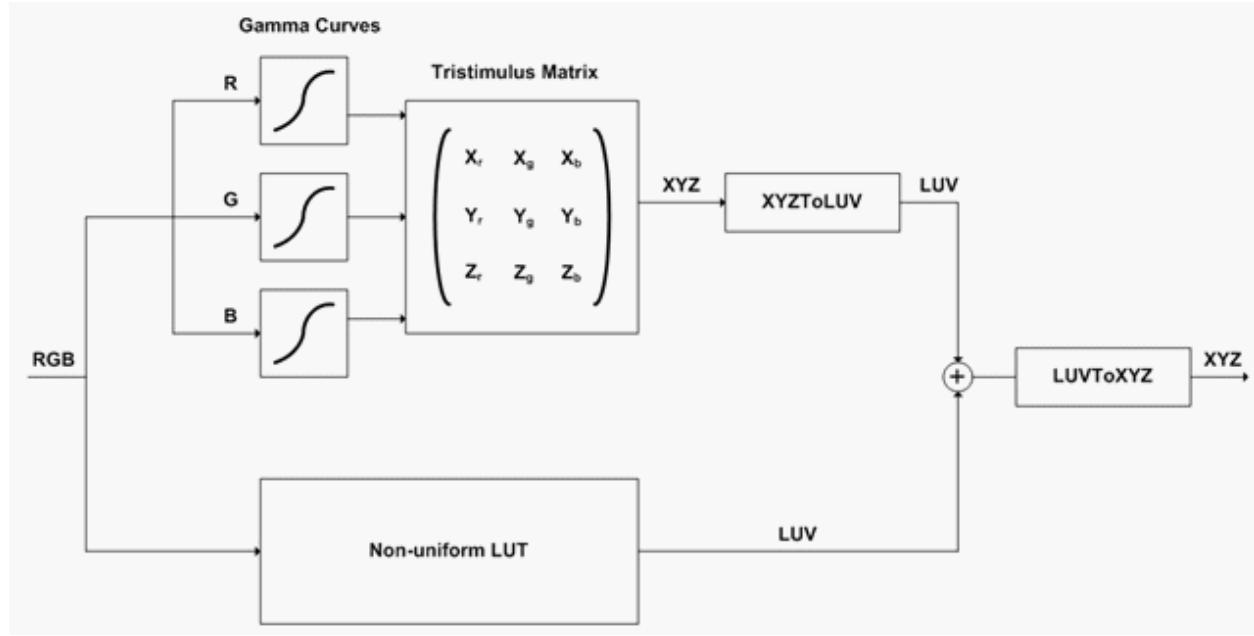
Step 5 is a procedure to obtain the CMYK value using the LUTs constructed in Step 4 for any input Luv point. The appropriate value of K is chosen by analyzing the lightness as well as the degree of color in the Luv requested. After the table is selected, the CMY values are obtained from the table by using Newton's method (as documented under the RGB printer device model earlier).

CIELUV space is used in the printer model instead of CIEJab because the device model should be based solely on colorimetric data available in the DMP. The DMP contains colorimetric data for each measured patch, including the media white point, so it is possible to convert CIEXYZ data into CIELUV data. However, it is not possible to convert to CIECAM02 JCh or Jab, because there is no access to the viewing condition information in the DMP.

## RGB Projector Device Model Baseline

Note: Many RGB projectors have more than one operating mode. In one mode, which is often the default and might be called something like "presentation," the color response of the projector is optimized for maximum brightness. However, in this mode, the projector loses the ability to reproduce light, slightly chromatic colors such as pale yellows and some flesh tones. In another mode, often called "film," "video," or "sRGB," the projector is optimized for reproduction of realistic images and natural scenes. In this mode, maximum brightness is traded off to improve the overall quality of the color reproduction. To obtain satisfactory color reproduction with RGB projectors, it is

necessary to place the projector in a mode where a smooth gamut of colors can be reproduced.



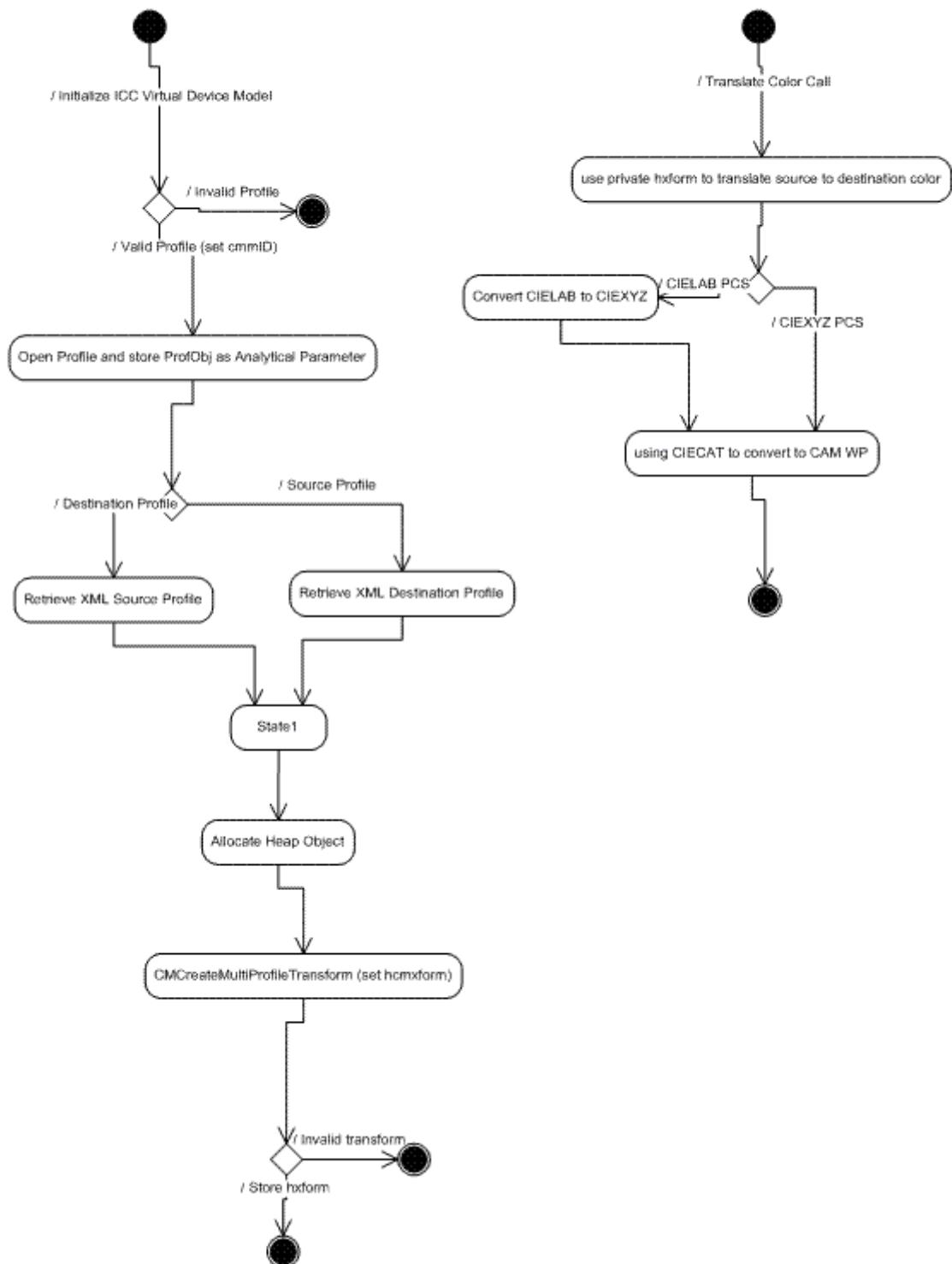
**Figure 9 : DLP device model**

An incoming RGB value passes through two computational pathways. The first is the matrix model, which results in an XYZ value. This is immediately followed by the conversion from XYZ to Luv. The second is the non-uniform LUT interpolation using tetrahedral interpolation. The output of the interpolation is already in Luv space by construction. The two outputs are added to obtain the predicted Luv value. This is finally converted to XYZ, which is the expected output of the colorimetric model for the DLP device.

Since projectors are display devices, they also support the inversion of the model, that is, the transform from XYZ to RGB. Because the device model transforms RGB space to XYZ space, which are both three dimensional, inversion is equivalent to solving three nonlinear equations in three unknowns. This can be done by standard equation solving techniques, such as Newton-Raphson. It is preferable, however, to first convert XYZ to Luv, and then invert the Luv To RGB transform, because Luv space is more perceptually linear than XYZ space.

## ICC Device Model Baseline

The CITE ICC workflow interoperability is enabled by creating a special ICC device baseline device model profile that stores the profile object and creates a ICC transform using a no-op XYZ profile. This transform is then used to translate between device and CIEXYZ colors.



**Figure 10 : CITE ICC Workflow Interoperability**

## Related topics

[Basic color management concepts](#)

[Windows Color System Schemas and Algorithms](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WCS Gamut Map Model Profile Schema and Algorithms

Article • 12/30/2021

- Overview
- Gamut Map Model Profile Architecture
- Generation of the Gamut Boundary
- The GMMP Schema
- The GMMP Schema Elements
- GamutMapModel
  - Namespace
  - Version
  - Documentation
  - DefaultBaselineGamutMapModel type
  - PlugInGamutMapType
  - ExtensionType
- The GMMP Baseline Algorithms
- Aligning the Neutral Axes
  - Minimum Color Difference (MinCD)
  - BasicPhoto
  - Overview
  - The case of single gamut shell
  - Black enhancement
  - The case of dual gamut shells
  - Reasons for changes from the CIE TC8-03 recommendations
  - Hue mapping
- Gamut Boundary Description and Gamut Shell Algorithms
  - Triangulation of the Gamut Boundary
  - Boundary Line Elements
  - Gamut Operation: CheckGamut
  - Full Hue Plane: Intersect
  - Gamut Operation: CheckGamut (continued)
  - Minimum Color Difference Gamut Mapping
  - Hue Smoothing
  - Setting Primaries and Secondaries in the Gamut Boundary Description
  - Setting the Neutral Axis in the Gamut Boundary Description
- Related topics

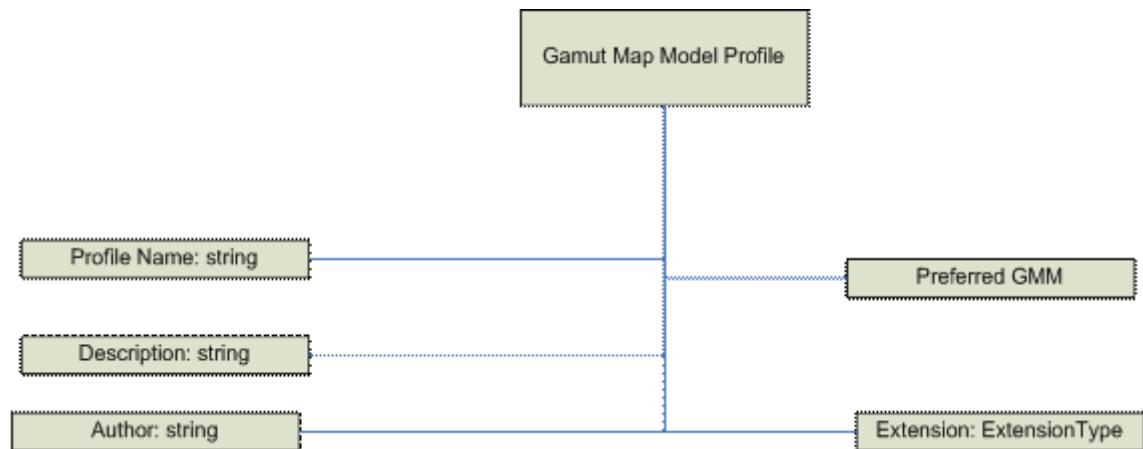
# Overview

This schema is used to specify the content of a gamut map model profile (GMMP). The associated baseline algorithms are described in the following topic.

The basic GMMP schema consists of common header information, an optional reference to a preferred Gamut Map Model plug-in, and extension tags.

In addition, the GMMP provides explicit information on the targeted Gamut Map Model, and provides a policy on the baseline fallback Gamut Map Model to use if the targeted model is unavailable. The schema can include private extensional information, but will include no other extraneous information.

## Gamut Map Model Profile Architecture



The sampling of the output device colorant space is done by iterating through the colorants from 0.0 to 1.0 with a fractional step, accumulating all combinations of each colorant at each step, and then converting them from device colorant space to color appearance space using the DM::DeviceToColorimetricColors method followed by the CAM::ColorimetricToAppearanceColors method. The following is an example of how this is done for RGB.

C++

```
For (red= 0.0; red <= 1.0; red += redStep) {
    For (green = 0.0; green <= 1.0; green += greenStep) {
        For (blue = 0.0; blue <= 1.0; blue += blueStep) {
            Colorants[0] = red; colorants[1] = green; colorants[2] =
            blue;
            pRGBDM->DeviceToColorimetricColors(1, colorants, &XYZ);
```

```
    pCAM->ColorimetricToAppearanceColors(1, &XYZ, &JCh);  
}  
}  
}
```

## Generation of the Gamut Boundary

There are three components to the gamut boundary: the primaries, the neutral samples, and the shells. The primaries are generated by taking the device primaries and applying the DeviceToColorimetric/ColorimetricToAppearance transformation. The neutral samples are generated by sampling the device colorant space in the neutral area and applying the same transformation. For three colorant devices (RGB or CMY), the neutral samples are defined as having all of the colorants equal, for example, R == G == B. For CMYK, the neutral samples are defined as having C == M == Y == 0.

The factors that influence the data used to create the gamut boundary are the data samples (baseline devices only) and the viewing conditions. The step size used to do the full sampling of the colorant space (for monitors and for plausible shell) is an internal constant and isn't available for outside manipulation. Changing the viewing conditions changes the behavior of the color appearance model (CAM) and alters the shape of the gamut boundary, so you must generate a gamut boundary tied to the viewing conditions profile. If sample data is used, as in the case of baseline printers and capture devices, then the samples will have a lot of impact on the shape of the reference gamut and affect the behavior of the model itself.

For input devices, such as cameras and scanners, different samplings are used to generate the reference shell and the plausible shell. The reference shell is generated from the measurements used to initialize the device model. The plausible shell is generated similar to the prior illustration for output devices. The difference is that when you input a typical target, you don't get fully saturated values (where R, G, or B = 255). You must extrapolate using the device model to estimate those values.

## The GMMP Schema

C++

```
<?xml version="1.0" encoding="UTF-8"?>  
<xss:schema
```

```
xmlns:gmm="http://schemas.microsoft.com/windows/2005/02/color/GamutMapModel"
xmlns:wcs="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfileTypes"

targetNamespace="http://schemas.microsoft.com/windows/2005/02/color/GamutMapModel"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  blockDefault="#all"
  version="1.0">

  <xs:annotation>
    <xs:documentation>
      Gamut Map Model profile schema.
      Copyright (C) Microsoft. All rights reserved.
    </xs:documentation>
  </xs:annotation>

  <xs:import
  namespace="http://schemas.microsoft.com/windows/2005/02/color/WcsCommonProfileTypes" />

  <xs:element name="GamutMapModel">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ProfileName" type="wcs:MultiLocalizedTextType"/>
        <xs:element name="Description" type="wcs:MultiLocalizedTextType"
minOccurs="0"/>
        <xs:element name="Author" type="wcs:MultiLocalizedTextType"
minOccurs="0"/>
        <xs:element name="DefaultBaselineGamutMapModel">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="HPMinCD_Absolute"/>
              <xs:enumeration value="HPMinCD_Relative"/>
              <xs:enumeration value="SGCK"/>
              <xs:enumeration value="HueMap"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="PlugInGamutMapModel" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:any namespace="##other" processContents="skip"
minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:attribute name="GUID" type="wcs:GUIDType" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="ID" type="xs:string" use="optional" />
    </xs:complexType>
```

```
</xs:element>  
</xs:schema>
```

# The GMMP Schema Elements

## GamutMapModel

This element is a sequence of the following sub-elements:

1. ProfileName string,
2. DefaultBaselineGamutMapModel,
3. Optional Description string,
4. Optional Author string,
5. Optional PlugInGamutMap, and
6. Optional ExtensionType.

**Validation conditions :** Each sub-element is validated by its own type.

## Namespace

xmlns:gmm="http://schemas.microsoft.com/windows/2005/02/color/GamutMapModel"

targetNamespace="http://schemas.microsoft.com/windows/2005/02/color/GamutMap  
Model"

## Version

Version "1.0" with the first release of Windows Vista.

**Validation conditions :** 1.0 in Windows Vista. Versions <2.0 are also valid in order to support non-breaking changes to the format.

## Documentation

Gamut Map Model Profile schema.

Copyright (C) Microsoft. All rights reserved.

**Validation conditions :** Each sub-element is validated by its own type.

## DefaultBaselineGamutMapModel type

UINT type

Enumeration values:

"MinCD\\_Absolute" "MinCD\\_Relative" "SIG\\_KNEE" "HueMap"

**Validation conditions :** The values must match one of the enumerations above.

## PlugInGamutMapType

This element is a sequence of a GUID GUIDType and any sub-elements.

**Validation conditions :** The GUID is used to match the GMM PlugIn DLL GUID. There are a maximum of 100,000 custom sub-elements.

## ExtensionType

This element is a optional sequence of any sub-elements.

**Validation conditions :** There can be a maximum of 100,000 sub-elements.

# The GMMP Baseline Algorithms

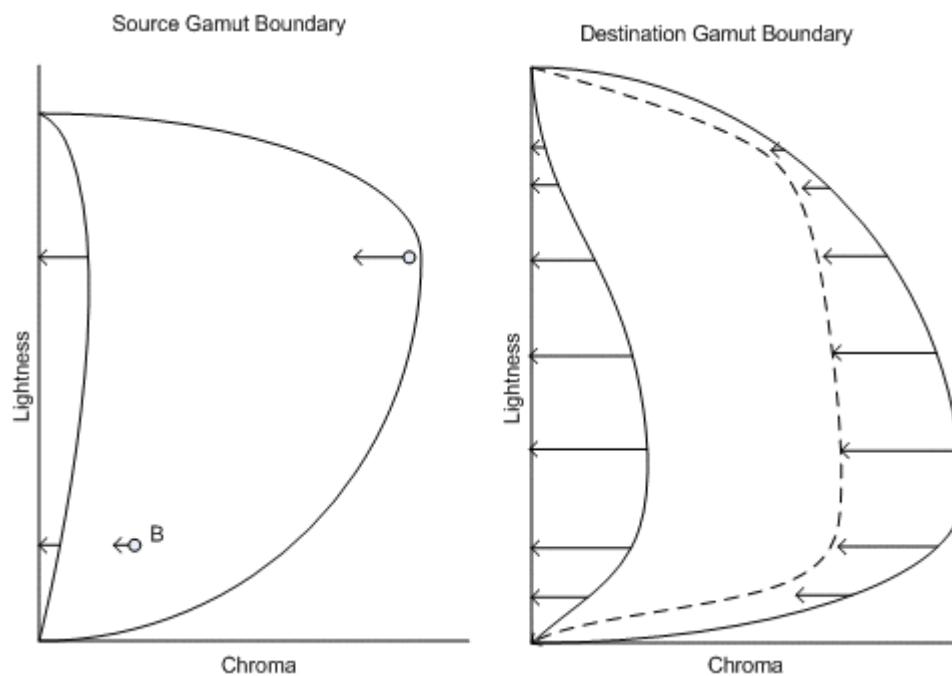
## Aligning the Neutral Axes

Most gamut mapping algorithms have a goal of mapping the neutral axis of the source device to the neutral axis of the destination device: that is, white goes to white, black to black, and gray to gray. This is addressed in part by scaling the lightness of source colors to match the lightness range of the destination device. But that does not completely address the problem.

It is a physical property of most imaging devices that the chromaticity of the device white does not exactly match the chromaticity of the device black. For example, monitor white depends on the sum of the chromaticities and relative luminances of the three primaries, while monitor black depends on the reflectance of the display surface. Printer white depends on the chromaticity of the paper, while printer black depends on the ink or toner used. An appearance model that maps device white exactly to the neutral axis of the appearance space (chroma exactly equal to zero) will not map device black to the neutral axis. Because the eye is more sensitive to chroma differences as lightness increases, mid-grays will be represented as even more chromatic than device black. (Figure 1 illustrates the curvature of the neutral axes in two dimensions. In fact, the neutral axes form a more complex curve in three dimensions.)

While the CAM predicts that these device neutral colors will appear chromatic, actual observers seem to compensate for this. Most people do not consider these device neutral values to be chromatic. For most gamut mapping models, therefore, you should continue to map source neutrals to device neutrals.

To map source neutrals to device neutrals, adjust the source and destination gamut boundaries and each pixel when you apply the gamut mapping algorithm. First adjust each value in the source color so that the source device's neutral axis at the source color's lightness falls directly on the neutral axis of the appearance space. (See the left side of Figure 1.) Then adjust the destination device's gamut boundary description so that each color on the destination device's neutral axis at the destination device gamut boundary color's lightness falls directly on the neutral axis of the appearance space. (See the right side of Figure 1.)



**Figure 1 :** Alignment of the neutral axes illustrated. Left: Adjusting source points relative to the source device neutral axis. Right: Adjusting the destination gamut boundary description relative to the destination gamut boundary description.

Note that you adjust each source pixel value relative to the neutral axis at that lightness value. This assures that source device neutrals will fall on the neutral axis of the appearance model. You also shift all other colors at that lightness by the same amount, so that there are no discontinuities in the representation of the source gamut. You have to shift by different amounts at different lightness levels, because the source device neutrals are not represented as equally chromatic at the different lightness levels. Clearly, this is not a trivial transformation.

Handling the destination device values is a little more tricky. Initially, you adjust the entire destination gamut boundary in a similar manner, but relative to the destination

device neutral axis. This is illustrated in Figure 1 on the right side. That adjustment assures that source gray values will map to destination gray values. This is the space in which the gamut mapping algorithms operate.

However, this space does not accurately describe the true behavior of the destination device. You must invert the mapping before gamut mapped colors are handed to the appearance model and destination device model. You offset all the mapped values by the opposite of the offset applied earlier to the destination device neutral axis. This maps the destination neutral axis back to the value represented originally by the CAM. It does the same for the gamut boundary and all intermediate values.

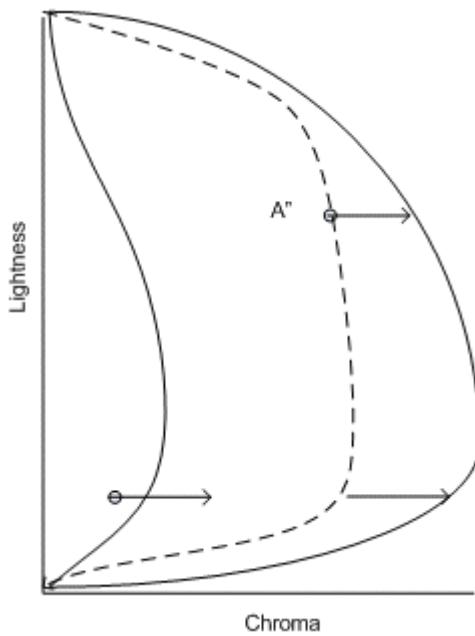


Figure 2 : Undoing the alignment of the destination device neutral axis

## Minimum Color Difference (MinCD)

Minimum Color Difference(MinCD) Relative and Absolute versions - equivalent to the ICC colorimetric intent.

### ⓘ Note

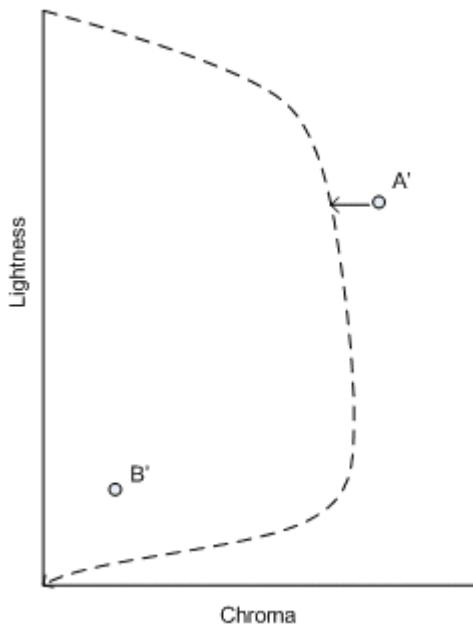
The MinCD GMM is suitable for mapping graphics and line art containing "logo" colors (spot colors), logo color gradients with some out-of-gamut colors, and for the final stage of proofing transforms. While the MinCD GMM could be used for photographic images that are entirely within the destination gamut, it is not recommended for general rendering of photographic images. The mapping of out-of-gamut colors to colors on the destination gamut surface can result in unwanted artifacts, such as tone or chroma irregularities in smooth gradients that cross the gamut boundary. BasicPhoto is recommended for photographic images. If a

photographic or contone image requires a gamut mapping other than BasicPhoto, then the alternative should be to create a plug-in GMM implementing that mapping, instead of using MinCD.

In-gamut colors are left unchanged. For out-of-gamut colors, lightness and chroma are adjusted by finding the point in the destination gamut that has the minimum color distance from out-of-gamut input points. The color distance is computed in JCh space. However, you weight the distance in lightness (J) and the distance in chroma (C) or hue (h) differently. A chroma-dependent weight function is used for the distance in lightness so that the weight is smaller for small chroma and larger for large chroma until a threshold chroma is reached, after which the weight stays at 1, that is, the same weight as distance in chroma or hue. This follows recommended usage for CMC and CIEDE2000. There are two variants: Relative colorimetric and absolute colorimetric.

**Relative colorimetric:** First, align the source and destination neutral axes as described previously. Then clip the adjusted source color to the adjusted destination gamut boundary. (See Figure 4. Chroma mapping along constant lightness.) Readjust the destination device values as described previously. In the case of a monochrome destination gamut boundary, the chroma clipping means that the chroma value (C) is set to zero (0.0).

**Absolute colorimetric:** This is similar to relative colorimetric, but without the alignment of the source and destination neutral axis. The source value is clipped directly to the destination neutral axis. Note that if both the source and destination gamut boundaries are monochrome, the behavior is identical to the relative colorimetric variant; that is, the neutral axes alignment is performed, and then the chroma is clipped to zero. This ensures that a reasonable output is attained even if the media and colorants are significantly different.



**Figure 3 : MinCD clipping to the adjusted gamut**

## BasicPhoto

### Overview

BasicPhoto - equivalent to the ICC preferred, pictorial, or perceptual intent.

This algorithm is a variant of the chroma-dependent sigmoidal lightness mapping and cusp knee scaling (SGCK) described by CIE TC8-03 in CIE156:2004. This variant algorithm supports gamut boundary descriptors (GBDs) with dual gamut shells; that is, GBDs with a reference shell and a plausible shell. The SGCK algorithm, which originally assumes only one gamut shell in the GBD, is based on the SIG\_KNEE algorithm (Braun 1999), that incorporates the sigmoidal lightness mapping and knee function scaling proposed by Braun and Fairchild (1999), combined with the chroma dependence from GCUSP (Morovic, 1998). It keeps perceived hue constant, for example, hue angle in hue-corrected Jab, and uses a generic (image-independent) sigmoidal lightness scaling which is applied in a chroma-dependent way and a 90 percent knee function chroma. The variant scales along lines of constant lightness.

### The case of single gamut shell

It is helpful to review the algorithm in the case where both source and destination GBDs have only one gamut shell. In this case, the algorithm consists of the following calculations.

First, perform initial lightness mapping using the following formula:

$$J_R = (1 - p_C) J_O + p_C J_S \quad (1)$$

where  $J_O$  is the original lightness and  $J_R$  is the reproduction lightness.

$$p_C = 1 - ((C^3) / (C^3 + 5 \times 10^5))^{1/2} \quad (2)$$

When the source gamut boundary is monochrome, the chroma value will be zero for the monochrome boundary due to neutral axis alignment. This will result in the degenerate case where  $C$  is equal to zero. In this case,  $p_C$  is set to 1.

$p_C$  is a chroma-dependent weighting factor (Morovic, 1998) that depends on the chroma of the original color,  $C$  and  $J_S$  are the result of the original lightness being mapped using a sigmoidal function.

To calculate  $J_S$  (Braun and Fairchild, 1999), a one-dimensional lookup table (LUT) between original and reproduction lightness values is first set up on the basis of a discrete cumulative normal function ( $S$ ).

$$S_i = \sum_{n=0}^{m-1} \frac{1}{\sqrt{2\pi}\Sigma} e^{-\left(\frac{(100n-x_0)}{2\Sigma}\right)^2} \quad (3)$$

where  $x_0$  and  $\Sigma$  are the mean and standard deviation of the normal distribution respectively and  $i = 0, 1, 2, \dots, m$ ,  $m$  is the number of points used in the LUT.  $S_i$  is the value of the cumulative normal function for  $i/m$  percent. The parameters depend on the lightness of the black point of the reproduction gamut and can be interpolated from Table 1. For details of calculating these parameters see Braun and Fairchild (1999, p. 391).

$J_{\text{minOut}}$

5.0

10.0

15.0

20.0

$x_0$

53.7

56.8

58.2

60.6

S

43.0

40.0

35.0

34.5

**Table 1 :** BasicPhoto lightness compression parameter calculation

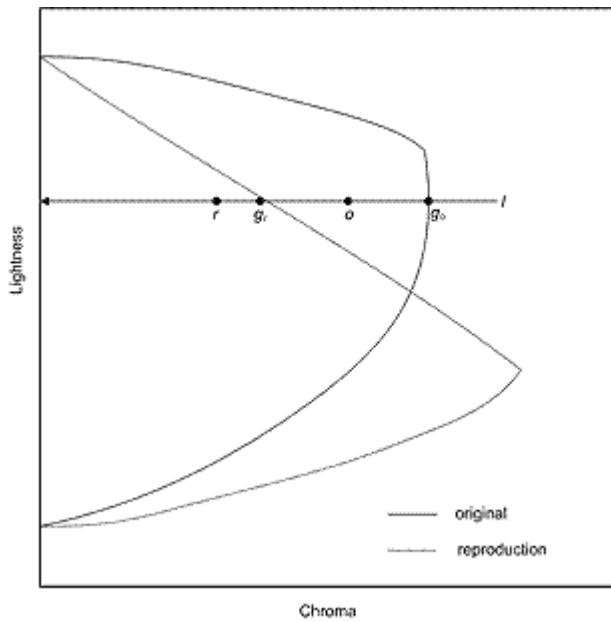
To use S as a lightness mapping LUT ( $S_{LUT}$ ) it must first be normalized into the lightness range of [0,100]. The normalized data is then scaled into the dynamic range of the destination device, as shown in Equation 4, where  $J_{min\ Out}$  and  $J_{max\ Out}$  are the values of lightness of the black point and the white point of the reproduction medium, respectively.

$$S_{LUT} = \frac{(S_i - \min(S))}{(\max(S) - \min(S))} (J_{max\ Out} - J_{min\ Out}) + J_{min\ Out} \quad (4)$$

At this point, the  $J_S$  values can be obtained from the  $S_{LUT}$  by interpolating between the  $m$  points of corresponding  $J_O'$  and  $J_S$  values it contains, and using the following equation as the input.

$$J_O' = 100(J_O - J_{min\ In}) / (J_{max\ In} - J_{min\ In}) \quad (5)$$

$J_{min\ In}$  is the lightness value of the black point of the original medium,  $J_{max\ In}$  is the lightness value of the white point of the original medium, and  $J_O$  is the original lightness. For later reference, you can denote by  $S$  the sigmoidal function defined in the manner just outlined, as illustrated in the following Figure 4.

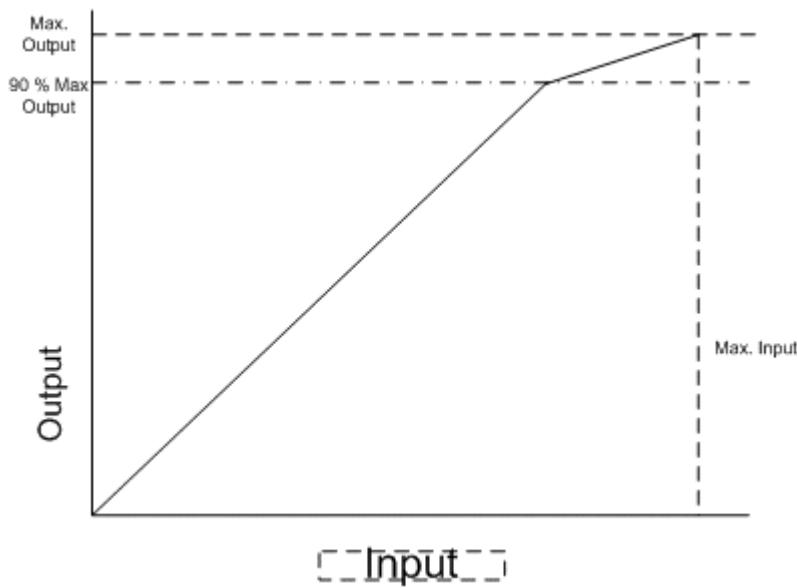


**Figure 4 : Chroma mapping along constant lightness**

Second, if the destination gamut boundary is chromatic, compress chroma along lines of constant lightness ( $l$ ) and perform the compression as follows.

$$d_r = \begin{cases} d_o; & d_o \leq 0.9d_{gr} \\ 0.9d_{gr} + (d_o - 0.9d_{gr})0.1d_{gr}/(d_{go} - 0.9d_{gr}); & d_o > 0.9 \times d_{gr} \end{cases} \quad (6)$$

where  $d$  represents distance from  $E$  on  $l$ ;  $g$  represents the medium gamut boundary;  $r$  represents the reproduction; and  $o$  the original Figure 5.



**Figure 5 : Chroma and lightness compression in BasicPhoto**

If the destination gamut boundary is monochrome, then the chroma value is clipped to zero.

Third, use a MinCD clip (described earlier) to eliminate any residual error.

## Black enhancement

The preceding algorithm can be modified to improve the black when the destination is a printer device. The issue has to do with the choice of  $J_{minOut}$ , which usually does not correspond to the darkest color a printer can produce.

More specifically, the color with highest density, obtained by putting 100 percent inks (or maximum possible coverage, if GCR/ink limiting is in effect), is usually not "neutral" in the color appearance space. See Figure 6. In other words, if neutral minimum lightness is used for the destination device, the lightness scaler constructed will map to a minimum lightness that is not the highest density that can be achieved by the printer. Consider the further use case of monitor to printer. The monitor black,  $R=G=B=0$ , will then be printed as not the highest density. The impact on the image quality is that there is a lack of depth and contrast.

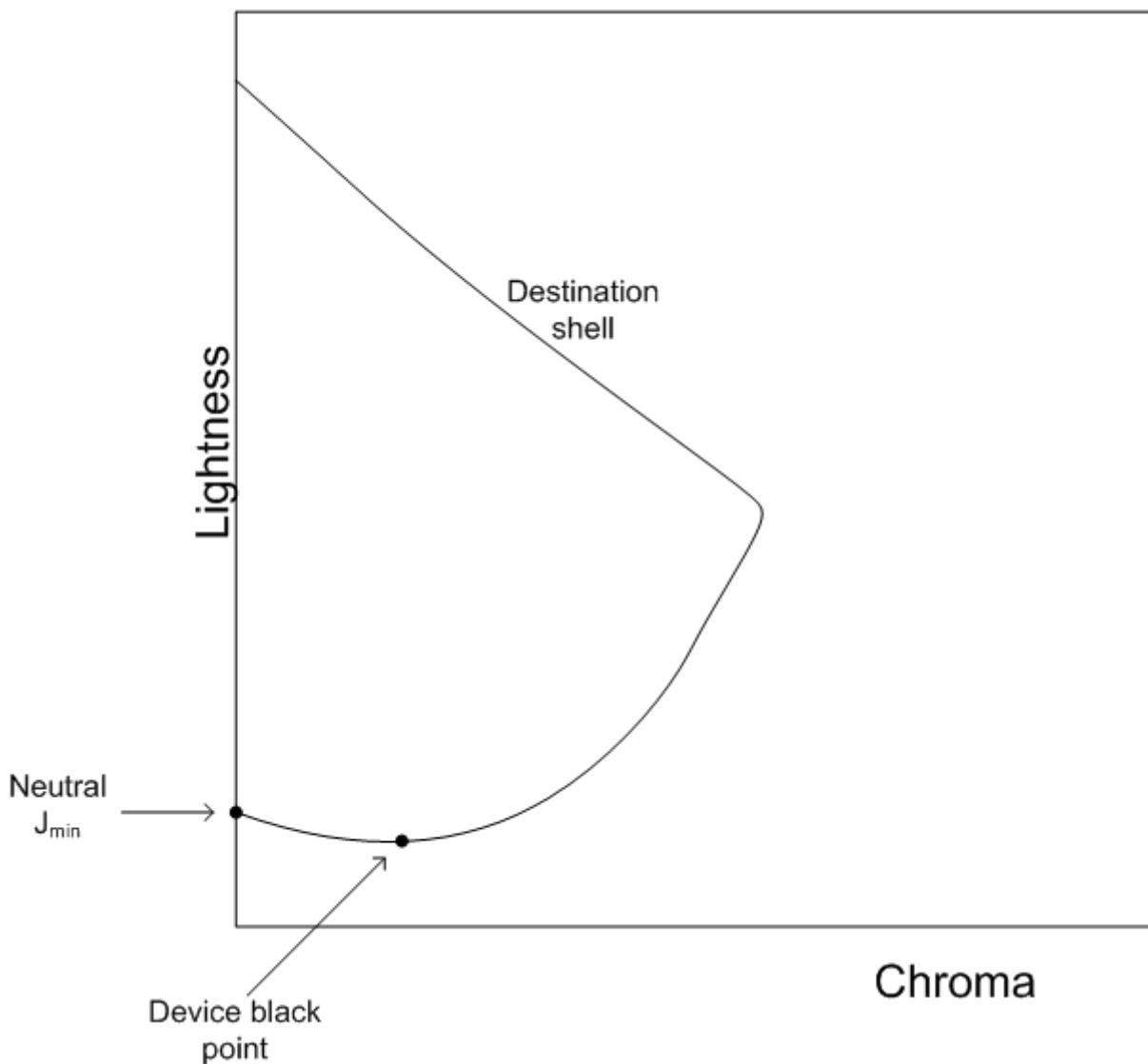


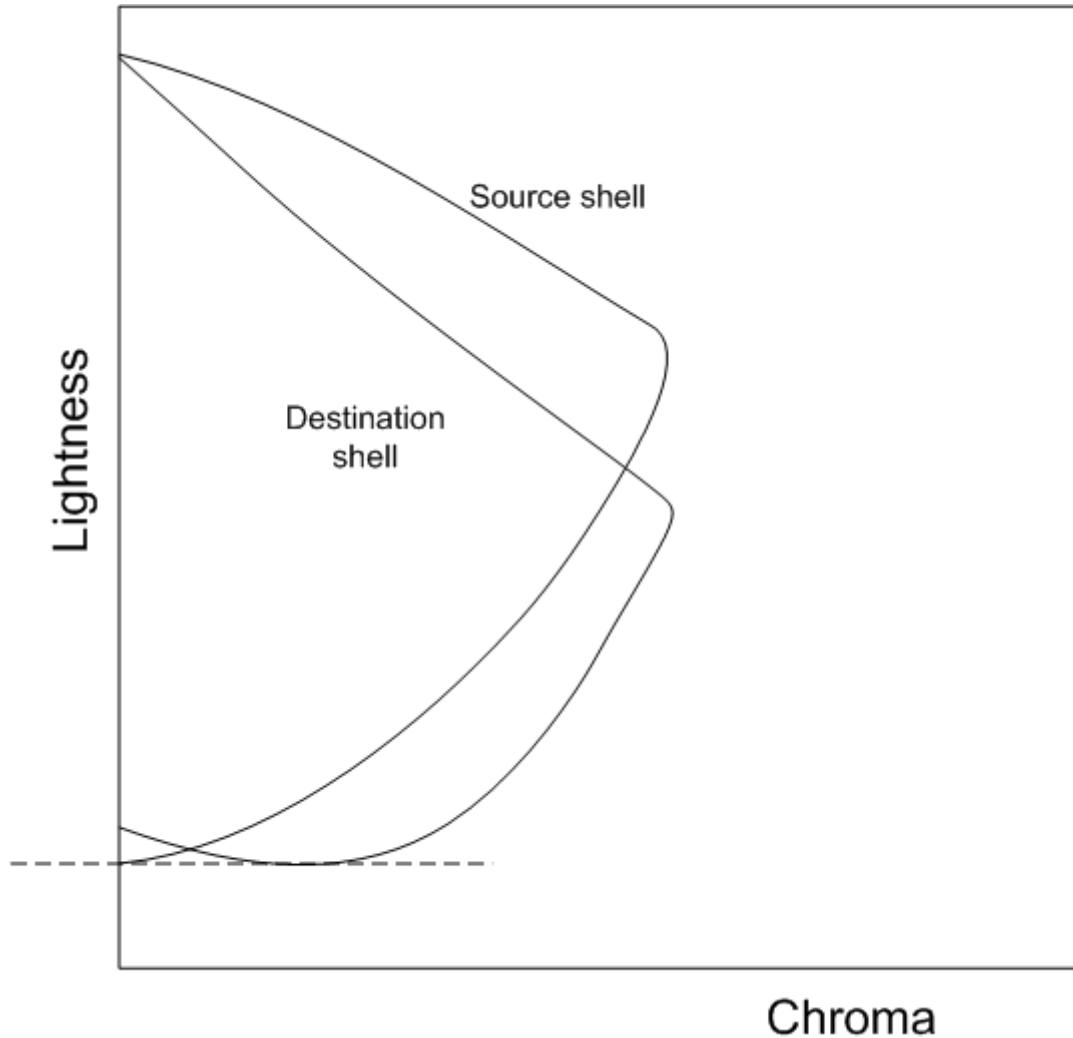
Figure 6 : The device black point may be darker than the neutral minimum lightness.

Suppose the destination "device black point" is  $J_k a_k b_k / J_k C_k h_k$ . If  $C_k$  is not zero, then the device black point is not neutral relative to CAM02. If you use  $J_k$  for the destination

"neutral minimum lightness" in the construction of the lightness scaler; that is, setting

$$J_{minOut} = J_k$$

and apply it to the source gamut shell, you obtain the configuration depicted in Figure 7. In the figure, the hue plane corresponds to  $h_k$ .



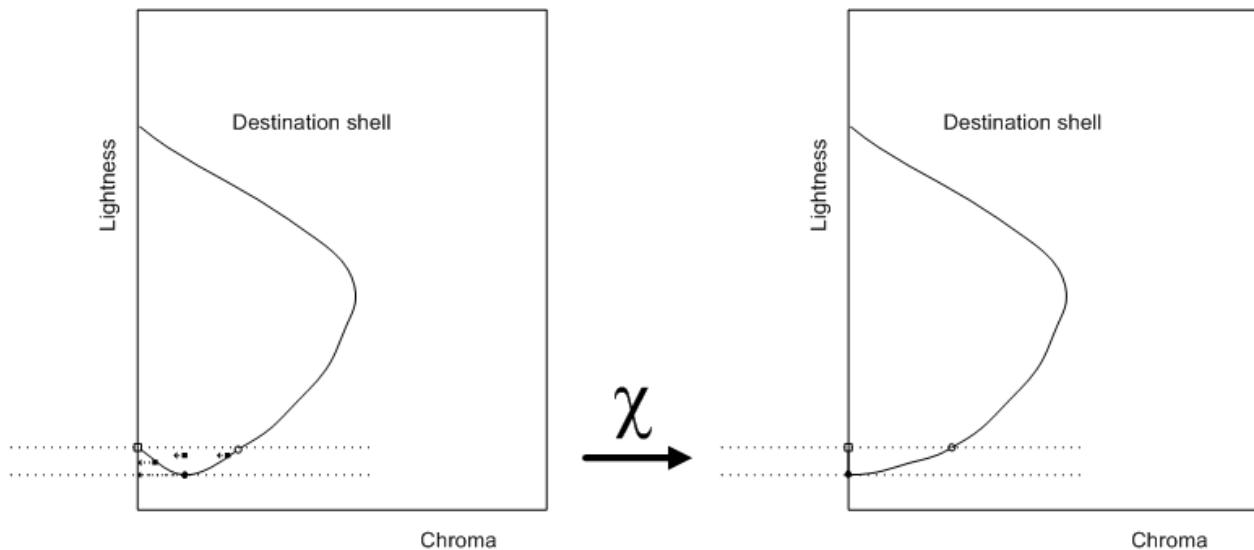
**Figure 7 :** Geometry using the modified lightness scaler with destination device black point

To allow the subsequent chroma compression algorithm to proceed, you want to align the maximum and minimum lightnesses on the source and destination shells. This can be achieved by adjusting the destination shell between  $J_{neutralMin}$  and  $J_k$  by shifting points to the left. Furthermore, this transformation must be applied on the whole Jab space, not just the hue plane corresponding to  $h_k$ .

The transformation is

$$\chi(J, \alpha, b) = \begin{cases} (J, \alpha, b) & \text{if } J \geq J_{neutralMin} \\ \left( J - \frac{J_{neutralMin} - J}{J_{neutralMin} - J_k} \alpha_k, b - \frac{J_{neutralMin} - J}{J_{neutralMin} - J_k} b_k \right) & \text{if } J < J_{neutralMin} \end{cases}$$

Figure 8 shows the effect of the transformation.



**Figure 8 :** Geometry using the modified lightness scaler with destination device black point

After applying the usual chroma compression algorithm, the point must then be "shifted back;" that is, the inverse transformation must be applied to obtain the final mapped color.

$$J_S = \begin{cases} J_{r_{\max, ref}} + \frac{J_{r_{\max, pla}} - J_{r_{\max, ref}}}{J_{o_{\max, pla}} - J_{o_{\max, ref}}}(J_O - J_{o_{\max, ref}}) & \text{if } J_O \geq J_{o_{\max, ref}} \\ S(J_O; J_{o_{\min, ref}}, J_{o_{\max, ref}}; J_{r_{\min, ref}}, J_{r_{\max, ref}}) & \text{if } J_{o_{\min, ref}} < J_O < J_{o_{\max, ref}} \\ J_{r_{\min, ref}} + \frac{J_{r_{\min, pla}} - J_{r_{\min, ref}}}{J_{o_{\min, pla}} - J_{o_{\min, ref}}}(J_O - J_{o_{\min, ref}}) & \text{if } J_O \leq J_{o_{\min, ref}} \end{cases}$$

## The case of dual gamut shells

The goal is to generalize SIG\_KNEE for single gamut shell to the case where either the source device GBD or the destination device GBD has a two-shell structure. The inner shell will be called the Reference Shell, while the outer shell will be called the Plausible Shell. You want to consider the following cases.

- (a) Both the source GBD and destination GBD have a two-shell structure.
- (b) The source GBD has a two-shell structure; the destination GBD has only one shell.
- (c) The source GBD has only one shell; the destination GBD has a two-shell structure.
- (d) Both the source GBD and destination GBD have only one shell.

Case (d) is the case of single gamut shell that was discussed previously. For cases (a), (b), and (c), you can generalize the lightness rescaling to use the extra information from the

dual shell structure. In cases (b) and (c) where either the source or the destination has only one shell, you introduce an "induced reference shell" which will be discussed in a subsequent section, "Induced Reference Shell." The general algorithm for two shells will be described for case (a). After the Induced Reference Shell construction is explained, the algorithm can be applied to case (b) and (c), as well. As for chroma compression, the compression ratio will be determined by the largest shells available. In other words, if both the Plausible Shell and Reference Shell are available, the Plausible Shell will be used; otherwise, the Reference Shell will be used.

### *Generalized Lightness Rescaling*

The existence of two shells for both source and destination GBD means that you must map a set of four points from the source GBD to a corresponding set in the destination GBD.

$$(J_{o,\min,pla}, J_{o,\min,ref}, J_{o,\max,ref}, J_{o,\max,pla}) \rightarrow (J_{r,\min,pla}, J_{r,\min,ref}, J_{r,\max,ref}, J_{r,\max,pla})$$

The subscripts have the following meanings.

o or r: "original" (source) or "reproduction" (destination)

min or max: minimum neutral lightness or maximum neutral lightness

pla or ref: Plausible Shell or Reference Shell

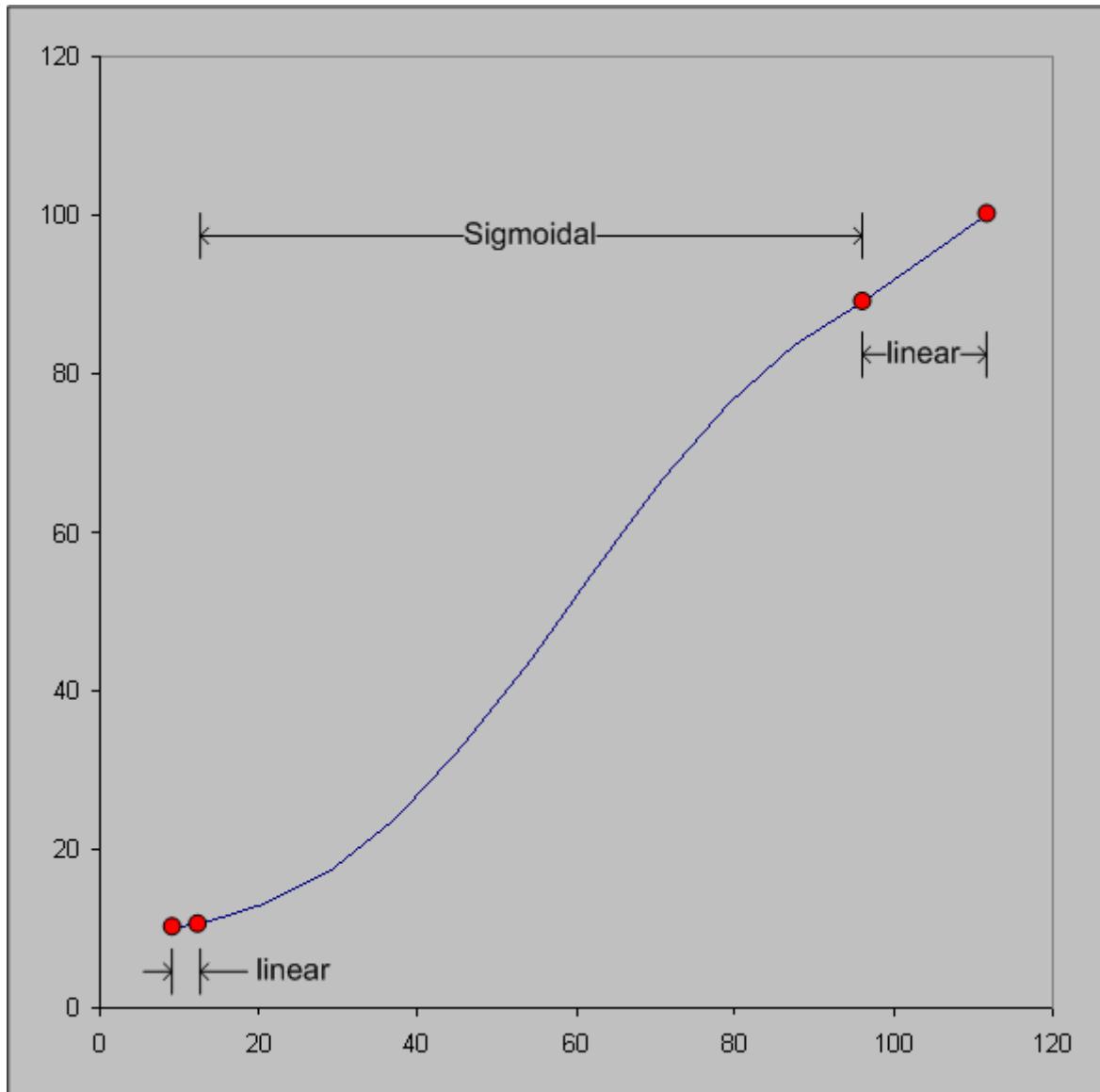
The ordering in each quadruple is also the expected relative magnitude of these points.

The Lightness Rescaling map uses the same first two equations as the single shell, but  $J_S$  is defined in a piecewise manner as follows.

$$J_S = \begin{cases} J_{r,\max,ref} + \frac{J_{r,\max,pla} - J_{r,\max,ref}}{J_{o,\max,pla} - J_{o,\max,ref}} (J_O - J_{o,\max,ref}) & \text{if } J_O \geq J_{o,\max,ref} \\ S(J_O; J_{o,\min,ref}, J_{o,\max,ref}; J_{r,\min,ref}, J_{r,\max,ref}) & \text{if } J_{o,\min,ref} < J_O < J_{o,\max,ref} \\ J_{r,\min,ref} + \frac{J_{r,\min,pla} - J_{r,\min,ref}}{J_{o,\min,pla} - J_{o,\min,ref}} (J_O - J_{o,\min,ref}) & \text{if } J_O \leq J_{o,\min,ref} \end{cases}$$

(7)

In other words, it is sigmoidal within the reference shell, and linear outside. See Figure 9.



**Figure 9 :** Lightness Rescaling function for two-shell GBDs

#### INDUCED REFERENCE SHELL

Where one GBD has one shell and the other GBD has two shells, you must create a "Reference Shell" for the GBD with only one shell. The existing shell, which would be called the Reference Shell, will be changed to the "Plausible Shell." In fact, you don't actually have to create a shell in the full Jab space. Because the lightness rescaling only uses  $J_{max}$  and  $J_{min}$ , you only have to make up these values for the induced Reference Shell. There are two cases, depending on which GBD has two shells.

Case 1: Source GBD has two shells; destination GBD has one shell.

Determine the destination induced Reference Shell on the neutral axis; that is, the  $J_{r,\text{min},\text{ref}}$  and  $J_{r,\text{max},\text{ref}}$  of the shell. This is done using the following algorithm.

$$J_{r,\min,ref} = \frac{J_{r,\min,pla} + J_{r,\max,pla}}{2} + \lambda_{low} \left( J_{r,\min,pla} - \frac{J_{r,\min,pla} + J_{r,\max,pla}}{2} \right)$$

$$J_{r,\max,ref} = \frac{J_{r,\min,pla} + J_{r,\max,pla}}{2} + \lambda_{high} \left( J_{r,\max,pla} - \frac{J_{r,\min,pla} + J_{r,\max,pla}}{2} \right)$$

The factors  $\lambda_{low}$  and  $\lambda_{high}$  control the separation between the Plausible Shell and the Reference Shell. A value of 1 means that the  $J_{\min}$  values or  $J_{\max}$  values coincide. Their values are "induced" from the source Reference Shell and source Plausible Shell.

$$J_{mid} = \frac{J_{o,\min,ref} + J_{o,\max,ref}}{2}$$

$$\lambda_{low} = F_{low} + (1 - F_{low}) \frac{J_{mid} - J_{o,\min,pla}}{J_{mid} - J_{o,\min,pla}}$$

$$\lambda_{high} = F_{high} + (1 - F_{high}) \frac{J_{o,\max,ref} - J_{mid}}{J_{o,\max,pla} - J_{mid}}$$

The "fudge factors"  $F_{low}$  and  $F_{high}$  are *tunable parameters* that must lie between 0 and 1. If the value is 0, then the  $J_{\min}$  or  $J_{\max}$  are directly induced from the source shells. For this case, choose  $F_{low} = 0.95$ , and  $F_{high} = 0.1$ .

Case 2: Source GBD has one shell; destination GBD has two shells.

Determine the source induced Reference Shell on the neutral axis; that is, the  $J_{o,\min,ref}$  and  $J_{o,\max,ref}$  of the shell. This is done using the following algorithm.

$$J_{o,\min,ref} = \frac{J_{o,\min,pla} + J_{o,\max,pla}}{2} + \lambda_{low} \left( J_{o,\min,pla} - \frac{J_{o,\min,pla} + J_{o,\max,pla}}{2} \right)$$

$$J_{o,\max,ref} = \frac{J_{o,\min,pla} + J_{o,\max,pla}}{2} + \lambda_{high} \left( J_{o,\max,pla} - \frac{J_{o,\min,pla} + J_{o,\max,pla}}{2} \right)$$

Again, the factors  $\lambda_{low}$  and  $\lambda_{high}$  control the separation between the Plausible Shell and Reference Shell. A value of 1 means the  $J_{\min}$  values or  $J_{\max}$  values coincide. Their values are "induced" from the source Reference Shell and source Plausible Shell:

$$J_{mid} = \frac{J_{r,\min,ref} + J_{r,\max,ref}}{2}$$

$$\lambda_{low} = \frac{J_{mid} - J_{r,\min,pla}}{J_{mid} - J_{r,\min,pla}}$$

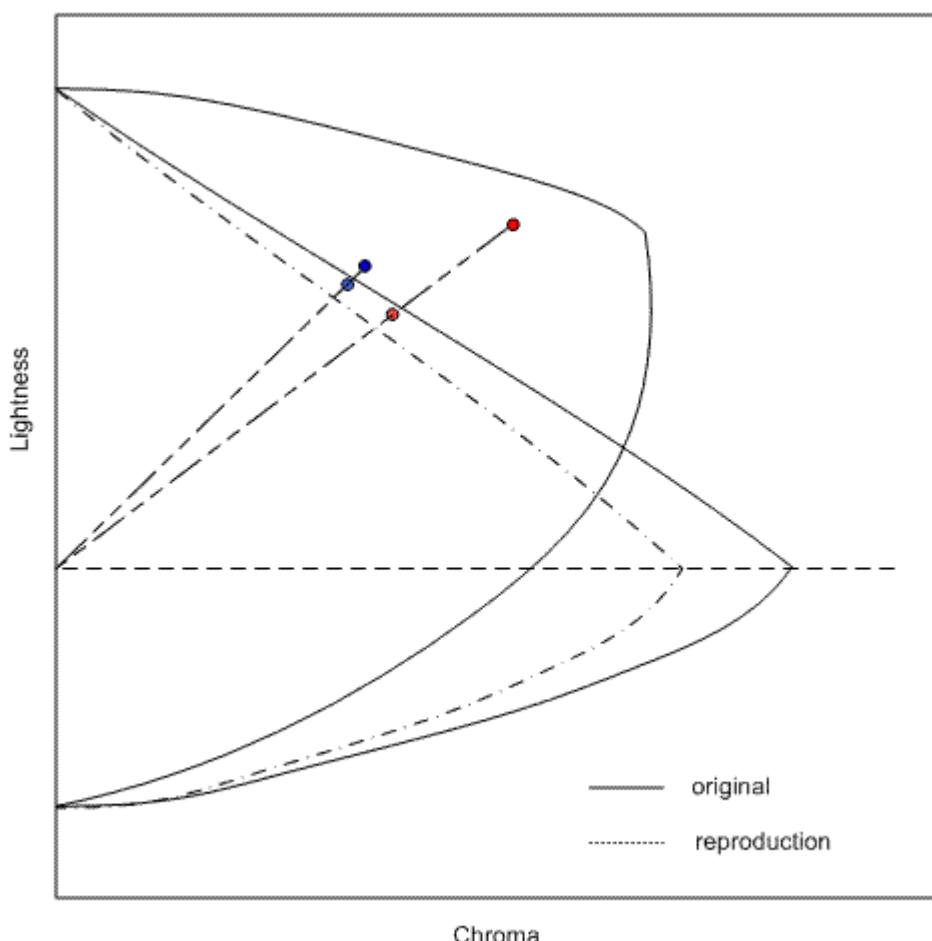
$$\lambda_{high} = \frac{J_{r,\max,ref} - J_{mid}}{J_{r,\max,pla} - J_{mid}}$$

# Reasons for changes from the CIE TC8-03 recommendations

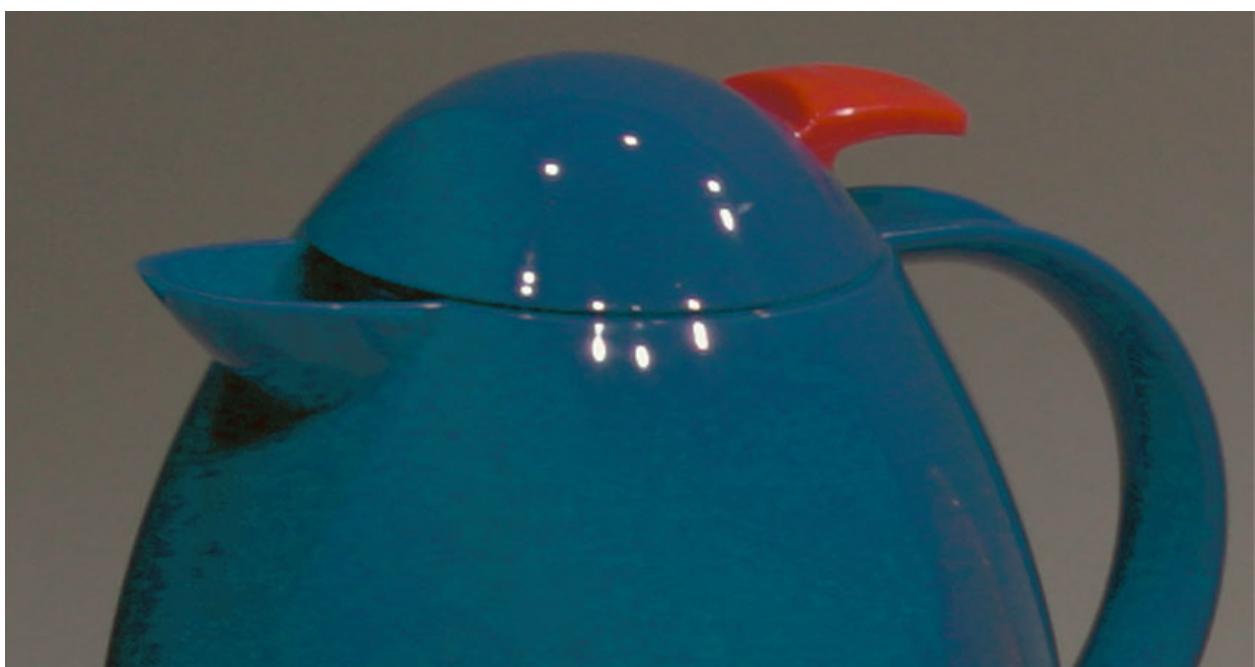
BasicPhoto differs from the CIE TC8-03 recommendations in the following ways.

1. Chroma is not compressed toward the cusp but along lines of constant lightness.
2. The lightness range uses the lightness of the darkest color in the gamut rather than the point at which the gamut boundary crosses the neutral axis.
3. BasicPhoto supports both a reference gamut shell and a plausible gamut shell, if either gamut boundary in the transform has two shells.
4. BasicPhoto uses CIECAM02; instead of using CIECAM97s to convert to D65 at 400 cd/m<sup>2</sup>, and then using RIT IPT color space.

The first change was made to prevent tone inversion problems that can occur when using compression toward a cusp. As shown in Figure 10, cusp compression can cause tone inversions. This can happen when colors of high chroma are lighter than colors of lower chroma. Because SGCK compresses each pixel independently in both lightness and chroma, it is not guaranteed to preserve the lightness relationship between the pixel values after the compression. The well-known downside to this decision to compress on lines of constant lightness is that you can suffer losses of chroma, particular in areas where the destination gamut boundary is very flat, as happens with bright yellows.



**Figure 10** : Tone inversion caused by SGCK







**Figure 11 :** Original image, SGCK result, and BasicPhoto result

Figure 11 illustrates this tone inversion. On the left is an original image captured by a digital camera; in the center, the image as reproduced by SGCK; and on the right, the image as reproduced by BasicPhoto. The image on the left is in the digital camera's color space, the center and right images are in the color space of an LCD video display. In the original image, the top part of the teapot is darker than the bottom, because the bottom is reflecting the tablecloth that it is sitting upon. In the SGCK image, the top part is actually lighter than the bottom, because of the tone inversion. Also, it is difficult to see the items reflected in the lower part of the teapot. On the right, BasicPhoto has fixed the tone inversion and the reflected articles are more clearly distinguishable.

The second change was made to improve the reproduction of nearly black colors on printers where the blackest black does not fall directly on the CIECAM02 neutral axis. The following Figure 12 shows an image converted to sRGB; reproduced for an RGB inkjet printer using SGCK; and reproduced for the same printer using BasicPhoto. The image in the center is not using the full device black, and so it lacks the contrast seen in the original. The contrast is restored with BasicPhoto.





**Figure 12 : Enhanced black**

The third change was made to improve the color reproduction for digital cameras. Particularly in cases where the digital camera has been profiled using a reference target, a gamut boundary description built from measured colors might not include all the colors that might be captured in a real world scene. Rather than clipping all colors to the gamut of the measured color target, you allow for extrapolation to produce a plausible gamut boundary. The BasicPhoto algorithm is designed to support such an extrapolated gamut boundary.

The fourth change was made because CIECAM02 works well for gamut mapping. The process recommended by TC8-03 of converting device colors to D65 at 400 cd/m<sup>2</sup>, and then using the RIT IPT color space is both computation-intensive and time-consuming.

## Hue mapping

HueMap is the equivalent of the ICC Saturation intent.

If either the source gamut boundary or the destination gamut boundary does not contain primaries, this model reverts to the MinCD (relative) model described in a preceding section; for example, devices for which the primaries cannot be determined (ICC profiles with more than four channels) or monochrome ICC profiles.

This algorithm first adjusts the hue of the input color value. Then it simultaneously adjusts the lightness and chroma, using a shearing mapping. Finally, it clips color value

to make sure it is within gamut.

The first step is to determine the "Hue Wheels." Find the JCh values for primary and secondary colors for both source and destination device. You are only considering the hue components. This results in a primary or secondary hue wheel with six color points for each device. (See Figure 13.)

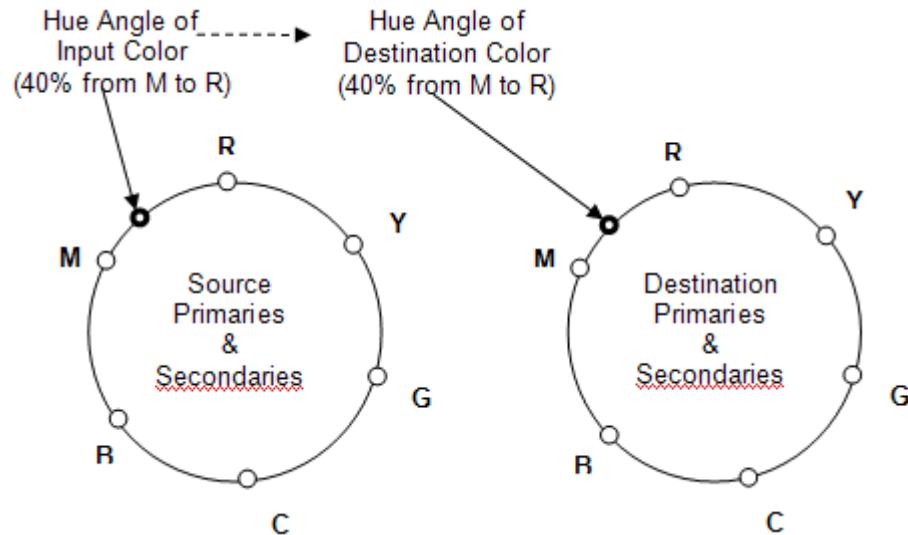


Figure 13 : Hue wheels

Better results can be obtained if the source blue primary is not rotated to the destination blue primary. Instead, the source blue primary hue angle is used as the destination blue primary hue angle.

Next, perform the hue rotations for each input color from the source image,

a)Using the hue angle of the input color, determine the location of the color on the source hue wheel relative to the two adjacent primary or secondary color. The location can be thought of as a percentage of the distance between the primaries. For example, the input color hue is 40 percent of the way from the hue value of Magenta to the hue value of Red.

b)On the destination hue wheel, find the associated hue angle, for example, 40 percent from Magenta to Red. This value will be the destination hue angle.

In general, the source primaries and secondaries will not be at the same hue angles as the destination primaries and secondaries; that is, the destination hue angle usually will be different from the source hue angle.

For example, suppose the hue wheels produce the following values:

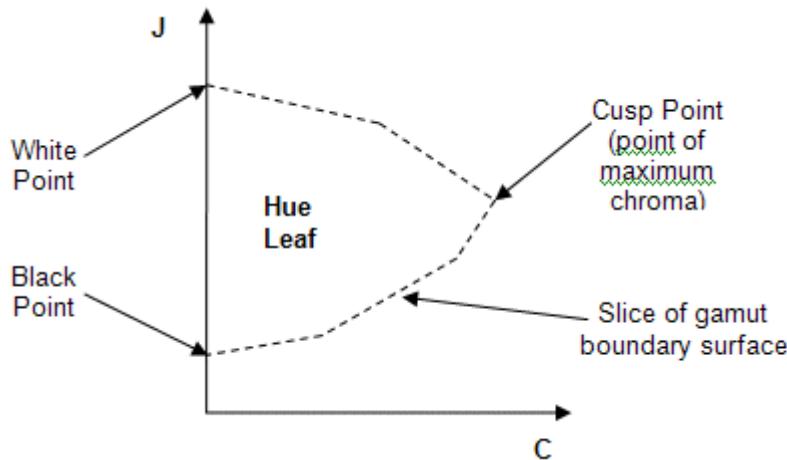
Source M = 295 degrees, Source R = 355 degrees.

Destination M = 290 degrees, Destination R = 346 degrees.

If the hue angle of the input color is 319 degrees, it is 40 percent of the angle (24 degrees) from source M to source R. The angle from M to R is 60 degrees, and the angle from M to input hue is 24 degrees. Calculate the angle on the destination that is 40 percent from destination M to destination R (22 degrees), so the hue angle of the destination color is 312 degrees.

Next, calculate the hue reference points for the source hue and the destination hue. To calculate the hue reference point for a particular h (hue) value, you want to find the J (lightness) value and C (chroma) value.

- Find the J value of the hue reference point by interpolating between the J values for the adjacent primary or secondary points, using the relative position of the hue; for example, 40 percent in this example.
- Find the maximum C value at this J value and h value. You now have the JCh of the hue reference point for that hue.



**Figure 14 :** A hue leaf (visualization of a gamut boundary slice at a specific hue)

The next step is to compute the shear mapping for each pixel. First, visualize a hue leaf from the source gamut for the source color hue angle, and a hue leaf from the destination gamut for the destination hue angle computed during hue rotation. The hue leaves are created by taking a "slice" from the JCh gamut boundary surface at a specific hue angle (see Figure 14).

NOTE: For performance optimization reasons, hue leaves are not actually created; they are described and shown here for visualization purposes only. The operations are performed directly on the gamut boundary surface at the specified hue. You then compute the hue reference points to determine the shear mapping.

- Perform a lightness rescaling to map the black and white points of the source leaf to the destination leaf (see Figure 15). The black and white points of the source hue leaf are mapped linearly to the black and white points of the destination hue

leaf, by scaling all the J coordinates of the source boundary. The hue-mapped input color value is scaled in the same manner.

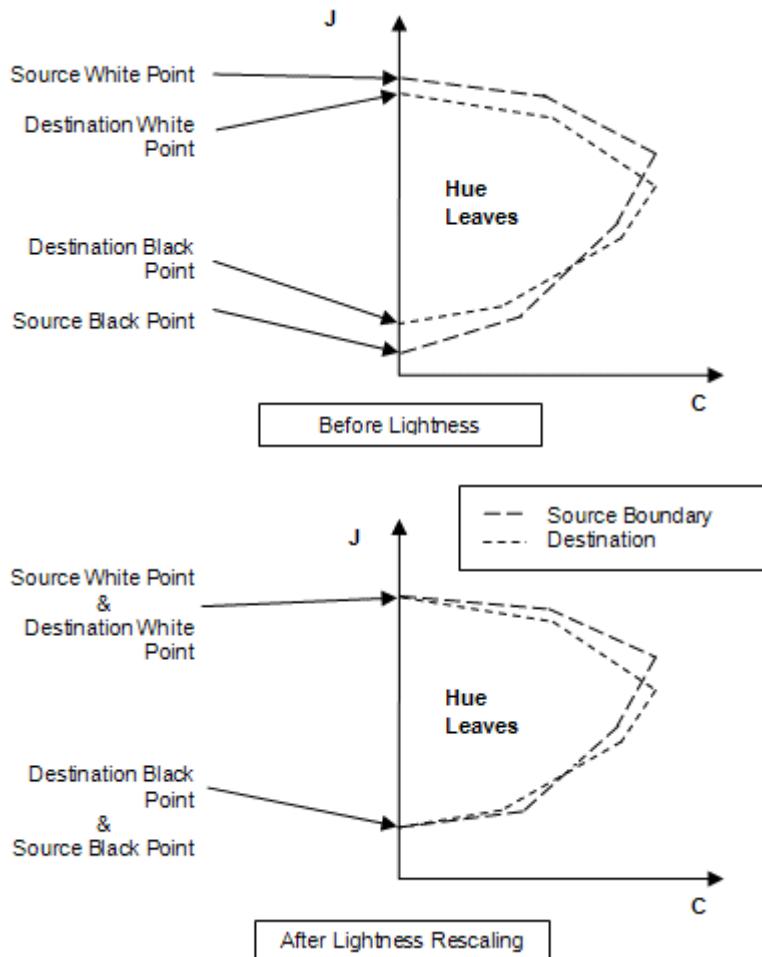


Figure 15 : Lightness mapping

- Determine the hue reference points for each hue leaf. Apply a shear mapping to the source leaf so that the source and destination reference points coincide (see Figure 16). The reference point for a gamut at a specific hue is the interpolated hue reference point between the adjacent primaries. The reference point of the source hue leaf is mapped linearly to the reference point of the destination hue leaf, using a "shear" operation that locks the J axis, keeping the black points and the white points stationary. The black points, white points, and reference points of the source and destination hue leaves should coincide.
- Apply the shear mapping to the lightness-adjusted input color value. The J and C coordinates of the source color value are proportionally scaled, relative to its distance from the J axis.
- Next, a subtle chroma-dependent lightness compression toward the J-value of the hue reference point is performed on the shear-mapped color point. The compression toward hue reference J is done in a gamma-like fashion, where white, black, grays, and points on the hue reference J are not affected. All other points tend toward the hue reference J in a smooth fashion, slightly bunching up near the

hue reference J, with chroma remaining constant. The chroma dependency assures that neutral colors are not affected, and the effect is increased on colors with higher chroma.

The following is a mathematical description of the lightness compression toward the hue reference J, or adjusting the J-value of the destination point. It is called the destination point because it has been shear mapped into the destination gamut.

First, compute "factorC" (chroma dependency factor) for the destination point, which determines how much effect the lightness compression will have. Points near or on the J-axis will have little or no compression; points farther away from the J-axis (high-chroma) will have more compression applied. Multiply by 0.5 to ensure that factorC is less than 1, because it is possible that sourceC could be slightly greater than referenceC, but not twice as great.

$$\text{factorC} = (\text{destinationC} / \text{referenceC}) ? 0.5$$

where:

destinationC is the C-value of the destination point.

referenceC is the C-value of the Hue Reference point.

Next, determine whether the destination point J is above or below the hue reference J. If it is above, do the following:

1. Compute "factorJ" for the destination point relative to the referenceJ. This factorJ value will be between 0 and 1 (0 if on the referenceJ; 1 if at maxJ).
2.  $\text{factorJ} = (\text{destinationJ} - \text{referenceJ}) / (\text{maxJ} - \text{referenceJ})$

where:

destinationJ is the J-value of the destination point.

referenceJ is the J-value of the hue reference point.

maxJ is the maximum J-value of the gamut.

3. Apply a gamma-like power function to factorJ, which will reduce factorJ by a certain amount. This example uses the power of 2 (the square). Subtract the reduced factorJ from the original factorJ and multiply the result by the total J-range above the primary referenceJ to find the "deltaJ," which represents the change in J after the lightness compression, but not including the chroma dependency.

4.  $\text{deltaJ} = (\text{factorJ} - (\text{factorJ} ? \text{factorJ})) ? (\text{maxJ} - \text{referenceJ})$
5. Apply factorC to the deltaJ (the higher the chroma, the greater the effect), and compute the new J-value for the destination point.
6.  $\text{destinationJ} = \text{destinationJ} - (\text{deltaJ} ? \text{factorC})$

If J-value for the destination point is below referenceJ, then a similar computation to the preceding steps A-C is performed, using minJ instead of maxJ to find the range in J to compute the factorJ, and taking into account the polarity of the operations "underneath" the referenceJ.

$$\text{factorJ} = (\text{referenceJ} - \text{destinationJ}) / (\text{referenceJ} - \text{minJ})$$

$$\text{deltaJ} = (\text{factorJ} - (\text{factorJ} ? \text{factorJ})) ? (\text{referenceJ} - \text{minJ})$$

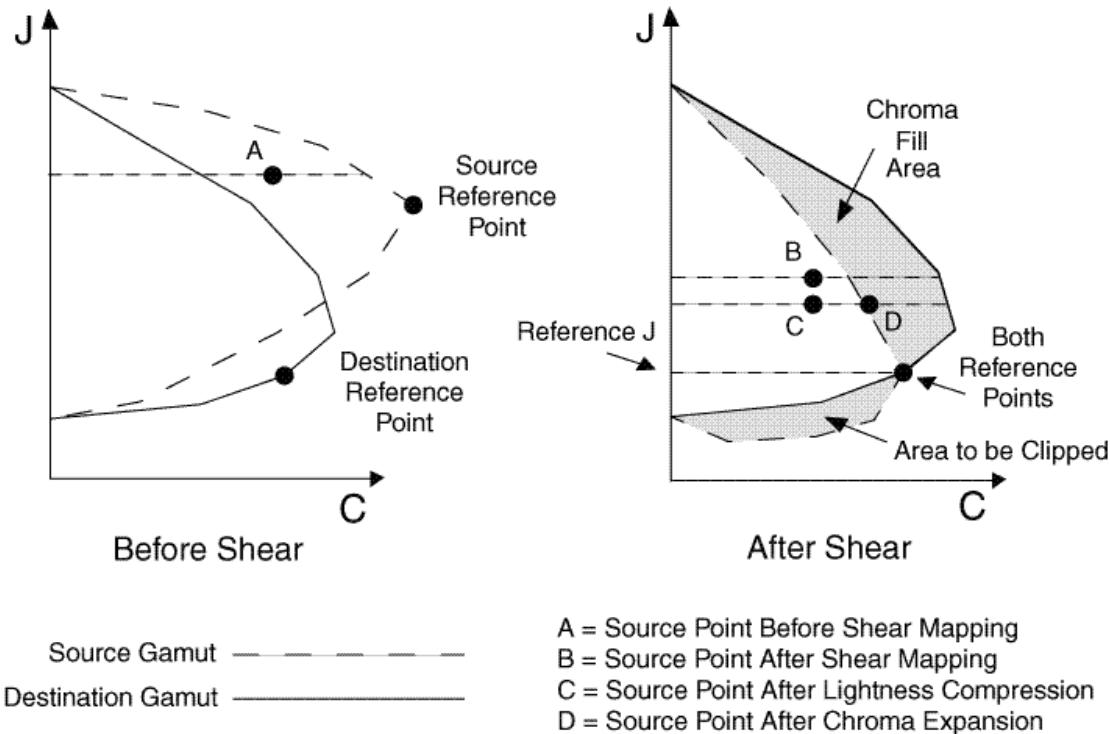
$$\text{destinationJ} = \text{destinationJ} + (\text{deltaJ} ? \text{factorC})$$

where:

minJ is the minimum J-value of the gamut.

The chroma for input color points is expanded linearly (when possible) along constant lightness proportional to the maximum chroma value of the source and destination gamuts at that hue and lightness. Combined with the preceding chroma-dependent lightness compression, this helps preserve the saturation because shear mapping using the reference points sometimes causes the source point to over-compress in chroma (see Figure 16).

**Shear Mapping to Match Hue Reference Points  
Followed by Lightness Compression Toward Hue Reference J  
and Chroma Expansion**



**Figure 16 :** Shear mapping, lightness compression toward hue reference J, and chroma expansion

The following is a mathematical description of the chroma expansion process, or adjusting the C-value of the destination point. It is called the destination point because it has been shear mapped and lightness compressed into the destination gamut.

1. Before the shear mapping, determine `sourceExtentC` (the chroma extent at the lightness and hue of the source point).
2. After the shear mapping and lightness compression which transforms the source point into the destination point, determine `destExtentC` (the chroma extent at the lightness and hue of the destination point).
3. If `sourceExtentC` is greater than `destExtentC`, no chroma adjustment to the destination point is necessary, and you can skip the next step.
4. Adjust `destinationC` (the chroma of the destination point) to fit within the destination chroma extent at this lightness and hue.
5. 
$$\text{destinationC} = \text{destinationC} ? (\text{destExtentC} / \text{sourceExtentC})$$

where:

destinationC is the destination point C-value.

sourceExtentC is the maximum C-value of the source gamut at the lightness and hue of the source point.

destExtentC is the maximum C-value of the destination gamut at the lightness and hue of the destination point.

Finally, perform the minimum distance clipping. If the hue-rotated, lightness-adjusted, and shear-mapped input color is still slightly outside the destination gamut, clip it (move it) to the closest point on the destination gamut boundary (see Figure 17).

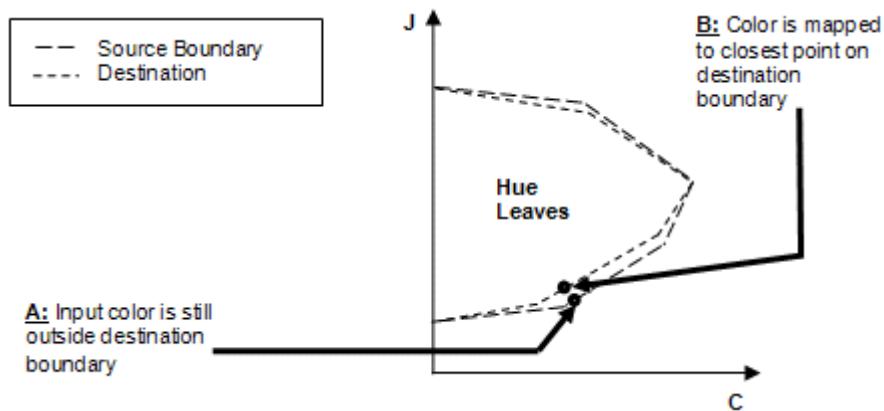


Figure 17 : Minimum distance clipping

## Gamut Boundary Description and Gamut Shell Algorithms

The device gamut boundary function uses the device model engine and analytical parameters to derive a color device gamut boundary, described as an indexed vertex list of the hull of the device gamut. The hull is computed differently depending on whether you are working with additive devices, such as monitors and projectors, or subtractive devices. The indexed vertex list is stored in CIEJab. The structure of the indexed vertex list is optimized for hardware acceleration by DirectX.

This approach has many well-known solutions. If you search for "convex hull DirectX" on the Web, you get more than 100 hits. For example, there is a reference from 1983 on this specific topic (Computer Graphics Theory and Application, "Ship hulls, b-spline surfaces, and cadcam," pp. 34-49) with references dating from 1970 to 1982 on the topic.

The collection of points can be determined from externally available information, as follows:

- Points for the reference shell for monitors are generated using a sampling of the color cube in device colorant space.
- Points for the reference shell for printers and capture devices are obtained from the sample data used to initialize the model.
- Points for the reference shell for scRGB and sRGB are generated using a sampling of the color cube for sRGB.
- Points for the plausible shell for capture devices are generated using a sampling of the color cube in device colorant space.
- Points for the reference shell for projectors are generated using a sampling of a polyhedron in the color cube in device colorant space.
- Points for the possible shell for wide dynamic range color spaces are generated using a sampling of the color cube in the space itself.

You can create a vertex list that efficiently describes the color device gamut, given a device profile and system support services.

For output devices, the gamut boundary describes the range of colors that can be displayed by the device. A gamut boundary is generated from the same data used to model the behavior of the device. Conceptually, you output a sampling of the range of colors the device can produce, measure the colors, convert the measurements into appearance space, and then use the results to create the gamut boundary.

Input devices are trickier. Each pixel in an input image must have some value. Each pixel must be able to represent any color found in the real world in some way. In this sense, no colors are "out of gamut" for an input device, because they can always be represented.

All digital image formats have some fixed dynamic range. Because of this limitation, there is always some distinct stimuli that map to the same digital value. So, you cannot establish a one-to-one mapping between real-world colors and digital camera values. Instead, the gamut boundary is formed by estimating a range of real-world colors that can produce the digital responses of the camera. You use that estimated range as the gamut for the input device.

Primaries are included to provide for business graphics intent-type gamut mapping.

In true object-oriented style, you abstract the underlying representation of the gamut boundary. This allows you the flexibility to change the representation in the future. To understand the gamut boundary descriptor (GBD) used in the new CTE, you must first understand how gamut mapping algorithms (GMAs) work. Traditionally, GMAs have been designed to meet the needs of the graphic art community; that is, to reproduce images that have already been properly rendered for the device on which the input image was created. The goal of graphic arts GMAs is to make the best possible

reproduction of the input image on the output device. The new CTE GBD is designed to solve four key problems.

Because the input image is rendered for the input device, all colors fit within the range between the medium's white point and black point. Suppose that the image is a photograph of a scene in which there is a diffuse white, such as a person in a white tee shirt, and a specular highlight, such as light reflecting off a window or chrome bumper. The scene will be rendered to the input medium so that the specular highlight is mapped to the medium's white point, and the diffuse white is mapped to some neutral color darker than the medium's white point. The choice of how to map colors from the scene to the input medium is both a scene-dependent decision and an aesthetic decision. If the specular highlight was missing from the original scene, then the diffuse white would probably be mapped to the medium's white point. In a scene with a lot of highlight detail, more range would be left between the specular white and the diffuse white. In a scene where the highlight is not significant, very little range might be left.

For pre-rendered images, gamut mapping is relatively straightforward. Basically, the original medium's white point is mapped to the reproduction medium's white point, the source black point is mapped to the destination black point, and most of the mapping is complete. The different GMAs in existence provide variations for mapping other points on the source medium's tone scale and different ways to handle out-of-gamut chroma values. But the mapping of white to white and black to black is consistent throughout these variations. This implementation requires that white be above a  $J^*$  of 50 and black below a  $J^*$  of 50.

Not all color encodings limit color ranges for input images. The IEC standard color encoding scRGB (IEC 61966-2-2) provides 16 bits for each of the three color channels red, green, and blue (RGB). In that encoding, reference black is not encoded as the RGB triple (0, 0, 0), but as (4096, 4096, 4096). Reference white is encoded as (12288, 12288, 12288). The scRGB encoding can be used to represent specular highlights and shadow detail. It includes RGB triples that are not physically possible because they require negative amounts of light, and encodings that are outside the CIE spectral locus. Clearly, no device can possibly produce all the colors in the scRGB gamut. In fact, no device can produce all the colors that a human being can see. So, devices cannot fill the scRGB gamut, and it would be useful to be able to represent the part of the gamut that they do fill. Each device has a range of values in scRGB space that it can produce. These are the "expected" colors for the device; it would be surprising for the device to produce colors outside this gamut. There is a defined transformation from scRGB space to appearance space, so each device also has a range of appearance values that it is expected to reproduce.

In both scRGB and input from capture devices characterized with a fixed target, it is possible to get a value outside the range of expected values. If someone calibrates a camera to a test target; and then captures a scene with specular highlights, there might be pixels that are brighter than the target's white point. The same thing can happen if a naturally occurring red is more chromatic than the target red. If someone takes an scRGB image from a device, and then manually edits the colors in the image, it is possible to create pixels that fall outside the expected range of the device gamut, even though they are within the full scRGB gamut.

A second problem may not, at first, seem to be related to this. It arises when you use a color target to characterize an input device, such as a camera or scanner. Reflective targets are typically produced on paper and contain a number of colored patches. Manufacturers provide data files with color measurements taken under a fixed viewing condition for each color patch. Color profiling tools create a mapping between these measured values and the values returned by the color sensors in the devices. The problem is that often these color targets do not cover the full range of device values. For example, the scanner or camera might return a value of (253, 253, 253) for the reference white point, and a reference red patch might have an RGB value of (254, 12, 4). These represent the range of expected values for the input device, based on the target values. If you characterize the input device based on the responses to the target, then you only expect colors within this narrow range. This range is not only smaller than the range of colors humans can see, it is smaller than the range of colors the device can produce.

In both cases, it is difficult to estimate the gamut of the input device or image, despite the existence of a reference gamut or measurements. In the first problem, the plausible gamut of the input device is less than the full gamut of scRGB. In the second problem, the reference gamut of the target is less than the full possible gamut of the input device.

The third problem involves tone mapping. Many models of gamut boundaries that can adequately represent pre-rendered images used in the graphic arts have been proposed, for example, the Braun and Fairchild Mountain range GBD (Braun[97]), and Morovic's Segment Maxima boundary descriptor (Morovic[98]). But these models only provide information about the extremes of the device's gamut; they are missing information about other points in the tonal mapping. Without such information, GMAs can only make rough estimates of optimal tone mapping. Worse, these models provide no help for the extended dynamic range in scRGB and in digital camera images.

How is this problem resolved in the photographic and videographic industries? The camera captures an image. Experts might debate how much rendering occurs in the capture device; but they agree that it is not a significant amount. Both technologies do not map a diffuse white in a captured scene to the medium's white point. Similarly, they

do not map the black point from the scene to the medium's black point. The behavior of photographic film is described in density space using a characteristic curve, often called a Hurter and Driffield, or H&D curve. The curve shows the density of the original scene and the resulting density on the film. Figure 18 shows a typical H&D curve. The x-axis represents increasing log exposure. The y-axis represents the density on the slide. Five reference points are marked on the curve: black without detail, which represents the minimum density on the negative; black with detail; reference mid-gray card; white with detail; and white without detail. Note that there is space between black without detail (which represents device black) and black with detail (shadow black). Similarly, there is space between white with detail (diffuse white) and white without detail (which represents device white).

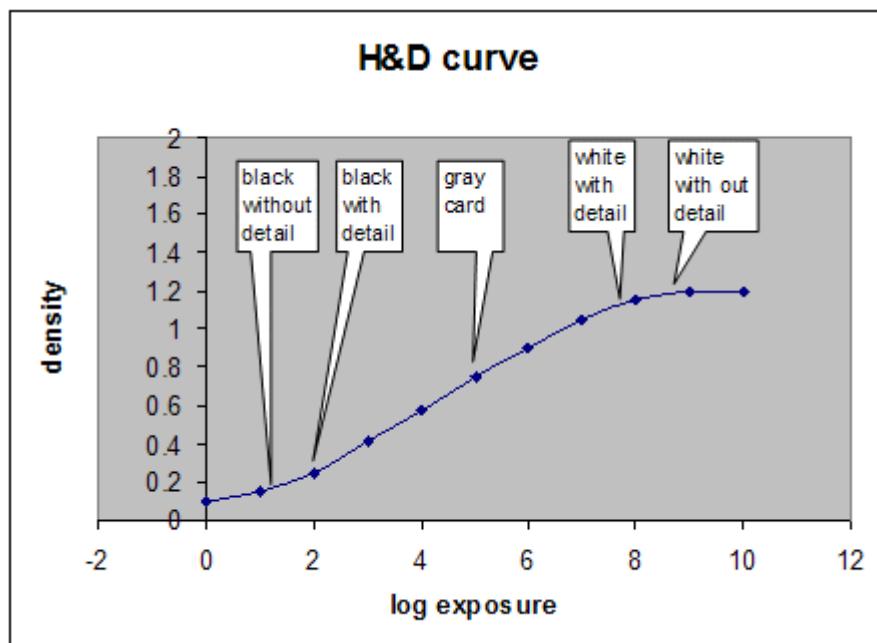
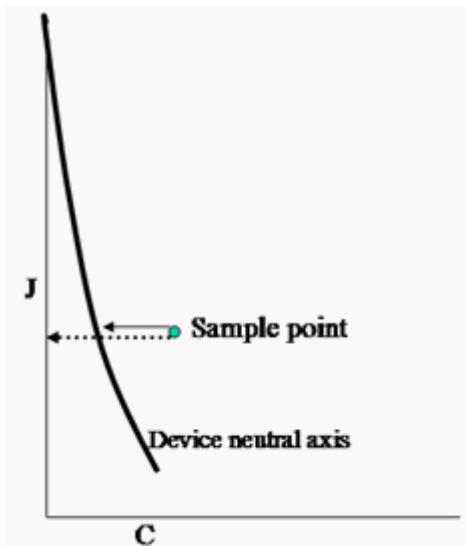


Figure 18 : H&D curve for slide film

The video industry provides "headroom" and "footroom" in images. In the ITU 709 specification, luminance (called Y) is encoded in 8 bits, with a range of 0 to 255. However, reference black is encoded at 15 and reference white is encoded as 235. This leaves the encoding range between 236 and 255 to represent specular highlights.

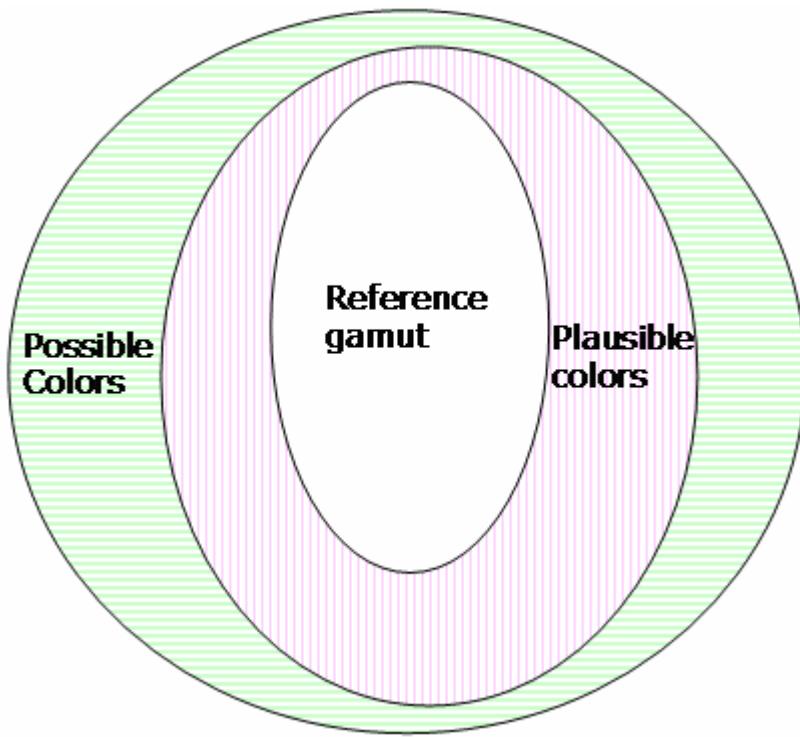
The video industry presents an essentially closed loop system. While there are many different equipment vendors, video systems are based upon reference devices. There is a standard encoding for video images. There is no need to communicate a gamut boundary with video images, because all images are encoded for reproduction on the same reference device. Film is also closed loop because there is no need to convey intermediate data between different components. You want a solution that enables images from devices with varying gamuts and representing both pre-rendered and unrendered scenes to be reproduced on output with varying gamuts.

A fourth problem that the new CTE must resolve is that the visually gray colors produced by a device, for example, when red=green=blue on a monitor, frequently do not fall upon the neutral axis of the CAM (when the chroma = 0.0). This causes great difficulties for GMAs. To make GMAs work well, you have to adjust the description of the device's gamut and of the input points so that the device's neutral axis falls on the appearance space's neutral axis. You have to adjust points off the neutral axis by a similar amount. Otherwise, you cannot make smooth gradations through the image. On the way out of the GMA, you undo this mapping, relative to the output device's neutral axis. This is referred to as a "chiropractic" straightening of the axis. Like a chiropractor, you not only straighten the skeleton (neutral axis), but you adjust the rest of the body to move along with the skeleton. Like a chiropractor, you do not adjust the skeleton by the same amount through the entire space. Instead, you adjust different sections differently.



**Figure 19:** Curvature of the device neutral axis relative to the CIECAM neutral axis

What the new CTE requires is a model of a gamut boundary that can be used to represent both rendered and unrendered source images, provide information about the appearance of device neutrals, and provide information for tone mapping images with a wide luminance range.



**Figure 20 : Three gamut shells**

The gamut boundary is composed of three shells that define three regions.

In the new CTE, the outer shell of the gamut is formed with a convex hull made from sample points in the device gamut. A hull is formed by taking a set of sample points and surrounding them by a surface. A convex hull has the additional property of being convex everywhere. Therefore, this is not the smallest possible hull that can be fit to the data. But experimentation has shown that fitting the sample points too tightly causes unappealing artifacts in images, such as a lack of smooth shading. The convex hull seems to solve these problems.

In the algorithm, color appearance values are obtained for a set of points sampled from the device. For monitors and printers, the color appearance values are obtained by outputting samples, and then measuring them. You can also create a device model, and then run synthetic data through the device model to predict measured values. The measured values are then converted from colorimetric space (XYZ) to appearance space (Jab), and the hull is wrapped around the points.

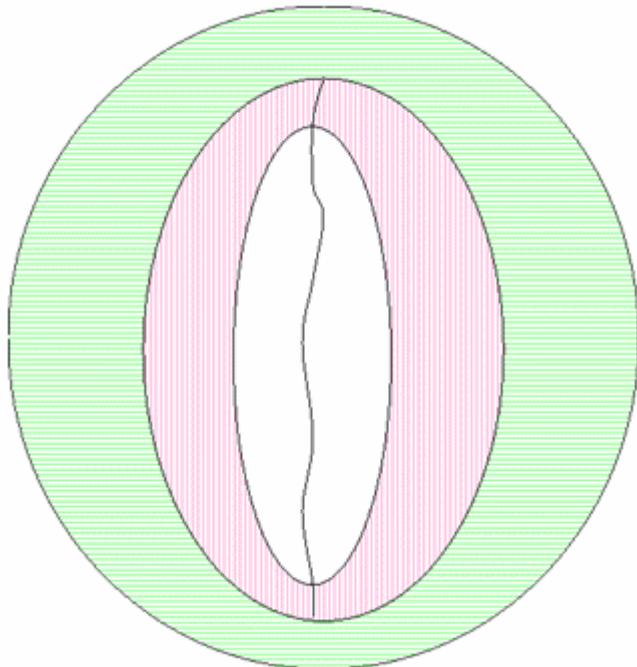
The key point to this algorithm is that the adopted white point used in the conversion from colorimetric to appearance space does not have to be the medium's white point. Instead, you can select a point farther inside the gamut and on (or near) the neutral axis. That point will then have a J-value of 100. Samples with a measured Y-value higher than the adopted white point will end up with a J-value greater than 100.

If you place the scene's diffuse white point as the adopted white point for the color space conversion, then specular highlights in the scene will be easily detected as having

a J-value greater than 100.

Because the CIECAM02 color model is based on the human visual system, after an adopted white is selected, the luminance level of the black point ( $J = 0$ ) is automatically determined by the model. If the input image has a wide dynamic range, it is possible that there might be values that map to J-values less than zero.

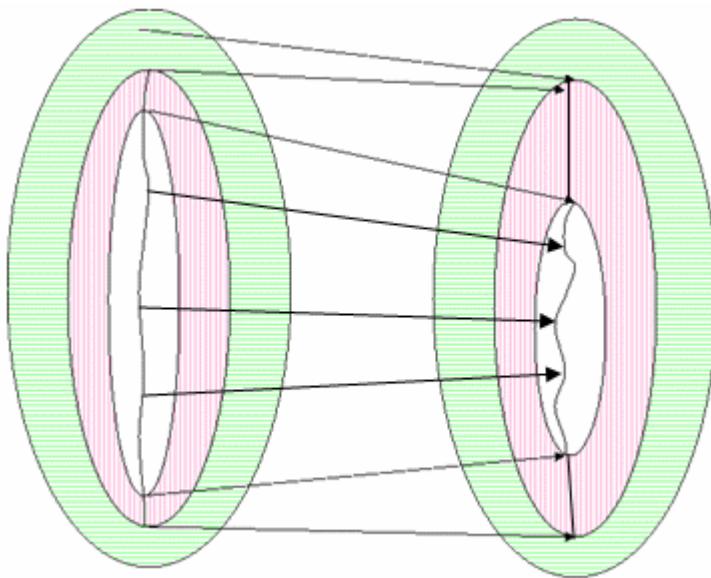
The following Figure 21 shows the device neutrals running through the center of the plausible and reference gamuts.



**Figure 21 :** Device neutral axis added to gamut boundary

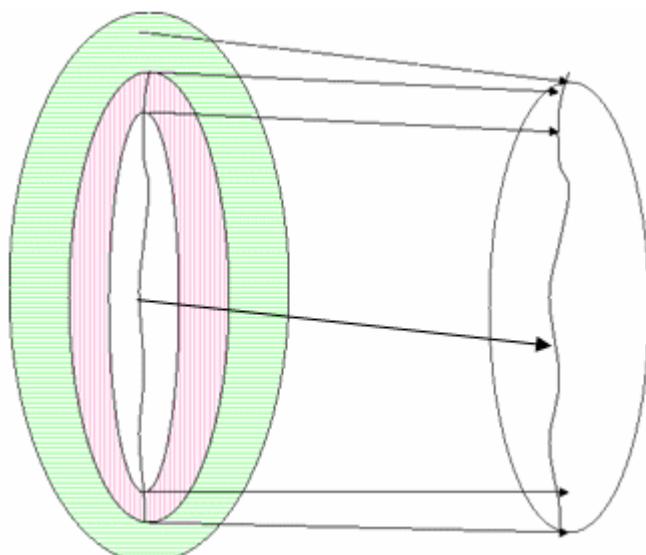
All gamut mappings involve either clipping an input range to an output gamut, or compressing the input gamut to fit within the output gamut. More complex algorithms are formed by compressing and clipping in different directions, or by dividing the gamut into different regions, and then performing clipping or compression in the different regions.

The new CTE extends this concept to support the regions of a possible gamut, a plausible gamut, and a reference gamut, and enables GMAs to map them in different ways. In addition, the GMAs have information about the device neutral axis. The following discussion addresses how to handle situations where the plausible gamuts and reference gamuts have collapsed on each other.



**Figure 22 : GMA with two un-collapsed gamut descriptors**

You might see this example if you map from an input device, such as a camera or scanner that is characterized with a reflective target, to scRGB space. Here the plausible colors that are lighter than reference white are specular highlights. In practice, characterizing a camera with a target might not generate the full range of values possible in the camera; however, specular highlights and very chromatic colors found in nature would. (Transmissive targets usually have a patch that is the minimum density possible on the medium. With such a target, specular highlights would fall within the target's range.) The reference black for a reflective target would be the beginning of the shadow black region. That is, there are likely to be colors in the shadows that are darker than the black on the target. If the image contains a lot of interesting content in that region, it may be worthwhile preserving that tonal variation.



**Figure 23 : GMA with collapsed destination gamut**

Figure 23 shows one possible gamut mapping algorithm when the destination gamut only provides the range from device white to black, and there are no possible colors outside this gamut. This is likely to happen for typical output devices, such as printers. The possible colors are mapped to the edge of the destination gamut. But it lacks a tone curve for the output device. The GMA must select some neutral point of lower luminance to use as a mapping destination for the reference white. A sophisticated algorithm can do this by histogramming the lightnesses in the source image and seeing how many fall in the range of expected but lighter than the reference white. The more lightnesses, the more space is required in the destination device between the mapped points for the specular highlights and reference white. A simpler algorithm might pick an arbitrary distance down the lightness scale from device white, such as 5 percent. A similar approach applies for the mapping of the maximum black and shadow black.

After you generate the destination tone curve, you can map in a method similar to that used in the preceding Figure 23. All points in the destination tone curve will fall within the device gamut, and all points in the mapping must fall within the device gamut.

If you reverse the left figures and right figures, and the directions of the arrows in Figure 23, you can describe the case where the source image has only a reference gamut, and the three gamuts of the output device have not collapsed onto each other. An example of this might be mapping from a monitor to scRGB. Again, the GMA must synthesize the control points for the five points on the tone curve for the source image. Some mappings might put all points of the tone curve within the scRGB device gamut, while other mappings might use more of the scRGB gamut by mapping diffuse white to the reference white and allowing specular white to map to a lighter value.

Finally, you have the case where both devices have only the reference gamut, which is how most gamut mapping algorithms work. So you can resolve this by just falling back to current algorithms. Alternatively, if you have a reasonable way to determine the five reference points for the source and destination devices, then you can arrange to map the reference points.

Device gamuts contain more than the five reference points on the neutral axis. These just represent the boundaries between potential regions in the image. Between each of the reference points, you can use any of the existing gamut mapping techniques. So you might clip the range of unexpected colors, and compress all colors between the expected white and black, or you might clip all colors outside the reference range and compress within that range. There are many possibilities, which can be implemented in different GMAs. Further, the GMAs can compress and clip in different ways. All of those combinations are covered within this invention.

So far in this discussion, the gamut has been treated as if it were solely a function of the device on which the image was created, captured, or displayed. However, there is no reason why all images for a device must have the same gamut. The GMAs depend on the data in the GBD. If the descriptor is changed between images, there is no way for the GMAs to know. In particular, if images have no specular highlights, GMAs perform better if the gamut descriptor does not show that there are colors lighter than diffuse white.

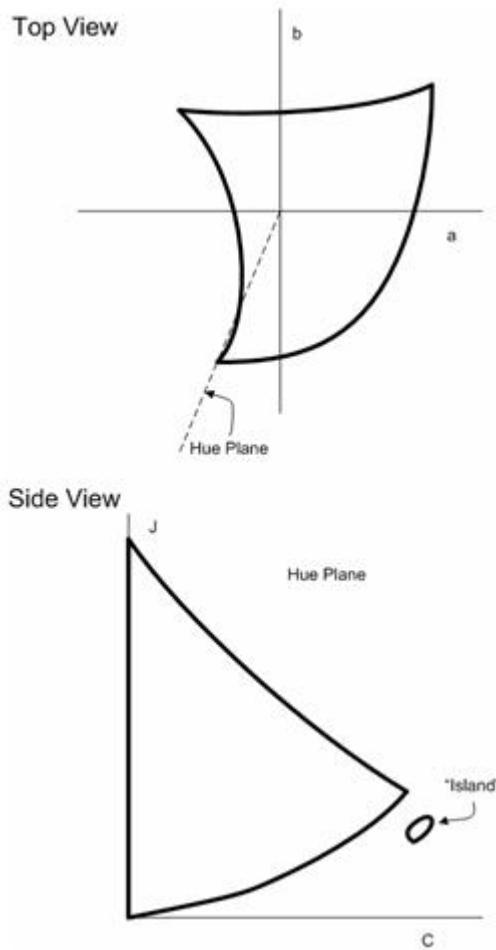
In the new CTE architecture, it is possible to use more than one GMA. Using multiple GMAs is inherently ill-defined. For example, if a capture device associates a GMA with its "look and feel," it tends to do so with a "targeted" destination gamut. The same is true for output devices and "targeted" source gamuts. The sRGB gamut is one commonly targeted implied gamut. Therefore, it is strongly recommended that you use a single GMA, if predictability is a priority. A single GMA workflow should be the default for all workflows, especially consumer and prosumer workflows. While gamut mapping for preferred reproduction should be done one time, there are instances where multiple mapping processes are included. First, for proofing, you do a preferred mapping to the gamut of the final target device, and then a colorimetric rendering to the gamut of the proofing device. Second, some types of mapping are used to alter the characteristics of the image, but aren't included to map to a device gamut, for example, adjusting the tone curve or the chromaticity. If multiple GMAs are used, the transform interface takes an array of bound gamut maps, that is, gamut maps that have been initialized with a pair of gamut boundary descriptions. When there is more than one gamut map, the input gamut boundary for a succeeding gamut map must be the same as the output gamut boundary of its predecessor.

The device gamut boundary function takes the device model engine and analytical parameters and derives a color device gamut boundary described as an ordered vertex list of the convex hull of the device gamut. The ordered vertex list is stored in CIEJab. The structure of the ordered vertex list is optimized for hardware acceleration by DirectX. This approach has many well known solutions (search for "convex hull DirectX" on the web and you get well over 100 hits). There is also a reference from 1983 on this topic (Computer Graphics Theory and Application, "Shiphulls, b-spline surfaces and cadcam" pp. 34-49), with references dating from 1970 to 1982 on the topic.

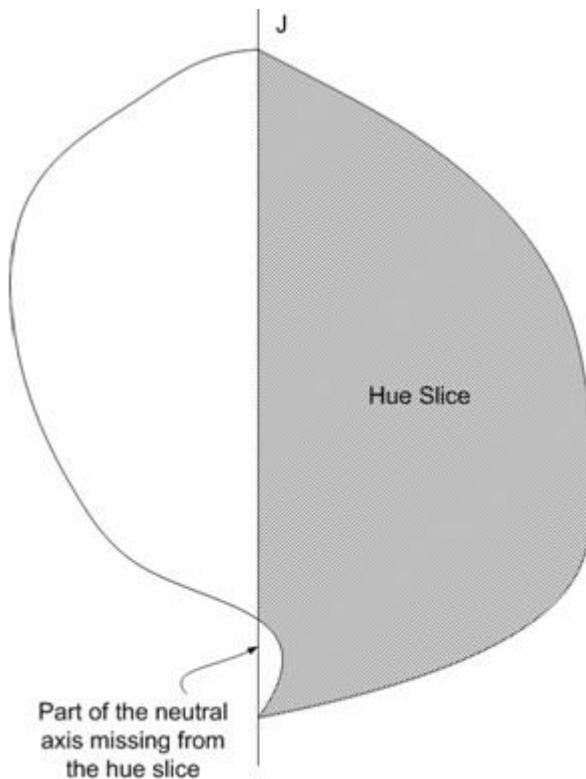
Two different techniques can be used to compute the triangles in the gamut shell. For other devices other than additive RGB devices, you compute a convex hull. You can consider investigating non-convex hull support for other devices if you have direct access to such devices to validate the robustness, performance, and fidelity of the algorithms. This is a well-known process that does not require further description. The technique used for additive RGB devices is described as follows.

Different GBDs have advantages and drawbacks. The convex hull representation guarantees nice geometric properties, such as convex hue slices that provide a unique intersection point with a ray emanating from a point on the neutral axis. The downside of the convex hull representation is also convexity. It is known that many devices, specifically display devices, have gamuts that are far from being convex. If the actual gamut deviates significantly from the convexity assumption, the convex hull representation would be inaccurate, possibly to the extent that it doesn't represent reality.

After you adopt a GBD that gives a reasonably accurate representation of the actual gamut, other problems arise, some due to the very concept of hue slice. There are at least two pathological situations. In the following Figure 24, a CRT gamut gives rise to hue slices with "islands." In Figure 25, a printer gamut gives rise to a hue slice with part of the neutral axis missing. The pathological hue slices are not caused by particularly pathological gamut boundaries in these cases. They are caused by the very concept of hue slice, because (a) it is taken along constant hue, and (b) it only takes one-half of the plane that corresponds to the hue angle.



**Figure 24 :** A typical CRT monitor has a gamut that shows peculiar "curving in" in the blue hues. If hue slices are taken within this hue range, isolated islands may appear in the hue slices.



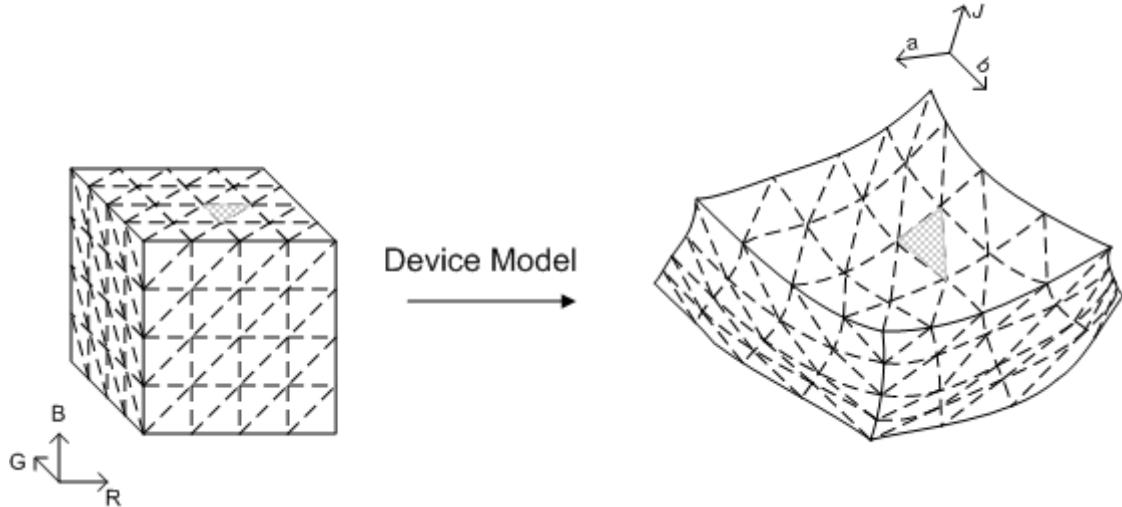
**Figure 25 :** A printer may have a gamut that has "gap" in its neutral axis. When a hue slice is taken (which is only one-half the plane), there is a "dent" on the part of the boundary that is the neutral axis. This can be hard to resolve algorithmically.

To resolve these pathologies, a new framework is proposed that abandons the concept of hue slice that used as the starting point. Instead, the framework uses the set of "boundary line elements," or lines that lie on the gamut boundary. They do not necessarily provide a coherent geometric visualization like hue slices, but they support all the common gamut operations. Besides solving the problems mentioned previously, this approach also suggests that the construction of hue slices, even when it is possible, is computationally wasteful.

## Triangulation of the Gamut Boundary

The starting point is a GBD consisting of a triangulation of the gamut boundary. Known methods of constructing GBDs usually provide that triangulation. For concreteness, one method of constructing GBDs for additive devices its device space is described here. These devices include monitors (both CRT-based and LCD-based) and projectors. The simple geometry of the cube enables you to introduce a regular lattice on the cube. The boundary faces of the cube can be triangulated in a number of fashions, such as the one shown in Figure 26. The architecture provides either a device model for the device so that colorimetric values of the lattice points can be obtained algorithmically, or measurements have been made directly for those points. The architecture also provides CIECAM02, so that you can assume the starting data has already been mapped into CIECAM02 Jab space. Then each lattice point on the boundary faces of the RGB cube has

a corresponding point in Jab space. The connections of points that form the set of triangles in RGB space also induces a set of triangles in Jab space. This set of triangles forms a reasonable triangulation of the gamut boundary if (a) the lattice on the RGB cube is fine enough, and (b) the transformation from device space to the uniform color space is topologically well-behaved; that is, it maps boundary to boundary, and it doesn't turn the gamut inside out so that interior points become boundary points.



**Figure 26 :** A simple method to triangulate the gamut boundary of a device with RGB as its device space

## Boundary Line Elements

Central to this framework is the concept of boundary line elements; a set of line segments that (a) lie on the gamut boundary, and (b) lie on a plane. In this case, the plane is a hue plane. Each line segment is the result of intersecting the plane with a gamut boundary triangle. Although many researchers have used the construction of intersecting a plane with boundary triangles, they generally analyze the relationship among these line segments, and attempt to construct a coherent geometric object out of the line segments. Different algorithms have been devised to follow these line segments one after the other until a whole hue slice is obtained, and many attempts have been made to speed up the process of searching.

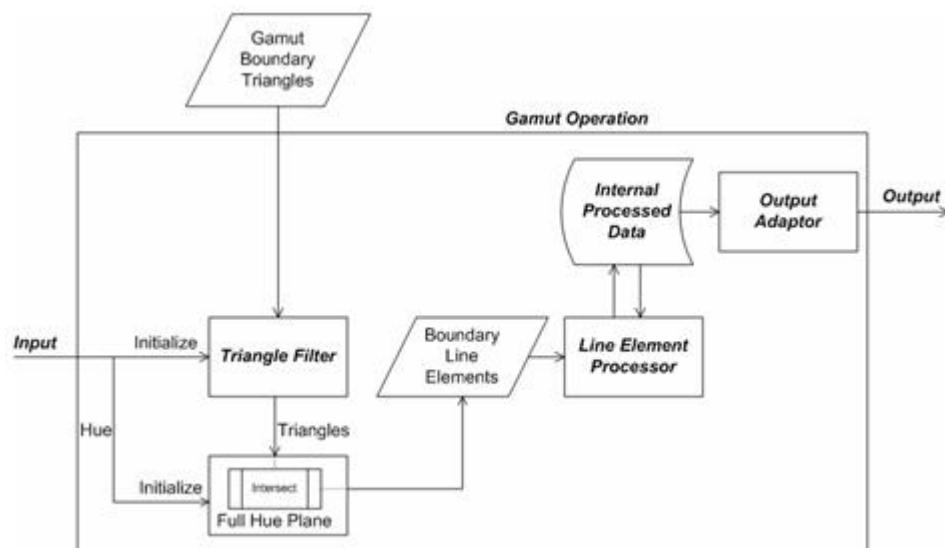
This approach is different. You intersect the plane with the triangles to obtain the line segments. You then consider these line segments as the *basic* conceptual objects. It is necessary to analyze the relationship among the line segments; you don't have to know how they are interconnected with each other. This point of view solves the problem of hue slice with islands. The known approaches that attempt to construct hue slice assume that, if one starts with one line segment and follows it to the next line segment, and so on; it eventually leads back to the starting point, at which point a whole hue slice would be constructed. Unfortunately, this approach would miss the island (and in the worst

scenario, the continent). By not insisting on obtaining a coherent geometric picture; that is, hue slice, you can handle the island problem effortlessly. Another important difference in this approach is that, to speed up the construction of line segments, it uses a "triangle filter." The triangle filter throws out certain triangles that will definitely not produce line segments that would be useful in the current gamut operation. Because intersecting a triangle with the plane is expensive computationally, this improves speed. A side effect is that, you cannot construct hue slice because some line segments would be missing due to the triangle filtering.

## Gamut Operation: CheckGamut

The following example will explain how the framework works, and how CheckGamut is carried out, that is, the operation of checking if a color is in-gamut.

The general framework is illustrated in the following Figure 27. There are various components. The components labeled in italics are components that may be different in implementation depending on the gamut operation in question. The other components are invariant across all gamut operations. To begin, the *Input* is a set of color attributes. In the case of CheckGamut, it is the query color. In Figure 27 and the following discussion, it is assumed that the hue angle is either among the input color attributes, or can be obtained from them. This is clearly the case if the input is the whole color point, either in Jab or JCh, from which you can then compute the hue angle. Note that the hue angle is only needed because hue planes are being used. Depending on the gamut operation in question, it might not be necessary to use the hue plane. For example, in the construction of the routine CheckGamut, you might want to use planes of constant J. This is a generality that will not be used or discussed further; but it might be helpful to remember this flexibility of the methodology to support other gamut operations when hue plane might not be the best choice.



**Figure 27** : The framework to support gamut operations

The hue angle, which is obtained directly from the inputs or computed from the inputs, is used to initialize the hue plane labeled **Full Hue Plane** in the figure. "Full" is emphasized because this is the full plane, not just the half-plane containing the hue. The full plane contains both the input hue angle and the angle 180 degrees opposite to it. The key functionality of the hue plane is the **Intersect** function, which is explained in the following subsection, **Full Hue Plane: Intersect**. Assume that the GBD has already been constructed, and the set of **Gamut Boundary Triangles** is available. Intersect the triangles that have survived the ***Triangle Filter*** with the hue plane using **Intersect**. The ***Triangle Filter*** component is labeled in italics, which means that the component varies in implementation for different gamut operations. The ***Triangle Filter*** for **CheckGamut** is explained in the section, **Gamut Operation: CheckGamut (continued)**. The result of intersecting a triangle with the hue plane is either empty or a **Boundary Line Element**, that is, a pair of distinct points. If the result is non-empty, it is passed into the ***Line Element Processor***, which again does different things depending on the gamut operation. The ***Line Element Processor*** updates the internal data structure, ***Internal Processed Data***, whose content or layout also depends on the gamut operation. Generally, the ***Internal Processed Data*** contains the "answer" to the problem, which is continually updated with each new Boundary Line Element found. When all the Boundary Line Elements have been processed, the answer has been found. It remains to access it through the ***Output Adaptor***. Because the ***Internal Processed Data*** is gamut operation specific, the ***Output Adaptor*** is also gamut operation specific.

## Full Hue Plane: Intersect

The **Intersect** function calculates the intersection of the hue plane and a triangle. As simple as it sounds, this function is important for two reasons.

First, intersecting each edge of the triangle with the plane might yield three intersection points, a geometrically impossible situation. The reason why this might happen in the computation is that, when calculations are done in floating point, for example, IEEE format, there are uncertainties, or "numeric noise," in each step that affects the conclusion of whether an edge intersects the plane. When the plane intersects the edges in a near-miss situation, the intersection points are close to each other, and the determination whether an intersection point lies within the edge is random. Although noise in the numeric values of the points is small, the qualitative conclusion that there are more than two intersection points is geometrically impossible and difficult to handle correctly in the algorithm.

Second, this function is in the critical loop for each edge of each filtered triangle, so it is important that you optimize its efficiency as much as possible.

To address the first issue of numeric noise, perform the calculations in integers. To address the second issue of optimizing its efficiency, cache the most used attribute of each vertex, or the "dot product" associated with each vertex. Passing into integers is a typical way to guarantee geometric consistency. The basic idea is that if you have to quantize, do it at the beginning. Then subsequent calculations can be performed in integers, and if the integers are wide enough so that there is no danger of overflow, the calculations can be done with infinite precision. The following quantization function useful for this purpose.

`ScaleAndTruncate(x) = Integer part of x*10000`

The scaling factor 10000 means that the input floating-point number has four decimal places, which is precise enough for this application. Depending on the range of values of the color appearance space, you want to choose an integer type with bits wide enough to hold the intermediate calculations. In most color appearance spaces, the range of each coordinate is within the range -1,000 to 1,000. The quantized coordinate has a maximum possible absolute value of  $1,000 \times 10,000 = 10,000,000$ . As you will see, the intermediate quantity is a dot product, which is a sum of two products of coordinates, so it has a maximum possible absolute value of  $2 \times (10,000,000)_2 = 2?10_{14}$ . The number of bits required is  $\log_2(2?10_{14}) = 47.51$ . A convenient choice for the integer type is, therefore, 64-bit integers.

To guarantee that intersecting a plane with a triangle always gives either empty set or a set of two points, you must consider the triangle as a whole, not as individual edges of the triangle separately. To understand the geometric situation, consider the "signed distances" of the vertices of the triangle from the hue plane. Do not calculate these signed distances directly; instead calculate the dot products of the position vectors of the vertices with the quantized normal vector to the plane. More specifically, during the initialization of the hue plane, the quantized normal vector is calculated as follows.

`NormalVector = (ScaleAndTruncate(-sin(hue)), ScaleAndTruncate(cos(hue)))`

Note that this vector is a two-dimensional vector. You can use a two-dimensional vector because the hue plane is vertical, so the third component of the normal vector is always zero. Moreover, a lookup table of dot products is initialized to have an entry for each vertex from the Gamut Boundary Triangles and the corresponding dot product set to an invalid value.

During one operation of intersecting the hue plane with a triangle, the dot product of each vertex of the triangle is looked up. If the value in the lookup table is the invalid value, then the dot product is computed using the following expression.

```
NormalVector.a*ScaleAndTruncate(vertex.a) +
NormalVector.b*ScaleAndTruncate(vertex.b)
```

Again, the J-component of the vertex is never used, because the normal vector is horizontal. This dot product is then saved in the lookup table so that it doesn't have to be computed again if the dot product of the vertex is queried later.

Caching allows for a quick determination of whether an edge intersects the plane, after the dot products are tabulated in the lookup table, which is built progressively as the vertices are processed.

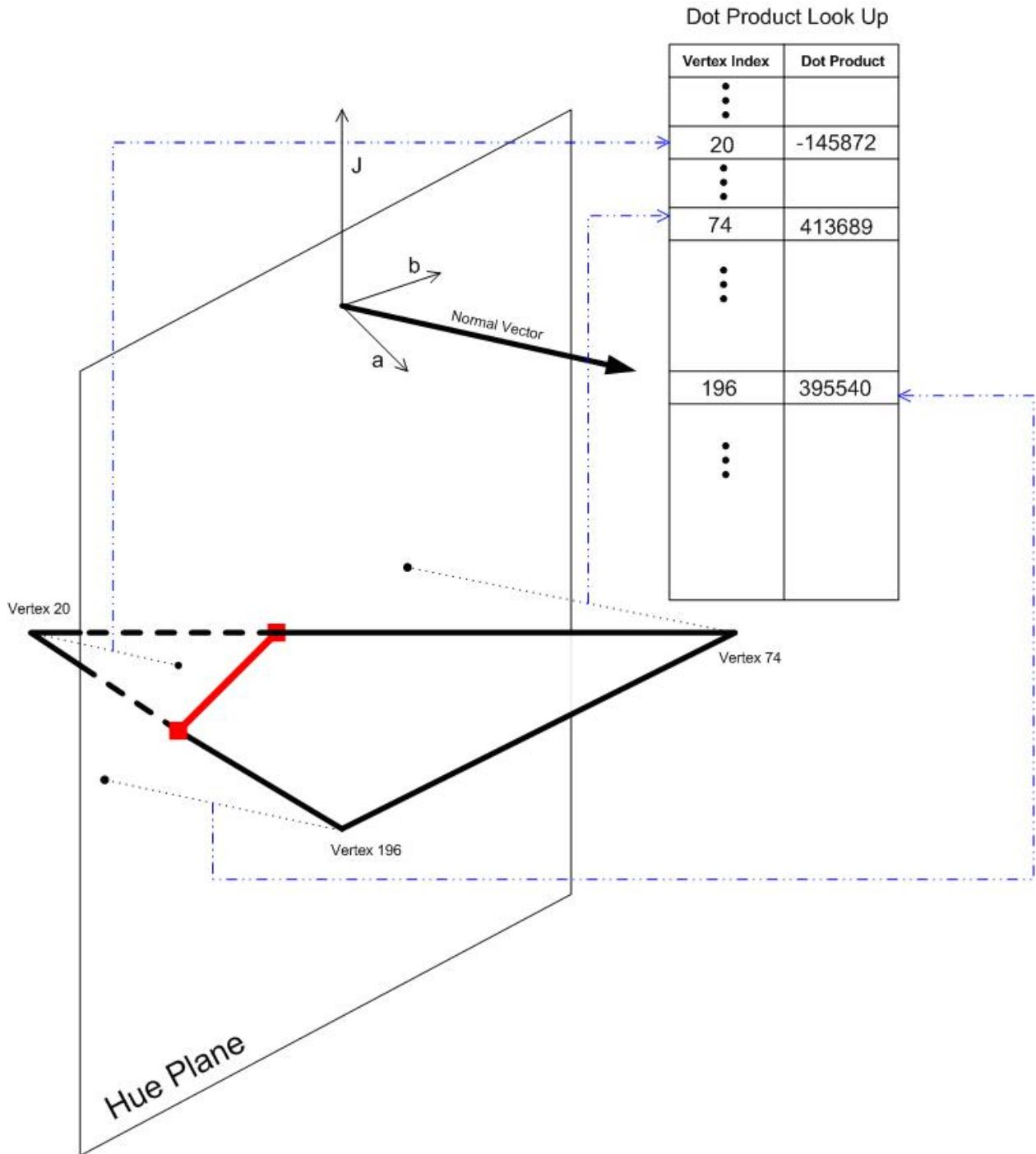


Figure 28 : Intersecting the hue plane with a triangle

For the triangle in Figure 28 to intersect the hue plane in a non-degenerate line segment, the dot products of the vertices must be in one of the following patterns, when sorted in ascending order.

0,0,+; -,0,0; -,0,+; --,+; -,+,+

An end point of the line segment arises when the plane is intersected by an edge with vertices that have different signs in the dot product. If the sign is zero, then the vertex lies right on the plane, and the intersection of the edge with the plane is the vertex itself. Note also that the cases 0,0,0; --,0; 0,+,+ are not reported. The first case (0,0,0) means the whole triangle lies on the plane. This is not reported because each edge of the triangle should belong to a neighboring triangle that does not also lie entirely on the plane. The edge will be reported when that triangle is considered. The other two cases (-,-,0 and 0,+,+) correspond to the geometric configuration that the triangle touches the plane in one vertex. These cases are not reported because they don't give rise to a non-degenerate line segment.

The preceding algorithm determines when to calculate an intersection between an edge of the triangle and the hue plane. After an edge is determined, the intersection is calculated using parametric equations. If one of the dot products is zero, the intersection is the vertex itself, so no calculation is necessary. Assuming that both dot products of the vertices of the edge are non-zero , vertex1 is the vertex with *negative* dot product dotProduct1; and vertex2 is the vertex with *positive* dot product dotProduct2. This order is important to ensure that the calculated intersection point does not depend on how the ordering of the vertices appears in the representation of the edge. The geometric concept of the edge is symmetrical with respect to its vertices. The computational aspect of using parametric equations of the edge introduces asymmetry (choice of starting vertex), which may give a slightly different intersection point due to numeric noise and conditioning of the linear equations to be solved. With this said, the intersection point, intersection, is given by the following.

$$t = \text{dotProduct1}/(\text{dotProduct1} - \text{dotProduct2})$$

$$\text{intersection.J} = \text{vertex1.J} + t * (\text{vertex2.J} - \text{vertex1.J})$$

$$\text{intersection.a} = \text{vertex1.a} + t * (\text{vertex2.a} - \text{vertex1.a})$$

$$\text{intersection.b} = \text{vertex1.b} + t * (\text{vertex2.b} - \text{vertex1.b})$$

## Gamut Operation: CheckGamut (continued)

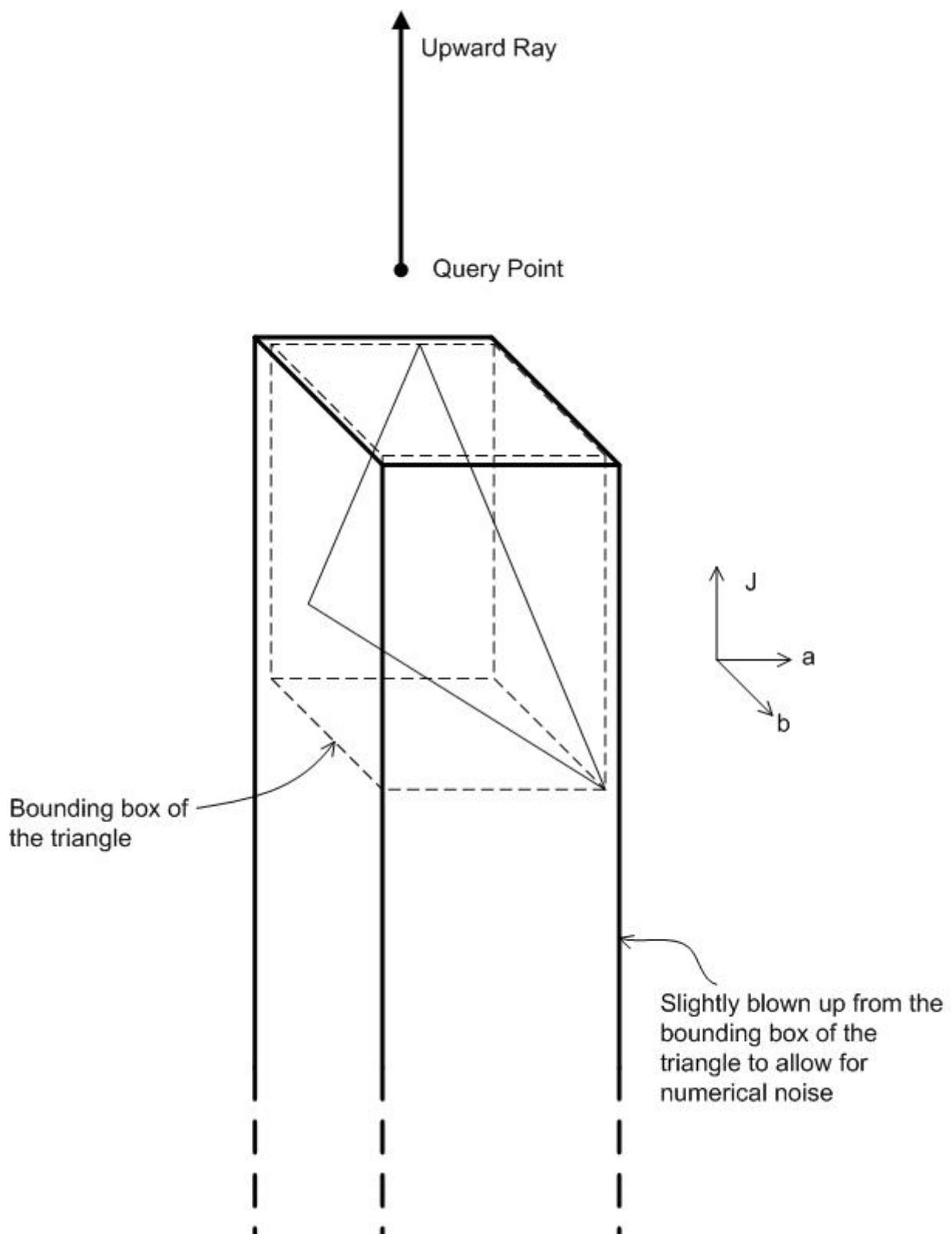
The basic geometric algorithm used for gamut checking is to count the number of ray crossings. For a given query point, consider a ray starting with the query point and

pointing upwards (J-direction). Count the number of times this ray crosses the gamut boundary. If this number is even, then the query point is out-of-gamut. If this number is odd, the point is inside. In principle, this algorithm can be implemented in 3-D, it is generally plagued by difficulties caused by degenerate situations, such as the ray lying (partly) on a boundary triangle, or lower dimensional degeneracy, such as the ray lying (partly) on an edge of a boundary triangle. Even in 2-D, you have to deal with these degenerate situations; but the problem is simpler and has been addressed in a satisfactory manner. See [O'Rourke].

For a given input point  $J_{ab}$ , determine its hue angle  $h$  as follows.

$$h = \text{atan}(b/a),$$

Initialize the hue plane, and then determine the Boundary Line Elements corresponding to this hue plane. Because the Boundary Line Elements are only relevant if they intersect the upward ray, set up a Triangle Filter to remove triangles that give line elements that definitely will not intersect the upward ray. In this case, consider the bounding box of the triangle. The upward ray will not intersect the triangle if the query point is outside the "shadow" cast by the bounding box if a light source was directly above. Inflate this slightly with a pre-fixed tolerance to allow for numeric noise so that you don't inadvertently throw away triangles that might give useful line elements. The result is the semi-infinite rectangular cylinder shown in Figure 29. Checking whether the query point is inside or outside this cylinder can be efficiently implemented using simple inequalities.



**Figure 29 : Triangle Filter for CheckGamut**

CheckGamut has three gamut operation specific components: *Internal Processed Data*, *Line Element Processor*, and *Output Adaptor*. The *Internal Processed Data* is a list of line elements that have been processed by *Line Element Processor*. In this case, the *Line Element Processor* simply adds a line element to the list. The internal data structure for *Internal Processed Data* can be either a linked list or an array that can grow in size.

The *Output Adaptor* is a module that accesses the list of line elements, determines if a line element crosses the upward ray (count 1) or not (count 0). Summing all these counts gives a total count. The *Output Adaptor* ultimately outputs an answer of "yes" (in-gamut) or "no" (out-of-gamut), depending on whether the total count is odd or even. The step where you determine if a line element crosses the upward ray deserves some attention because this is where the problem of degeneracy arises and also the problem of over-counting arises. Following [O'Rourke], for a line element to cross the ray, the right end point (the end point with larger chroma) must be strictly on the right side of the ray. This guarantees that, if an end point ever lies exactly on the ray, it is only counted one time. The same rule also solves the degenerate situation where the line element lies exactly on the ray. You do not increment the count for this line element.

Figure 30 shows the resulting line elements of a sample gamut with the query point in various positions.

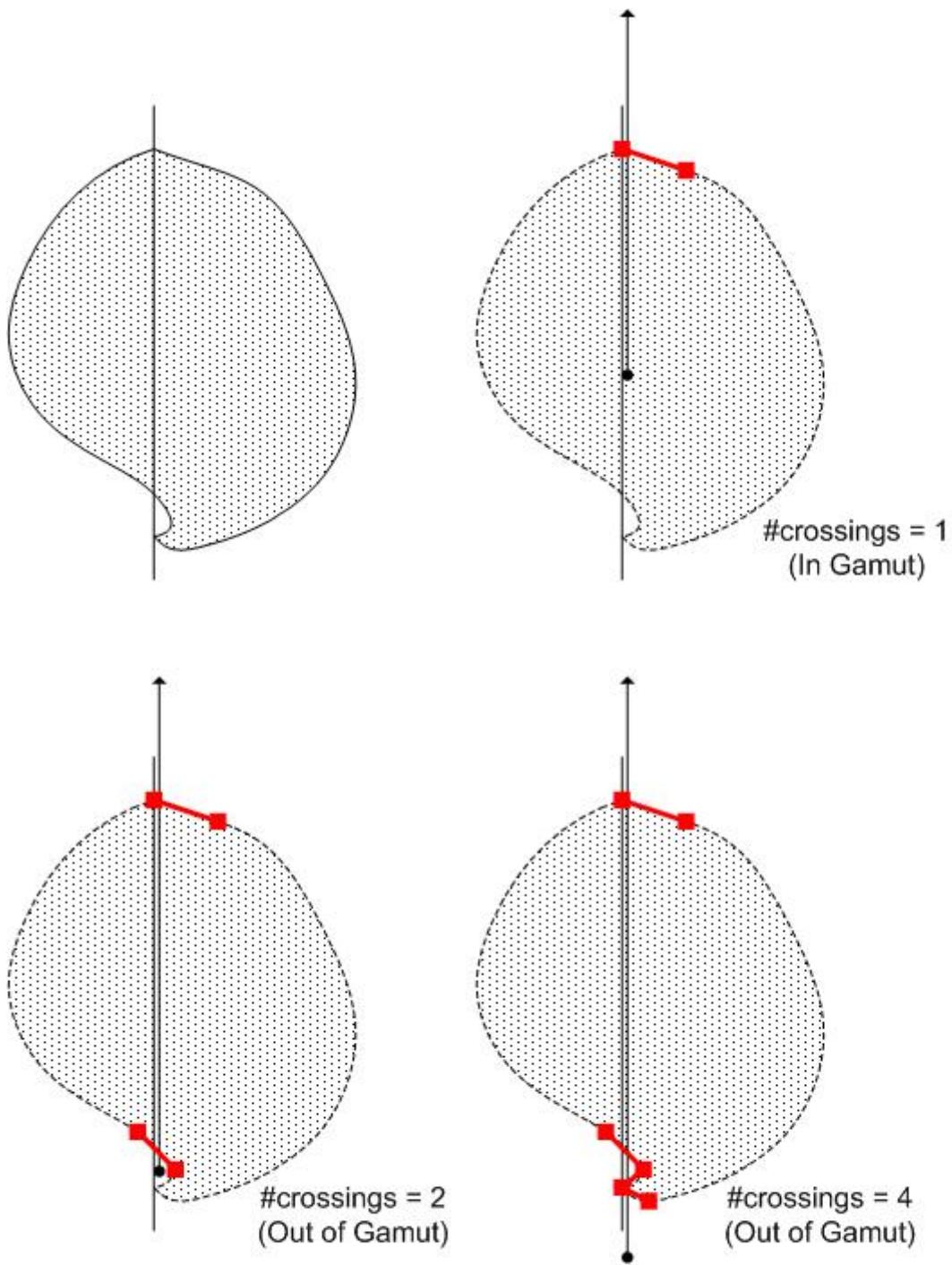


Figure 30 : How CheckGamut works

## Minimum Color Difference Gamut Mapping

The Minimum Color Difference Gamut Mapping, MinDEMap, has a simple specification: If a color is in-gamut, do nothing. If a color is out-of-gamut, project it to the "nearest" point on the gamut boundary. The keyword "nearest" is not well-defined until you specify which color difference equation to use. In practice, to make computation easier

and faster, the Euclidean distance of the chosen color appearance space, or a variant of it, is used as the color difference metric. The advantage of the Euclidean metric is that it is compatible with the dot product of the space, which makes it possible to use linear algebra. In detail, if a "dot product" is defined in the space, then a distance can be defined as the square root of the dot product of the difference vector with itself. A dot product can generally be defined by a positive definite  $3 \times 3$  matrix  $A$ .

$$u \cdot v = u^T A v$$

where the right hand side is the usual matrix multiplication. If  $A$  is the identity matrix, the standard dot product is recovered. In practice, if  $Jab$  is the color space, you do not want to mix up the components, so a diagonal matrix other than the identity matrix can be used. In addition, you might want to keep the scale on  $a$  and  $b$  unchanged so that the measure of hue is preserved. So a useful variation of the standard Euclidean dot product is the following.

$$w_J (J \text{ component of } u)(J \text{ component of } v) + (a \text{ component of } u)(a \text{ component of } v) + (b \text{ component of } u)(b \text{ component of } v)$$

where  $w_J$  is a positive number. One further variation is to let  $w_J$  vary with the input query point:

$$w_{Jq} = w_J(\text{queryPoint})$$

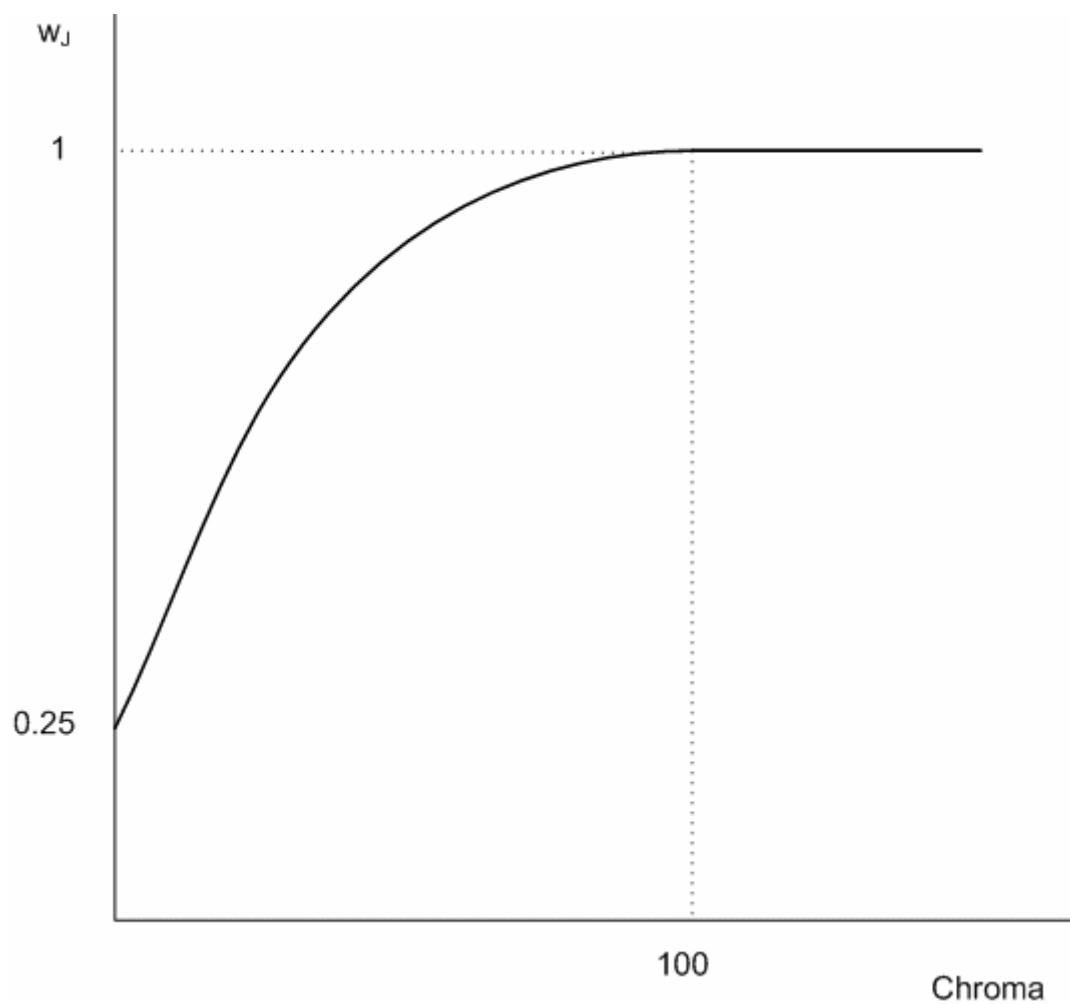
The end result is a measure of distance that is asymmetric with respect to the two points, and with different relative weights on lightness and chroma or hue as the input query point varies. This is in accordance with some observations about human color perception that color differences are not weighted equally in all dimensions. It has been found that people are less sensitive to differences in lightness than they are to differences in hue and chroma.

The following weight function is useful.

$$w_J = k_2 - k_1 (C - C_{\max})^n$$

where  $k_2 = 1$ ,  $k_1 = 0.75/(C_{\max})^n$ ,  $C_{\max} = 100$ ,  $n = 2$  and  $C$  is the smaller of chroma of the query point and  $C_{\max}$ .

so that a weight of 0.25 is put on the  $J$  term when chroma is zero, and a weight of 1 when chroma is 100. The trend of putting less weight on  $J$  when chroma is small, and more weight on  $J$  when chroma is large follows the recommended usage for CMC and CIEDE2000.



**Figure 31 :** The weight function on the J component of the metric

Use Jab space for the following example. It is computationally demanding to search through all the boundary triangles to determine the closest point in the Euclidean metric. The following is a simple approach to make this process as efficient as possible, without introducing additional assumptions that may speed up the process but also end up in only an approximate answer. First, it is necessary to understand the geometric procedure of projecting a point onto the given triangle. A description is given here.

An orthogonal projection onto the infinite plane containing the triangle is first performed. The shortest distance of the query point from the plane can be determined in two steps.

(a) Calculate the unit normal vector to the triangle.

(b) Calculate the dot product of the unit normal vector and a vector formed from the query point and a point on the triangle; that is, one of its vertices. Because the normal vector has unit length, the absolute value of this dot product is the distance of the query point from the plane.

The projected point might not be the answer, because it might lie outside the triangle. So, you must perform a check first. The calculation is equivalent to calculating the

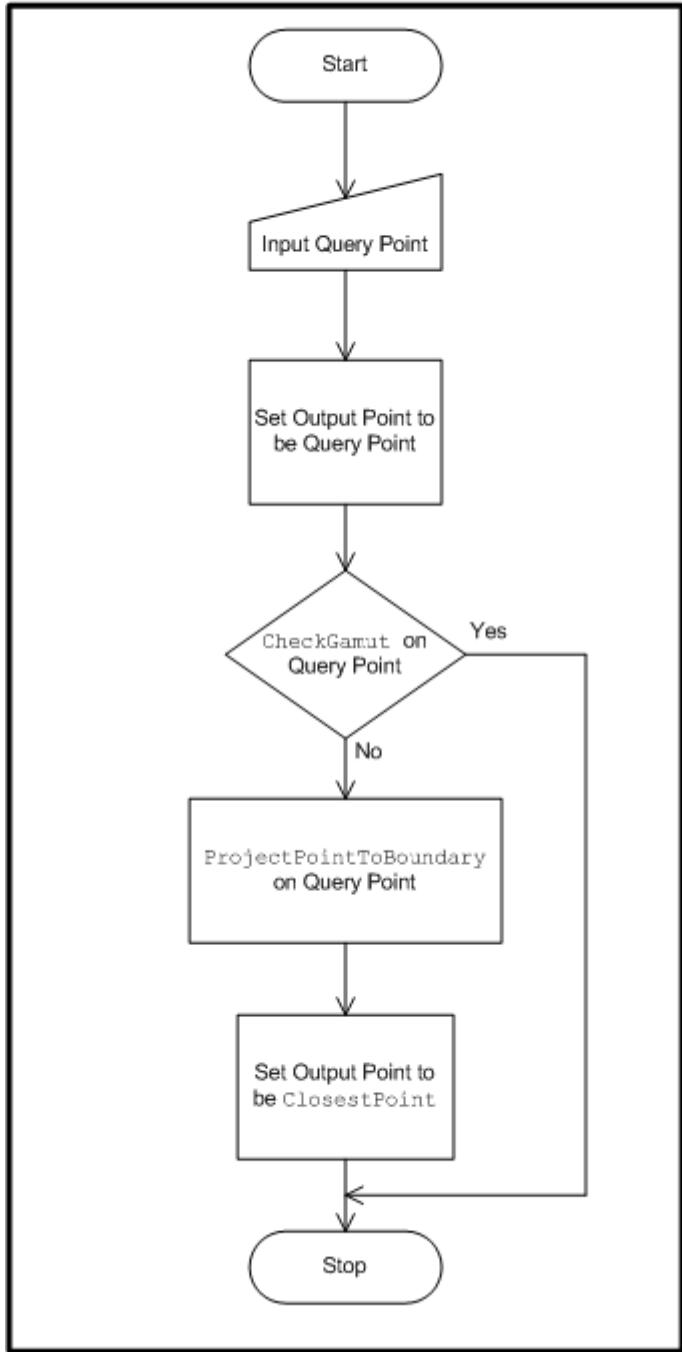
barycentric coordinates of the projected point relative to the triangle. If the projected point is determined to be inside the triangle, it is the answer. If not, the closest point is acquired on one of the edges of the triangle. Perform a search on each of the three edges. Determining the projection of the query point onto an edge is a process similar to projection onto the triangle, but one dimension less. An orthogonal projection is first calculated. If the projected point lies on the edge, it is the answer. If not, the closest point is acquired on one of the two end points. Perform a search on the two end points; that is, calculate the distance of query point from each one, and compare which one is smaller.

Careful examination reveals that there is a lot of repeated searching when you go through all the triangles because an edge is always shared by two triangles, and a vertex shared by at least three edges. In addition, you are not much interested in finding the closest point to one particular triangle; instead you are interested in finding the closest point to the whole gamut boundary. However, one particular triangle would be the one in which this is achieved. There are two strategies that you can use to speed up the search.

**Strategy I.** Each vertex will be processed, at most, once. Each edge will be processed, at most, once.

**Strategy II.** At any point in the search, you have a best candidate with the corresponding best distance. If you can determine, by a quick check, that a triangle is not capable of giving a better distance, there is no need to continue the calculation further. You do not need the closest point and distance for this triangle.

## MinDEMap



**Figure 32 : Minimum DE Mapping schematics**

Figure 32 shows the general flow of logic for the gamut map MinDEMap. For a query point, the CheckGamut function is first invoked. If the point is in-gamut, the map is a no-op. If the point is out-of-gamut, call ProjectPointToBoundary. Now move on to Figure 33. At this point, it is assumed that the following values have been computed.

- (a) Unit normal vector to each Gamut Boundary Triangle with respect to the standard dot product.
- (b) Vertex list and edge list, in addition to the triangle list.

### ProjectPointToBoundary

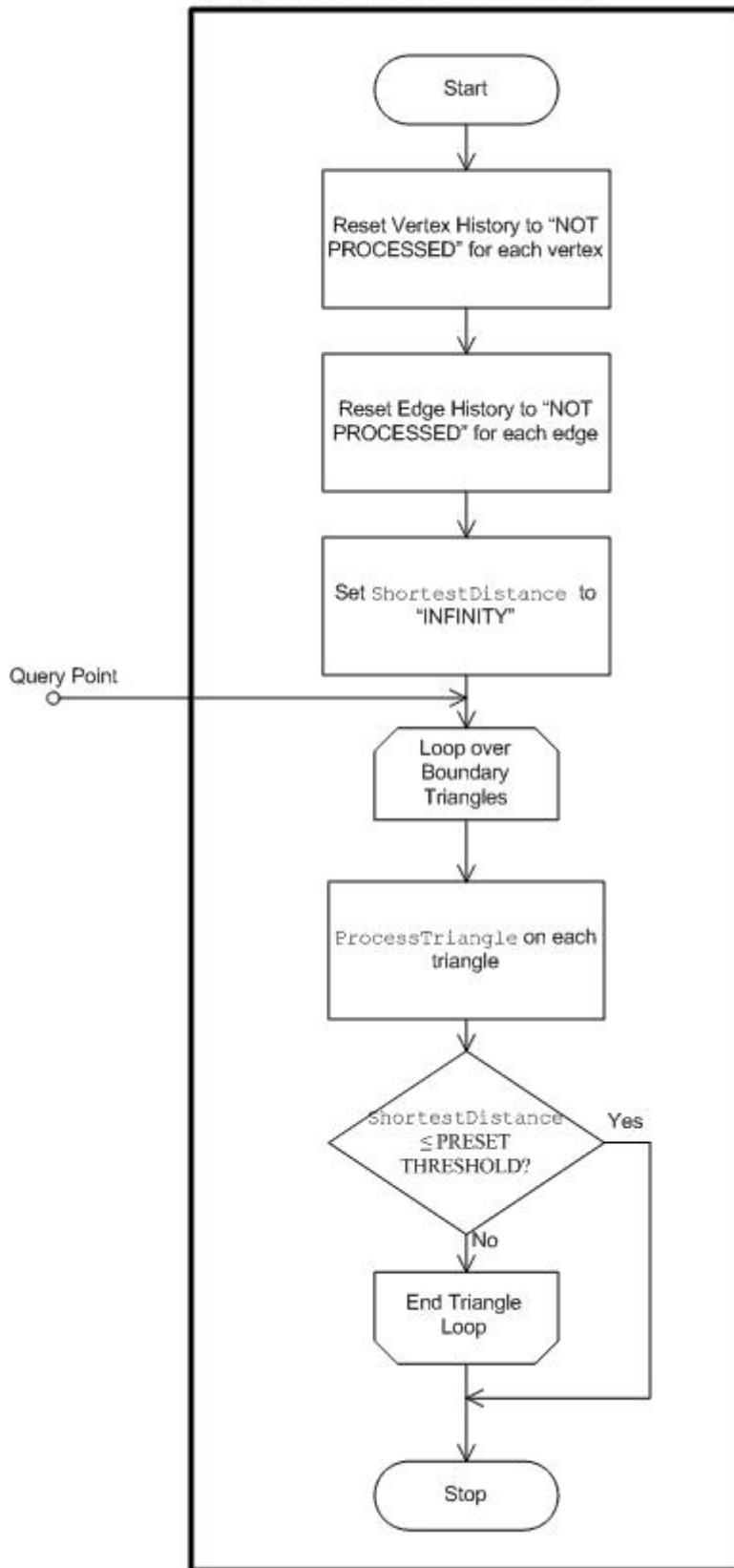


Figure 33 : The ProjectPointToBoundary routine

All these are constant overhead, and would have diminishing cost if sufficient queries to this gamut boundary are made. Usually, this is the case when you build a transform LUT from one device to another, where there are only two fixed gamuts, and the transform LUT runs through points on the uniformly sampled grid. You pre-compute the normal vectors with respect to the standard dot product, even though the notion of

perpendicularity will be based on the weighted dot product, which depends on the query point as explained previously. The reason is because a normal vector with respect to the weighted dot product can be obtained easily from the normal vector with respect to the standard dot product. If  $n_0$  is a normal vector with respect to the standard dot product, then

$$n = (J\text{-component of } n_0 / w_J, a\text{-component of } n_0, b\text{-component of } n_0)$$

is normal to the triangle with respect to the weighted dot product. Because of this relationship, it is still beneficial to pre-compute  $n_0$  even though it must be adjusted based on the query point.

The ProjectPointToBoundary routine starts by resetting the "processed history" of the vertices and edges. These are tables of BOOLEAN flags that track whether a vertex or edge has been visited before. It also resets the variable ShortestDistance to "INFINITY," which is the maximum encoded value in the floating-point number system used. Then it runs through a loop, searching for the closest point from each triangle using the ProcessTriangle call. ProcessTriangle is the routine to update the ShortestDistance variable and is clearly in the critical loop. One optimization is to stop when the result is good enough. After each call to ProcessTriangle, the variable ShortestDistance is examined. If it satisfies a predefined threshold, you can stop. The predefined threshold is dependent on the color space used and on the required accuracy of the color imaging system. For a typical application, you do not want to do unnecessary work if the color difference is less than what can be discerned by human vision. For CIECAM02, this color difference is 1. Use a threshold value of 0.005 in the implementation, however, to preserve the precision of computations, because this might be only an intermediate step in a chain of transforms.

ProcessTriangle implements the preceding Strategy II. Obtaining a normal vector from the pre-computed unit normal vector to the triangle with respect to the standard dot product, it computes the distance of the query point to the infinite plane containing the triangle by forming the dot product of the unit normal vector and the queryVector, the vector from one of the vertices of the triangle, vertex1, to the query point, queryPoint.

$$\text{queryVector} = \text{queryPoint} - \text{vertex1}$$

$$\text{distance} = |\text{normalVector} * \text{queryVector}| / \|\text{normalVector}\|$$

This is a relatively inexpensive calculation, and the distance is required to carry out further calculations. If this distance is not less than the current best distance, ShortestDistance, this triangle will not produce a better distance, because it will not give a better distance than the plane containing it. In this case, you return control to the triangle loop. If the distance is smaller than ShortestDistance, potentially, you have a

closer point, if this point lies inside the triangle. You must perform some "hard" calculations (though nothing beyond linear algebra) to determine that. If the other two vertices of the triangle are vertex2 and vertex3, then form the basis vectors firstBasisVector and secondBasisVector.

firstBasisVector = vertex2 - vertex1

secondBasisVector = vertex3 - vertex1

Use the following linear system of equations to solve for unknowns u and v.

firstBasisVector \* queryVector = (firstBasisVector \* firstBasisVector)u + (firstBasisVector \* secondBasisVector)v

secondBasisVector \* queryVector = (secondBasisVector \* firstBasisVector)u + (secondBasisVector \* secondBasisVector)v

and the conditions for the projected point to lie inside the triangle are:

$0 \leq u \leq 1$ ,  $0 \leq v \leq 1$  and  $u + v \leq 1$

After this calculation, if it is determined that the projected point lies within the triangle, then you have found a new closest point; the distance that you calculated at the beginning is the new shortest distance. In this case, update the variables ShortestDistance and ClosestPoint. If the projected point lies outside the triangle, you might find a closer point on one of its edges. So, you can call the ProcessEdge routine on each of the three edges.

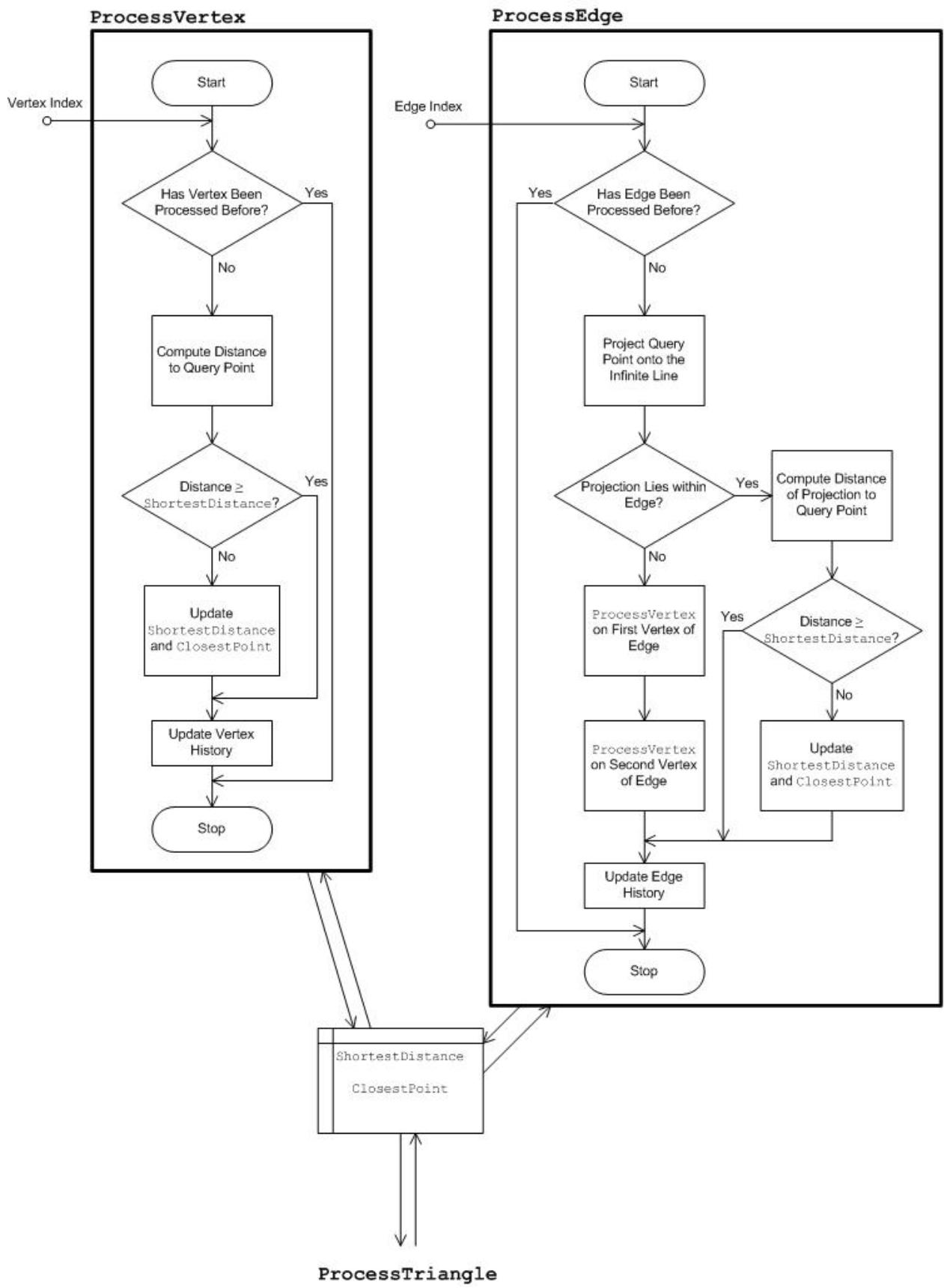


Figure 34 : ProcessEdge and ProcessVertex routines

The ProcessEdge routine implements Strategy I, which is illustrated in Figure 34. ProcessEdge starts by checking whether the edge has been processed before. If so, no further action is taken. If not, it proceeds to calculate the orthogonal projection of the query point onto the infinite line containing the edge. The linear algebra involved in the

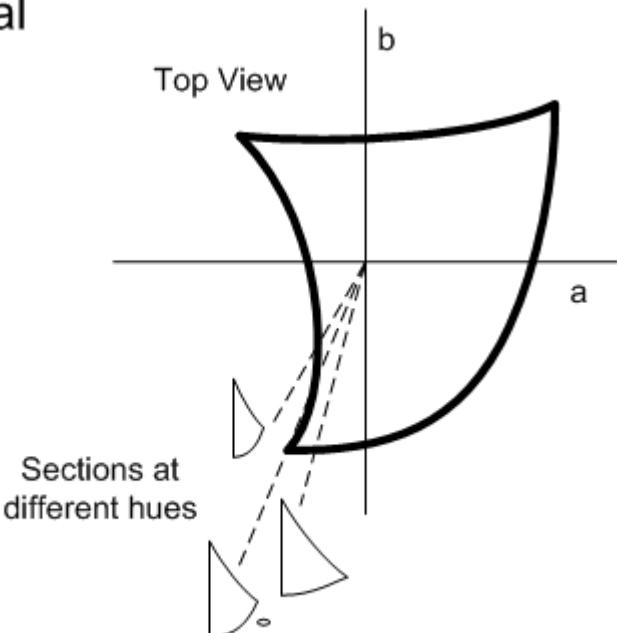
calculation is similar to the previous triangle equations. However, the calculation is simpler, it is not described here. If the projected point lies within the edge, you find the distance of the projected point from the query point. If this distance is smaller than ShortestDistance, you have found a new closest point. Update both ShortestDistance and ClosestPoint. If the projected point lies outside the edge, call ProcessVertex on the two end points. Before returning control, update the edge history so that this edge is marked as "PROCESSED."

Finally, you give a description of ProcessVertex. The ProjectVertex routine also implements Strategy I and maintains a vertex history table. As illustrated in Figure 34, it first checks whether the vertex has been processed before. If so, no further action is taken. If not, it proceeds to calculate the distance of the vertex from the query point. If the distance is smaller than ShortestDistance, update both ShortestDistance and ClosestPoint. At the end, it updates the vertex history so that this vertex is marked as "PROCESSED."

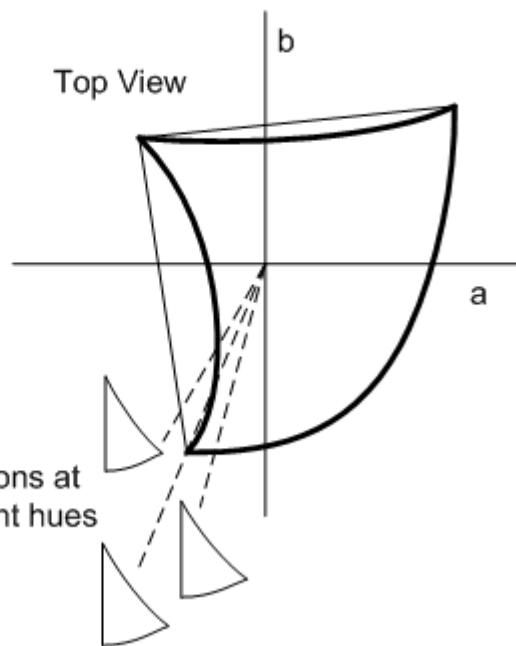
When the outer control loop has either exhausted all the triangles or exited before color difference threshold has been met, the variable ClosestPoint is accessed. This is the result of MinDEMMap. The caller can also retrieve ShortestDistance if interested in how far the mapped color is from the query color.

## Hue Smoothing

## Original



## Hue Smoothed



**Figure 35 : Hue Smoothing**

An issue arises with operations that are hue constrained; that is, the operation only considers variables within a hue plane. Figure 35 shows an example of a gamut exhibiting "discontinuous" hue slices in the blue hues. Within this hue range, for certain hue angles, the gamut boundary is tangential to the hue plane. In effect, this causes a change in the topological structure of the hue slices. In the example shown, as the hue plane sweeps across this hue range, an "island" emerges and submerges. This change in topology will cause hue specific operations to be discontinuous. For example, the cusp at fixed hue will change abruptly as the hue angle changes across this range.

There is a color science reason why it is desirable to preserve hue in certain operations. To resolve the preceding issue, the original Gamut Boundary Triangles must be "hue smoothed." Generally speaking, a hue smoothing of a set of Gamut Boundary Triangles is a set of triangles such that (a) it forms the boundary of a new "gamut," which might not correspond to the actual device gamut, and that contains the gamut defined by the original set of triangles; and (b) the triangles in the new set are bounded away from being parallel to the hue planes.

One practical way to obtain a hue smoothed set of triangles is to take the convex hull of the original vertices. As illustrated in Figure 35, the hue slices of the convex hull vary smoothly in the problematic hue range without a sudden change in topology.

## Setting Primaries and Secondaries in the Gamut Boundary Description

Certain gamut mapping methods, such as HueMap, depend on the location of the device primaries and secondaries. For additive devices, the primaries are red, green, and blue (R, G, and B); and the secondaries are cyan, magenta, and yellow (C, M, and Y). For subtractive devices, the primaries are C, M, and Y; and the secondaries are R, G, and B. The GBD keeps track of all six of those values, plus white and black (W and K), in a array of Jab color values. These values are set into the gamut boundary description when it is created. For output devices, the primaries can be determined by running combinations of device control values through the device model. For capture devices, this approach is not well-suited for creating the reference GBD, because it is almost impossible to capture an image that yields a fully saturated pure device value, such as (0.0, 0.0, 1.0). WCS device profiles contain the indices of the primaries in the capture target. Because these values are not contained in an ICC profile, use values measured from a typical scanner target after converting to Jab, relative to the ICC viewing conditions.

## Setting the Neutral Axis in the Gamut Boundary Description

The HueMap and the Relative MinCD gamut mapping methods use the device neutral axis for straightening. For the baseline output devices, the neutral axis can be determined by running device neutral values (R=G=B or C=M=Y) through the DeviceToColorimetric method, and then through the ColorimetricToAppearance method of the CIECAM02 object. However, capture devices do not always return a device neutral value when presented with a neutral sample. This is particularly true when the ambient lighting is not perfectly neutral. WCS device profiles contain the indices of the neutral samples in the target. Use those samples to set the neutral axis. Because this

information is not available for ICC profiles, you must use the same method that is used for output devices; run device neutral samples through the DeviceToColorimetric method, and then couple the input values and the colorimetric results.

## Related topics

[Basic color management concepts](#)

[Windows Color System Schemas and Algorithms](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WCS Transform Creation Algorithms

Article • 10/12/2022

## [Creation of Transforms](#)

### [Sequential Transform Execution](#)

### [Creation of Optimized Transforms](#)

#### [ICCProfileFromWCSProfile](#)

#### [Black Preservation and Black Generation](#)

#### [Check Gamut](#)

## Creation of Transforms

To properly explain how color transforms work, it is helpful to explain the complete processing path through both ICM 2.0 and the internals of the CTE. The ICM 2.0 [CreateColorTransformW](#) function creates a color transform that applications can use to perform color management. This function creates a color context from the [LOGCOLORSPACE](#) and intent inputs. The intents are mapped to baseline ICC gamut mapping algorithm correlates. The function then calls ICM 2.0 function [CreateMultiProfileTransform](#) for consistent color processing. The [CreateColorTransform](#) function generally copies data into the internal optimized transform structure.

The ICM 2.0 CreateMultiProfileTransform function accepts an array of profiles and an array of intents or a single device link profile and creates a color transform that applications can use to perform color mapping. It processes those input profiles and intents to create device models, color appearance models, gamut boundary descriptions, and gamut mapping models. Here's how this is done:

- Device models are initialized directly from DM profiles. There is one device model created for each profile in the call to [CreateMultiProfileTransform](#).

- Color appearance models are initialized directly from CAM profiles. There is one CAM profile for each profile in the call to [CreateMultiProfileTransform](#). The same CAM profile can be specified for more than one profile, however.
- Gamut boundary descriptions are initialized from a device model object and a CAM object. There is one gamut boundary description for each profile in the call to [CreateMultiProfileTransform](#).
- Gamut mapping models are initialized from two gamut boundaries and an intent. You must create a gamut mapping model between each pair of device models created from the call to [CreateMultiProfileTransform](#). Note that this means that you use one fewer gamut map model than device model. Because the number of intents matches the number of device models, there is also one more intent than required. The first intent in the list is skipped. You walk through the list of device models and intents, creating gamut mapping models. Pick up the first and second device models and the second intent, and then initialize the first gamut mapping model. Pick up the second and third device models and the third intent, and then initialize the second gamut mapping model. Continue in this manner until you have created all the gamut mapping models.

When the profiles have been properly processed and all intermediate objects have been created and initialized, then you can create the CITE transform with the following call. The *pDestCAM* and *pDestDM* are those associated with the last profile in the call to [CreateMultiProfileTransform](#).

C++

```
HRESULT CreateCITEColorTransform(
    __inout     IDeviceModel          *pSourceDM,
    __inout     IColorAppearanceModel *pSourceCAM,
    __in       GamutMapArray         *pGamutMapArray,
    __inout     IColorAppearanceModel *pDestCAM,
    __inout     IDeviceModel          *pDestDM,
    EColorTransformMode      eTransformMode,
    __deref_out  IColorTransform      **ppCTS
);
```

## Support for plug-ins

One issue involved in setting up the transform list is to validate whether a required plug-in is available. The following model switch provides this policy to control this behavior. The management of this transform list is a method in the internal optimized transform structure, but each model method provides the pointer to itself and its own set of parameter values.

The mode must be one of the following.

- TfmRobust: If a measurement profile specifies a preferred plug-in and the plug-in is not available, the new CTE system will use the baseline plug-in. If neither plug-in is available, the transform will report an error.
- TfmStrict: If the ColorContext specifies a preferred plug-in, the plug-in must be available. If no preferred plug-in is found, the baseline plug-in will be used. If neither plug-in is available, the transform will report an error.
- TfmBaseline: Only baseline plug-ins can be used in AddMeasurementStep. If the ColorContext specifies a preferred plug-in, the plug-in will be ignored. If the baseline plug-in is not available, the transform will report an error.

## Transform execution

The ICM 2.0 API [TranslateColors](#) function

translates an array of colors from the source [color space](#) to the destination color space as defined by a color transform. This function internally checks against an array of cached colors to enable immediate matching of commonly transformed colors. This transform supports 8-bit per channel byte arrays and 32-bit per channel float arrays. All other formats will be converted prior to passing off to the new CTE.

The ICM 2.0 API [TranslateBitmapBits](#) function translates the colors of a bitmap having a defined format to produce another bitmap in a requested format. This function internally checks against an array of cached colors to enable immediate matching of commonly transformed colors. To avoid too many code paths, support, and testing complexity, only a limited number of bitmap formats are actually supported in the transform and interpolation engine. This function must translate the non-native incoming and outgoing bitmap formats into natively supported formats for processing. This transform only supports 8-bit per channel byte bitmaps and 32-bit per channel float bitmaps. All other formats will be converted prior to passing off to the new CTE.

## Sequential Transform Execution

If the *dwFlags* parameter has the SEQUENTIAL\_TRANSFORM bit set when the ICM functions [CreateColorTransformW](#) or [CreateMultiProfileTransform](#) are called, then the transform steps are executed sequentially. This means that the code steps through each device model, color appearance model, and gamut mapping model separately, as specified by the [CreateColorTransform](#) or [CreateMultiProfileTransform](#) call. This may be helpful for debugging plug-in modules, but it is much slower than executing through an

optimized transform. Running in sequential mode is therefore not recommended for production software. Also, there may be slight differences in the results obtained in sequential mode and in optimized mode. This is due to variations introduced when the functions are concatenated together.

## Creation of Optimized Transforms

An optimized transform is a multi-dimensional lookup table. The table can be processed by a multi-dimensional interpolation engine, such as tetrahedral interpolation, that applies the input colors to the transform. The following section describes how the optimized lookup tables are created. The section after that describes how to interpolate within the optimized lookup tables.

## Sparse lookup tables

Conventional printers have CMYK inks. To extend the gamut, one approach is to add new inks to the system. The inks typically added are colors that CMYK inks have difficulty reproducing. Common choices are orange, green, red, blue, etc. To increase the "apparent resolution," inks with different tints can be used, for example, light cyan, light magenta, and so on. In effect, the printer device has more than four channels.

Although printers are output devices, they also perform the color conversion from the device space to another color space. In the case of a CMYK printer, this would be a transformation from CMYK to XYZ, or the "forward model" of the printer. By combining the forward model with other transformations, it is possible to emulate a CMYK print on another device. For example, a printer CMYK to a monitor RGB would make possible a proofing mechanism that emulates a print of that CMYK printer on a monitor. Similarly, the same also applies for hi-fi printers. A CMYKOG to RGB conversion allows proofing of the CMYKOG printer on a monitor.

The conventional approach to implementing such color conversion is by using a uniform LUT. For example, in an ICC profile for a CMYKOG printer, the ICC specification mandates an A2B1 tag that stores a uniform LUT representing a uniform sampling in the CMYKOG device space of the forward model, which goes from CMYKOG to the ICC profile connection space (either CIELAB or CIEXYZ). The ICC device link profile enables a direct transformation from CMYKOG device space to any color space including a device space, also in the form of a LUT sampled uniformly in CMYKOG space. Sampling is never done with 256 levels (bit depth 8) because of the huge LUT resulting, except in the case of monochrome devices (1 channel). Instead, sampling with lower bit depth is used; some typical choices are 9 (bit depth 3), 17 (bit depth 4), 33 (bit depth 5). With the number of

levels less than 256 in each channel, the LUT is used in conjunction with an interpolation algorithm to produce the result if a level is between two sampled levels.

While a uniform LUT is conceptually simple to implement, and interpolation on a uniform LUT is generally efficient, the LUT size increases exponentially with the input dimension. In fact, if  $d$  is the number of steps used in the uniform LUT, and  $n$  is the number of channels in the source color space, then the number of nodes in the LUT is  $d^n$ . Clearly, the number of nodes quickly demands so much storage in memory that even top-of-the-line computing systems have difficulty handling the demand. For devices with six or eight channels, an ICC implementation of the device profile requires using few steps in the LUT, sometimes even down to five steps in the A2B1 table to keep the profile within megabytes instead of gigabytes. Clearly, using a smaller number of steps increases interpolation error, because there are now fewer samples. Because the LUT must be uniform, accuracy over the whole color space is degraded even in those regions of the space where a significant color difference can be caused by small change in the device value.

In devices with more than four colorants, certain subspaces of the whole device space are more important than others. For example, in CMYKOG space, cyan and green inks are seldom used together because their hues largely overlap each other. Similarly, yellow and orange inks largely overlap each other. A uniform reduction in the number of steps can be viewed as an overall degradation in quality across the whole color space, which is something you can afford for the improbable ink combinations, but not for the likely or important combinations.

While a uniformly sampled LUT is simple and efficient for interpolation, it imposes huge memory requirements as dimension increases. In reality, while a device might have six or eight channels, they are rarely used simultaneously. In most cases, the input color to color transformation has only a few "active" colorants and so resides in a lower dimensional color space. This also means that interpolation can be done more efficiently in that lower dimensional space because interpolation is faster when the dimension is lower.

Therefore, the approach is to stratify the whole device space into subspaces of various dimensions. And because lower dimensions (those combining three or four colorants) are more important, by stratifying the space, you can also apply different sampling rates; that is, a different number of steps, to the pieces; increasing sampling rates for lower dimensions, reducing them for higher dimensions.

To fix notations,  $n$  is the number of channels in the source color space of the color transformation that you want to sample. You can also refer to  $n$  as the input dimension, and  $n \geq 5$ , unless otherwise specified.

The basic building blocks are LUTs of various input *dimensions* and sizes, instead of one uniform LUT with input dimension  $n$ . To be more precise, aLUT is a rectangular lattice imposed on a unit cube; that is, all the device coordinates are normalized to the range  $[0, 1]$ . If  $v$  is the input dimension of the lut (note that does not have to be equal to  $n$ , although  $v \leq n$  is required), then it consists of  $v$  one-dimensional sampling grids:

Samp  $i: x_1, x_2, \dots, x_{d(i)}$

where all the  $x_j$ s must lie in the range  $[0, 1]$ ,  $d(i)$  is the number of steps for the  $i$  th channel sampling which must be at least 1, and  $x_{d(i)}$  must be 1. On the other hand,  $x_1$  is not required to be 0.

Only the following two special cases of LUT will be defined.

*Closed LUT*: This is a LUT with the additional requirement that for each Samp $^*_i$ ,  $x_1 = 0$ , and  $d(i) \geq 2$ . A uniform closed LUT is a closed LUT that has the same  $d(i)$  for each channel, and the nodes are uniformly spaced between 0 and 1.

*Open LUT*: This is a LUT with the additional requirement that for each Samp  $i$ ,  $x_1 > 0$ . It is OK to have  $d(i) = 1$ .

The goal is to stratify the unit cube  $[0, 1]^n$  into a collection of closed LUTs and open LUTs, such that the whole collection will cover the unit cube. It is conceptually simpler to organize these "LUT strata" by their dimensions, so that on the top level:

$$[0,1]^n = \bigcup_{k=3}^n \Sigma_k$$

where  $\Sigma_k$  is the " $k$ -dimensional strata collection." Note that the strata dimension starts from 3 instead of 0; that is, points, because interpolation of three-colorant combinations can be handled without too much memory requirement.

## Description of the LUT strata

In this implementation:

1.  $\Sigma_3$  consists of closed LUTs with three inputs, one from every possible combination of three colorants chosen out of the  $n$  colorants.
2.  $\Sigma_4$  consists of one closed LUT for the combination CMYK (or the first four colorants), together with  $\binom{n}{4} - 1$  open LUTs for all other four-colorant

combinations. By singling out the CMYK combination, you acknowledge that it is an important combination.

3. For  $k = 5, \dots, n$ ,  $\Sigma_k$  consists of  $\binom{n}{k}$  open LUTs, one for each possible combination of choosing  $k$  colorants from the total of  $n$  colorants.

It remains to specify the sizes of the LUTs. The key difference between open and closed LUTs is that open LUTs don't overlap, and closed LUTs might overlap at the boundary faces. The fact that the one-dimensional sampling in an open LUT does not contain 0, essentially means that an open LUT is missing half of the boundary faces, hence the name "open." If two LUTs don't overlap, you can use a different number of steps or node locations in each channel. The same is not true if two LUTs overlap. In that case, if the number of steps or the node locations are different, a point lying in the intersection of the two LUTs will receive a different interpolation value depending on which LUT is used in the interpolation. A simple solution to this problem is to use uniform sampling with the same number of steps whenever two LUTs overlap. In other words:

All closed LUTs (all three-colorant LUTs and the CMYK LUT in this implementation) must be uniform and have the same number of steps, which are denoted  $d$ .

The following two algorithms can be used to determine the number of steps  $d$  for closed LUTs and the number of steps for open LUTs.

## Algorithm #1

This algorithm does not require external input.

All the closed LUTs will be uniform with  $d$  number of steps.

All the open LUTs of dimension  $k$  will have same number of steps  $d(k)$  in each input channel, and the nodes are equally spaced; that is, for each  $i = 1, 2, \dots, k$ .

Samp  $i$ :  $1/d(k), 2/d(k), \dots, (d(k)-1)/d(k), 1$

Finally, specify  $d$  and  $d(k)$  in the following Table 1. The three modes, "proof," "normal," and "best" are the ICM 2.0 quality settings. In this implementation, the proof mode has the smallest memory footprint, and best mode has the largest memory footprint.

To implement this algorithm, you must call the following Algorithm #2. Users can specify their own sampling locations, using the tables as a guide.

## Algorithm #2

This algorithm requires external input in the form of a list of "important" sampling locations, but it is more adaptive and can save memory space potentially.

The required input is an array of device values supplied by the user. These device values indicate which region of the device color space is important; that is, which region should be sampled more.

All the closed LUTs will be uniform with  $d$  number of steps as described in Algorithm #1. Values for  $d$  are provided in Table 1.

(a) *Uniform Closed LUT*

	<b>Proof mode</b>	<b>Normal mode</b>	<b>Best mode</b>
$d$	9	17	33

(b) *Open LUT*

<b>Input dimension</b>	<b>Proof mode</b>	<b>Normal mode</b>	<b>Best mode</b>
4	5	7	9
5	2	3	3
6	2	3	3
7	2	2	2
8 or more	2	2	2

**Table 1:** LUT sizes used in the algorithm

Each open LUT can have a different number of steps in each input channel, and the sampling locations do not have to be equally spaced. For a given open LUT stratum, there is an associated colorant combination, for example,  $c_1, \dots, c_k$ , where the  $c_i$  s are distinct integers between 1 and  $n$ . They are the channel indices corresponding to the "active" colorants in this strata.

STEP 1: Filter out the inputted array of device values that are not contained in this strata. A device value  $(x_1, x_2, \dots, x_n)$  is contained in the strata, if and only if

$x_{c_1} > 0, x_{c_2} > 0, \dots, x_{c_k} > 0$  and all other channels are 0. If the filtered set has  $N$  entries, let

$$d_{\text{tentative}}(k) = \min(\text{dMax}(k), \max(1, \text{int}(N^{1/k})))$$

For each  $i = 1, 2, \dots, k$ , iterate the following steps 2-5:

STEP 2: If  $d_{\text{tentative}}(k) = 1$ , Samp  $i$  has only 1 point, which must be 1.0. Move on to the next  $i$ . Otherwise, continue to STEP 3.

STEP 3: Sort the filtered samples in ascending order in the  $c_i$  channel.

STEP 4: Define the "tentative" sampling grid using the nodes

$$x_j = (j - 1) \cdot 100 / (d_{\text{tentative}}(k) - 1)$$

where  $j = 1, 2, \dots, d_{\text{tentative}}(k)$ .

STEP 5: Regularize the tentative grid to ensure that it conforms with strict monotonicity and also that it ends with 1.0. Because the array is already sorted, the nodes in the tentative grid are already monotonic nondecreasing. However, adjacent nodes might be identical. You can fix this by removing identical nodes, if necessary. Finally, after this procedure, if the end point is less than 1.0, replace it with 1.0.

Note that STEP 5 is the reason that the LUT strata may have a different number of steps in each channel. After the regularization, the number of steps in a channel may be less than  $d_{\text{tentative}}(k)$ .

## Interpolation

You can construct the stratification of the unit cube by open LUT strata and closed LUT strata. To perform interpolation using this "sparse LUT structure," follow these steps.

Assume a given input device value  $(x_1, x_2, \dots, x_n)$ .

STEP 1: Determine the number of "active" channels. This is the number of non-zero channels. This determines the strata dimension  $k$  to search for the containing stratum. More precisely, the strata dimension is 3 if the number of active channels is  $\leq 3$ , otherwise, the strata dimension is the same as the number of active channels.

STEP 2: Within  $\Sigma_k$ , search for the containing stratum. A device value is contained in an open stratum if all the channels corresponding to the stratum have non-zero value, and all other channels are zero. A device value is contained in a closed stratum if every channel not represented by the stratum is zero. If no containing stratum is found, there

is an error condition. Cancel and report failure. If a containing stratum is found, proceed to the next step.

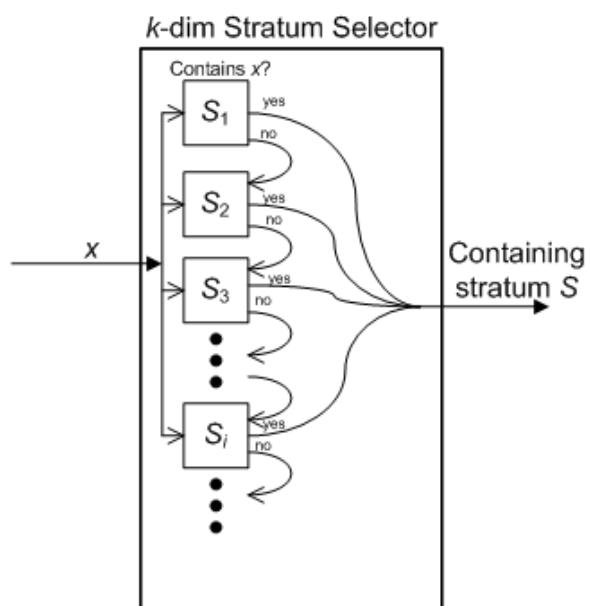
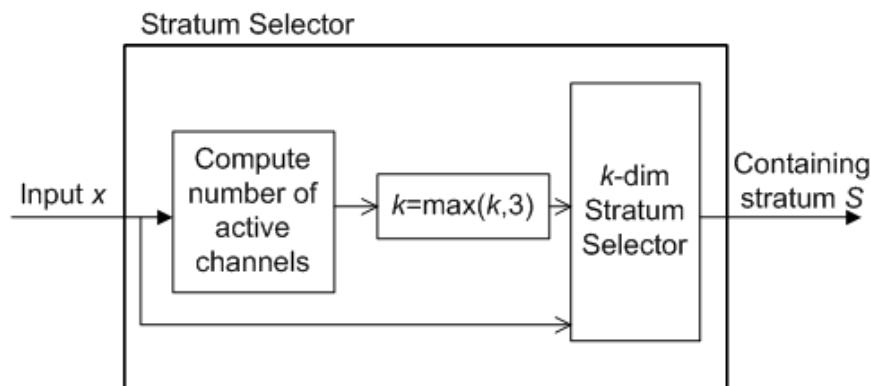
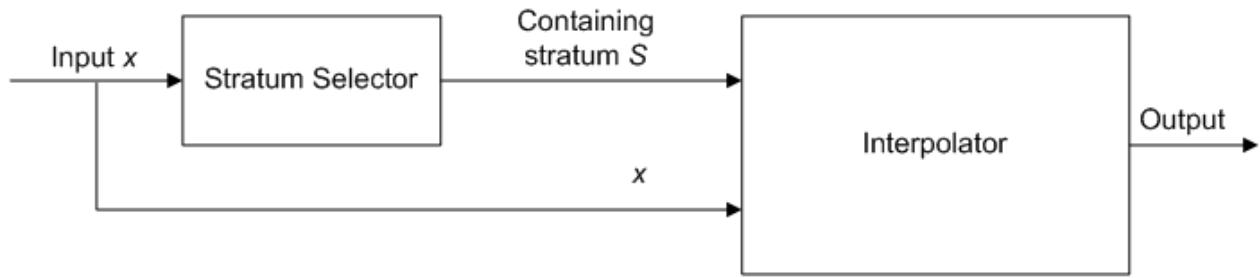
STEP 3: If the containing stratum is closed, then interpolation within the stratum can be done by any known interpolation algorithm. In this implementation, the choice of algorithm is tetrahedral interpolation. If the containing stratum is open, and the device value lies strictly within the stratum, that is,

$$x_i \geq \text{first node in } i \text{ th channel}$$

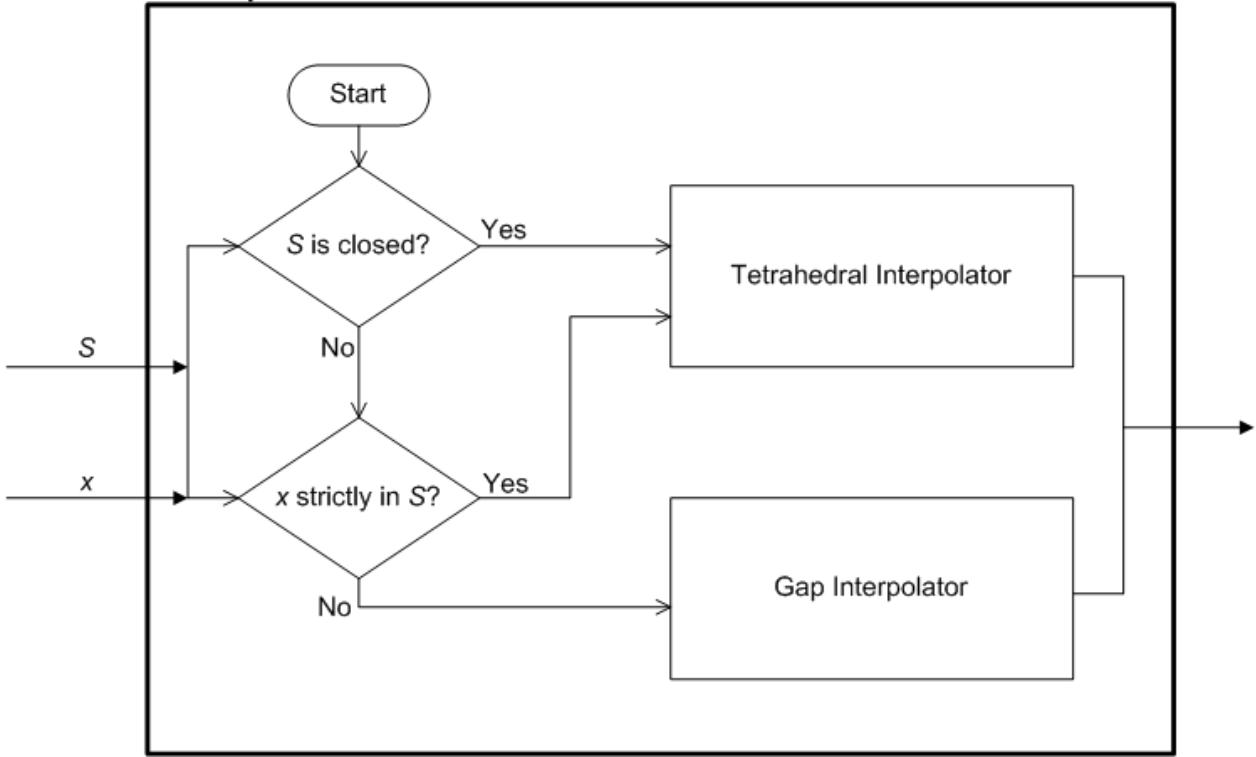
where  $i$  is a channel index for the stratum, then standard interpolation algorithm, such as tetrahedral interpolation, works.

If  $x_i <$  first node in  $i$  th channel for some  $i$ , then the device value falls into the "gap" between the stratum and the lower dimensional subspaces. This MOI is not concerned with an interpolation algorithm per se, so any interpolation algorithm can be used to interpolate within this "gap," although the preferred algorithm is the following transfinite interpolation.

The architecture of the interpolation module is illustrated in the two parts of Figure 1.



## Interpolator



**Figure 1:** Intepolation module architecture

As explained earlier, this algorithm is able to achieve reasonably dense sampling in regions of the device space that contain important combination of colorants, while minimizing the total size of LUTs needed. The following table shows a comparison of the number of nodes needed for the sparse LUT implementation (using Algorithm #1 and normal mode) and the corresponding uniform LUT implementation.

Number of input channels	Sparse LUT	Uniform LUT
5	142498	1419857
6	217582	24137567
7	347444	410338673
8	559618	6975757441

## Interpolation within a unit cube

A basic step in the case of rectangular grid is interpolation within an enclosing cell. For an input point, you can determine the enclosing cell easily. In a rectangular grid, the output value at each of the vertices (corner points) of the enclosing cell is specified. They are also the only boundary conditions (BCs) that an interpolant must satisfy: The

interpolant must pass through all these points. Note that these boundary conditions are on "discrete" points, in this case the  $2n$  corner points of the cell, where  $n$  is the dimension of the color space.

It is useful to formalize the concept of boundary conditions before moving on. For any subset  $S$  of the boundary of the enclosing cell (the unit cube in  $n$  dimensions), a boundary condition on  $S$  is a specification of a function  $BC: S \rightarrow R^m$ , where  $m$  is the output dimension. In other words, an interpolant, which may be denoted  $\text{Interp}: [0,1]^n \rightarrow R^m$ , is required to satisfy:  $\text{Interp}(x) = BC(x)$  for all  $x$  in  $S$ .

In the standard scenario of interpolation on the unit cube,  $S$  is the set of discrete points that are the  $2n$  vertices of the cube.

You can now generalize the boundary conditions to solve the issues described earlier and provide a new interpolation algorithm within the unit cube. Instead of allowing only discrete boundary points, boundary conditions can be imposed on a whole boundary face of the cube. The precise assumptions are as follows:

- (a) The point  $v_n = (1, 1, \dots, 1)$  is special and only a discrete boundary condition is allowed. In other words, no continuous boundary conditions can be imposed on the  $n$  boundary faces  $x_i=1$  ( $i=1, \dots, n$ ).
- (b) For each of the remaining  $n$  boundary faces  $x_i=0$  ( $i=1, \dots, n$ ), boundary condition can be imposed on the whole face, with the compatibility condition that if two faces intersect, the boundary conditions on the faces should agree on the intersection.
- (c) Any vertices not contained in the faces with boundary condition will have an individual (discrete) boundary condition.

You can refer to a discrete boundary condition as finite data, and a continuous boundary condition as transfinite data in discussing interpolation on finite and transfinite data.

First, review the standard tetrahedral interpolation (such as that used in Sakamoto's patent) which helps set the notations for this particular formulation of the problem. It is known that the unit cube  $[0,1]^n$  can be subdivided into  $n!$  tetrahedra, parameterized by the set of permutations on  $n$  symbols. More specifically, each such tetrahedron is defined by inequalities

$$x_{\sigma(1)} \geq x_{\sigma(2)} \geq \dots \geq x_{\sigma(n)}$$

where  $\sigma: \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, n\}$  is a permutation of "symbols" 1, 2, ...,  $n$ , that is, it is a bijective mapping of the set of  $n$  symbols. For example, if  $n = 3$  and  $\sigma = (3, 2, 1)$ , meaning  $\sigma(1)=3$ ,  $\sigma(2)=2$ ,  $\sigma(3)=1$ , then the corresponding tetrahedron is defined by  $z \geq y \geq x$ , where the common notation  $x, y, z$  is used for  $x_1, x_2, x_3$ . Note that these tetrahedrons are not

disjoint from each other. For the purpose of interpolation, points lying on a common face of two distinct tetrahedrons will have the same interpolation value regardless of which tetrahedron is used in the interpolation. Still, in the standard scenario of interpolating on finite points, for a given input point  $(x_1, \dots, x_n)$ , first determine which tetrahedron it lies in, or equivalently, the corresponding permutation  $\sigma$ , then the tetrahedral interpolant is defined as

$$Interp(\mathbf{x}) = BC(\mathbf{v}_0) + \sum_{i=1}^n x_i [BC(\mathbf{v}_i) - BC(\mathbf{v}_{i-1})]$$

$$\mathbf{v}_0 = \mathbf{0}, \mathbf{v}_i = \sum_{j=1}^i \mathbf{e}_{\sigma(j)}$$

where  $\mathbf{v}_i$  for  $i=1, \dots, n$ , and  $\mathbf{e}_1, \dots, \mathbf{e}_n$  are the standard basis vectors.

Before moving on to the generalization, note that  $v_0, v_1, \dots, v_n$  are the vertices of the tetrahedron, and  $1 - x_{\sigma(1)}, x_{\sigma(1)} - x_{\sigma(2)}, x_{\sigma(2)} - x_{\sigma(3)}, \dots, x_{\sigma(n)}$  are the "barycentric coordinates."

For the general case of BCs on boundary faces, you can use the concept of barycentric projection. As before, for a given input point  $(x_1, \dots, x_n)$ , first determine in which tetrahedron it lies, or equivalently, the corresponding permutation  $\sigma$ . Then perform a series of barycentric projections, as follows. The first projection  $BProj_1(\mathbf{x})$  sends the point to the plane  $x_{\sigma(1)} = 0$  unless  $\mathbf{x} = \mathbf{v}_*$  in which case it is not changed. The precise definition of the map  $BProj$  is defined as follows:

$$BProj_k(\mathbf{x}) = \begin{cases} \mathbf{p}_k + (\mathbf{x} - \mathbf{p}_k) / (1 - x_{\sigma(k)}) & \text{if } \mathbf{x} \neq \mathbf{p}_k \\ \mathbf{x} & \text{if } \mathbf{x} = \mathbf{p}_k \end{cases}$$

$$\mathbf{p}_k = \mathbf{v}_* - \sum_{i=1}^{k-1} \mathbf{e}_{\sigma(n+1-i)}$$

with and  $k = 1, 2, \dots, n$ .

In the case  $\mathbf{x} = \mathbf{v}_*$ , you can stop, because BC is defined at  $v_n$  by Assumption (a). In the case  $\mathbf{x} \neq \mathbf{v}_*$ , it is clear that  $BProj_1(\mathbf{x})$  has the  $\sigma(1)$ th component annihilated. In other words, it is on one of boundary faces. Either it is on a face on which BC is defined, in which case you can stop, or you perform another barycentric projection  $BProj_2(\mathbf{x}')$  where  $\mathbf{x}' = BProj_1(\mathbf{x})$ . And if  $\mathbf{x}'' = BProj_1(\mathbf{x}')$  is on a face on which BC is defined, you can stop; otherwise, perform yet another projection  $BProj_3(\mathbf{x}'')$ . Because every projection annihilates one component, the effective dimension decreases, so you know the process must eventually stop. In the worst case scenario, you perform  $n$  projections down to dimension 0, that is, vertices on the cube, which according to Assumption (c), you know BC will be defined on.

Assuming that  $K$  projections have been performed, with

$$\mathbf{x}^{(k)} = \text{BProj}_k(\mathbf{x}^{(k-1)}), k = 1, \dots, K,$$

$\mathbf{x}(0) = \mathbf{x}$ , the input point, and BC is defined at  $\mathbf{x}(k)$ . Then unwind the projections by defining a series of output vectors:

$$\mathbf{y}^{(k-1)} = x_{\pi(k)}^{(k-1)} BC(\mathbf{p}_k) + (1 - x_{\pi(k)}^{(k-1)}) \mathbf{y}^{(k)}, k = K, K-1, \dots, 1.$$

where  $\mathbf{y}^{(K)} = BC(\mathbf{x}^{(K)})$ , and you finally obtain the answer

$$Interp(\mathbf{x}) = \mathbf{y}^{(0)}$$

## Worked example

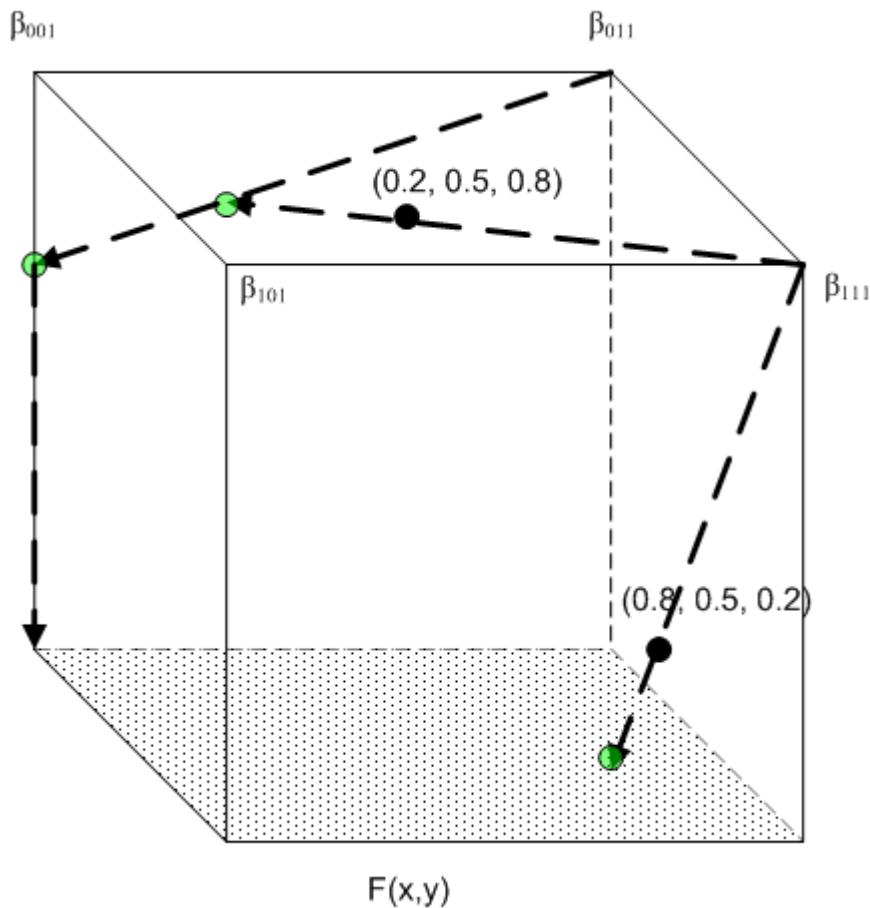


Figure 2: Worked example

Consider the situation depicted in Figure 2, where  $n = 3$ ,  $m = 1$ , and you have the following BCs:

- (a) Four discrete BCs on the vertices

$$(0, 0, 1): \beta_{001}$$

$$(0, 1, 1): \beta_{011}$$

(1, 0, 1):  $\beta_{101}$

(1, 1, 1):  $\beta_{111}$

(b) A continuous BC on the face  $x_3=0$ :  $F(x_1, x_2)$

Computation #1: Input point  $x = (0.8, 0.5, 0.2)$ . The enclosing tetrahedron is associated with the permutation  $<1, 2, 3>$ .

$$1\text{st projection: } \mathbf{p}_1 = (1,1,1), \mathbf{x}^{(1)} = \mathbf{p}_1 + (\mathbf{x} - \mathbf{p}_1) / (1 - 0.2) = (0.75, 0.375, 0)$$

This is already on the face  $x_3=0$ , so you can stop. Backward substitution then gives

$$\mathbf{y}^{(0)} = 0.2\beta_{111} + 0.8F(0.75, 0.375) \text{ which is the answer.}$$

Computation #2 : Input point  $x = (0.2, 0.5, 0.8)$ . The enclosing tetrahedron is associated with the permutation  $<3, 2, 1>$ .

$$1\text{st projection: } \mathbf{p}_1 = (1,1,1), \mathbf{x}^{(1)} = \mathbf{p}_1 + (\mathbf{x} - \mathbf{p}_1) / (1 - 0.2) = (0, 0.375, 0.75)$$

$$2\text{nd projection: } \mathbf{p}_2 = (0,1,1), \mathbf{x}^{(2)} = \mathbf{p}_2 + (\mathbf{x}^{(1)} - \mathbf{p}_2) / (1 - 0.375) = (0, 0, 0.6)$$

3rd projection:  $\mathbf{p}_3 = (0,0,1), \mathbf{x}^{(3)} = \mathbf{p}_3 + (\mathbf{x}^{(2)} - \mathbf{p}_3) / (1 - 0.6) = (0,0,0)$ , which is on the face  $x_3=0$ . Backward substitution then gives

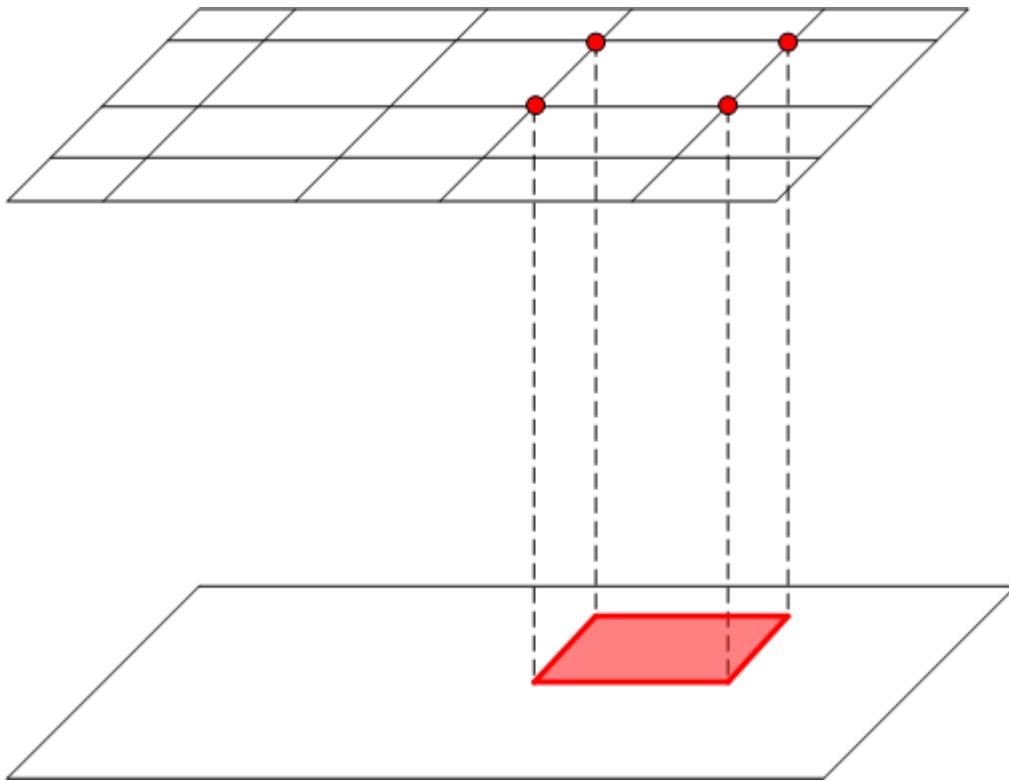
$$\mathbf{y}^{(2)} = 0.6\beta_{001} + 0.4F(0,0)$$

$$\mathbf{y}^{(1)} = 0.375\beta_{011} + 0.625\mathbf{y}^{(2)} = 0.375\beta_{011} + 0.375\beta_{001} + 0.25F(0,0)$$

$$\mathbf{y}^{(0)} = 0.2\beta_{111} + 0.8\mathbf{y}^{(1)} = 0.2\beta_{111} + 0.3\beta_{011} + 0.3\beta_{001} + 0.2F(0,0), \text{ which is the final answer.}$$

## Applications

(a) Sequential Tetrahedral Interpolation

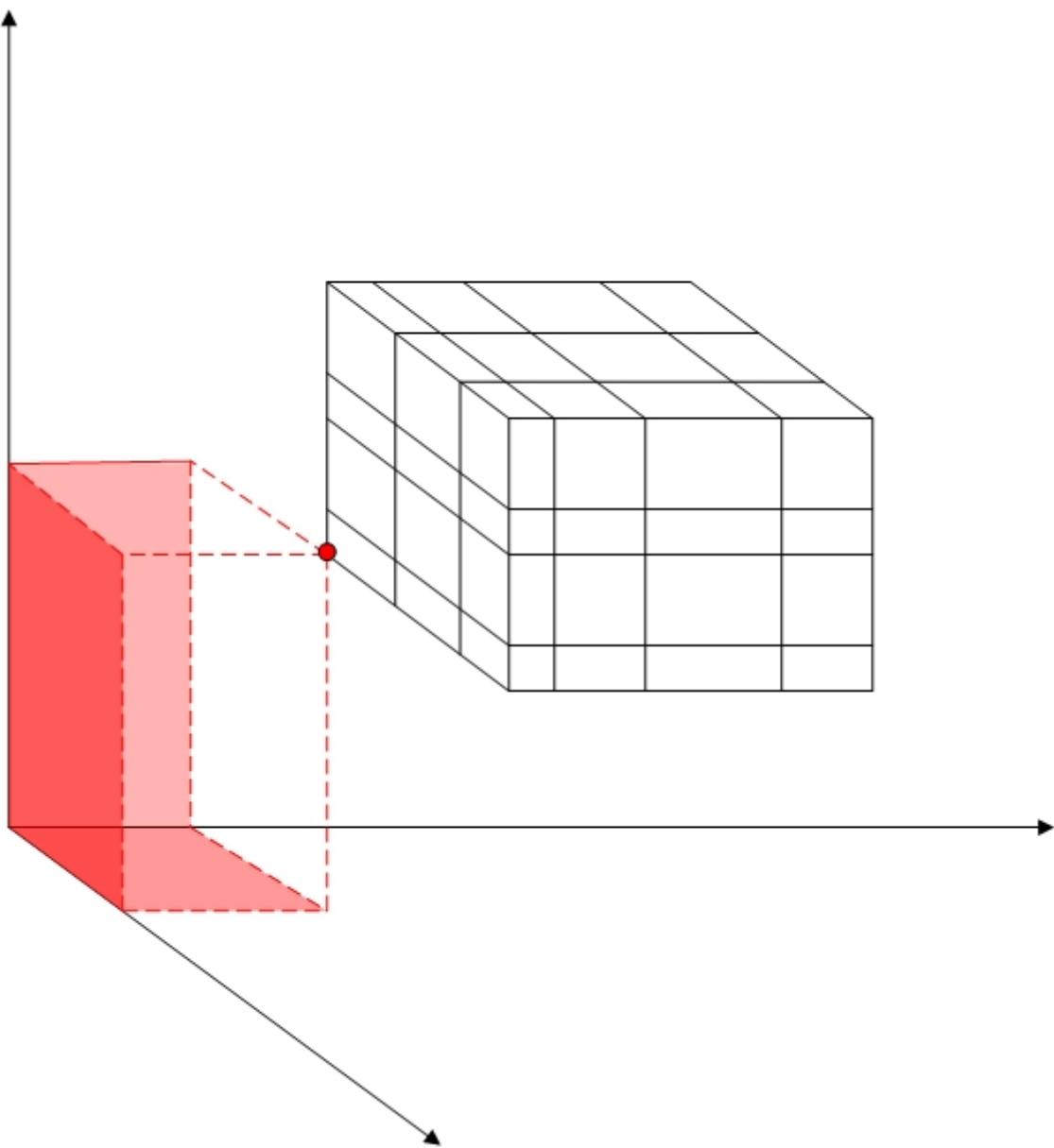


**Figure 3:** Sequential tetrahedral interpolation

Refer to Figure 3. To interpolate between two planes on which incompatible grids have been imposed, consider a cell enclosing a given point  $P$  shown in the figure. The "top" vertices of the cell come directly from the grid in the top plane. The vertices in the bottom face are not compatible with the grid in the bottom plane, so the whole face is treated as having a BC with values obtained by interpolation on the grid in the bottom plane. It is then clear that this setup satisfies Assumptions (a), (b), and (c) above, and you can apply the interpolation algorithm.

It is also clear that the algorithm has reduced the dimension of the interpolation problem by 1, because the result is a linear combination of values at the vertices in the upper grid, and interpolation in the lower plane, which has dimension less 1. If a similar sandwiching plane configuration exists in the lower plane, you can apply the procedure in that plane, further reducing the dimension by 1. This procedure can continue until you reach dimension 0. This cascade of projections and interpolations can be called "Sequential Tetrahedral Interpolation."

(b) *Gap Interpolation*



**Figure 4:** Gap interpolation

This is a grid imposed on a cube sitting strictly inside the positive quadrant. The cube itself has a grid on it, and each coordinate plane has grids that are not necessarily compatible. The "gap" between the cube and the coordinate planes has a cross-section that is "L-shaped" and is not amenable to standard techniques. However, with the technique introduced here, you can easily introduce cells that cover this gap. Figure 4 depicts one of these. The grids on the coordinate planes support interpolation that provides the necessary BCs for all the bottom faces of the cell, with one remaining vertex whose BC is provided by the lower corner of the cube.

## Final note on implementation

In actual application, the "unit cube" that is the basic setting of the algorithm is extracted from larger lattices, and the values at the vertices may require expensive calculation. On the other hand, it is also clear that tetrahedral interpolation requires only

the values at the vertices of the tetrahedron, which is a subset of all the vertices of the unit cube. Therefore, it is more efficient to implement what can be called "deferred evaluation." In a software implementation of the preceding algorithm, it is typical to have a subroutine that takes the unit cube and values at its vertices as input. Deferred evaluation means that instead of passing the values at the vertices, the necessary information to evaluate the values of the vertices is passed, without actually carrying out the evaluation. Inside the subroutine, actual evaluation of these values will be carried out only for those vertices that belong to the enclosing tetrahedron, after the enclosing tetrahedron is determined.

## **Lookup table for use with high dynamic range virtual RGB source devices**

In the case where a transform is constructed with a source device that is modeled as a virtual RGB device, it is possible that source colorant values may be negative or greater than unity (1.0). When this occurs, the source device is referred to as having a high dynamic range(HDR). Special consideration is made for this case.

In the case of HDR transforms, the minimum and maximum values for each colorant channel can be determined from the gamut boundary of the device. By using these values, a simple scaling for each colorant channel is applied so that colorant values equal to the minimum colorant would be converted to 0.0, and colorant values equal to the maximum colorant would be converted to 1.0, with a linear scaling of values between to map linearly between 0.0 and 1.0.

## **ICCProfileFromWCSProfile**

Because the main purpose of this feature is to support pre-Vista versions of Windows, you must generate Version 2.2 ICC profiles as defined in ICC Specification ICC.1:1998-09. In certain cases (see the following table "Baseline Device To ICC Profile Class Mapping"), you can create a matrix or TRC-based ICC profile from a WCS profile. In other cases, the ICC profile consists of LUTs. The following process describes how to create the AToB and BToA LUTs. Of course, ICC profiles have other fields as well. Some of the data can be derived from the WCS profile. For other data, you'll have to develop intelligent defaults. The copyright will be assigned to Microsoft; since it is Microsoft technology that is being used to create the LUTs.

This design should work for all types of device models, including plug-ins. As long as the plug-in has an associated baseline device model, the underlying device type can be determined.

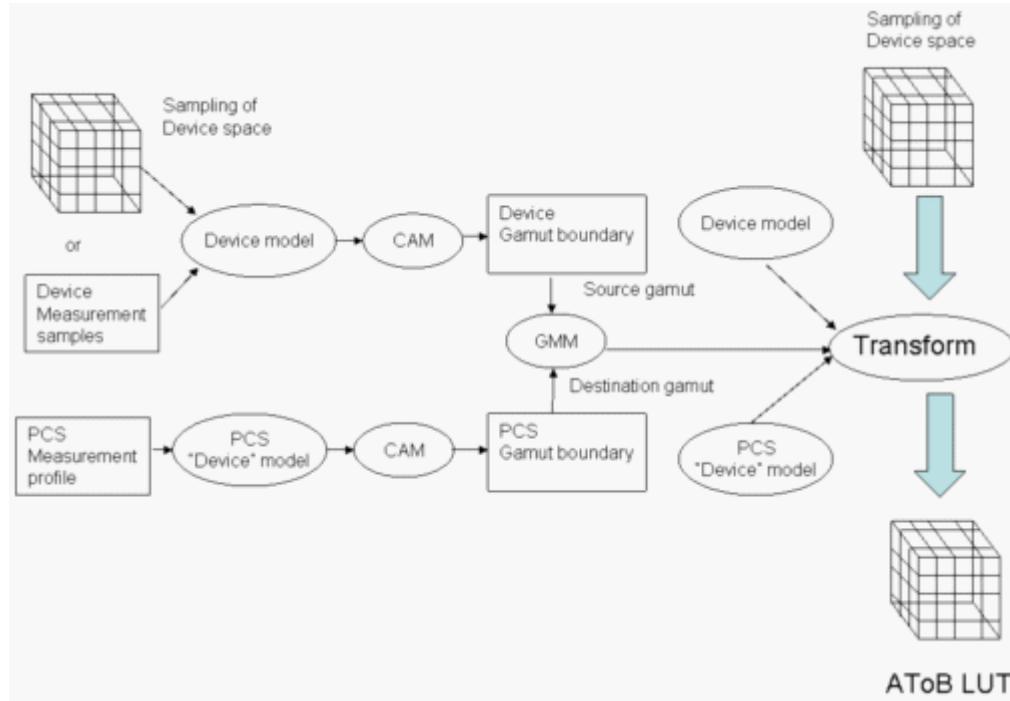
The hard part of creating an ICC profile is creating the AToB and BToA lookup tables. These tables map between the device space, for example, RGB or CMYK, and the Profile Connection Space (PCS), which is a variant of CIELAB. This is fundamentally the same as the color management process used in the CITE transform to map from device space to device space. However, you must have the following information to make the transformation.

1. Reference viewing conditions for the PCS.
2. Reference PCS gamut.
3. Device model that converts between PCS values and colorimetry.

The WCS profile and its associated CAM are provided as parameters. There are two baseline device models that convert between colorimetry and the PCS encoding. The reason you need two is explained below.

1. You can obtain the reference viewing conditions for the PCS from the ICC profile format specification. The information provided in the ICC profile format specification is sufficient to compute all the data required to initialize the CAM used by the CMS. For consistency and flexibility, this information is stored in a WCS color profile.
2. You can also use a WCS profile to store samples that define the reference gamut of the PCS. The CITE color management system (CMS) has two ways to create gamut boundaries. One is to sample the complete device space and use the device model to create measurement values. The second method is to use measured samples from the profile to create a reference gamut boundary. Because the gamut of the ICC PCS is too large to make a useful reference gamut, the first method is inappropriate. But the second method is a flexible, profile-based approach. To redefine the reference PCS gamut, you can change the measurement data in the PCS device profile.
3. The ICC PCS is a modeling of an ideal device. By creating a model of the PCS as a real device, you can take advantage of the color management process used in the Smart CMM. Creating a device model from colorimetry to the PCS encoding is straightforward. You simply map between the true colorimetric values and the PCS encoded values. Since the CMS interface for device models only supports XYZ values, you might also have to map between XYZ and LAB. This is a well-known transformation. This model is described in the document 2.2.02 "Baseline Device Models" in sections 7.9 and 7.10.

You might have to perform some gamut mapping, if the gamut of the device is larger than the gamut of the PCS. The baseline GMMs can be used for this purpose. Note that a properly created ICC profile has lookup tables for the Relative Colorimetric, Perceptual, and Saturation intents, although these may all point to the same LUT internally.

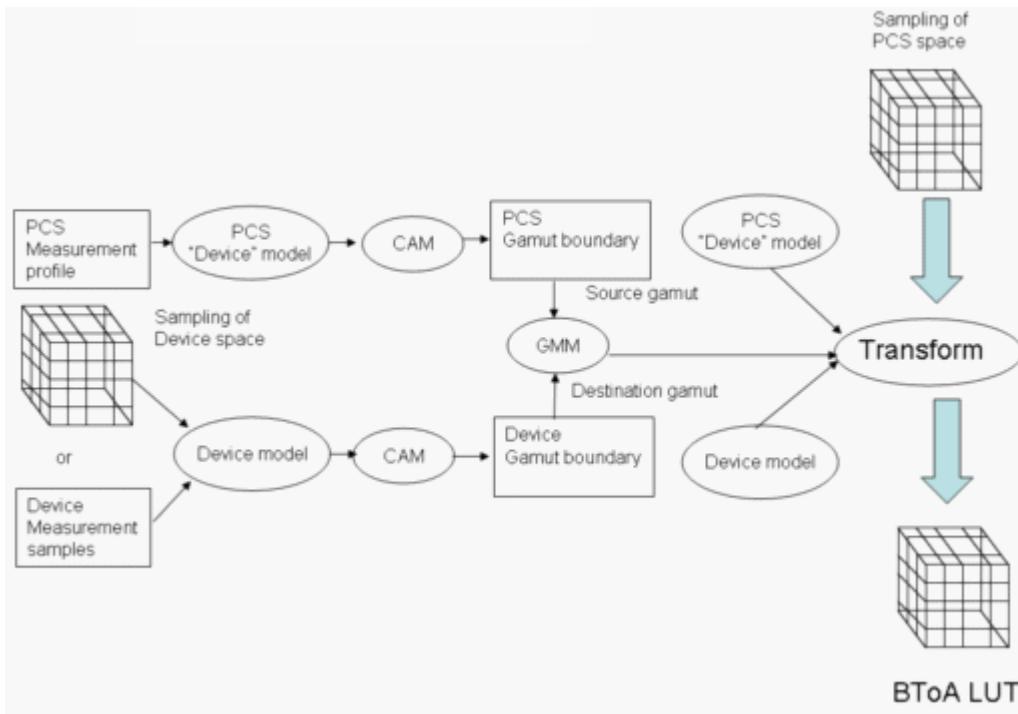


**Figure 5:** Creation of an AToB LUT

This process is illustrated in Figure 5. First, the device model is initialized from the data in the DM profile. Then, construct a device gamut boundary as follows. A sampling of data from the device model is run through the device model to obtain colorimetric data. The colorimetric data is run through the CAM to create appearance data. The appearance data is used to create the device gamut boundary.

Next, use data from the reference PCS measurement profile to create a gamut boundary for the PCS.

Use the two gamut boundaries just created to initialize a GMM. Then, use the device model, the GMM, and the PCS device model to create a transform. Run a sampling of device space through the transform to create an AToB LUT.



**Figure 6:** Creation of a BToA LUT

Figure 6 illustrates the creation of the BToA LUT. This is almost identical to creating an AToB LUT, with the roles of source and destination exchanged. Also, you must sample the full PCS gamut to create the LUT.

Note that because the CAM (CIECAM02 in WCS) is involved in the process, chromatic adaptation between the media white point and the PCS white point (mandated by ICC to be that of D50) is effected transparently by the CAM.

## HDR virtual RGB devices

Special consideration must be given when generating profiles for HDR virtual RGB devices; that is, devices for which the colorant values may be less than 0.0 or greater than 1.0. In the generation of the ATOB LUT, a larger set of 1D input LUTs is built. Colorant values are scaled and offset to the range 0 .. 1 using the minimum and maximum colorant values in the WCS profile.

Because the colorant space for HDR devices is not likely to be completely populated, special support is provided in the 3-D LUT for the tag as well. In order to handle colors in the sparsely populated region, the colorants are recoded so that extrapolation beyond 0.0 and 1.0 can be achieved. The range used is -1 .. +4.

Because of the rescaling applied for the 3-D LUT, a set of 1D output LUTs is built to map the result back to the range 0 .. 1.

# More than one PCS

The ICC found that one PCS was not sufficiently flexible to meet all the intended uses of a CMS. In version 4 of the Profile Specification, the ICC clarified that there are actually two PCS encodings. One is used for the colorimetric intents; another is used for the perceptual intent. (No PCS is specified for the Saturation intent. The ICC has left this part ambiguous.) The colorimetric PCS has a minimum and maximum lightness specified, but the chroma and hue values range to roughly  $\pm 127$ . This PCS looks like a rectangular prism. As mentioned previously, the perceptual PCS volume resembles the gamut of an inkjet printer.

The two ICC PCSs also have two different digital encodings. In the perceptual PCS, a value of zero represents a lightness of zero. In the colorimetric PCS, a value of zero represents the minimum lightness of the PCS, which is greater than zero. You can solve this problem by having a different device model for each of the PCS encodings.

## Gamut mapping

To create the AToB LUTs in an ICC profile, you map from the device gamut to the appropriate PCS space. To create the BToA LUTs, you map from the PCS space to the device gamut. The mapping for the AToB LUTs is quite similar to that used in a measurement-based CMS. For the perceptual PCS, map the device plausible gamut to the perceptual PCS gamut boundary, using either clipping or compression for any out-of-gamut colors. For the colorimetric intents, you might have to clip lightness, but the chroma and hue values are all going to fit in the colorimetric PCS gamut.

The mapping for the BToA LUTs is a little different. The colorimetric intents are still easy; you just clip PCS values to the device gamut. But the ICC requires that all possible PCS values map to some device value, not merely those within the reference gamut of the perceptual PCS. So, you must make sure that the GMMs can handle source colors that are outside the reference gamut. This can be handled by clipping those colors to the device gamut boundary.

## Baseline device to ICC profile class mapping

Baseline Device Type	ICC Profile Class	Remark
RGB Capture Device	Input Device ("scnr")	PCS is CIELAB. AToB0Tag is Device to PCS with relative colorimetric intent.

Baseline Device Type	ICC Profile Class	Remark
CRT, LCD monitor	Display Device ("mntr")	PCS is CIEXYZ. See the following for model conversion.
RGB Projector	Color Space ("spac")	PCS is CIELAB.
RGB and CMYK printer	Output Device ("prtr")	PCS is CIELAB.
RGB Virtual Device (non-HDR case)	Display Device ("mntr")	PCS is CIEXYZ.
RGB Virtual Device (HDR case)	Color Space ("spac")	PCS is CIELAB.

The conversion of monitor profiles does not involve building LUTs, but instead consists of building a matrix or TRC model. The model used in ICC is slightly different from the one used in the WCS CRT or LCD modeling in that the "black correction" term is missing. Specifically,

$$\text{WCS model: } \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = M_{WCS} \begin{pmatrix} \rho_{WCS}(R) \\ \gamma_{WCS}(G) \\ \beta_{WCS}(B) \end{pmatrix} + \begin{pmatrix} X_{black} \\ Y_{black} \\ Z_{black} \end{pmatrix}$$

$$\text{ICC model: } \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = M_{ICC} \begin{pmatrix} \rho_{ICC}(R) \\ \gamma_{ICC}(G) \\ \beta_{ICC}(B) \end{pmatrix}$$

The conversion from WCS model to ICC model is done as follows.

Define new curves:

$$\begin{pmatrix} \bar{\rho}_{WCS}(R) \\ \bar{\gamma}_{WCS}(G) \\ \bar{\beta}_{WCS}(B) \end{pmatrix} = \begin{pmatrix} \rho_{WCS}(R) \\ \gamma_{WCS}(G) \\ \beta_{WCS}(B) \end{pmatrix} + M_{WCS}^{-1} \begin{pmatrix} X_{black} \\ Y_{black} \\ Z_{black} \end{pmatrix}$$

These are not tone reproduction curves because they do not map 1 to 1. A normalization will achieve that. The final definitions of the ICC model are:

$$\rho_{ICC}(R) = \bar{\rho}_{WCS}(R) / \bar{\rho}_{WCS}(1)$$

$$\gamma_{ICC}(G) = \bar{\gamma}_{WCS}(G) / \bar{\gamma}_{WCS}(1)$$

$$\beta_{ICC}(B) = \bar{\beta}_{WCS}(B) / \bar{\beta}_{WCS}(1)$$

$$M_{ICC} = M_{WCS} \cdot \begin{pmatrix} \bar{\rho}_{WCS}(1) & 0 & 0 \\ 0 & \bar{\gamma}_{WCS}(1) & 0 \\ 0 & 0 & \bar{\beta}_{WCS}(1) \end{pmatrix}$$

For non-HDR RGB virtual devices, you are also generating a display ICC profile for space efficiency. In that case, the tristimulus matrix  $M_{ICC}$  can be obtained directly from the primaries of the WCS profile without the above model conversion. One final, but important, note is that this tristimulus matrix must be chromatically adapted to D50 to conform with the ICC specification of the PCS. In other words, the entries on each row of the matrix to be encoded in the ICC profile must sum respectively to 96.42, 100, and 82.49. In the current implementation, the chromatic adaptation is done by CAT02, which is also the chromatic adaptation transform used in CAM02.

## Black Preservation and Black Generation

Implementation of black preservation is tied together with the generation of the black channel in devices that support a black channel. To accomplish this, information about each source color is collected to allow device models that support a black channel to determine how best to set the black channel on output. While black preservation is pertinent for color transforms that convert between one black-channel device to another, black generation is implemented for all transform involving a black-channel destination device.

Black channel information is recorded in a data structure called **BlackInformation**. The **BlackInformation** structure contains a boolean indicating whether the color contains only black colorant and a numeric value indicating the degree of "blackness" called black weight. For source devices that support a black channel, the black weight is the percentage of black colorant in the source color. For source devices that do not contain a black channel, the black weight is computed using the other colorants and the appearance value. A value called "color purity" is computed by taking the difference between the maximum colorant value and the minimum colorant value divided by the maximum colorant value. A value called "relative lightness" is computed by taking the difference between the lightness of the color and the minimum lightness for the destination device divided by the difference between the minimum and maximum lightness for the destination device. If the source device is an additive device (monitor or projector), the black weight is determined to be the 1.0 minus the color purity multiplied

by the relative lightness. For example, if the source device is an RGB monitor, the maximum value and minimum value of R, G, and B for each color is computed and the black weight is determined by the formula:

$$BW = (1.0 - (\max(R,G,B) - \min(R,G,B)) / \max(R, G, B)) * \text{relative lightness}$$

If the source device supports subtractive coloration, for example, a CMY printer, then the individual colorants must be "ones complemented" (subtracted from 1.0) before use in the preceding formula. Therefore, for a CMY printer,  $R = 1.0 - C$ ,  $G = 1.0 - M$ , and  $B = 1.0 - Y$ .

The black information for each color processed by the color transform is determined during the color translation process. The black-only information is only determined if black preservation is specified. Black weight is always determined if the destination device model support a black colorant. The black information is passed to the destination device model via the [ColorimetricToDeviceColorsWithBlack](#) method, which uses the resulting LUT.

Note that, because of color transform optimization, the above process occurs only during the creation of the optimized transform LUT, not during the execution of the TranslateColors method.

## Optimization for transforms with more than three source channels

The size of the optimized transform is determined by several factors: the number of color channels in the source device, the number of steps in the table for each source color channel, and the number of color channels in the output device. The formula for determining the transform table size is:

$$\text{Size} = \text{Number of steps per channel}_{\text{source}\backslash \text{device}}(\text{Number}\backslash \text{of}\backslash \text{channels}\backslash \text{in}\backslash \text{source}\backslash \text{device}) \times \text{number of channels in output device}$$

As you can see, the size of the table grows exponentially depending on the number of channels in the source device. Many source devices support three color channels, for example, Red, Green, and Blue. However, if a source device supports four channels, such as CMYK, the size of the table and the time required to construct the table grow by a factor of the number of steps. In a measurement-based CMS where transforms are constructed "on the fly," this time may well be unacceptable.

To reduce the time required to construct the color conversion table, it is possible to take advantage of two facts. First, while the source device may support more than three color

channels, the intermediate device-independent color space (CIECAM02  $J_a \ C_b \ C_g$ ) has only three color channels. Second, the most time-consuming part of the processing is not the device modeling (converting from device color coordinates to tristimulus values), but the gamut mapping. Using these facts, you can construct a preliminary color conversion table that converts colors in the device-independent color space through the gamut mapping steps, and finally, through the output device color model. Construction of this table is of dimension three. Then we construct the dimension four final color-conversion table by converting the source color combinations to intermediate device-independent space, and then, using the preliminary color-conversion table, finish the conversion to the output device color space. So you reduce from computing (number of steps in the lookup table)  $\text{number of channels}$  gamut mapping computations to the number of steps in the intermediate table,  $\text{number of channels}$ , gamut mapping computations. Even though you have to perform number of steps in the (lookup table)  $\text{number of channels}$  computations of device modeling and three-dimensional table lookups, this is still much faster than the original calculation.

The preceding process will work well provided that there is no need for information to pass between the source device model and any other component in the color transform. However, if the output device and the source device both support a black colorant, and the source black colorant is used in determining the output black colorant, the process will fail to properly communicate the source black information. An alternative process is to construct a preliminary color conversion table that converts colors in the device-independent color space through the gamut mapping steps only. Then construct the dimension four final color-conversion table using the following steps: a) convert the source color combinations to intermediate device-independent space, b) perform the gamut mapping steps by interpolating in the preliminary color table instead of applying the actual gamut mapping processes, and c) use the resulting values from the gamut mapping steps and any source black channel information to compute the output device colorants using the output device model. This process also can be used when there is information transferred between the source and output device models even if there is no black channel; for example, if the two modules are implemented with a plug-in architecture that allows for data interchange between modules.

The preceding two processes can be used to effectively improve the time required to construct the four-dimensional color transform table.

## CheckGamut

The ICM calls `CreateTransform` and `CreateMultiProfileTransform` take a word of flag values, one of which is `ENABLE_GAMUT_CHECKING`. When this flag is set, CITE must create the transform differently. The initial steps are the same: the source and

destination CAMs must be initialized, then the source and destination gamut boundary descriptors must be initialized. Regardless of the specified intent, the CheckGamut GMM must be used. The CheckGamut GMM should be initialized using the source and destination device models and gamut boundary descriptors. However, the transform should then create a truncated transform comprising the source device model, the source CAM, any intervening GMMs, and the CheckGamut GMM. This assures that the delta J, delta C, and delta h values output by the CheckGamut CMM become the final resulting values.

The meaning of CheckGamut is clear when there are only two device profiles in the transform. When there are more than two device profiles and more than two GMMs, then CheckGamut reports whether the colors which have been transformed through the first device model and all but the last GMM fall within the gamut of the destination device.

## Related topics

[Basic color management concepts](#)

[Windows Color System Schemas and Algorithms](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# About Windows Color System Version 1.0

Article • 12/30/2021

Microsoft Windows Color System (WCS) technology ensures that a color image, graphic or text object is rendered as close as possible to its original intent on any device, despite differences in imaging technologies and color capabilities between devices. Whether you are scanning an image or other graphic on a color scanner, downloading it over the Internet, viewing or editing it on the screen, or outputting it to paper, film, or other media, WCS 1.0 helps you keep its colors consistent and accurate.

WCS 1.0 is an integral part of Microsoft Windows. Because WCS 1.0 is built into the Windows family of operating systems as a set of Win32 API functions, it is readily available to any application, device driver, device calibration tool, or color management module (CMM).

Version 1.0 of Image Color Management (ICM) was delivered in Microsoft Windows 95, and provides basic color management capabilities within Windows device contexts.

ICM version 2.0 is included in Windows 98, Windows Millennium Edition, Windows 2000, and Windows XP and included functions that implemented color management outside of device contexts. These functions were suitable for more demanding color management requirements, and give applications greater control over the color-management process.

WCS version 1.0 is included in Windows Vista and includes a variety of new functions that provides significant improvements in flexibility, transparency, predictability and extensibility for vendors.

What kinds of applications will benefit from using WCS 1.0? Almost any application running in a Windows Vista environment today. Basic color management capability is becoming a requirement for applications of all kinds.

Color display and printing have improved in recent years to the point where photo-realistic color images and complex color graphics are now commonly used not only in desktop publishing, but also across a broad spectrum of data and presentation applications. Object technologies such as COM and ActiveX have embedded color objects in virtually every kind of data space.

Clothing catalogs, online art, family photo albums, business logos, charts, graphs, presentations, print previews, and color simulations are just a few examples of everyday

applications where color matters.

As a result, more and more users are requiring their applications to be capable of accurate color display. Poor color rendering ruins the images and graphics that are increasingly important to users.

On a fundamental level, almost any application should be able to adjust color automatically so that its output looks the same on different monitors and printers. WCS 1.0 provides a set of functions to deliver this kind of color management that is transparent to a user and requires little overhead in the application.

On a higher level, WCS 1.0 provides additional functions that deliver more complex and controllable color management. Graphics and desktop publishing applications need these additional capabilities to let their users work precisely with consistent color across many devices throughout a production process.

In general, software vendors whose products deal with color input or output and hardware vendors of color peripherals will find WCS 1.0 a key technology for simplifying the delivery of successful products. The following sections include:

- [What's New in Version 1.0 of WCS](#)
- [WCS 1.0 Availability](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# What's New in Windows Vista

Article • 10/12/2022

Version 1.0 of Image Color Management (ICM) was delivered in Microsoft Windows 95, and provides basic color management capabilities within Windows device contexts.

ICM version 2.0 was delivered in Windows 98, Windows Millennium Edition, Windows 2000, and WindowsXP and included a variety of new functions that implemented color management outside of device contexts. These new functions were suitable for more demanding color management requirements, and gave applications greater control over the color-management process.

With the release of Windows Vista, ICM 2.0 is now included in Windows Color System (WCS) 1.0, which adds more functionality. The following table lists new application programming interfaces (API) that ship in Windows Vista.

## New API Shipping in Windows Vista

Enumerations

API Name

Header

Library

### [COLORDATATYPE](#)

icm.h

mscms.lib

### [COLORPROFILESUBTYPE](#)

icm.h

mscms.lib

### [COLORPROFILETYPE](#)

icm.h

mscms.lib

### [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#)

icm.h

mscms.lib

Functions

API Name

Header

Library

### [\*\*WcsAssociateColorProfileWithDevice\*\*](#)

icm.h

mscms.lib

### [\*\*WcsCheckColors\*\*](#)

icm.h

mscms.lib

### [\*\*WcsCreateIccProfile\*\*](#)

icm.h

mscms.lib

### [\*\*WcsDisassociateColorProfileFromDevice\*\*](#)

icm.h

mscms.lib

### [\*\*WcsEnumColorProfiles\*\*](#)

icm.h

mscms.lib

### [\*\*WcsEnumColorProfilesSize\*\*](#)

icm.h

mscms.lib

**WcsGetDefaultColorProfile**

icm.h

mscms.lib

**WcsGetDefaultColorProfileSize**

icm.h

mscms.lib

**WcsGetDefaultRenderingIntent**

icm.h

mscms.lib

**WcsGetUsePerUserProfiles**

icm.h

mscms.lib

**WcsOpenColorProfileW**

icm.h

mscms.lib

**WcsSetDefaultColorProfile**

icm.h

mscms.lib

**WcsSetDefaultRenderingIntent**

icm.h

mscms.lib

**WcsSetUsePerUserProfiles**

icm.h

mscms.lib

**WcsTranslateColors**

icm.h

mscms.lib

## Interfaces and Their Functions

API Name

Header

Library

### **[IDeviceModelPlugin](#)**

WcsPlugIn.h

N/A

#### **[IDeviceModelPlugin::ColorimetricToDeviceColors](#)**

WcsPlugIn.h

N/A

#### **[IDeviceModelPlugin::ColorimetricToDeviceColorsWithBlack](#)**

WcsPlugIn.h

N/A

#### **[IDeviceModelPlugin::DeviceToColorimetricColors](#)**

WcsPlugIn.h

N/A

#### **[IDeviceModelPlugin::GetGamutBoundaryMesh](#)**

WcsPlugIn.h

N/A

#### **[IDeviceModelPlugin::GetGamutBoundaryMeshSize](#)**

WcsPlugIn.h

N/A

**IDeviceModelPlugin::GetNeutralAxis**

WcsPlugIn.h

N/A

**IDeviceModelPlugin::GetNeutralAxisSize**

WcsPlugIn.h

N/A

**IDeviceModelPlugin::GetNumChannels**

WcsPlugIn.h

N/A

**IDeviceModelPlugin::GetPrimarySamples**

WcsPlugIn.h

N/A

**IDeviceModelPlugin::Initialize**

WcsPlugIn.h

N/A

**IDeviceModelPlugin::SetTransformDeviceModelInfo**

WcsPlugIn.h

N/A

**IGamutMapModelPlugin**

WcsPlugIn.h

N/A

**IGamutMapModelPlugin::Initialize**

WcsPlugIn.h

N/A

**IGamutMapModelPlugin::SourceToDestinationAppearanceColors**

WcsPlugIn.h

N/A

Structures

API Name

Header

Library

### **BlackInformation**

WcsPlugIn.h

N/A

### **GamutBoundaryDescription**

WcsPlugIn.h

N/A

### **XYZColorF** ↗

WcsPlugIn.h

N/A

### **JChColorF** ↗

WcsPlugIn.h

N/A

### **JabColorF** ↗

WcsPlugIn.h

N/A

### **GamutShell**

WcsPlugIn.h

N/A

## GamutShellTriangle

WcsPlugIn.h

N/A

## PrimaryJabColors

WcsPlugIn.h

N/A

## PrimaryXYZColors

WcsPlugIn.h

N/A

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# What's New in Version 1.0 of WCS

Article • 12/30/2021

Version 1.0 of Image Color Management (ICM) was delivered in Microsoft Windows 95, and provides basic color management capabilities within Windows device contexts.

ICM version 2.0 is included in Windows 98, Windows Millennium Edition, Windows 2000, and Windows XP and includes a variety of new functions that implement color management outside of device contexts. These new functions are suitable for more demanding color management requirements, and give applications greater control over the color-management process.

Additional functionality is included in the Windows Vista release of Microsoft Windows.

- choice of static/monolithic and dynamic/modular CMM/Profile workflow
  - static provided by HeidelbergerDruckmaschinen in Win98 ICM2, dynamic provided by Canon Inc. using Canon's Kyuanos technology
  - baseline device, color appearance, and gamut mapping model profiles,
  - black generation support,
  - black preservation support,
  - High dynamic range support,
  - floating point/greater than 16 bits per channel precision,
  - default wide gamut working space profile support,
  - flexibility to control objective intra-device characteristics separate from inter-device characteristics,
  - ability to control single gamut mapping algorithm between two devices,
  - support for mapping source device primaries to destination primaries in a color managed workflow,
- choice of binary and text color profile formats
  - binary conforming to ICC TIFF-tagged-based profile format and text using XML schemas
- ICC Version 4 support,
- central color management control panel,
- legacy compatibility maintained for applications and devices,
- extensibility via 3rd party plug-in device models and gamut mapping models,
- support for advanced device/gamut mapping by enabling direct communication between DM and GMM plugins,
- static (ICM2) CMM now defaults to Relative Colorimetric intent instead of Absolute Colorimetric intent.

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WCS 1.0 Availability

Article • 12/30/2021

ICM1 is available for Microsoft Windows 95. It is not available for MS DOS, Windows 3.x, or Windows NT 4.0 or earlier.

ICM2 is available for Microsoft Windows 98, Windows Millennium Edition (Me), Windows 2000, and Windows XP. It is not available for MS DOS, Windows 3.x, Windows 95, or Windows NT 4.0 or earlier.

WCS 1.0 is available for Microsoft Windows Vista. It is not available for Windows XP or earlier.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Using WCS 1.0

Article • 12/30/2021

This section presents an overview of how WCS works, and how it can be used to create applications that utilize color management. It contains the following topics:

- [Using Device Profiles with WCS](#)
- [Using Color Management Modules \(CMM\)](#)
- [Rendering Intents](#)
- [Using The Color Mapping Process with WCS](#)
- [Using Structures in WCS](#)
- [Using Color Management on the Internet](#)
- [Using GDI Functions with WCS](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Using Device Profiles with WCS

Article • 12/30/2021

Device profiles are a basic tool for color management. A [device profile](#) is a file that contains information about how to convert colors in the color space and the color [gamut](#) of a specific device into a device-independent color space. The device-independent color space that ICM2 uses is called a Profile Connection Space (PCS). A device profile also contains information about how to convert colors from the PCS into the color space and color gamut of a specific device.

These two sets of conversion information are used for [color conversion](#) and color management. For instance, an image may be created in the color space and color gamut of a video display. The information in the device profile files can be used to display a representation of a printed image. Users often want to see on the screen how colors will change when an image is printed. This is called proofing. An image can be proofed by converting the colors from the source color gamut (the screen) into [PCS](#) colors, and then converting them from the PCS into the destination color gamut (the printer). The resulting image can be displayed on the screen, allowing the user to see what the final image will look like when it is printed.

Device profiles allow more complex uses as well. For example, they can be used to get an idea of what an image created on a video display would look like when printed on a high resolution laser printer. The example gets more complex if there is only a standard inkjet printer on which to proof it. ICM2 will convert the image from the [gamut](#) of the display, into the gamut of the inkjet printer. From there it is converted into the gamut of the laser printer. The resulting image can be printed on the inkjet printer. Of course the image would be at a higher resolution when printed on the color laser printer. However, the colors of the proofing image printed on the inkjet printer would be a close match to the colors that the laser printer would print.

The conversions from a device profile can be concatenated together into a single file, called a *device link profile*. If a series of conversions is used repeatedly, creating a device link profile for them will shorten conversion time.

Device profiles themselves can be managed. Profile management is the process of associating color profiles with device instances. Any color output device can have a set of one or more color profiles associated with it. Hardware manufacturers and third parties who provide color profiles need to perform profile management.

Profile sets can be shared between devices, or can be dedicated to specific devices. For example, suppose that you have two color printers of the same model. Each printer

would require a set of color profiles to be associated with it. The set of color profiles for a printer corresponds to the various configurations possible in that model. Being of the same model, both printers could share one set of profiles. The alternative is to give each printer its own distinct profile set. In the latter case, the color profiles in the set can be calibrated precisely to the printer's individual output, not just the general output characteristics of the model.

There are three different classes of ICC profiles: input profiles, display profiles, and output profiles. Input profiles are usually associated with a device, such as a scanner. Display profiles are typically associated with a computer monitor. Output profiles are commonly associated with printers.

The specification also allows for device profiles, abstract profiles, device link profiles, named color profiles, and color space profiles.

A device profile describes the color space of a particular device.

An abstract profile provides a generic method for users to make subjective color changes to images or graphic objects by transforming the color data within the PCS.

As mentioned previously, a device link profile concatenates a series of profiles into one transformation.

Named color profiles can be thought of as sibling profiles to device profiles. For a given device there would be one or more device profiles to handle process color conversions and one or more named color profiles to handle named colors. There might be multiple named color profiles to account for different consumables or multiple named color vendors.

A color space profile describes a device-independent color space.

The following table summarizes the various types of profiles.

<b>Profile Type</b>	<b>Description</b>
Abstract Profile	A profile that has been adjusted to suit a user's particular preferences.
Color Space Profile	A profile that describes a device-independent color space.
Device Link Profile	A series of profiles that are concatenated together into a single profile.
Device Profile	A profile that describes the color space of a particular device.

Profile Type	Description
Display Profile	A class or category of profiles that includes any type of profile associated with a display.
Input Profile	A class or category of profiles that includes any type of profile associated with an input device such as a scanner.
Named Color Profile	A profile for a color space that consists of named colors.
Output Profiles	A class or category of profiles that includes any type of profile associated with a hardcopy output device such as a printer.

Color transforms can be created with one or more profiles. If a transform is created with only one profile, the profile must be a device link profile. A transform may also have two or more profiles in a chain. If it does, it can contain no device link profiles. Abstract profiles can only be in the middle of the chain. The first and last profiles must be device profiles or color space profiles.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Using Color Management Modules (CMM)

Article • 12/30/2021

Color Management Modules (CMMs) are modules of WCS code that use the information in device profiles to perform color conversion and color mapping. Application developers should not have to implement CMMs. Microsoft provides the default CMM. However, if you do write software that requires the use of specialized color conversion and color mapping algorithms, you may create one.

## ⓘ Note

CMM entry points are *not* API functions and should not be called by applications.

When CMMs are installed, the installation program registers them in the Windows registry. Applications can enumerate the registered CMMs and select one using the [SelectCMM](#) function. The following sample application demonstrates how to enumerate all registered CMMs.

C++

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <mbstring.h>
#include <windows.h>
#include <icm.h>

#ifdef WINDOWS_98
TCHAR gszICMatcher[] = __TEXT(
    "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\ICM\\ICMatchers");
#else
TCHAR gszICMatcher[] = __TEXT(
    "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\ICM\\ICMatchers");
#endif

_CRTAPI1 main (int argc, char *argv[])
{
    DWORD dwNumCMM = 0;

    HANDLE hkCMM;
    DWORD dwErr = RegCreateKey(HKEY_LOCAL_MACHINE,
```

```

        gszICMatcher, &hkCMM);
DWORD dwMaxName, dw.MaxValue;
DWORD dwInfoErr = RegQueryInfoKey(&hkCMM, NULL, NULL,
                                   NULL, NULL, NULL, NULL, NULL,
                                   &dwMaxName, &dw.MaxValue,
                                   NULL, NULL);
TCHAR chCMM[dwMaxName];
ULONG cjCMM = sizeof(chCMM)/sizeof(chCMM[0]);
DWORD dwType;
TCHAR chCMMFile[dw.MaxValue];
ULONG cjCMMFile = sizeof(chCMMFile)/sizeof(chCMMFile[0]);

if (dwErr != ERROR_SUCCESS)
{
    printf("Could not open ICMatcher registry key: %d\n", dwErr);
}

if (dwErr == ERROR_SUCCESS)
{
    while (RegEnumValue(
        hkCMM,dwNumCMM,chCMM,
        &cjCMM,NULL,&dwType,
        chCMMFile,&cjCMMFile) == ERROR_SUCCESS)
    {
        if (dwType == REG_SZ)
        {
            printf("%d: %-80s - %-80s\n",dwNumCMM,chCMM,chCMMFile);
        }
        else
        {
            printf("%d: error\n");
        }

        dwNumCMM++;
        cjCMM = sizeof(chCMM);
        cjCMMFile = sizeof(chCMMFile);
    }
}

RegCloseKey(hkCMM);
}

```

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Rendering Intents

Article • 12/30/2021

The International Color Consortium (ICC) has defined four different values called *rendering intents*. These represent four different approaches to creating a color rendering. These four intents, and the constants used to refer to them in code are as follows.

Intent	ICC Name	Description
Picture	Perceptual	INTENT_PERCEPTUAL
Graphic	Saturation	INTENT_SATURATION
Proof	Relative Colorimetric	INTENT_RELATIVE_COLORIMETRIC
Match	Absolute Colorimetric	INTENT_ABSOLUTE_COLORIMETRIC

The ICC Profile Format Specification Version 3.4, which describes these intents, can be downloaded from [color.org](http://color.org).

## Picture Intent

Called perceptual intent in the ICC specification clause 4.9, a Picture intent causes the full [gamut](#) of the image to be compressed or expanded to fill the gamut of the destination device, so that gray balance is preserved but colorimetric accuracy may not be preserved.

In other words, if certain colors in an image fall outside of the range of colors that the output device can render, the picture intent will cause all the colors in the image to be adjusted so that every color in the image falls within the range that can be rendered and so that the relationship between colors is preserved as much as possible.

This intent is most suitable for display of photographs and images, and is generally the default intent.

## Graphic Intent

The ICC specification clause 4.12 calls the Graphic intent a [saturation](#) intent. It preserves the chroma of colors in the image at the possible expense of [hue](#) and [lightness](#).

Implementation of this intent remains somewhat problematic, and the ICC is still working on methods to achieve the desired effects.

This intent is most suitable for business graphics such as charts, where it is more important that the colors be vivid and contrast well with each other rather than a specific color.

## Proof Intent

The Proof intent, called the colorimetric intent in the ICC specification, is defined such that any colors that fall outside the range that the output device can render are adjusted to the closest color that can be rendered, while all other colors are left unchanged.

Proof intent does not preserve the [white point](#).

For example, the whitest white of a paper is more yellow than the whitest white of a computer monitor. An image converted into the gamut of the printer using relative colorimetric intent would result in all colors becoming more yellow. The white point of the image is moved to match the white point of the printer. All other colors in the image keep their position relative to the white point. This produces an image that more accurately reflects what the printed image will look like. However, the user may find it visually disconcerting.

## Match Intent

In a Match intent, any colors that fall outside the range that the output device can render are adjusted to the closest color that can be rendered, while all other colors are left unchanged. The ICC specification calls the match intent absolute colorimetric intent.

Match intent preserves the white point.

For example, the whitest white of a paper is more yellow than the whitest white of a computer monitor. An image converted into the [gamut](#) of the printer using match intent would result in all colors being converted and matched into the gamut of the printer. The white point of the image is not moved to match the white point of the printer. Therefore, the distance of the colors to the white point may change. This produces an image that is less visually disconcerting to the user, but is also a less accurate rendition of printer output.

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Using The Color Mapping Process with WCS

Article • 12/30/2021

WCS color mapping is based on [device profiles](#). These are supplied by vendors of color hardware devices and installed when a device is installed. When color mapping is used by an application program, WCS accesses the device profile of the image to get the information needed to convert the image to the PCS. The conversion is done by the CMM.

A device profile can be embedded into the image itself. So the device profile travels with the image, even across the Internet. A user does not need the source device to get an accurate color mapping. If an image does not have a device profile, the sRGB space is used as a default. For more details, see [Using Color Management on the Internet](#).

Note that applications which use WCS should never embed the sRGB profile into an image. The sRGB color space provides a standardized color space that is portable across all devices. It is automatically available to users of Windows 98 and later as well as Windows 2000 and later. Therefore, it does not need to travel with the image.

After the image colors are in the [PCS](#), WCS accesses the device profile of the destination device. It gets the CMM to convert the image colors from PCS to the gamut of the destination device.

More complex color mapping can also be done with WCS. For example, it can be used to get an idea of what an image created on a video display would look like when printed on a high resolution laser printer. The example gets more complex if there is only a standard inkjet printer on which to preview it. The image can be converted from the gamut of the display, into the gamut of the inkjet printer. From there it can be converted into the gamut of the laser printer. The resulting image can be printed on the inkjet printer. Of course the image would be at a higher resolution when printed on the color laser printer. However, the colors of the proofing image printed on the inkjet printer would be a close match to the colors that the laser printer would print.

The way the conversions in the example are accomplished is to convert the image colors from the display's gamut into the PCS. After the image colors are converted into the PCS, the device profile of the inkjet printer would be used to transform them into the gamut of the inkjet printer. Next, the gamut to PCS transform would be used to move the colors back to the PCS. From there, the laser printer's device profile is used to convert the colors from PCS into the gamut of the laser printer.

The ability to easily transform colors from a device gamut to the PCS and back again enables image colors intended for one color output device to be proofed on almost any other.

In the preceding example, the description varies from the actual procedure somewhat for clarity. In reality, all of the transformations mentioned in the preceding paragraph would be concatenated into a single transformation.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Using Structures in WCS 1.0

Article • 06/07/2022

Most of the structures used by WCS 1.0 are very straightforward and require little explanation. They are documented in the WCS 1.0 Reference section entitled [Structures](#).

Exceptions are the [COLORMATCHSETUPW](#) structure used by the [SetupColorMatchingW](#) function, and the following Windows structures defined in Wingdi.h:

- [BITMAPV5HEADER](#)
- [LOGCOLORSPACE](#)
- [CIEXYZ](#)
- [CIEXYZTRIPLE](#)

The following topics are discussed at greater length:

- [Windows Bitmap Header Structures](#)
- [Differences Between V4 and V5 Headers](#)
- [Bitmap.exe: a Command-Line Utility for Converting Bitmap Headers](#)

## Windows Bitmap Header Structures

WCS 1.0 allows ICC color profiles to be linked or embedded in device-independent bitmaps (DIBs). This allows DIB colors to be characterized more accurately than was possible using WCS in Windows 95. [BITMAPV5HEADER](#), the new bitmap header structure, is defined in Wingdi.h in the release of Windows 98. For development purposes, it is also included in the file Icm.h with this Programmer's Reference. The [BITMAPV5HEADER](#) structure is as follows:

C++

```
typedef struct {
    DWORD      bV5Size;
    LONG       bV5Width;
    LONG       bV5Height;
    WORD       bV5Planes;
    WORD       bV5BitCount;
    DWORD      bV5Compression;
    DWORD      bV5SizeImage;
    LONG       bV5XPelsPerMeter;
    LONG       bV5YPelsPerMeter;
    DWORD      bV5ClrUsed;
    DWORD      bV5ClrImportant;
```

```

    DWORD      bV5RedMask;
    DWORD      bV5GreenMask;
    DWORD      bV5BlueMask;
    DWORD      bV5AlphaMask;
    DWORD      bV5CSType;
    CIEXYZTRIPLE bV5Endpoints;
    DWORD      bV5GammaRed;
    DWORD      bV5GammaGreen;
    DWORD      bV5GammaBlue;
    DWORD      bV5Intent;           // Rendering intent for bitmap
    DWORD      bV5ProfileData;     // Offset to profile data
    DWORD      bV5ProfileSize;     // Size of embedded profile data
    DWORD      bV5Reserved;        // Should be zero
} BITMAPV5HEADER, FAR *LPBITMAPV5HEADER, *PBITMAPV5HEADER;

```

The member **bV5CSType** can have the values PROFILE\_EMBEDDED or PROFILE\_LINKED to specify whether a profile is embedded or linked with the DIB. The member **bV5ProfileData** is the offset in bytes from the beginning of the **BITMAPV5HEADER** structure to the start of the profile data. If the profile is embedded, profile data is the actual profile, and if it is linked, the profile data is the null-terminated file name of the profile. This cannot be a Unicode string. It must be composed exclusively of characters from the Windows character set (code page 1252).

When a DIB is loaded into memory, the profile data (if present) should follow the color table, and **bV5ProfileData** should give the offset of the profile data from the beginning of the **BITMAPV5HEADER** structure. The value of this member will be different now, as the bitmap bits do not follow the color table in memory. Applications should modify the **bV5ProfileData** member after loading the DIB into memory.

For packed DIBs, the profile data should follow the bitmap bits similar to the file format. The **bV5ProfileData** member should still give the offset of the profile data from the beginning of the **BITMAPV5HEADER** structure.

Applications should access the profile data only when **bV5Size == sizeof (BITMAPV5HEADER)** AND **bV5CSType** is PROFILE\_EMBEDDED or PROFILE\_LINKED.

If a profile is linked, the path of the profile can be any fully qualified name (including a network path) that can be opened using the Win32 **CreateFile** function.

## Differences Between V4 and V5 Headers

In working with the new bitmap structure, it is useful to recognize differences in how **BITMAPV4HEADER** and **BITMAPV5HEADER** structures are set up:

V4 Header	Meaning
-----------	---------

V4 Header	Meaning
bV4CSType	LCS_CALIBRATED_RGB. This value implies that end points and gammas are given in the appropriate fields. Bogus values cause trouble.
bV4CSType	LCS_sRGB. This value implies that the bitmap is in sRGB color space (gammas and endpoints ignored).
bV4CSType	LCS_WINDOWS_COLOR_SPACE. This value implies that the bitmap is in Windows default color space.

V5 Header	Meaning
bV5CSType	LCS_CALIBRATED_RGB. This value implies that end points and gammas are given in the appropriate fields. Bogus values cause trouble.
bV5CSType	LCS_sRGB. This value implies that the bitmap is in sRGB color space (gammas and endpoints ignored).
bV5CSType	PROFILE_EMBEDDED. This value implies that <b>bV5ProfileData</b> points to a memory buffer that contains the profile to use (gammas and endpoints are ignored).
bV5CSType	PROFILE_LINKED. This value implies that <b>bV5ProfileData</b> points to the file name of the profile to use (gammas and endpoints are ignored).
bV5CSType	LCS_WINDOWS_COLOR_SPACE. This value implies that the bitmap is in Windows default color space.

In order to convert older bitmaps to and from the new **BITMAPV5HEADER** structure, a command-line conversion utility file named **Bitmap.exe** is included in the WCS 1.0 Programmer's Reference.

## BitMap.exe: a Command-Line Utility for Converting Bitmap Headers

**Bitmap.exe** is a command-line utility located in the \Bin folder under the installation folder that you specified. It modifies the headers of Windows bitmaps, allowing you to convert existing bitmaps from **BITMAPINFOHEADER** and **BITMAPV4HEADER** header structures to the newer **BITMAPV5HEADER** structure and back again. The command-line syntax is as follows:

C++

```
BITMAP [/d] [/1|4|5] [/s] [/f]
filename
```

The command-line switches have the following effects.

Switch	Meaning
/d	Default values are automatically entered in the converted headers.
/1	Convert the specified bitmaps to <b>BITMAPINFOHEADER</b>
/4	Convert the specified bitmaps to <b>BITMAPV4HEADER</b>
/5	Convert the specified bitmaps to <b>BITMAPV5HEADER</b>
/f	Forces conversion, even if the bitmap already has the right header
/s	Converts bitmaps in the specified folder and all subdirectories under it

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Using Color Management on the Internet

Article • 12/30/2021

Color images and graphics are an increasingly important way of communicating information for users of the Internet. Color conversion information can be embedded in some file formats. The embedding techniques currently used are not compact. Internet bandwidth considerations often make them impractical. Internet considerations include the following topics:

- sRGB: A Standard Color Space
- WCS 1.0 Defaults for Input Color Space and Output Profile
- sRGB and Embedded Profiles

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# sRGB: A Standard Color Space

Article • 03/15/2023

As a result of Internet bandwidth considerations, Hewlett-Packard and Microsoft have proposed the adoption of a standard predefined color space known as *sRGB* (IEC 61966-2-1), so as to allow accurate color mapping with very little data overhead.

A help-file version of a white paper discussing the technical details of sRGB, sRGB.hlp, is available in the \Help folder of the WCS 1.0 Programmer's Reference.

Different file formats may use or add a flag to specify that the image is in sRGB color space. In the Windows device-independent bitmap (DIB) format, setting the **bV5CSType** member of the **BITMAPV5HEADER** structure to **LCS\_sRGB** specifies that the DIB colors are in the sRGB color space.

WCS 1.0 provides native support for sRGB. There are two ways to use WCS 1.0 for rendering an image defined in the sRGB color space:

## To render an image inside the device context

1. Create a device context (DC) on the display device.
2. Set color management using the **SetICMMode** function.
3. Use the **SetDIBitsToDevice** function to transfer the DIB into the DC. As long as the **bV5CSMType** member of the DIBs **BITMAPV5HEADER** structure is set to **LCS\_sRGB**, the system will perform the appropriate color management.

## To render an image outside the device context

1. Create a transform using **CreateColorTransformW**. The **IcsCSType** member of the **LOGCOLORSPACE** structure pointed to by the *pLogColorSpace* parameter should be set to **LCS\_sRGB**. The *hDestProfile* parameter indicates the display device's color space.
2. Use the created color transform to color match the image before displaying it on the device.

## WCS 1.0 Defaults for Input Color Space and Output Profile

When no input color space is specified, by default WCS 1.0 uses the sRGB color space as the input color space for color mapping.

When no output profile is specified, but a default device is specified, WCS 1.0 selects a default output profile. If the default device does not have an associated profile, WCS 1.0 uses the sRGB color space as the output profile.

The following table shows the resulting color transforms when a default device is not available.

	<b>Output Profile Specified</b>	<b>Output Profile Not Specified</b>
Input Color Space Specified	Transform uses the specified profiles.	Transform converts from known input color space to sRGB.
Input Color Space Not Specified	Transform converts from sRGB to known output profile.	Transform from sRGB to sRGB is assumed; nothing is done.

## sRGB and Embedded Profiles

Beginning with ICM version 2.0, applications that utilize WCS can embed profiles in images. Embedded profiles assist users' applications in maintaining a consistent color appearance even if images are transmitted across the Internet.

Images that use the sRGB color space do not need an embedded color profile. Because they have no embedded profile, sRGB-based images are smaller and more readily transferable across data channels with limited bandwidth.

Applications should set the **LCS\_sRGB** flag in the image's bitmap header to indicate that the image uses the sRGB color space. For details, see [Windows Bitmap Header Structures](#) and [LOGCOLORSPACE](#).

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Using GDI Functions With WCS

Article • 12/30/2021

There are various functions in the graphics device interface (GDI) that use or operate on color data. Some are enabled for use with WCS and some are not. The following GDI functions are relevant to ICM:

- [Device Context Functions with WCS](#)
- [Pen And Brush Functions with WCS](#)
- [Text Output Functions with WCS](#)
- [Bitmap Functions with WCS](#)

## Device Context Functions with WCS

Function	Description
CreateCompatibleDC	If the device context (DC) that is passed to this function through its hdc parameter is enabled for ICM, then the DC the function creates is also ICM-enabled. The source and destination color spaces are specified in the DC.
CreateDC	ICM can be enabled by setting the dmICMMETHOD member of the DEVMODE structure pointed to by the pInitData parameter to the appropriate value. For details, see the documentation in the Platform SDK on the DEVMODE structure.
ResetDC	The color profile of the device context specified by the hdc parameter will be reset based on the information in the DEVMODE structure specified by the lpInitData parameter.

## Pen And Brush Functions with WCS

Function	Description
Brush Functions	No color management is done at brush creation. However, color management will be performed when the brush is selected into an ICM-enabled DC.
CreatePen	No color management is done at pen creation. However, color management will be performed when the brush is selected into an ICM-enabled DC.
ExtCreatePen	No color management is done at pen creation. However, color management will be performed when the brush is selected into an ICM-enabled DC.

<b>Function</b>	<b>Description</b>
SelectObject	If the object being selected is a brush or a pen, color management is performed.
SetDCBrushColor	Color management is performed if WCS is enabled.
SetDCPenColor	Color management is performed if WCS is enabled.

## Text Output Functions with WCS

<b>Function</b>	<b>Description</b>
SetBkColor	Color management is performed if WCS is enabled.
SetTextColor	Color management is performed if WCS is enabled.

## Bitmap Functions with WCS

<b>Function</b>	<b>Description</b>
BitBlt	No color management is performed when blits occur.
CreateDIBitmap	The fuUsage parameter specifies that the bmiColors member of the BITMAPINFO structure pointed at by the lpbmi parameter does or does not contain color information. If it does not, no color management is performed for this bitmap. The bitmap must use version 4 or version 5 of the BITMAPINFO structure for color management to be enabled. The contents of the resulting bitmap are not color matched after the bitmap has been created.
CreateDIBSection	If the BITMAPINFO structure passed through the pbmi parameter is not version 4 or version 5, no color management is done. If it is version 4 or 5, color management is enabled, and the specified color space is associated with the bitmap.
MaskBlt	No color management is performed when blits occur.
SelectObject	If the object is a bitmap created with CreateDIBSection, color management is performed. The bitmap's color space is used as the destination color space.

Function	Description
SetDIBits	Color management is performed. If the specified BITMAPINFO structure is not version 4 or version 5, the color profile of the current DC is used as the source color space profile. If it does not have one, the sRGB space is used. If the specified BITMAPINFO structure is version 4 or version 5, the color space profile specified in the bitmap header is used as the source color space profile.
SetDIBitsToDevice	Color management is performed. If the specified BITMAPINFO structure is not version 4 or version 5, the color profile of the current device context is used as the source color space profile. If it doesn't have one, the sRGB color space is used. If the specified BITMAPINFO structure is version 4 or version 5, the color space profile associated with the bitmap is used as the source color space.
SetDIBColorTable	No color management is performed.
StretchBlt	No color management is performed when blits occur.
StretchDIBits	Color management is performed. If the specified BITMAPINFO structure is not version 4 or version 5, the color profile of the current DC is used as the source color space profile. If it does not have one, the sRGB space is used. If the specified BITMAPINFO structure is version 4 or version 5, the color space profile specified in the bitmap header is used as the source color space profile.

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ICC profile behavior with Advanced Color

Article • 10/12/2022

Advanced Color is an umbrella term of OS technologies for displays with significantly higher color fidelity than standard displays. For more information, refer to [Use DirectX with Advanced Color on high/standard dynamic range displays](#). Advanced Color and auto color management ensure consistent and colorimetrically accurate display color for all apps: both legacy and modern. However, your app might already perform its own explicit color management using International Color Consortium (ICC) color profiles.

When Advanced Color is active on either SDR or HDR displays, the behavior of display ICC profiles changes in non-backwards compatible ways. If your app works with display ICC profiles, then Windows offers compatibility helpers to ensure that your app continues to get correct behavior. Advanced Color-aware apps should transition away from directly interacting with display ICC profiles since Windows provides replacement app-facing APIs that abstract away the profile; full guidance is available at [Use DirectX with Advanced Color on high/standard dynamic range displays](#).

This topic describes the changes in ICC profile behavior. In addition, if your color-managed app needs to continue using display ICC profiles, then this topic will show how to adapt your app to incrementally leverage Advanced Color benefits.

## Legacy Windows color management behavior

When Advanced Color is inactive, Windows doesn't perform any color management on your app's visual content output (for example, GDI hDC, DirectX swap chain, or composition visual); in practice, it assumes that your app content is in the standard sRGB color space. If you want accurate color reproduction on the active display, then your app must perform its own color management, most often using International Color Consortium (ICC) color profiles. The main conceptual steps are:

1. Get the display's color characteristics.
2. Perform color space conversion to the display's color space.
3. Perform gamut mapping in order to constrain to the display's gamut.

Here are more details on each of the three steps.

### Get the display's color characteristics

A Win32 app uses the [Windows Color System profile management functions](#) to obtain the default ICC profile, which tells you the display's color characteristics, including its available color gamut.

A Universal Windows Platform app uses [DisplayInformation.GetColorProfileAsync Method](#) instead.

## Perform color space conversion to the display's color space

If the display's color space doesn't match your content's color space, then you must do a color space conversion. For example, digital content is often encoded as sRGB, but your display might be wide gamut DCI-P3. You'd typically use an ICC color management library that reads the ICC profile, and transforms your content's color values to match. Windows provides multiple ICC color management engines; for example, the [Direct2D color management effect](#).

It's important to note that ICC profile-based color management is *display-referred* or *output-referred*. That means that color values are not stored as absolute (*scene-referred*) colors, but instead are encoded relative to the color space of the display (the output device). For example, if your app is rendering sRGB red, then that's represented as `RGB(1, 0, 0)` in your rendered output. But if you're rendering that content on an Adobe RGB display, then `RGB(1, 0, 0)` is simply interpreted by the display as its most saturated red (Adobe RGB red), which is incorrect. When you apply an ICC color transform, it will re-encode the color as `RGB(0.858659, 0, 0)`, and when that's rendered by the Adobe RGB display, it will be reproduced correctly as sRGB red.

## Perform gamut mapping to constrain to the display's gamut

In addition to reinterpreting color values to match the display's color space, you need to handle the case where the display can't physically reproduce all the colors in your content; if your content's color gamut is larger than the display's. That process is called gamut mapping.

Gamut mapping is lossy because you have to make a tradeoff on how to approximate the larger gamut of the content. The most straightforward method is colorimetric, where colors that are within the display's gamut are preserved, and colors that are out of the gamut are clipped to the nearest in-gamut value.

In an ICC profile-based workflow, gamut mapping is typically handled automatically in the color management library. You have some control over the mapping behavior by selecting the rendering intent (see [Rendering intent modes](#)).

### Note

When you're in an advanced color workflow, we generally don't recommend using the perceptual rendering intent, neither for source or destination, since it was designed for SDR sources and destinations that have smaller color gamuts than the ones used for HDR and some WCG displays; so using them can result in unexpected behavior.

## Windows automatic system color management

When Advanced Color is active, Windows performs automatic system color management—it ensures that your app's color content is accurately reproduced on the display. That dramatically simplifies the required actions on your app, although advanced apps might continue to perform additional processing for maximum color and perceptual accuracy. For more information, refer to [Use DirectX with Advanced Color on high/standard dynamic range displays](#).

### Get the display's color characteristics

Advanced Color-aware apps should not directly interact with the display ICC profile. Instead you can obtain the display's color properties using either [DisplayInformation::GetAdvancedColorInfo](#) or [IDXGIOOutput6](#).

### Perform color space conversion to the display's color space

Windows will perform the color space conversion to the display's color space determined by the current default color profile. If there is no profile, then EDID colorimetry data will be used. Your app automatically gets *scene-referred* color behavior—for example, if you render sRGB red encoded as `RGB(1, 0, 0)` and display to an Adobe RGB monitor, then Windows will correctly reproduce it as sRGB red. Advanced Color-aware apps should tag their content with the correct color space to inform Windows using [IDXGISwapChain3::SetColorSpace1](#). For all non Advanced Color-aware apps that render to a standard integer pixel format (for example, 8-bit RGBA), Windows will explicitly treat the app as sRGB. If you wish to render AdobeRGB red in an Advanced

Color scenario, then you'll need to render `RGB(1.158157,0,0)` in a scRGB tagged surface (it's constrained by the display's gamut).

## Perform gamut mapping to constrain to the display's gamut

The GPU's display pipeline will perform numeric clipping on out-of-gamut colors. If your app wishes to use a more sophisticated mapping, then you need to do that yourself.

## ICC profile default behavior with Advanced Color

Automatic system color management necessarily impacts the way that existing ICC profile-based apps behave, since they're performing many actions themselves that are now handled by the operating system (OS.) Windows applies the default behavior (described below) to ICC profile-based apps. That ensures that those apps don't have incorrect behavior. However, without further work, they won't get access to any of the extended color capabilities.

In particular, by default your ICC profile-based app is restricted to the sRGB gamut, even if the monitor is actually wider gamut. Windows also provides an ICC compatibility helper that can give your ICC app access to the display's entire gamut. For more info, see the [Display ICC profile compatibility helper](#) section in this topic.

## Get the display's color characteristics

When Advanced Color is active, any calls to the color profile management APIs to get the default profile for a display will return "no profile", regardless of what profiles are actually installed. By convention "no profile" should be interpreted as sRGB.

Display ICC profiles are still valid and used with Advanced Color, but they're used only at the system level, and most apps shouldn't directly interact with them. The below information is generally needed only if your app is a utility that enumerates all display profiles, or is authoring/installing profiles.

To enforce that, Windows adds the concept of `STANDARD` and `EXTENDED` color profile subtypes. That applies to any color profile management APIs that use the [COLORPROFILESUBTYPE](#):

CPST\_STANDARD\_DISPLAY\_COLOR\_MODE  
CPST\_EXTENDED\_DISPLAY\_COLOR\_MODE

### ⓘ Note

STANDARD and EXTENDED subtypes are not a property stored within the profile itself; rather, they apply to the profile's association to a display (that is, when the profile is added to the display's profile association list). A single profile could be associated with both STANDARD and EXTENDED subtypes for a display, meaning that it would be available both for standard and Advanced Color scenarios.

Display profile associations that are intended for use in SDR—whether regular SDR or Advanced Color SDR—have subtype STANDARD (that's the default if no value is specified). Display profile associations for use in HDR mode are subtype EXTENDED. If your app doesn't specify a subtype, then that's interpreted as STANDARD.

Any *getter* API using COLORPROFILESUBTYPE will return only profiles with the matching STANDARD or EXTENDED subtype. For example, if HDR is active, the only display profiles with the EXTENDED subtype are valid for use, and STANDARD subtype profiles aren't used. *Setter* APIs can specify the subtype (STANDARD is the default).

## Perform color space conversion to the display's color space

Because the ICC profile management APIs return sRGB when Advanced Color is active, your ICC profile-based app will color-manage to sRGB, and Windows will correctly reproduce that as sRGB on the display.

## Perform gamut mapping to constrain to the display's gamut

Any existing gamut-mapping behavior is preserved.

## Display ICC profile compatibility helper

When Advanced Color is active, Windows provides a compatibility helper for display ICC profiles that provides access to the display's entire gamut. In that way your app continues to get accurate and wide gamut colors up to the reported capability of the

display—the same functionality that's available on calibrated wide gamut monitors in legacy non-Advanced Color mode today. Without that helper, your app will be limited to default behavior, which is sRGB (see [ICC profile default behavior with Advanced Color](#)).

That helper is available starting with Windows 11. It doesn't provide other benefits of Advanced Color including access to higher precision/bit-depth or high dynamic range—you'll need to modify your app to be Advanced Color-aware.

## Enabling the display ICC compatibility helper

The display ICC compatibility helper is enabled on a per-app basis. It isn't enabled by default.

Users can enable it for an app by going to the Compatibility tab of the executable's properties, and selecting **Use legacy display ICC color management**. The compatibility helper is applied to the entire process, and is active only when Advanced Color is enabled for the display—it has no effect on a standard SDR display.

General Compatibility Security Details Previous Versions

If this program isn't working correctly on this version of Windows, try running the compatibility troubleshooter.

[Run compatibility troubleshooter](#)

[How do I choose compatibility settings manually?](#)

Compatibility mode

Run this program in compatibility mode for:

Windows 8

Settings

Reduced color mode

8-bit (256) color

Run in 640 x 480 screen resolution

Disable fullscreen optimizations

Run this program as an administrator

Register this program for restart

Use legacy display ICC color management

[Change high DPI settings](#)



[Change settings for all users](#)

OK

Cancel

Apply

Windows automatically enables the helper for some popular apps that are known to use ICC color profile management.

There's no programmatic way to enable that compatibility helper for your app.

## Get the display's color characteristics

If the compatibility helper is active, then when your app queries for the default **STANDARD** color profile using the [Windows Color System profile management functions](#), Windows constructs a synthetic ICC profile using the same data that populates the Advanced Color display capabilities APIs. The synthetic profile's data can come from a combination of the current color profile, from the display's EDID or DisplayID, or from other sources.

If your app queries for the default **EXTENDED** color profile, then that indicates that your app is Advanced Color-aware, and it will receive the actual **EXTENDED** profile.

## Perform color space conversion to the display's color space

If the compatibility helper is active, then your app is expected to use ICC color management to target the synthetic display profile. Windows assumes that your app is targeting that color space, and will perform the correct color space conversion to ensure it is accurately rendered on the display.

The color space conversion applies to the entire app process, so all of your app's visual content is treated as targeting the display's color space, even if some of it is not color managed and nominally targets sRGB (for example, UI). The color space conversion also is applied regardless of the graphics API (GDI, DirectX, XAML, and so on), pixel format, or other characteristics of your rendered content.

## Perform gamut-mapping to constrain to the display's gamut

Any existing gamut-mapping behavior is preserved.

## User-visible behavior changes

Users can verify whether the display ICC compatibility helper is active for an executable by checking its Compatibility properties tab. If your app shows info about the default display ICC profile, then users will see that it is a synthetic profile. The descriptive contents of the profile (including name) are an implementation detail.

The actual color behavior should be identical to when Advanced Color is disabled. In both cases, your app will render accurate colors that can access the full gamut of the display, as described by the ICC profile.

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Windows hardware display color calibration pipeline

Article • 08/28/2024

This topic covers display color calibration using a new GPU display color transform pipeline that's supported by Windows 10, version 2004 (20H1) and later. The pipeline provides significantly improved color accuracy over existing paths such as the GDI *gamma ramp* pipeline, and adds support for HDR displays.

This topic is for display and PC manufacturers and display calibration providers who want to better calibrate their customers' displays. Most Windows apps don't need to do anything to benefit from the pipeline; but if you develop color-managed apps, then you might want to be aware of how this technology works.

The new color pipeline is available for any display if the GPU meets the system requirements. If the display is HDR or uses auto color management, then there are additional considerations and requirements, which can be found in [Use DirectX with Advanced Color on high/standard dynamic range displays](#).

## Introduction

Display color calibration is the process of ensuring that a display accurately matches its reported color space; for example, sRGB or DCI-P3 D65. Because of variations in the manufacturing process and other sources, an individual display panel might deviate from its specification. Once a display has been calibrated, your apps and content can confidently target the display's color space without worrying about that variability or inaccuracy.

At a high level, display color calibration involves these steps:

1. Perform optical measurements of the actual color output of a display when rendering a set of known color values.
2. Based on the measurement data, generate a color transform that corrects for any inaccuracies in the display, and generate metadata that describes the display's resulting color volume.
3. Store the color transform data and display metadata for later use.
4. At runtime, load and apply the color transform to the display framebuffer (color values sent to the display), and report the display metadata to apps.

Windows 10, version 2004 provides enhanced functionality for steps 3 and 4, while display manufacturers and calibration providers are responsible for steps 1 and 2.

## System requirements

The new color transform pipeline requires a capable GPU and display driver. Supported GPU architectures include:

- AMD:
  - AMD RX 500 400 Series, or later
  - AMD Ryzen processors with Radeon Graphics
- Intel:
  - Integrated: Intel 10th Gen GPU (Ice Lake), or later
  - Discrete: Intel DG1, or later
- NVIDIA GTX 10xx, or later (Pascal+)
- Qualcomm 8CX Gen 3, or later; 7C Gen 3, or later

### Note

Intel codename Comet Lake (5 digit model code) chipsets are not supported.

A Windows Display Driver Model (WDDM) 2.6 or later driver is needed (released with Windows 10, version 1903). Some GPU vendors need a newer driver, potentially as new as WDDM 3.0 (released with Windows 11, version 21H2).

See [New display ICC profile management APIs](#) for info about how an app can determine whether the new color transform pipeline is available on a system.

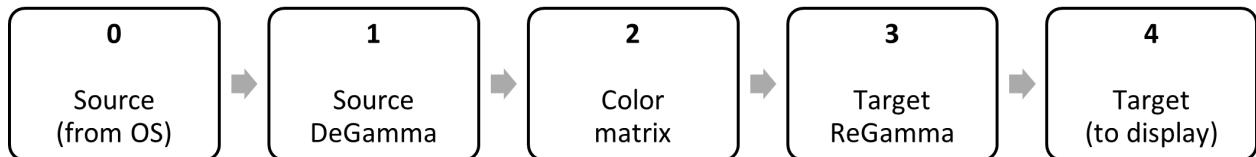
## New GPU color transform pipeline

Windows 10, version 2004 exposes a GPU-accelerated display color transform pipeline consisting of a linear gamma color matrix and 1DLUT. Compared to the existing *gamma ramp* pipeline, it offers superior accuracy, precision, and support for wide color gamut displays. In addition, it adds support for new technologies such as HDR displays that use BT.2100 signaling.

The pipeline isn't directly programmable by apps, and instead is exposed only via MHC profiles; see below for more details. Other operating system (OS) features such as night light might also use this pipeline, and the OS manages how to share (compose) and/or rationalize pipeline access between multiple scenarios.

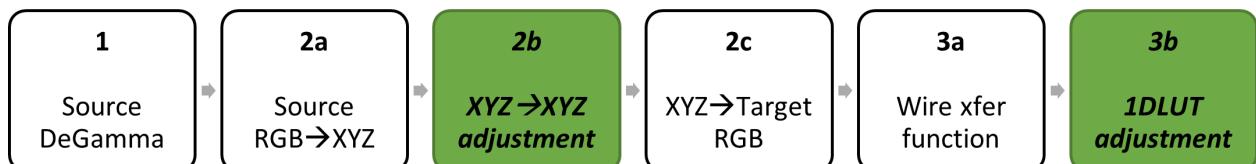
# Color transform pipeline description

The color transform pipeline is based on the standard conceptual model for color space conversions:



The model can convert between any two RGB (or other 3-channel) color spaces, such as sRGB to P3 D65. It can also correct the most common types of panel color variation.

The Windows color transform pipeline takes the conceptual model, expands stages 2 (color matrix) and 3 (target regamma) into sub-stages, and exposes a subset of the stages (**2b and 3b, in green**) for apps to program, while leaving the remainder (white) controlled by the driver:



Those modifications allow the color pipeline to be agnostic to the color space of the source content, which may change on a frame-to-frame basis. Also, it improves compatibility with display color spaces such as BT.2100 ST.2084, which require opaque optimizations in order to preserve precision.

## Stage 0: Source (graphics input)

The input is the rendered framebuffer from the OS. It can be in one of several color spaces depending on the scenario, including sRGB, sYCC, HDR10, or scRGB, and can change on a frame-to-frame basis.

## Stage 1: Source DeGamma

The display driver automatically converts the source content into linear gamma, and this stage is not programmable by apps.

## Stage 2: Color space conversion matrix

In the standard color space conversion model, the matrix stage can be broken down into three matrices, which are composed (multiplied) together:

- **2a:** Convert from the source content RGB color space (linear gamma) to an absolute color space; in the Windows pipeline, the absolute color space is CIEXYZ.
- **2b:** Perform any adjustments in CIEXYZ space, such as calibration.
- **2c:** Convert from CIEXYZ to the target RGB color space (linear gamma). The target RGB color space is defined as the encoding used when transmitting colors over the display wire, typically BT.709 or BT.2020 primaries. It is not the actual, measured primaries of the physical panel.

Matrix 2a is determined by the source content, and matrix 2c is determined by the display's signaling mode; only matrix 2b is accessible to apps. The driver multiplies the three together to generate the actual matrix to be executed in hardware:

```
FinalMatrix = SourceRGBtoXYZ * XYZtoXYZAdjust * XYZtoTargetRGB
```

### Note

Because the display driver is responsible for the source RGB to XYZ, and target XYZ to RGB conversions, the matrix that you program (stage 2b) should not include either.

**Example 1:** If you're performing no adjustments to colors (pass-through), then your matrix should be identity, regardless of the type of display you're outputting to.

**Example 2:** If you're outputting to an SDR P3 D65 display, and are implementing an "sRGB proofing" profile that emulates sRGB on the panel, then your matrix should consist of a primaries rotation from sRGB into P3 D65.

## Stage 3: Target ReGamma

This stage can be broken down into two RGB 1DLUTs, which are composed together:

- **3a:** Encode the linear RGB data from stage 2c into the transfer function/gamma of the signal over the display wire.
- **3b:** Perform any adjustments in the target gamma space, such as calibration.

1DLUT 3a is determined by the display wire format color space; most commonly it's sRGB for SDR displays, and ST.2084 for HDR displays. 3b is programmable by apps and

occurs after the wire format transfer function is applied. The driver composes the two 1DLUTs to generate the actual 1DLUT to be executed in hardware:

```
Final1DLUT = Adjustment1DLUT(TargetReGamma(input))
```

#### ⓘ Note

Because the driver is responsible for programming the display signal transfer function, the 1DLUT that you program (3b) should not include that encoding. For example, if you're performing no adjustments to colors (pass-through), then your 1DLUT should be identity, regardless of the display wire format color space.

## Stage 4: Target (output to scanout)

This is the framebuffer to be scanned out over the wire by the GPU; in the display's native color space, and after any adjustments you have programmed. Additional operations such as YCbCr encoding might occur afterwards.

## Higher precision and accuracy

The linear gamma matrix stage (XYZ to XYZ adjustment) capability was introduced in Windows 10, version 1709. The capability enables you to perform adjustments to color primaries and white point, as well as arbitrary RGB color space conversions.

The 1DLUT adjustment stage is conceptually similar to the existing *gamma ramp* 1DLUT, but offers improved precision, with up to 4096 LUT entries at up to 16-bit fixed point precision.

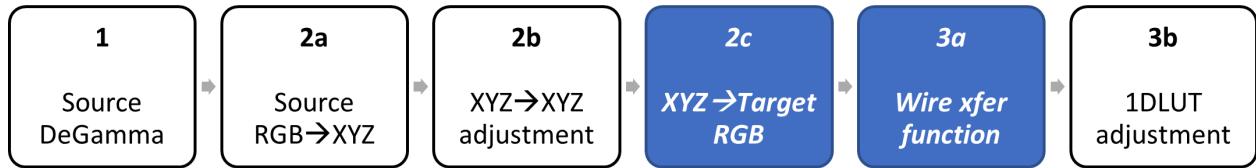
#### ⓘ Note

Not all hardware supports the full count of entries or precision exposed by the color pipeline.

## Support for HDR (BT.2100) displays

A limitation of the existing *gamma ramp* pipeline is that it has undefined behavior when the display is using HDR (BT.2100 ST.2084) signaling. The new color transform pipeline explicitly supports both SDR (BT.1886 or sRGB) and HDR signaling, and scales to support

future wire format color spaces. It accomplishes this via the "XYZ to Target RGB" and "Wire transfer function" (blue) stages in the block diagram:



Those two stages, which are controlled automatically by the driver, are responsible for encoding colors into the wire format color space: for example, sRGB or BT.2020 ST.2084.

Therefore, when you're programming the color transform pipeline, you get well-defined behavior based on the active wire format color space of the display.

## New "MHC2" tag for ICC profiles

Windows doesn't provide an API for directly controlling the new color transform pipeline at runtime. Instead, your app accesses the pipeline by writing a properly formatted International Color Consortium (ICC) color profile with extra data stored in a new "Microsoft Hardware Calibration" ("MHC2") private tag. It's a similar model to the existing *gamma ramp* pipeline, which uses "VCGT" private ICC tags. ICC profiles with valid MHC2 tag data are referred to as "MHC ICC profiles" or "MHC profiles".

### ⓘ Note

MHC2 refers to the second version of the private tag, which is available to all Windows 10, version 2004 devices; MHC1 shipped on an earlier release of Windows with specific OEM PCs.

## Supplemental ST.2086 HDR static metadata

In addition to programming the new color transform pipeline, MHC ICC profiles also contain ST.2086 HDR static metadata. Those are values that describe a display's dynamic range (luminance) and color gamut. They are widely implemented with HDR displays but are useful for any display. The values are:

- Peak luminance (nits)
- Max full frame luminance (nits)
- Min luminance (nits)
- RGB color primaries (xy coordinates)
- White point (xy coordinates)

The white point, max full frame luminance, and RGB color primaries are described using standard ICC tags. Peak and minimum luminance are described in the MHC2 tag. A profile must contain all of this information for the OS to accept the profile, and use it for Advanced Color scenarios.

Windows rationalizes ST.2086 metadata from multiple sources, including the MHC ICC profile, graphics driver, and EDID or DisplayID firmware. MHC ICC profiles are treated as the most trusted source, and will override other sources. Windows exposes this information via the HDR capability APIs as described in [Use DirectX with Advanced Color on high/standard dynamic range displays](#)—in that way, HDR apps are given the best available HDR display information.

## Definition of ST.2086 luminance for adjustable backlight displays

A display might have an adjustable backlight, for example controlled by the user, or controlled automatically by an ambient light sensor. That introduces ambiguity with how the ST.2086 luminance values should be interpreted.

For displays where Windows has control over the backlight (typically for laptops and integrated panel devices), the luminance values must describe when this OS-controlled backlight is at its maximum or brightest setting.

For displays where Windows doesn't have control over the backlight (typically for external monitors), the luminance values are accurate only for the display state at the time of measurement.

## ICC profile requirements

An MHC ICC profile must use either ICC spec version 2 ([ICC.1:2001-04](#)) or version 4 ([ICC.1:2010-12/ISO 15076-1:2010](#)). An MHC ICC profile must be a display device profile.

An MHC ICC profile may include color transform pipeline data. The portions of the MHC2 structure that define the color transform may be empty, which explicitly indicates an identity transform.

An MHC ICC profile must include ST.2086 metadata. A profile containing only ST.2086 metadata and no transform data is used for HDR display calibration scenarios—in that case HDR calibration means providing more accurate min/max luminance and color gamut information for HDR apps and games.

## Reuse of existing public tags

MHC ICC profiles use existing public tags to define some of the ST.2086 metadata values. All of these tags are already required for display device profiles. Tag and data type definitions can be found in the [ICC specifications](#).

[+] Expand table

Tag name	Data type	ST.2086 value	Unit reported by Windows
redColorantTag	XYZNumber	Red primary	Chromaticity (xy)
greenColorantTag	XYZNumber	Green primary	Chromaticity (xy)
blueColorantTag	XYZNumber	Blue primary	Chromaticity (xy)
mediaWhitePointTag	XYZNumber	White point	Chromaticity (xy)
luminanceTag	XYZNumber	Max full frame luminance	Luminance (nits)

## "MHC2" private tag definition

An MHC ICC profile must contain one MHC2 tag structure. The matrix and 1DLUT color transform elements may be set to 0 (NULL), which explicitly indicate an identity transform for the respective stage. The ST.2086 metadata values must be filled in with valid data.

[+] Expand table

Byte Position	Field Length (bytes)	Content	Data type
0 to 3	4	'MHC2' (4D484332h) Type Signature	MHC2Type
4 to 7	4	Offset to beginning of tag data element	UInt32Number
8 to 13	4	Size of tag data element	UInt32Number

## MHC2Type structure definition

[+] Expand table

Byte Position	Field Length (bytes)	Content	Data type
0 to 3	4	'MHC2' (4D484332h) Type Signature	
4 to 7	4	Reserved, set to 0	

<b>Byte Position</b>	<b>Field Length (bytes)</b>	<b>Content</b>	<b>Data type</b>
8 to 11	4	Number of 1DLUT entries (4096 or less) [1] OPTIONAL: 0 = Identity Transform	ulInt32Number
12 to 15	4	ST.2086 min luminance in nits	s15Fixed16Number
16 to 19	4	ST.2086 peak luminance in nits	s15Fixed16Number
20 to 23	4	Offset in bytes to matrix [2] OPTIONAL: 0 = Identity Transform	ulInt32Number
24 to 27	4	Offset in bytes to red 1DLUT [2]	ulInt32Number
28 to 31	4	Offset in bytes to green 1DLUT [2]	ulInt32Number
32 to 35	4	Offset in bytes to blue 1DLUT [2]	ulInt32Number

[1] The OS will interpolate data to the count of hardware-supported entries.

[2] Offsets within the MHC2Type structure are relative to the beginning of the structure, not the file.

## Matrix definition

[\[+\] Expand table](#)

<b>Byte Position</b>	<b>Field Length (bytes)</b>	<b>Content</b>	<b>Data type</b>
0 to 47	48	3x4 XYZ to XYZ adjustment matrix stored in row major order, column 4 is ignored [1]	s15Fixed16Number

[1] The matrix structure is sized to fit 12 elements for a 3x4 matrix in row major order. However, Windows uses only data from the left three columns, effectively defining a 3x3 matrix. For example, storing these 12 values in linear order:

```
[a, b, c, 0, d, e, f, 0, g, h, i, 0]
```

produces the following matrix:

[\[\] Expand table](#)

First column	Second Column	Third Column
a	b	c
d	e	f
g	h	i

### ① Note

As described in [Color space conversion matrix](#), don't include the source RGB to XYZ or XYZ to target RGB matrix transforms, since they are handled automatically by the driver. Target RGB is defined as the encoding used when transmitting colors over the display wire; typically BT.709 or BT.2020 primaries.

## 1DLUT definition

[\[\] Expand table](#)

Byte Position	Field Length (bytes)	Content	Data type
0 to 3	4	'sf32' (73663332h) Type Signature	
4 to 7	4	Reserved, set to 0	
8 to end	Variable (0 to 16384)	Calibration LUT values normalized to [0.0, 1.0]	s15Fixed16Number

### ① Note

As described in [Target ReGamma](#), this LUT operates in the wire format color space after the transfer function is encoded.

### ① Note

If your measurements or calibration curve need fewer than 4096 LUT entries, then store only the count of entries you actually need, and specify the count in the MHC2Type structure. For example, the simplest identity LUT requires only two

entries set to 0.0 and 1.0. The OS will interpolate to the count of hardware-supported entries.

## New display ICC profile management APIs

### Note

The guidance in this section applies to any display ICC profile, whether or not it contains MHC data.

Once you've generated an MHC ICC profile, you provision it on the Windows system for the targeted display. In earlier versions of Windows, you would use [Windows Color System \(WCS\) profile management functions](#) to do that. While you can continue to use these existing APIs, Windows 10, version 2004 adds a set of new, modernized APIs to WCS that are specialized for managing display ICC color profiles. These APIs are all prefixed with "ColorProfile":

- `ColorProfileAddDisplayAssociation`
- `ColorProfileRemoveDisplayAssociation`
- `ColorProfileSetDisplayDefaultAssociation`
- `ColorProfileGetDeviceCapabilities`

### Note

The above API provides functionality for which there's no existing WCS API equivalent.

- `ColorProfileGetDisplayList`
- `ColorProfileGetDisplayDefault`
- `ColorProfileGetDisplayUserScope`

A typical workflow using the ColorProfile APIs to provision an MHC ICC profile on the system is:

1. Use `ColorProfileGetDeviceCapabilities` to determine whether the system supports the new color transform pipeline. Even if it doesn't, it might still be beneficial to provision the profile to provide supplemental ST.2086 metadata.
2. Use `InstallColorProfile` (an existing WCS API) to install the color profile. That adds the profile to the list of profiles available for use on the system.

3. Use `ColorProfileGetDisplayUserScope` to determine whether the Windows user has overridden the system's default profile associations, and is using their own per-user association lists.
4. Use `ColorProfileAddDisplayAssociation` to associate the color profile with a display (make an installed profile selectable for that display), and optionally set the profile as default (the currently active profile).

## Enhanced Windows display calibration loader

Windows has offered an inbox display color calibration loader since Windows 7. That calibration loader supports reading ICC profiles with *gamma ramp* pipeline data stored in either VCGT or MS00 private ICC profile tags. The *gamma ramp* loader must be explicitly turned on by calling `WcsSetCalibrationManagementState`.

Windows 10, version 2004 enhances the inbox calibration loader by adding support for MHC ICC profiles and the new color transform pipeline. Writing and provisioning an MHC ICC profile, and having the Windows loader apply its state, is the only method for apps to access the color transform pipeline: there are no direct access APIs. Unlike with the *gamma ramp* profiles, reading from MHC ICC profiles is always enabled, so once an MHC ICC profile is set as default on a capable system, its calibration state is automatically loaded.

## HDR and Advanced Color scenarios with automatic system color management

New Advanced Color technologies such as HDR and auto color management add new capabilities to Windows including superior color accuracy and access to much larger display color gamuts; for more information, see [Use DirectX with Advanced Color on high/standard dynamic range displays](#).

Advanced color and auto color management ensure consistent and colorimetrically accurate display color for all apps: both legacy and modern. However, some apps might perform their own explicit color management using International Color Consortium (ICC) color profiles.

When Advanced Color is active on either SDR or HDR displays, the behavior of display ICC profiles changes in non-backwards compatible ways. If your app works with display ICC profiles, then Windows offers compatibility behaviors to ensure that your app continues to get correct behavior.

For info about the changes to ICC profile behavior and how you can adapt your app to maximize compatibility with Advanced Color, refer to [ICC profile behavior with Advanced Color](#).

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# API elements reference for WCS

Article • 12/30/2021

The API elements that WCS provides fall into the following sections:

- [Functions](#)
- [Obsolete WCS Functions](#)
- [Enumerations](#)
- [Structures](#)
- [Macros for CMYK Values and Colors](#)
- [WCS Constants](#)
- [WCS Schemas](#)
- [WCS Interfaces](#)
- [WCS Registry Keys](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# WCS functions

Article • 12/30/2021

In the following topics, WCS functions are organized by category. This allows you to browse functions that have related uses in a single topic. The function reference pages are arranged in alphabetical order.

The functions are grouped as follows:

- Basic Functions for Use Within a Device Context
- Advanced Functions for Use Outside of a Device Context
- Device Calibration and Characterization Functions
- Profile Management Functions
- ICM2 Functions for Color Management Modules (CMMs) to Implement
- Alphabetical List of All WCS Functions

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Basic Functions for Use Within a Device Context

Article • 06/09/2022

The following WCS functions deliver basic [color mapping](#) capabilities within device contexts. They are useful to all applications that need to implement color management with low overhead and minimal user intervention.

Function	Description
<a href="#">CheckColorsInGamut</a>	Checks if given colors are in a device's gamut.
<a href="#">ColorCorrectPalette</a>	Corrects the entries in a palette for a device context.
<a href="#">ColorMatchToTarget</a>	Performs color mapping for preview purposes.
<a href="#">CreateColorSpace</a>	Creates a color space.
<a href="#">DeleteColorSpace</a>	Deletes a color space.
<a href="#">EnumICMProfiles</a>	Enumerates output color profiles available for a given device context.
<a href="#">EnumICMProfilesProcCallback</a>	Application-defined callback function for <a href="#">EnumICMProfiles</a> . The name of this function is also defined by the application.
<a href="#">GetColorSpace</a>	Gets the current input color space in a device context.
<a href="#">GetICMProfile</a>	Gets the current output color profile of a device context.
<a href="#">GetLogColorSpace</a>	Gets the <a href="#">LOGCOLORSPACE</a> structure of a device context.
<a href="#">SetColorSpace</a>	Sets a device context's input color space.
<a href="#">SetICMMode</a>	Turns color management on or off in a device context.
<a href="#">SetICMProfile</a>	Sets the output color profile for a given device context.
<a href="#">WcsEnumColorProfiles</a>	Enumerates all color profiles that satisfy the enumeration criteria in the specified profile management scope.

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Advanced Functions for Use Outside of a Device Context

Article • 12/30/2021

These functions provide advanced color management capabilities outside of device contexts.

Function	Description
<a href="#">PCMSCALLBACKW</a>	*PCMSCALLBACKW* (or <a href="#">ApplyCallbackFunction</a> ) is a callback function that you implement that updates the WCS configuration data while the dialog box displayed by the <a href="#">SetupColorMatchingW</a> function is executing.
<a href="#">CheckBitmapBits</a>	Checks whether the pixels in a specified bitmap lie within the output <a href="#">gamut</a> of a specified transform.
<a href="#">CheckColors</a>	Determines whether the colors in an array lie within the output <a href="#">gamut</a> of a specified transform.
<a href="#">ConvertColorNameToIndex</a>	Converts color names in a named color space to index numbers in an International Color Consortium (ICC) color profile.
<a href="#">ConvertIndexToColorName</a>	Transforms indices in a color space to an array of names in a named color space.
<a href="#">CreateColorTransformW</a>	Transforms indices in a color space to an array of names in a named color space.
<a href="#">CreateMultiProfileTransform</a>	Accepts an array of profiles or a single <a href="#">device link profile</a> and creates a color transform that applications can use to perform color map operations.
<a href="#">DeleteColorTransform</a>	Deletes a given color transform.
<a href="#">GetCMMInfo</a>	Retrieves various information about the color management module (CMM) that created the specified color transform.
<a href="#">GetNamedProfileInfo</a>	Retrieves information about the International Color Consortium (ICC) named color profile that is specified in the first parameter.
<a href="#">ICMProgressProcCallback</a>	Application-supplied callback function to report progress. The name of this function is also defined by the application.
<a href="#">SelectCMM</a>	Allows you to select the preferred color management module (CMM) to use.
<a href="#">SetupColorMatchingW</a>	Provides user control over color management by way of a dialog box.
<a href="#">TranslateBitmapBits</a>	Converts bitmap colors using a color transform.
<a href="#">TranslateColors</a>	Translates an array of colors from the source <a href="#">color space</a> to the destination color space as defined by a color transform.
<a href="#">WcsCheckColors</a>	Determines whether the colors in an array are within the output gamut of a specified WCS color transform.
<a href="#">WcsTranslateColors</a>	Translates an array of colors from the source color space to the destination color space as defined by a color transform.

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Device Calibration and Characterization Functions

Article • 12/30/2021

The following functions are useful for writing device calibration and characterization tools needed for installing and calibrating color display devices such as monitors and printers.

Function	Description
<a href="#">CloseColorProfile</a>	Closes an open profile handle.
<a href="#">CreateDeviceLinkProfile</a>	Creates an International Color Consortium (ICC) <i>device link profile</i> from a set of color profiles, using the specified intents.
<a href="#">GetColorProfileElement</a>	Copies data from a specified tagged profile element of a specified color profile into a buffer.
<a href="#">GetColorProfileElementTag</a>	Retrieves the tag name specified by <i>dwlIndex</i> in the tag table of a given International Color Consortium (ICC) color profile, where <i>dwlIndex</i> is the index of the tag to be retrieved.
<a href="#">GetColorProfileFromHandle</a>	Retrieves the color profile contents given a handle to an open color profile.
<a href="#">GetColorProfileHeader</a>	Retrieves or derives ICC header structure from either ICC color profile or WCS XML profile. Drivers and applications should assume that the ICC header is valid.
<a href="#">GetCountColorProfileElements</a>	Retrieves the number of tagged elements in a given color profile.
<a href="#">GetPS2ColorRenderingDictionary</a>	Retrieves the PostScript Level 2 color rendering dictionary from the specified ICC color profile.
<a href="#">GetPS2ColorRenderingIntent</a>	Retrieves the PostScript Level 2 color <a href="#">rendering intent</a> from an ICC color profile.
<a href="#">GetPS2ColorSpaceArray</a>	Retrieves the PostScript Level 2 <a href="#">color space</a> array from an ICC color profile.
<a href="#">IsColorProfileTagPresent</a>	Reports whether a specified International Color Consortium (ICC) tag is present in the specified color profile.
<a href="#">IsColorProfileValid</a>	Allows you to determine whether the specified profile is a valid International Color Consortium (ICC) profile, or a valid Windows color profile.
<a href="#">OpenColorProfileW</a>	Creates a handle to a specified color profile. The handle can then be used in other profile management functions.
<a href="#">SetColorProfileElement</a>	Sets the element data for a tagged profile element in an ICC color profile.
<a href="#">SetColorProfileElementReference</a>	Creates in a specified ICC color profile a new tag that references the same data as an existing tag.
<a href="#">SetColorProfileElementSize</a>	Sets the size of a tagged element in an ICC color profile.
<a href="#">SetColorProfileHeader</a>	Sets the header data in a specified ICC color profile.
<a href="#">WcsGetCalibrationManagementState</a>	Determines whether system management of the display calibration state is enabled.
<a href="#">WcsSetCalibrationManagementState</a>	Determines whether system management of the display calibration state is enabled.

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Profile Management Functions

Article • 12/30/2021

## Profile Management Functions

The following API functions are useful in profile management.

Function	Description
<a href="#">AssociateColorProfileWithDeviceW</a>	Associates a specified color profile with a specified device.
<a href="#">[CreateProfileFromLogColorSpaceW]</a> ((/windows/win32/api/icm/nf-icm-createprofilefromlogcolorspacew)	Converts a logical <a href="#">color space</a> to a <a href="#">device profile</a> .
<a href="#">DisassociateColorProfileFromDeviceW</a>	Disassociates a specified color profile with a specified device on a specified computer.
<a href="#">EnumColorProfilesW</a>	Enumerates all the profiles satisfying the given enumeration criteria.
<a href="#">GetColorDirectoryW</a>	Retrieves the path of the Windows COLOR directory on a specified machine.
<a href="#">GetDeviceGammaRamp</a>	Gets the gamma ramp from direct color display boards.
<a href="#">GetStandardColorSpaceProfileW</a>	Retrieves the color profile registered for the specified standard <a href="#">color space</a> .
<a href="#">InstallColorProfileW</a>	Installs a given profile for use on a specified machine. The profile is also copied to the COLOR directory.
<a href="#">RegisterCMMW</a>	Associates a specified identification value with the specified color management module dynamic link library (CMM DLL). W
<a href="#">SetDeviceGammaRamp</a>	Sets the gamma ramp on direct color display boards.
<a href="#">SetStandardColorSpaceProfileW</a>	Registers a specified profile for a given standard <a href="#">color space</a> . The profile can be queried using <a href="#">GetStandardColorSpaceProfileW</a> .
<a href="#">UninstallColorProfileW</a>	Removes a specified color profile from a specified computer. Associated files are optionally deleted from the system.
<a href="#">UnregisterCMMW</a>	Dissociates a specified ID value from a given color management module dynamic-link library (CMM DLL).
<a href="#">WcsAssociateColorProfileWithDevice</a>	Associates a specified WCS color profile with a specified device.
<a href="#">WcsCreateIccProfile</a>	Converts a WCS profile into an ICC profile.
<a href="#">WcsDisassociateColorProfileFromDevice</a>	Disassociates a specified WCS color profile with a specified device on a specified computer.
<a href="#">WcsEnumColorProfiles</a>	Enumerates all color profiles that satisfy the enumeration criteria in the specified profile management scope.
<a href="#">WcsEnumColorProfilesSize</a>	Returns the size, in bytes, of the buffer required by the <a href="#">WcsEnumColorProfiles</a> function to enumerate color profiles.
<a href="#">WcsGetDefaultColorProfile</a>	Retrieves the default color profile for a device, or the device-independent default if the device is not specified.
<a href="#">WcsGetDefaultColorProfileSize</a>	Returns the size, in bytes, of the default color profile name for a device, including the <b>NULL</b> terminator.
<a href="#">WcsGetDefaultRenderingIntent</a>	Retrieves the default rendering intent in the specified profile management scope.
<a href="#">WcsGetUsePerUserProfiles</a>	Determines whether the user has chosen to use a per-user profile association list for the specified device.
<a href="#">WcsOpenColorProfileW</a>	Creates a handle to a specified color profile.
<a href="#">WcsSetDefaultColorProfile</a>	Sets the default color profile name of the specified profile type in the specified profile management scope.
<a href="#">WcsSetDefaultRenderingIntent</a>	Sets the default rendering intent in the specified profile management scope.
<a href="#">WcsSetUsePerUserProfiles</a>	Allows the user to specify whether to use a per-user profile association list for the specified device.

## Profile Consumption Functions

The profile consumption APIs are those APIs in ICM2 that take ICC or WCS XML profiles, profile handles, or rendering intents as parameters, and a set of new APIs for WCS profile support for application color management code.

## Profiles and Profile Management Functions

The profile management workflow is based on existing ICM2 APIs that are augmented to provide additional functionality for revising application code.

Profiles contain information used by color processing algorithms to translate color between different color spaces. Profile management provides a way to query and specify which profiles get used at different stages by the color processing model to manage color output of various peripheral devices with diverse color characteristics.

Profile management provides the following set of functionalities:

1. Installing color profiles for use in the system.
2. Associating one or more installed color profiles with any particular device.
3. Choosing a default color profile of a particular type among the profiles available for use in a particular stage of color processing. This could be for a device among the profiles associated with it, or among the profiles installed in the system and not device specific.
4. Enumerating color profiles that satisfy particular criteria among the profiles installed in the system.

The WCS profile filename extensions are ".cdmp" for DMPs, ".camp" for CAMPs, and ".gmmp" for GMMPs.

#### **Per-user profile management and enabling execution in LUA context**

The goal of the design described in the current document is as follows:

1. Legacy ICM2 implementation does not provide support for per-user profile management. Different users cannot have their own profile settings. In Vista, the WCS profile management infrastructure enables users to configure individual profile settings for most functionalities.
2. All legacy ICM2 profile management APIs modify settings system-wide and require administrative privileges. In Windows Vista, all users run in Least-privileged User Account (LUA) settings most of the time, and administrators can elevate privilege selectively to run applications that modify system-wide settings. In WCS profile management, all per-user profile settings are configurable in LUA context. Profile management applications can run as LUA settings, increasing their scope of use and ensuring that security of the system is not compromised.

Profile management in Vista provides the following enhancements over legacy ICM2 infrastructure:

1. It enables profile association with devices, default profile settings, and enumeration of profiles in both per-user and system-wide scope.
2. Installing a profile remains system wide and requires administrator privileges. This is consistent with profile installation during device installation because device installation is system wide and requires administrative privileges.

Whether devices can be installed from LUA context is particular to what is supported for that class of device. For example, in Vista, it is possible to do printer installation from LUA context if the user has been granted rights to copy files into the driver store by a domain administrator using driver store policies. Color profile management infrastructure doesn't need to do anything special in this regard since the installation happens in spooler context.

3. Modifying profile settings in per-user scope can be done in LUA context; system-wide modifications required administrative privileges. Profile management operations that require reading configuration information can be done in LUA context for both per-user and

system-wide settings.

Profile management scope indicates the scope of the operations performed; either per-user or system wide.

For each operation, it is indicated whether it can be done from LUA context. If an operation cannot be performed in LUA context, the corresponding profile management API returns failure with access denied. Applications using the API, such as Color Management Control Panel, can enable the user to elevate to administrative context (using OTS or Consent UI), and then call the API from the elevated context so that the operation will succeed.

Operation

Profile Management Scope

Pre-condition

Post-condition

Executable in LUA context

  \${{ROWSPAN2}}\$Install profile\${{REMOVE}}\$

System wide

Profile copied, installed into the system, and available for use. Profile is enumerable in system-wide and current-user scope for all users.

During device driver installation, governed by driver installation policies. No, otherwise.

Current user

Not supported

  \${{ROWSPAN2}}\$Uninstall profile\${{REMOVE}}\$

System wide

Profile is installed in the system

Profile uninstalled from the system and optionally deleted from the profile store. Profile is no longer available for use and is not enumerable in any scope.

No

Current user

Not supported

  \${{ROWSPAN2}}\$Associate profile with device\${{REMOVE}}\$

System wide

Profile is installed and is of type ICC or CDMP

Profile is available for use with the device by all users. It is enumerable, in system-wide scope and also current-user scope for all users, as associated with the device.

No

Current user

Profile is installed. It does not matter whether the profile is already associated to the device in system-wide scope and is of type ICC or CDMP.

Profile is available for use with the device by current user. It is enumerable only in current-user scope (unless there is a system-wide association as well) as associated with the device.

Yes

  \${{ROWSPAN2}}\$Disassociate profile from device\${{REMOVE}}\$

System wide

Profile is associated with the device in system-wide scope and is of type ICC or CDMP

Profile is no longer available for use (except for users who have this association in their current-user scope as well). It is not enumerable in system-wide scope. It could be enumerable in current-user scope though, for a user who has this association in its scope.

No

Current user

Profile is associated with the device in current-user scope (irrespective of whether it is associated in system-wide scope) and is of type ICC or CDMP.

Profile is no longer available for use, or enumerable as associated to the device, by current user (unless it is also associated in system-wide scope to the device).

Yes

\${\{ROWSPAN2}\\$Set profile for a type (DMP or ICC) as default for a device\\${REMOVE}\\$

System wide

Profile is of type ICC or CDMP

The profile is used by default, for the particular type with the device, for all users except for those who have overridden this setting in their current-user scope. (The profile is installed and associated to the device system wide, if that is not already the case.)

No

Current user

Profile is of type ICC or CDMP

The profile is used by default for the particular type with the device in case of current user, irrespective of the system-wide default for this. (The profile is installed and associated to the device for current user, if that is not already the case.)

Yes, if the profile is already installed

\${\{ROWSPAN2}\\$Set profile for a type (ICC, DMP, CAMP, GMMP) and subtype combination as global default\\${REMOVE}\\$

System wide

Only ICC and CDMP profiles can be associated with devices.

The profile is used by default for the particular type. Users can override this setting in the current-user scope. (The profile is installed, if that is not already the case.)

No

Current user

Only ICC and CDMP profiles can be associated with devices.

The profile is used by default for the particular type for current user. (The profile is installed, if that is not already the case.)

Yes, if the profile is already installed.

\${\{ROWSPAN2}\\$Erase the current-user override for a particular default profile setting, so that the system default always gets used (as fallback) even for current-user scope.\\${REMOVE}\\$

System wide

Not applicable

Current user

Even for current-user queries on default profile settings, system-wide settings are returned for use.

Yes

\${\{ROWSPAN2}\\$Enumerate installed profiles satisfying particular criteria (like device class, profile class, etc.)\\${REMOVE}\\$

System wide

Only ICC and CDMP profiles can be associated with and enumerated for devices.

Profiles that are installed and satisfy the specified criteria in system-wide scope are enumerated.

Yes

Current user

Only ICC and CDMP profiles can be associated with devices and thus enumerated for devices.

Profiles which are installed and satisfy the specified criteria in system-wide scope are enumerated.

Yes

\${\{ROWSPAN2\}}\$ Enumerate profiles associated with a particular device satisfying particular criteria, such as device class, and profile class \${\{REMOVE\}}\$

System wide

Only ICC and CDMP profiles can be associated with and enumerated for devices.

Profiles that are associated with the device in system-wide scope and satisfies the specified criteria in system-wide scope are enumerated.

Yes

Current user

Only ICC and CDMP profiles can be associated with and enumerated for devices.

Profiles that are available as associated with the device in current-user scope, which includes the system-wide associations and satisfies the specified criteria in current-user scope are enumerated.

Yes

The valid color profile types are provided by the COLORPROFILETYPE enumeration.

The valid color profile subtypes are provided by the COLORPROFILESUBTYPE enumeration.

The valid profile type/subtype combinations are shown in the following table.

#### COLORPROFILETYPE

##### Valid COLORPROFILESUBTYPE

##### Notes

Device Default

Global Default

Intended Use

Intended Use

CPT\_ICC

CPST\_NONE

Get/Set default ICC profile associated with a device

CPST\_RGBWorkingSpace or CPST\_CustomWorkingSpace

Get/Set ICC profile as global RGB or custom working space profile. See Note.

The COLORPROFILETYPE CPT\_ICC and CPT\_DMP are mutually exclusive. The default color profile you set for a given working space (RGB or Custom) can be either an ICC profile or a DMP profile, but not both.

CPT\_DMP

CPST\_NONE

Get/Set default DMP profile associated with a device

CPST\_RGBWorkingSpace or CPST\_CustomWorkingSpace

Get/Set DMP profile as global RGB or custom working space profile. See Note.

The COLORPROFILETYPE CPT\_ICC and CPT\_DMP are mutually exclusive. The default color profile you set for a given working space (RGB or Custom) can be either an ICC profile or a DMP profile, but not both.

#### ⚠ Note

When WcsSetDefaultColorProfile is called to set a DMP profile as the default profile for the RGB working space or a custom working space, only a DMP profile that is of type RGBVirtualDevice, LCD, or CRT is valid.

When WcsSetDefaultColorProfile is called to set an ICC profile as the default profile for the RGB working space or a custom working space, only an ICC profile whose class is "spac" or "disp," and whose color space is "RGB" is valid.

The architecture is designed according to the requirements of the operations as mentioned in the enumerations and tables above.

## Profile management public API layer

Because profile management scope is not supported by legacy ICM2 APIs, a new set of WCS profile management APIs is required that defines profile management scope as system wide or current user. Legacy ICM2 APIs continue to be supported for backward compatibility, and work on profile management scope that is implicit for the call. o ICM2 APIs that work on current-user scope ? This is for operations that are supported for both system wide and current-user scope in WCS profile management. Legacy ICM2 APIs call on new WCS APIs with profile management scope as current user. This makes sense from user perspective because this enables per-user settings from legacy applications and also executing most of the operations in LUA context. o ICM2 APIs that work on system-wide scope ? This is for operations (install profiles and uninstall profiles) that support only system-wide scope. No new WCS profile management APIs are created and existing APIs can be modified.

The underlying implementations of the profile management operations work on the following configuration data entities to create the context for color processing algorithms to provide color management functionalities. They are either device specific or global (device independent) settings. o Device specific configuration data: ? List of profiles associated with a particular device. ? Default profile for different profile types associated with a device. ? Matching mode of profiles used for enumeration. o Global configuration data: ? List of profiles installed in the system. ? Global default profile for different profile types. ? The underlying implementations of configuration data storage take in storage scope for configuration data (either device independent or device specific), which can be either system wide or current user. This is different from profile management scope. An operation with current-user profile management scope can cause a read from a system-wide storage scope if the current-user setting for that operation is not present. ? The ICM2/WCS API layer calls in this storage layer for getting and setting data with appropriate storage scope. The storage layer is transparent to profile management scope. The logic for combining data from current-user and system-wide storage scopes to create or update a configuration according to the profile management scope specified by the API caller. This logic is present in the ICM2/WCS API layer.

## Device-specific storage layer

The storage for different classes of devices like print, capture or display could be different from each other. For example, configuration data for a print device must be stored using standard print APIs, such as SetPrinterDataEx and GetPrinterDataEx, to enable the profiles to be copied and settings to be transferred to a client machine during Point-and-Print connection. ? This layer exports functionality to open store, get data, set data and close store using common pre-defined interfaces so that the profile management configuration storage layer can call into them while being transparent to the way the data gets stored for that device.

The following diagram illustrates this architecture.

### Profile Management Public API Layer

Legacy ICM2 APIs for operations that support only system-wide profile management scope in Vista (install, uninstall, and get color directory). They call the configuration storage layer with appropriate storage scope. \${REMOVE}

Legacy ICM2 API for operations that support both system-wide and current-user profile management scope in Vista (all operations other than install, uninstall, and get color directory). They implicitly work on current-user scope, and call into new WCS API with profile management scope as current user.

New WCS API with system-wide and current-user profile management scope support. They call the configuration storage layer with appropriate storage scope.

Profile Management Configuration Storage Layer

Device-independent global configuration routines

Device-specific configuration routines

Profile installation and device-independent default profile settings management, supported in system-wide and current-user storage scope.~~MOVE~~

Device association and device-specific default profile settings management, supported in system-wide and current-user storage scope.

Device-Specific Storage layer

Print specific storage

Display specific storage

Capture specific storage

Legacy ICM2 APIs for operations that support only system-wide profile management scope in Vista have no changes in behavior. Install and uninstall operations fall in this category.

Legacy ICM2 APIs for operations that support both system-wide and current-user profile management scope have their behavior changed to query and configure current-user settings. All operations other than install and uninstall fall in this category.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WCS Functions for Color Management Modules (CMMs) to Implement

Article • 12/30/2021

The following functions are to be implemented by color management modules (CMMs), and exported for the operating system to call.

Function	Description
<a href="#">CMCheckColors</a>	Determines whether given colors lie within the output <a href="#">gamut</a> of a specified transform.
<a href="#">CMCheckColorsInGamut</a>	Determines whether specified RGB triples lie in the output <a href="#">gamut</a> of a specified transform.
<a href="#">CMCheckRGBs</a>	Checks bitmap colors against an output gamut.
<a href="#">CMConvertColorNameToIndex</a>	Converts color names in a named color space to index numbers in a color profile
<a href="#">CMConvertIndexToColorName</a>	Transforms indices in a color space to an array of names in a named color space.
<a href="#">CMCreateDeviceLinkProfile</a>	Creates a <a href="#">device link profile</a> in the format specified by the International Color Consortium in its ICC Profile Format Specification.
<a href="#">CMCreateMultiProfileTransform</a>	Accepts an array of profiles or a single <a href="#">device link profile</a> and creates a color transform. This transform is a mapping from the color space of the first profile to the color space of the last profile.
<a href="#">CMCreateProfile</a>	Creates a display color profile from a <a href="#">LOGCOLORSPACEA</a> structure.
<a href="#">CMCreateProfileW</a>	Creates a display color profile from a <a href="#">LOGCOLORSPACEW</a> structure.
<a href="#">CMCreateTransform</a>	Deprecated. There is no replacement API because this one was no longer being used. Developers of alternate CMM modules are encouraged to implement this function.
<a href="#">CMCreateTransformExt</a>	Creates a color transform that maps from an input <a href="#">LOGCOLORSPACEA</a> to an optional target space and then to an output device link profile.
<a href="#">CMCreateTransformExtW</a>	Creates a color transform that maps from an input <a href="#">LOGCOLORSPACEW</a> to an optional target space and then to an output device link profile.
<a href="#">CMCreateTransformW</a>	Deprecated. There is no replacement API because this one was no longer being used. Developers of alternate CMM modules are encouraged to implement this function.
<a href="#">CMDelateTransform</a>	Deletes a specified color transform, and frees any memory associated with it.
<a href="#">CMGetInfo</a>	Retrieves various information about the color management module (CMM).
<a href="#">CMGetNamedProfileInfo</a>	Retrieves information about the specified named color profile.
<a href="#">CMGetPS2ColorRenderingDictionary</a>	Gets a PostScript color rendering dictionary.
<a href="#">CMGetPS2ColorRenderingIntent</a>	Retrieves the PostScript Level 2 color <a href="#">rendering intent</a> from a profile.
<a href="#">CMGetPS2ColorSpaceArray</a>	Gets a PostScript color space array.
<a href="#">CMIsProfileValid</a>	Reports whether the given profile is a valid ICC profile that can be used for color management.
<a href="#">CMTranslateColors</a>	Translates an array of colors from a source <a href="#">color space</a> to a destination color space using a color transform.
<a href="#">CMTranslateRGB</a>	Translates an application-supplied RGBQuad into the device <a href="#">color space</a> .
<a href="#">CMTranslateRBGs</a>	Translates a bitmap from one <a href="#">color space</a> to another using a color transform.
<a href="#">CMTranslateRBGsExt</a>	Translates a bitmap from one defined format into a different defined format and calls a callback function periodically, if one is provided.

## Feedback

Was this page helpful? [!\[\]\(1ac551012a870e156bf017228aac2819\_img.jpg\) Yes](#) [!\[\]\(f32c2a98be68ed521a1b9e1297719f10\_img.jpg\) No](#)

Get help at Microsoft Q&A

# Alphabetical List of All WCS Functions

Article • 06/09/2022

The following is a complete alphabetical list of the WCS 1.0 API functions provided by Windows 98 and later and Windows 2000 and later.

Function or Structure	Description
<b>PCMSCALLBACKW</b>	*PCMSCALLBACKW* (or <a href="#">ApplyCallbackFunction</a> ) is a callback function that you implement that updates the WCS configuration.
<b>AssociateColorProfileWithDeviceW</b>	Associates a specified color profile with a specified device.
<b>CheckBitmapBits</b>	Checks whether the pixels in a specified bitmap lie within the output <a href="#">gamut</a> of a specified transform.
<b>CheckColors</b>	Determines whether the colors in an array lie within the output <a href="#">gamut</a> of a specified transform.
<b>CheckColorsInGamut</b>	Checks if given colors are in a device's gamut.
<b>CloseColorProfile</b>	Closes an open profile handle.
<b>CMCheckColors</b>	Determines whether given colors lie within the output <a href="#">gamut</a> of a specified transform.
<b>CMCheckColorsInGamut</b>	Determines whether specified RGB triples lie in the output <a href="#">gamut</a> of a specified transform.
<b>CMCheckRGBs</b>	Checks bitmap colors against an output gamut.
<b>CMConvertColorNameToIndex</b>	Converts color names in a named color space to index numbers in a color profile
<b>CMConvertIndexToColorName</b>	Transforms indices in a color space to an array of names in a named color space.
<b>CMCreateDeviceLinkProfile</b>	Creates a <a href="#">device link profile</a> in the format specified by the International Color Consortium in its ICC Profile Format Specification.
<b>CMCreateMultiProfileTransform</b>	Accepts an array of profiles or a single <a href="#">device link profile</a> and creates a color transform. This transform is a mapping from the input color space to the output color space.
<b>CMCreateProfile</b>	Creates a display color profile from a <a href="#">LOGCOLORSPACEA</a> structure.
<b>CMCreateProfileW</b>	Creates a display color profile from a <a href="#">LOGCOLORSPACEW</a> structure.
<b>CMCreateTransform</b>	Deprecated. There is no replacement API because this one was no longer being used. Developers of alternate CMM modules should use <a href="#">CMCreateTransformExt</a> .
<b>CMCreateTransformExt</b>	Creates a color transform that maps from an input <a href="#">LOGCOLORSPACEA</a> to an optional target space and then to an output color space.
<b>CMCreateTransformExtW</b>	Creates a color transform that maps from an input <a href="#">LOGCOLORSPACEW</a> to an optional target space and then to an output color space.
<b>CMCreateTransformW</b>	Deprecated. There is no replacement API because this one was no longer being used. Developers of alternate CMM modules should use <a href="#">CMCreateTransformExtW</a> .
<b>CMDeleteTransform</b>	Deletes a specified color transform, and frees any memory associated with it.
<b>CMGetInfo</b>	Retrieves various information about the color management module (CMM).
<b>CMGetNamedProfileInfo</b>	Retrieves information about the specified named color profile.
<b>CMGetPS2ColorRenderingDictionary</b>	Gets a PostScript color rendering dictionary.
<b>CMGetPS2ColorRenderingIntent</b>	Retrieves the PostScript Level 2 color <a href="#">rendering intent</a> from a profile.
<b>CMGetPS2ColorSpaceArray</b>	Gets a PostScript color space array.
<b>CMIsProfileValid</b>	Reports whether the given profile is a valid ICC profile that can be used for color management.
<b>CMTranslateColors</b>	Translates an array of colors from a source <a href="#">color space</a> to a destination color space using a color transform.
<b>CMTranslateRGB</b>	Translates an application-supplied RGBQuad into the device <a href="#">color space</a> .
<b>CMTranslateRBGs</b>	Translates a bitmap from one <a href="#">color space</a> to another using a color transform.
<b>CMTranslateRBGsExt</b>	Translates a bitmap from one defined format into a different defined format and calls a callback function periodically, if one is specified.
<b>ColorCorrectPalette</b>	Corrects the entries in a palette for a device context.
<b>ColorMatchToTarget</b>	Performs color mapping for preview purposes.
<b>ConvertColorNameToIndex</b>	Converts color names in a named color space to index numbers in an International Color Consortium (ICC) color profile.
<b>ConvertIndexToColorName</b>	Transforms indices in a color space to an array of names in a named color space.
<b>CreateColorSpace</b>	Creates a color space.
<b>CreateColorTransformW</b>	Transforms indices in a color space to an array of names in a named color space.

Function or Structure	Description
CreateColorTransformW	Transforms indices in a color space to an array of names in a named color space.
CreateMultiProfileTransform	Accepts an array of profiles or a single <a href="#">device link profile</a> and creates a color transform that applications can use to perform color space conversion.
[CreateProfileFromLogColorSpaceW] ((/windows/win32/api/icm/nf-icm-createprofilefromlogcolorspacew)	Converts a logical <a href="#">color space</a> to a <a href="#">device profile</a> .
DeleteColorSpace	Deletes a color space.
DeleteColorTransform	Deletes a given color transform.
DisassociateColorProfileFromDeviceW	Disassociates a specified color profile with a specified device on a specified computer.
EnumColorProfilesW	Enumerates all the profiles satisfying the given enumeration criteria.
EnumICMProfiles	Enumerates output color profiles available for a given device context.
EnumICMProfilesProcCallback	Application-defined callback function for <a href="#">EnumICMProfiles</a> .
GetCMMInfo	Retrieves various information about the color management module (CMM) that created the specified color transform.
GetColorDirectoryW	Retrieves the path of the Windows COLOR directory on a specified machine.
GetColorProfileElement	Copies data from a specified tagged profile element of a specified color profile into a buffer.
GetColorProfileElementTag	Retrieves the tag name specified by <i>dwIndex</i> in the tag table of a given International Color Consortium (ICC) color profile, version 4.1 or later.
GetColorProfileFromHandle	Retrieves the color profile contents given a handle to an open color profile.
GetColorProfileHeader	Retrieves or derives ICC header structure from either ICC color profile or WCS XML profile. Drivers and applications should use this function to get the ICC header structure.
GetColorSpace	Gets the current input color space in a device context.
GetCountColorProfileElements	Retrieves the number of tagged elements in a given color profile.
GetDeviceGammaRamp	Gets the gamma ramp from direct color display boards.
GetICMProfile	Gets the current output color profile of a device context.
GetLogColorSpace	Gets the <a href="#">LOGCOLORSPACE</a> structure of a device context.
GetNamedProfileInfo	Retrieves information about the International Color Consortium (ICC) named color profile that is specified in the first parameter.
GetPS2ColorRenderingDictionary	Retrieves the PostScript Level 2 color rendering dictionary from the specified ICC color profile.
GetPS2ColorRenderingIntent	Retrieves the PostScript Level 2 color <a href="#">rendering intent</a> from an ICC color profile.
GetPS2ColorSpaceArray	Retrieves the PostScript Level 2 <a href="#">color space</a> array from an ICC color profile.
GetStandardColorSpaceProfileW	Retrieves the color profile registered for the specified standard <a href="#">color space</a> .
ICMProgressProcCallback	Application-supplied callback to report progress.
InstallColorProfileW	Installs a given profile for use on a specified machine. The profile is also copied to the COLOR directory.
IsColorProfileTagPresent	Reports whether a specified International Color Consortium (ICC) tag is present in the specified color profile.
IsColorProfileValid	Allows you to determine whether the specified profile is a valid International Color Consortium (ICC) profile, or a valid Windows color profile.
OpenColorProfileW	Creates a handle to a specified color profile. The handle can then be used in other profile management functions.
RegisterCMMW	Associates a specified identification value with the specified color management module dynamic link library (CMM DLL). Windows Vista and later.
SelectCMM	Allows you to select the preferred color management module (CMM) to use.
SetColorProfileElement	Sets the element data for a tagged profile element in an ICC color profile.
SetColorProfileElementReference	Creates in a specified ICC color profile a new tag that references the same data as an existing tag.
SetColorProfileElementSize	Sets the size of a tagged element in an ICC color profile.
SetColorProfileHeader	Sets the header data in a specified ICC color profile.
SetColorSpace	Sets a device context's input color space.
SetDeviceGammaRamp	Sets the gamma ramp on direct color display boards.
SetICMMode	Turns color management on or off in a device context.

Function or Structure	Description
<a href="#">SetICMProfile</a>	Sets the output color profile for a given device context.
<a href="#">SetStandardColorSpaceProfileW</a>	Registers a specified profile for a given standard <a href="#">color space</a> . The profile can be queried using <a href="#">GetStandardColorSpaceProfile</a> .
<a href="#">SetupColorMatchingW</a>	Provides user control over color management by way of a dialog box.
<a href="#">TranslateBitmapBits</a>	Converts bitmap colors using a color transform.
<a href="#">TranslateColors</a>	Translates an array of colors from the source <a href="#">color space</a> to the destination color space as defined by a color transform.
<a href="#">UninstallColorProfileW</a>	Removes a specified color profile from a specified computer. Associated files are optionally deleted from the system.
<a href="#">UnregisterCMMW</a>	Dissociates a specified ID value from a given color management module dynamic-link library (CMM DLL).
<a href="#">WcsAssociateColorProfileWithDevice</a>	Associates a specified WCS color profile with a specified device.
<a href="#">WcsCheckColors</a>	Determines whether the colors in an array lie within the output gamut of a specified WCS color transform.
<a href="#">WcsCreateIccProfile</a>	Converts a WCS profile into an ICC profile.
<a href="#">WcsDisassociateColorProfileFromDevice</a>	Disassociates a specified WCS color profile with a specified device on a specified computer.
<a href="#">WcsEnumColorProfiles</a>	Enumerates all color profiles that satisfy the enumeration criteria in the specified profile management scope.
<a href="#">WcsEnumColorProfilesSize</a>	Returns the size, in bytes, of the buffer required by the <a href="#">WcsEnumColorProfiles</a> function to enumerate color profiles.
<a href="#">WcsGetCalibrationManagementState</a>	Determines whether system management of the display calibration state is enabled.
<a href="#">WcsGetDefaultColorProfile</a>	Retrieves the default color profile for a device, or the device-independent default if the device is not specified.
<a href="#">WcsGetDefaultColorProfileSize</a>	Returns the size, in bytes, of the default color profile name for a device, including the NULL terminator.
<a href="#">WcsGetDefaultRenderingIntent</a>	Returns the user or system-wide rendering intent.
<a href="#">WcsGetUsePerUserProfiles</a>	Determines whether the user has chosen to use a per-user profile association list for the specified device.
<a href="#">WcsOpenColorProfileW</a>	Creates a handle to a specified color profile.
<a href="#">WcsSetCalibrationManagementState</a>	Enables or disables system management of the display calibration state.
<a href="#">WcsSetDefaultColorProfile</a>	Sets the default color profile name of the specified profile type in the specified profile management scope.
<a href="#">WcsSetDefaultRenderingIntent</a>	Sets the user or system-wide rendering intent.
<a href="#">WcsSetUsePerUserProfiles</a>	Allows the user to specify whether or not to use a per-user profile association list for the specified device.
<a href="#">WcsTranslateColors</a>	Translates an array of colors from the source color space to the destination color space as defined by a color transform.

## Feedback

Was this page helpful? [!\[\]\(b24841ad71cf80fb25f2748899463961\_img.jpg\) Yes](#) [!\[\]\(1c28f022a490046d9dd99b794b0dedec\_img.jpg\) No](#)

Get help at Microsoft Q&A

# PBMCALLBACKFN callback function (icm.h)

Article 02/22/2024

TBD

## Syntax

C++

```
PBMCALLBACKFN Pbmcallbackfn;

BOOL Pbmcallbackfn(
    ULONG unnamedParam1,
    ULONG unnamedParam2,
    LPARAM unnamedParam3
)
{...}
```

## Parameters

unnamedParam1

TBD

unnamedParam2

TBD

unnamedParam3

TBD

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	icm.h

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# PCMSCALLBACKA callback function (icm.h)

Article 10/05/2021

\***PCMSCALLBACKA**\* (or **ApplyCallbackFunction**) is a callback function that you implement that updates the WCS configuration data while the dialog box displayed by the **SetupColorMatchingW** function is executing. The name **ApplyCallbackFunction** is a placeholder. The actual name of this callback function is supplied by your application using ICM.

## Syntax

C++

```
PCMSCALLBACKA Pcmcallback;  
  
BOOL Pcmcallback(  
    _tagCOLORMATCHSETUPA *unnamedParam1,  
    LPARAM unnamedParam2  
)  
{...}
```

## Parameters

unnamedParam1

Pointer to a **COLORMATCHSETUPW** structure that contains WCS configuration data.

unnamedParam2

Contains a value supplied by the application.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. The callback function can set the extended error information by calling **SetLastError**.

## Remarks

The [ApplyCallbackFunction](#) function is used to change the WCS configuration for a device while the Color Management dialog box is displayed. The Color Management dialog box is displayed by the [SetupColorMatchingW](#) function.

If the callback function is provided, an **Apply** button is displayed in the lower right of the dialog box. When you select the **Apply** button, the callback function immediately updates the configuration for the device being set up. The Color Management dialog box remains on the screen.

An application supplies a callback function to WCS by storing the address of the callback function in the [COLORMATCHSETUPW](#) structure that is passed to the [SetupColorMatchingW](#) function. The address is stored in the [IPfnApplyCallback](#) ↗ member of the [COLORMATCHSETUP](#) structure. The **dwFlags** member should be set to **CMS\_USEAPPLYCALLBACK**, or the callback function will be ignored.

A value supplied by the application may be passed to the callback function. Prior to invoking the [SetupColorMatchingW](#) function, the application can store a value in the [IParamApplyCallback](#) ↗ member of the [COLORMATCHSETUPW](#) structure. When the callback function is invoked, the value in the [IParamApplyCallback](#) structure member will be passed to the callback function in its *IParam* parameter.

The callback function is completely optional. If it is not supplied, the **Apply** button does not appear in the Color Management dialog box. Microsoft strongly recommends that your application supplies a callback function.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [SetupColorMatchingW](#)
- [COLORMATCHSETUPW](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# PCMSCALLBACKW callback function (icm.h)

Article 10/05/2021

\***PCMSCALLBACKW**\* (or **ApplyCallbackFunction**) is a callback function that you implement that updates the WCS configuration data while the dialog box displayed by the **SetupColorMatchingW** function is executing. The name **ApplyCallbackFunction** is a placeholder. The actual name of this callback function is supplied by your application using ICM.

## Syntax

C++

```
PCMSCALLBACKW Pcmcallbackw;

BOOL Pcmcallbackw(
    _tagCOLORMATCHSETUPW *unnamedParam1,
    LPARAM unnamedParam2
)
{...}
```

## Parameters

unnamedParam1

Pointer to a **COLORMATCHSETUPW** structure that contains WCS configuration data.

unnamedParam2

Contains a value supplied by the application.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. The callback function can set the extended error information by calling **SetLastError**.

## Remarks

The [ApplyCallbackFunction](#) function is used to change the WCS configuration for a device while the Color Management dialog box is displayed. The Color Management dialog box is displayed by the [SetupColorMatchingW](#) function.

If the callback function is provided, an **Apply** button is displayed in the lower right of the dialog box. When you select the **Apply** button, the callback function immediately updates the configuration for the device being set up. The Color Management dialog box remains on the screen.

An application supplies a callback function to WCS by storing the address of the callback function in the [COLORMATCHSETUPW](#) structure that is passed to the [SetupColorMatchingW](#) function. The address is stored in the [IPfnApplyCallback](#) ↗ member of the [COLORMATCHSETUP](#) structure. The **dwFlags** member should be set to **CMS\_USEAPPLYCALLBACK**, or the callback function will be ignored.

A value supplied by the application may be passed to the callback function. Prior to invoking the [SetupColorMatchingW](#) function, the application can store a value in the [IParamApplyCallback](#) ↗ member of the [COLORMATCHSETUPW](#) structure. When the callback function is invoked, the value in the [IParamApplyCallback](#) structure member will be passed to the callback function in its *IParam* parameter.

The callback function is completely optional. If it is not supplied, the **Apply** button does not appear in the Color Management dialog box. Microsoft strongly recommends that your application supplies a callback function.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [SetupColorMatchingW](#)
- [COLORMATCHSETUPW](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# AssociateColorProfileWithDeviceA function (icm.h)

Associates a specified color profile with a specified device.

## ! Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileAddDisplayAssociation](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL AssociateColorProfileWithDeviceA(
    PCSTR pMachineName,
    PCSTR pProfileName,
    PCSTR pDeviceName
);
```

## Parameters

pMachineName

Reserved. Must be **NULL**. This parameter is intended to point to the name of the machine on which to associate the specified profile and device. A **NULL** pointer indicates the local machine.

pProfileName

Points to the file name of the profile to associate.

pDeviceName

Points to the name of the device to associate.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

The **AssociateColorProfileWithDevice** function will fail if the profile has not been installed on the computer using the [InstallColorProfileW](#) function.

Note that under Windows (Windows 95 or later), the PostScript device driver for printers assumes a CMYK color model. Therefore, all PostScript printers must use a CMYK color profile. Windows 2000 does not have this limitation.

If the specified device is a monitor, this function updates the default profile.

Several profiles are typically associated with printers, based on paper and ink types. There is no default. The GDI selects the best one from the associated profiles when your application creates a device context (DC).

Scanners also have no default profile. However, it is atypical to associate more than one profile with a scanner.

**AssociateColorProfileWithDevice** always adds the specified profile to the current user's per-user profile association list for the specified device. Before adding the profile to the list, **AssociateColorProfileWithDevice** determines whether the user has previously expressed the desire to use a per-user profile association list for the device. If so, then **AssociateColorProfileWithDevice** simply adds the specified profile to the existing per-user profile association list for the device. If not, then **AssociateColorProfileWithDevice** creates a new per-user profile association list for the device by copying the system-wide association list for that device. It then appends the specified profile to the per-user list. From that point on, the current user will be using a per-user profile association list for the specified device, as if [WcsSetUsePerUserProfiles](#) had been called for *pDevice* with the *usePerUserProfiles* parameter set to TRUE.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Mscms.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [DisassociateColorProfileFromDevice](#)

---

Last updated on 10/31/2025

# AssociateColorProfileWithDeviceW function (icm.h)

Article07/27/2022

Associates a specified color profile with a specified device.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileAddDisplayAssociation](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL AssociateColorProfileWithDeviceW(
    PCWSTR pMachineName,
    PCWSTR pProfileName,
    PCWSTR pDeviceName
);
```

## Parameters

pMachineName

Reserved. Must be **NULL**. This parameter is intended to point to the name of the machine on which to associate the specified profile and device. A **NULL** pointer indicates the local machine.

pProfileName

Points to the file name of the profile to associate.

pDeviceName

Points to the name of the device to associate.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

The **AssociateColorProfileWithDevice** function will fail if the profile has not been installed on the computer using the [InstallColorProfileW](#) function.

Note that under Windows (Windows 95 or later), the PostScript device driver for printers assumes a CMYK color model. Therefore, all PostScript printers must use a CMYK color profile. Windows 2000 does not have this limitation.

If the specified device is a monitor, this function updates the default profile.

Several profiles are typically associated with printers, based on paper and ink types. There is no default. The GDI selects the best one from the associated profiles when your application creates a device context (DC).

Scanners also have no default profile. However, it is atypical to associate more than one profile with a scanner.

**AssociateColorProfileWithDevice** always adds the specified profile to the current user's per-user profile association list for the specified device. Before adding the profile to the list, **AssociateColorProfileWithDevice** determines whether the user has previously expressed the desire to use a per-user profile association list for the device. If so, then **AssociateColorProfileWithDevice** simply adds the specified profile to the existing per-user profile association list for the device. If not, then **AssociateColorProfileWithDevice** creates a new per-user profile association list for the device by copying the system-wide association list for that device. It then appends the specified profile to the per-user list. From that point on, the current user will be using a per-user profile association list for the specified device, as if [WcsSetUsePerUserProfiles](#) had been called for *pDevice* with the *usePerUserProfiles* parameter set to **TRUE**.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [DisassociateColorProfileFromDeviceW](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# CheckBitmapBits function (icm.h)

Article 10/05/2021

Checks whether the pixels in a specified bitmap lie within the output [gamut](#) of a specified transform.

## Syntax

C++

```
BOOL CheckBitmapBits(
    HTRANSFORM    hColorTransform,
    PVOID         pSrcBits,
    BMFORMAT      bmInput,
    DWORD         dwWidth,
    DWORD         dwHeight,
    DWORD         dwStride,
    PBYTE         paResult,
    PBMCALLBACKFN pfnCallback,
    LPARAM        lpCallbackData
);
```

## Parameters

`hColorTransform`

Handle to the color transform to use.

`pSrcBits`

Pointer to the bitmap to check against the output gamut.

`bmInput`

Specifies the format of the bitmap. Must be set to one of the values of the [BMFORMAT](#) enumerated type.

`dwWidth`

Specifies the number of pixels per scan line of the bitmap.

`dwHeight`

Specifies the number of scan lines of the bitmap.

`dwStride`

Specifies the number of bytes from the beginning one scan line to the beginning of the next one. If set to zero, the bitmap scan lines are assumed to be padded so as to be **DWORD**-aligned.

`paResult`

Pointer to an array of bytes where the test results are to be placed. This results buffer must contain at least as many bytes as there are pixels in the bitmap.

`pfnCallback`

Pointer to a callback function called periodically by **CheckBitmapBits** to report progress and allow the calling process to cancel the bitmap test. (See [ICMProgressProcCallback](#)).

`lpCallbackData`

Data passed back to the callback function, for example, to identify the bitmap test about which progress is being reported.

## Return value

If this function succeeds, the return value is a nonzero value.

If this function fails, the return value is zero. For extended error information, call **GetLastError**.

## Remarks

If the input format is not compatible with the color transform, the **CheckBitmapBits** function fails.

This function places results of the tests in the buffer pointed to by *paResult*. Each byte in the buffer corresponds to a pixel in the bitmap, and has an unsigned value between 0 and 255. The value 0 denotes that the color is in gamut, while a nonzero value denotes that it is out of gamut. For any integer *n* such that  $0 < n < 255$ , a result value of *n* + 1 indicates that the corresponding color is at least as far out of gamut as would be indicated by a result value of *n*.

When either of the floating point BMFORMATs, **BM\_32b\_scARGB** or **BM\_32b\_scRGB** is used, the color data being checked should not contain NaN or infinity. NaN and infinity are not considered to represent legitimate color component values, and the result of

checking pixels containing NaN or infinity is meaningless in color terms. NaN or infinity values in the color data being processed will be handled silently, and an error will not be returned.

The out-of-gamut information in the gamut tags created in WCS use the perceptual color distance in CIECAM02, which is the mean square root in CIECAM02 Jab space. The distance in the legacy ICC profile gamut tags is the mean square root in CIELAB space. We recommend that you use the CIECAM02 space when it is available because it provides more perceptually accurate distance metrics.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [ICMProgressProcCallback](#)
- [BMFORMAT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CheckColors function (icm.h)

Article 02/22/2024

Determines whether the colors in an array lie within the output [gamut](#) of a specified transform.

## Syntax

C++

```
BOOL CheckColors(
    HTRANSFORM hColorTransform,
    PCOLOR     paInputColors,
    DWORD      nColors,
    COLORTYPE   ctInput,
    PBYTE      paResult
);
```

## Parameters

`hColorTransform`

Handle to the color transform to use.

`paInputColors`

Pointer to an array of *nColors* [COLOR](#) structures to translate.

`nColors`

Contains the number of elements in the arrays pointed to by *paInputColors* and *paResult*.

`ctInput`

Specifies the input color type.

`paResult`

Pointer to an array of *nColors* bytes that receives the results of the test.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

If the input color type is not compatible with the color transform, [CheckColors](#) fails.

The function places results of the tests in the array pointed to by *paResult*. Each byte in the array corresponds to a **COLOR** element in the array pointed to by *palnputColors* and has an unsigned value between 0 and 255. The value 0 denotes that the color is in gamut, while a nonzero value denotes that it is out of gamut. For any integer *n* such that  $0 < n < 255$ , a result value of *n* + 1 indicates that the corresponding color is at least as far out of gamut as would be indicated by a result value of *n*.

The out-of-gamut information in the gamut tags created in WCS use the perceptual color distance in CIECAM02, which is the mean square root in CIECAM02 Jab space. The distance in the legacy ICC profile gamut tags is the mean square root in CIELAB space. We recommend that you use the CIECAM02 space when it is available because it provides more perceptually accurate distance metrics.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [COLOR structure](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CheckColorsInGamut function (wingdi.h)

Article04/02/2021

The **CheckColorsInGamut** function determines whether a specified set of RGB triples lies in the output [gamut](#) of a specified device. The RGB triples are interpreted in the input logical color space.

## Syntax

C++

```
BOOL CheckColorsInGamut(
    HDC      hdc,
    LPRGBTRIPLE lpRGBTriple,
    LPVOID   dlpBuffer,
    DWORD    nCount
);
```

## Parameters

hdc

Handle to the device context whose output gamut to be checked.

lpRGBTriple

Pointer to an array of RGB triples to check.

dlpBuffer

Pointer to the buffer in which the results are to be placed. This buffer must be at least as large as *nCount* bytes.

nCount

The number of elements in the array of triples.

## Return value

If this function succeeds, the return value is a nonzero value.

If this function fails, the return value is zero.

## Remarks

The function places the test results in the buffer pointed to by *lpBuffer*. Each byte in the buffer corresponds to an *RGB triple*, and has an unsigned value between CM\_IN\_GAMUT (= 0) and CM\_OUT\_OF\_GAMUT (= 255). The value 0 denotes that the color is in gamut, while a nonzero value denotes that it is out of gamut. For any integer *n* such that  $0 < n < 255$ , a result value of *n* + 1 indicates that the corresponding color is at least as far out of gamut as would be indicated by a result value of *n*, as specified by the ICC Profile Format Specification. For more information on the ICC Profile Format Specification, see the sources listed in [Further information](#).

Note that for this function to succeed, WCS must be enabled for the device context handle that is passed in through the *hDC* parameter. WCS can be enabled for a device context handle by calling the [SetICMMode](#) function.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [SetICMMode](#)

---

## Feedback



Was this page helpful?  

[Get help at Microsoft Q&A](#)

# CloseColorProfile function (icm.h)

Article02/22/2024

Closes an open profile handle.

## Syntax

C++

```
BOOL CloseColorProfile(  
    HPROFILE hProfile  
) ;
```

## Parameters

`hProfile`

Handle to the profile to be closed. The function determines whether the HPROFILE contains ICC or WCS profile information.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib

Requirement	Value
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CMCheckColors function (icm.h)

Determines whether given colors lie within the output [gamut](#) of a specified transform.

## Syntax

C++

```
BOOL CMCheckColors(
    HCMTRANSFORM hcmTransform,
    LPCOLOR     lpaInputColors,
    DWORD       nColors,
    COLORTYPE   ctInput,
    LPBYTE      lpaResult
);
```

## Parameters

`hcmTransform`

Handle to the color transform to use.

`lpaInputColors`

Pointer to an array of [COLOR](#) structures to check against the output gamut.

`nColors`

Specifies the number of elements in the array.

`ctInput`

Specifies the input color type.

`lpaResult`

Pointer to a buffer in which to place an array of bytes containing the test results. Each byte in the buffer corresponds to a [COLOR](#) structure, and on exit has been set to an unsigned value between 0 and 255. The value 0 denotes that the color is in gamut, while a nonzero value indicates that it is out of gamut. For any integer  $n$  such that  $0 < n < 255$ , a result value of  $n + 1$  indicates that the corresponding color is at least as far out of gamut as would be indicated by a result value of  $n$ . These values are usually generated from the *gamutTag* in the ICC profile.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. If the function is not successful, the CMM should call [SetLastError](#) to set the last error to a valid error value defined in Winerror.h.

## Remarks

Every CMM is required to export this function.

If the input color type is not compatible with the color transform **CMCheckColors** fails.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

Last updated on 10/31/2025

# CMCheckColorsInGamut function (icm.h)

Article02/22/2024

[**CMCheckColorsInGamut** is no longer available for use as of Windows Vista.]

Determines whether specified RGB triples lie in the output [gamut](#) of a specified transform.

## Syntax

C++

```
BOOL CMCheckColorsInGamut(
    HCMTRANSFORM hcmTransform,
    RGBTRIPLE    *lpaRGBTriple,
    LPBYTE        lpaResult,
    UINT          nCount
);
```

## Parameters

`hcmTransform`

Specifies the transform to use.

`lpaRGBTriple`

Points to an array of RGB triples to check.

`lpaResult`

Points to the buffer in which to put results.

The results are represented by an array of bytes. Each byte in the array corresponds to an RGB triple and has an unsigned value between 0 and 255. The value 0 denotes that the color is in gamut, while a nonzero value denotes that it is out of gamut. For any integer  $n$  in the range  $0 < n < 255$ , a result value of  $n + 1$  indicates that the corresponding color is at least as far out of gamut as would be indicated by a result value of  $n$ .

`nCount`

Specifies the number of elements in the array.

## Return value

Beginning with Windows Vista, the default CMM (Icm32.dll) will return **FALSE** and [GetLastError](#) will report ERROR\_NOT\_SUPPORTED.

**Windows Server 2003, Windows XP and Windows 2000:**

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. Call [GetLastError](#) to retrieve the error.

## Remarks

Beginning with Windows Vista, CMM Implementors are no longer required to implement this method.

**Windows Server 2003, Windows XP and Windows 2000:**

CMM Implementors are required to implement this method.

Every CMM is required to export this function.

If the function is not successful, custom CMMs should call [SetLastError](#) to set the last error to a valid error value defined in Winerror.h.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CMCheckRGBs function (icm.h)

Checks bitmap colors against an output gamut.

## Syntax

C++

```
BOOL CMCheckRGBs(
    HCMTRANSFORM hcmTransform,
    LPVOID        lpSrcBits,
    BMFORMAT      bmInput,
    DWORD         dwWidth,
    DWORD         dwHeight,
    DWORD         dwStride,
    LPBYTE        lpaResult,
    PBMCALLBACKFN pfnCallback,
    LPARAM        ulCallbackData
);
```

## Parameters

hcmTransform

lpSrcBits

bmInput

dwWidth

dwHeight

dwStride

lpaResult

pfnCallback

ulCallbackData

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Icm32.Lib

---

Last updated on 10/31/2025

# CMConvertColorNameToIndex function (icm.h)

Converts color names in a named color space to index numbers in a color profile.

## Syntax

C++

```
BOOL CMConvertColorNameToIndex(
    HPROFILE     hProfile,
    PCOLOR_NAME  paColorName,
    PDWORD       paIndex,
    DWORD        dwCount
);
```

## Parameters

`hProfile`

The handle to a named color profile.

`paColorName`

Pointer to an array of color name structures.

`paIndex`

Pointer to an array of **DWORDS** that this function fills with the indices.

`dwCount`

The number of color names to convert.

## Return value

If this function succeeds with the conversion, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. When this occurs, the CMM should call **SetLastError** to set the last error to a valid error value defined in **Winerror.h**.

## Remarks

This function is required in the default CMM. It is optional for all other CMMs.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

Last updated on 10/31/2025

# CMConvertIndexToColorName function (icm.h)

Transforms indices in a color space to an array of names in a named color space.

## Syntax

C++

```
BOOL CMConvertIndexToColorName(
    HPROFILE     hProfile,
    PDWORD       paIndex,
    PCOLOR_NAME  paColorName,
    DWORD        dwCount
);
```

## Parameters

`hProfile`

The handle to a color space profile.

`paIndex`

Pointer to an array of color-space index numbers.

`paColorName`

Pointer to an array of color name structures.

`dwCount`

The number of indices to convert.

## Return value

If this conversion function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE. When this occurs, the CMM should call `SetLastError` to set the last error to a valid error value defined in Winerror.h.

## Remarks

This function is required in the default CMM. It is optional for all other CMMs.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

Last updated on 10/31/2025

# CMCreateDeviceLinkProfile function (icm.h)

Creates a [device link profile](#) in the format specified by the International Color Consortium in its ICC Profile Format Specification.

## Syntax

C++

```
BOOL CMCreateDeviceLinkProfile(
    PHPROFILE pahProfiles,
    DWORD      nProfiles,
    PDWORD     padwIntents,
    DWORD      nIntents,
    DWORD      dwFlags,
    LPBYTE     *lpProfileData
);
```

## Parameters

`pahProfiles`

Pointer to an array of profile handles.

`nProfiles`

Specifies the number of profiles in the array.

`padwIntents`

An array of rendering intents.

`nIntents`

The number of elements in the array of intents.

`dwFlags`

Specifies flags to used control creation of the transform. For details, see [CMM Transform Creation Flags](#).

`lpProfileData`

Pointer to a pointer to a buffer. If successful the function allocates and fills this buffer. The calling application must free this buffer when it is no longer needed. Use the [GlobalFree](#)

function to free this buffer.

## Return value

If the function succeeds, the return value is a nonzero value.

If this function fails, the return value is zero. If the function is not successful, the CMM should call [SetLastError](#) to set the last error to a valid error value defined in Winerror.h.

## Remarks

Only the Windows default CMM is required to export this function; it is optional for all other CMMs.

If a CMM does not support [CMCreateDeviceLinkProfile](#), Windows uses the default CMM to create a device link profile.

The first and the last profiles in the array must be [device profiles](#). The other profiles can be [color space](#) or abstract profiles. Each profile's output color space must be the next profile's input color space.

The calling application must free the buffer allocated by this function and pointed to by the *lpProfileData* parameter. Use the [GlobalFree](#) function to free the buffer.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

- GlobalFree
- 

Last updated on 10/31/2025

# CMCreateMultiProfileTransform function (icm.h)

Accepts an array of profiles or a single [device link profile](#) and creates a color transform. This transform is a mapping from the color space specified by the first profile to that of the second profile and so on to the last one.

## Syntax

C++

```
HCMTRANSFORM CMCreateMultiProfileTransform(  
    PHPROFILE pahProfiles,  
    DWORD nProfiles,  
    PDWORD padwIntents,  
    DWORD nIntents,  
    DWORD dwFlags  
) ;
```

## Parameters

`pahProfiles`

Points to an array of profile handles.

`nProfiles`

Specifies the number of profiles in the array.

`padwIntents`

Points to an array of rendering intents. Each rendering intent is represented by one of the following values:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering intents](#).

`nIntents`

Specifies the number of intents in the intent array. Can be 1, or the same value as `nProfiles`.

## dwFlags

Specifies flags to used control creation of the transform. For details, see [CMM Transform Creation Flags](#).

## Return value

If this function succeeds, the return value is a color transform in the range 256 to 65,535. Since only the low **WORD** of the transform is retained, valid transforms cannot exceed this range.

If this function fails, the return value is an error code having a value less than 256. When the return value is less than 256, signaling an error, the CMM should use **SetLastError** to set the last error to a valid error value as defined in Winerror.h.

## Remarks

Every CMM is required to export this function.

The array of intents specifies how profiles should be combined. The *n*th intent is used for combining the *n*th profile in the array. If only one intent is specified, it is used for the first profile, and all other profiles are combined using Match intent.

The profile handles used to create the color transform can be closed after the call to **CMCreateMultiProfileTransform** completes.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)

- Functions

---

Last updated on 10/31/2025

# CMCreateProfile function (icm.h)

[CMCreateProfile is no longer available for use as of Windows Vista.]

Creates a display color profile from a [LOGCOLORSPACEA](#) structure.

## Syntax

C++

```
BOOL CMCreateProfile(
    LPLOGCOLORSPACEA lpColorSpace,
    LPDEVCHARACTER    *lpProfileData
);
```

## Parameters

`lpColorSpace`

Pointer to a color logical space, of which the `IcsFilename` member will be **NULL**.

`lpProfileData`

Pointer to a pointer to a buffer. If successful the function allocates and fills this buffer. It is the calling application's responsibility to free this buffer when it is no longer needed.

## Return value

Beginning with Windows Vista, the default CMM (Icm32.dll) will return **FALSE** and [GetLastError](#) will report **ERROR\_NOT\_SUPPORTED**.

**Windows Server 2003, Windows XP and Windows 2000:**

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. Call [GetLastError](#) to retrieve the error.

## Remarks

Beginning with Windows Vista, CMM Implementors are no longer required to implement this method.

**Windows Server 2003, Windows XP and Windows 2000:**

The Unicode version of this function is [CMCreateProfileW](#).

Only the Windows default CMM is required to export this function; it is optional for all other CMMs.

If a CMM does not support [CMCreateProfile](#), Windows uses the default CMM to create the profile.

The CMM should set all header fields to sensible defaults. This profile should be usable as the input profile in a transform.

The calling application must free the buffer allocated by this function and pointed to by the *lpProfileData* parameter. Use [GlobalFree](#) to free the buffer.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [CMCreateProfileW](#)

# CMCreateProfileW function (icm.h)

[CMCreateProfileW is no longer available for use as of Windows Vista.]

Creates a display color profile from a [LOGCOLORSPACEW](#) structure.

## Syntax

C++

```
BOOL CMCreateProfileW(  
    LPLOGCOLORSPACEW lpColorSpace,  
    LPDEVCHARACTER    *lpProfileData  
)
```

## Parameters

`lpColorSpace`

Pointer to a color logical space, of which the `IcsFilename` member will be **NULL**.

`lpProfileData`

Pointer to a pointer to a buffer. If successful the function allocates and fills this buffer. It is the calling application's responsibility to free this buffer when it is no longer needed.

## Return value

Beginning with Windows Vista, the default CMM (Icm32.dll) will return **FALSE** and [GetLastError](#) will report **ERROR\_NOT\_SUPPORTED**.

**Windows Server 2003, Windows XP and Windows 2000:**

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. Call [GetLastError](#) to retrieve the error.

## Remarks

Beginning with Windows Vista, CMM Implementors are no longer required to implement this method.

**Windows Server 2003, Windows XP and Windows 2000:**

The Unicode version of this function is [CMCreateProfileW](#).

Only the Windows default CMM is required to export this function; it is optional for all other CMMs.

If a CMM does not support `CMCreateProfileW`, Windows uses the default CMM to create the profile.

The CMM should set all header fields to sensible defaults. This profile should be usable as the input profile in a transform.

The calling application must free the buffer allocated by this function and pointed to by the `lpProfileData` parameter. Use [GlobalFree](#) to free the buffer.

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

Last updated on 10/31/2025

# CMCreateTransform function (icm.h)

Deprecated. There is no replacement API because this one was no longer being used.  
Developers of alternate CMM modules are not required to implement it.

## Syntax

C++

```
HCMTRANSFORM CMCreateTransform(
    LPLOGCOLORSPACEA lpColorSpace,
    LPDEVCHARACTER   lpDevCharacter,
    LPDEVCHARACTER   lpTargetDevCharacter
);
```

## Parameters

lpColorSpace

lpDevCharacter

lpTargetDevCharacter

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Icm32.Lib

Last updated on 10/31/2025

# CMCreateTransformExt function (icm.h)

Article 08/23/2022

Creates a color transform that maps from an input [LOGCOLORSPACEA](#) to an optional target space and then to an output device, using a set of flags that define how the transform should be created.

## Syntax

C++

```
HCMTRANSFORM CMCreateTransformExt(
    LPLOGCOLORSPACEA lpColorSpace,
    LPDEVCHARACTER   lpDevCharacter,
    LPDEVCHARACTER   lpTargetDevCharacter,
    DWORD             dwFlags
);
```

## Parameters

`lpColorSpace`

Pointer to an input logical color space structure.

`lpDevCharacter`

Pointer to a memory-mapped device profile.

`lpTargetDevCharacter`

Pointer to a memory-mapped target profile.

`dwFlags`

Specifies flags to used control creation of the transform. For details, see [CMM transform creation flags](#).

## Return value

If this function succeeds, the return value is a color transform in the range 256 to 65,535. Since only the low **WORD** of the transform is retained, valid transforms cannot exceed this range.

If this function fails, the return value is an error code having a value less than 256. When the return value is less than 256, signaling an error, the CMM should use **SetLastError** to set the last error to a valid error value as defined in Winerror.h.

## Remarks

The Unicode equivalent of **CMCreateTransformExt** is [CMCreateTransformExtW](#).

Every CMM is required to export this function.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [CMCreateTransformExtW](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CMCreateTransformExtW function (icm.h)

Creates a color transform that maps from an input [LOGCOLORSPACEW](#) to an optional target space and then to an output device, using a set of flags that define how the transform should be created.

## Syntax

C++

```
HCMTRANSFORM CMCreateTransformExtW(
    LPLOGCOLORSPACEW lpColorSpace,
    LPDEVCHARACTER   lpDevCharacter,
    LPDEVCHARACTER   lpTargetDevCharacter,
    DWORD             dwFlags
);
```

## Parameters

`lpColorSpace`

Pointer to an input logical color space structure.

`lpDevCharacter`

Pointer to a memory-mapped device profile.

`lpTargetDevCharacter`

Pointer to a memory-mapped target profile.

`dwFlags`

Specifies flags to used control creation of the transform. For details, see [CMM transform creation flags](#).

## Return value

If this function succeeds, the return value is a color transform in the range 256 to 65,535. Since only the low **WORD** of the transform is retained, valid transforms cannot exceed this range.

If this function fails, the return value is an error code having a value less than 256. When the return value is less than 256, signaling an error, the CMM should use [SetLastError](#) to set the

last error to a valid error value as defined in Winerror.h.

## Remarks

Every CMM is required to export this function.

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [CMCreateTransformExtW](#)

---

Last updated on 10/31/2025

# CMCreateTransformW function (icm.h)

Deprecated. There is no replacement API because this one was no longer being used.  
Developers of alternate CMM modules are not required to implement it.

## Syntax

C++

```
HCMTRANSFORM CMCreateTransformW(
    LPLOGCOLORSPACEW lpColorSpace,
    LPDEVCHARACTER   lpDevCharacter,
    LPDEVCHARACTER   lpTargetDevCharacter
);
```

## Parameters

lpColorSpace

lpDevCharacter

lpTargetDevCharacter

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Icm32.Lib

Last updated on 10/31/2025

# CMDeleteTransform function (icm.h)

Deletes a specified color transform, and frees any memory associated with it.

## Syntax

C++

```
BOOL CMDeleteTransform(  
    HCMTRANSFORM hcmTransform  
)
```

## Parameters

`hcmTransform`

Identifies the color transform to be deleted.

## Return value

If this function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. If the **CMDeleteTransform** function is not successful, the CMM should call **SetLastError** to set the last error to a valid error value defined in Winerror.h.

## Remarks

Every CMM is required to export this function.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

Requirement	Value
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

Last updated on 10/31/2025

# CMGetInfo function (icm.h)

Retrieves various information about the color management module (CMM).

Every CMM is required to export this function.

## Syntax

C++

```
DWORD CMGetInfo(  
    DWORD dwInfo  
) ;
```

## Parameters

`dwInfo`

Specifies what information should be retrieved. This parameter can take one of the following constant values.

 Expand table

Constant	Significance of the function's return value
CMM_DESCRIPTION	A text string that describes the color management module.
CMM_DLL_VERSION	Version number of the CMM DLL.
CMM_DRIVER_LEVEL	Driver compatibility information.
CMM_IDENT	The CMM identification signature registered with the International Color Consortium (ICC).
CMM_LOGOICON	The logo icon for this CMM.
CMM_VERSION	Version of Windows supported.
CMM_WIN_VERSION	Backward compatibility with Windows 95.

## Return value

If this function succeeds, the return value is the same nonzero value that was passed in through the `dwInfo` parameter. If the function fails, the return value is zero.

## Remarks

The **CMGetInfo** function can be called by applications directly to obtain information about the CMM. Applications should not call other CMM functions directly. To obtain CMM information, get the path to the CMM from the registry. Invoke the Windows API function [GetModuleHandle](#) and pass the file name of the CMM as the value of its parameter. Call the **CMGetInfo** function and pass it the constant **CMM\_DESCRIPTION** as the value of its parameter. Call the [LoadString](#) function. Pass the module handle as the first parameter, and the return value of the **CMGetInfo** function as the value of the second parameter.

CMMs that do not run on Windows 95 should return 0x0050000 for **CMM\_WIN\_VERSION**.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

Last updated on 10/31/2025

# CMGetNamedProfileInfo function (icm.h)

Retrieves information about the specified named color profile.

## Syntax

C++

```
BOOL CMGetNamedProfileInfo(  
    HPROFILE           hProfile,  
    PNAMED_PROFILE_INFO pNamedProfileInfo  
) ;
```

## Parameters

`hProfile`

The handle to the profile from which the information will be retrieved.

`pNamedProfileInfo`

A pointer to a **NAMED\_PROFILE\_INFO** structure.

## Return value

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE. When this occurs, the CMM should call **SetLastError** to set the last error to a valid error value defined in Winerror.h.

## Remarks

This function is required in the default CMM. It is optional for all other CMMs.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [NAMED\\_PROFILE\\_INFO](#)

---

Last updated on 10/31/2025

# CMGetPS2ColorRenderingDictionary function (icm.h)

Article02/22/2024

## Syntax

C++

```
BOOL CMGetPS2ColorRenderingDictionary(
    HPROFILE hProfile,
    DWORD     dwIntent,
    LPBYTE    lpBuffer,
    LPDWORD   lpcbSize,
    LPBOOL    lpbBinary
);
```

## Parameters

hProfile

dwIntent

lpBuffer

lpcbSize

lpbBinary

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CMGetPS2ColorRenderingIntent function (icm.h)

Article 02/22/2024

Retrieves the PostScript Level 2 color rendering intent from a profile.

## Syntax

C++

```
BOOL CMGetPS2ColorRenderingIntent(
    HPROFILE hProfile,
    DWORD     dwIntent,
    LPBYTE    lpBuffer,
    LPDWORD   lpcbSize
);
```

## Parameters

`hProfile`

Specifies the profile to use.

`dwIntent`

Specifies the desired rendering intent to retrieve. Can be one of the following values:

- INTENT\_PERCEPTUAL
- INTENT\_SATURATION
- INTENT\_RELATIVE\_COLORIMETRIC
- INTENT\_ABSOLUTE\_COLORIMETRIC

For more information, see [Rendering Intents](#).

`lpBuffer`

Points to a buffer in which the color rendering intent is to be placed. If the pointer is NULL, the function returns the size required for this buffer in `*lpcbSize`.

`lpcbSize`

Points to a variable specifying the size of the buffer. On return, the variable contains has the number of bytes actually copied to the buffer.

## Return value

If this function succeeds, the return value is TRUE. It also returns TRUE if it is called with *lpBuffer* set to NULL and the size of the required buffer is copied into *lpcbSize*.

If this function fails, the return value is FALSE. When this occurs, the CMM should call **SetLastError** to set the last error to a valid error value defined in Winerror.h.

## Remarks

This function is optional for all CMMs.

If a CMM does not support this function, Windows uses the default CMM to get the color rendering intent.

If the tag is not present in the profile indicated by *hProfile*, the CMM creates it. The resulting rendering intent can be used as the operand for the PostScript Level 2 **findcolorrendering** operator.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CMGetPS2ColorSpaceArray function (icm.h)

Article02/22/2024

## Syntax

C++

```
BOOL CMGetPS2ColorSpaceArray(
    HPROFILE hProfile,
    DWORD     dwIntent,
    DWORD     dwCSAType,
    LPBYTE    lpBuffer,
    LPDWORD   lpcbSize,
    LPBOOL    lpbBinary
);
```

## Parameters

hProfile

dwIntent

dwCSAType

lpBuffer

lpcbSize

lpbBinary

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	icm.h

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CMIsProfileValid function (icm.h)

Reports whether the given profile is a valid ICC profile that can be used for color management.

## Syntax

C++

```
BOOL CMIsProfileValid(
    HPROFILE hProfile,
    LPBOOL    lpbValid
);
```

## Parameters

`hProfile`

Specifies the profile to check.

`lpbValid`

Pointer to a variable that is set on exit to TRUE if the profile is a valid ICC profile, or FALSE if not.

## Return value

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE. If the function fails, the CMM should call `SetLastError` to set the last error to a valid error value defined in Winerror.h.

## Remarks

Only the Windows default CMM is required to export this function; it is optional for all other CMMs.

If a CMM does not support this function, Windows uses the default CMM to validate the profile.

## Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

Last updated on 10/31/2025

# CMTranslateColors function (icm.h)

Translates an array of colors from a source [color space](#) to a destination color space using a color transform.

## Syntax

C++

```
BOOL CMTranslateColors(
    HCMTRANSFORM hcmTransform,
    LPCOLOR     lpaInputColors,
    DWORD       nColors,
    COLORTYPE   ctInput,
    LPCOLOR     lpaOutputColors,
    COLORTYPE   ctOutput
);
```

## Parameters

`hcmTransform`

Specifies the color transform to use.

`lpaInputColors`

Points to an array of [COLOR](#) structures to translate.

`nColors`

Specifies the number of elements in the array.

`ctInput`

Specifies the color type of the input.

`lpaOutputColors`

Points to a buffer in which an array of translated [COLOR](#) structures is to be placed.

`ctOutput`

Specifies the output color type.

## Return value

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE. The CMM should call **SetLastError** to set the last error to a valid error value defined in Winerror.h.

## Remarks

Every CMM is required to export this function.

If the input and the output color types are not compatible with the color transform, this function should fail.

Note that this function must support in-place translation. That is, whenever the memory footprint of the output is less than or equal to the memory footprint of the input, this function must be able to translate the bitmap colors even if the source and destination buffers are the same.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

# CMTranslateRGB function (icm.h)

Translates an application-supplied RGBQuad into the device [color space](#).

## Syntax

C++

```
BOOL CMTranslateRGB(
    HCMTRANSFORM hcmTransform,
    COLORREF     ColorRef,
    LPCOLORREF   lpColorRef,
    DWORD        dwFlags
);
```

## Parameters

`hcmTransform`

Specifies the transform to be used.

`ColorRef`

The RGBQuad to translate.

`lpColorRef`

Points to a buffer in which to place the translation.

`dwFlags`

Specifies how the transform should be used to make the translation. This parameter can take one of the following meanings.

 [Expand table](#)

Value	Meaning
<code>CMS_FORWARD</code>	Use forward transform
<code>CMS_BACKWARD</code>	Use reverse transform

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. The CMM should call **SetLastError** to set the last error to a valid error value defined in **Winerror.h**.

## Remarks

Every CMM is required to export this function.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icm32.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

Last updated on 10/31/2025

# CMTranslateRGBs function (icm.h)

Article10/21/2021

[**CMTranslateRGBs** is no longer available for use as of Windows Vista.]

Translates a bitmap from one [color space](#) to another using a color transform.

## Syntax

C++

```
BOOL CMTranslateRGBs(
    HCMTRANSFORM hcmTransform,
    LPVOID        lpSrcBits,
    BMFORMAT      bmInput,
    DWORD         dwWidth,
    DWORD         dwHeight,
    DWORD         dwStride,
    LPVOID        lpDestBits,
    BMFORMAT      bmOutput,
    DWORD         dwTranslateDirection
);
```

## Parameters

**hcmTransform**

Specifies the color transform to use.

**lpSrcBits**

Points to the bitmap to translate.

**bmInput**

Specifies the input bitmap format.

**dwWidth**

Specifies the number of pixels per scan line in the input bitmap.

**dwHeight**

Specifies the number of scan lines in the input bitmap.

`dwStride`

Specifies the number of bytes from the beginning of one scan line to the beginning of the next in the input bitmap. If `dwStride` is set to zero, the CMM should assume that scan lines are padded so as to be **DWORD**-aligned.

`lpDestBits`

Points to a destination buffer in which to place the translated bitmap.

`bmOutput`

Specifies the output bitmap format.

`dwTranslateDirection`

Specifies the direction of the transform being used for the translation. This parameter must take one of the following values.

<b>Value</b>	<b>Meaning</b>
<code>CMS_FORWARD</code>	Use forward transform
<code>CMS_BACKWARD</code>	Use reverse transform

## Return value

Beginning with Windows Vista, the default CMM (`Icm32.dll`) will return **FALSE** and [GetLastError](#) will report `ERROR_NOT_SUPPORTED`.

**Windows Server 2003, Windows XP and Windows 2000:**

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. If the function is not successful, the CMM should call [SetLastError](#) to set the last error to a valid error value defined in `Winerror.h`.

## Remarks

Beginning with Windows Vista, CMM Implementors are no longer required to implement this method.

**Windows Server 2003, Windows XP and Windows 2000:**

Every CMM is required to export this function.

When writing into the destination buffer, the CMM should make sure that scan lines are padded to be **DWORD**-aligned.

If the input and output formats are not compatible with the color transform, this function fails.

If both input and output bitmap formats are 3-channel, 4 bytes-per-pixel as in the case of BM\_xRGBQUADS, the 4th byte should be preserved and copied to the output buffer.

Note that this function must support in-place translation. That is, whenever the memory footprint of the output is less than or equal to the memory footprint of the input, this function must be able to translate the bitmap colors even if the source and destination buffers are the same.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# CMTranslateRBGsExt function (icm.h)

Article 08/23/2022

Translates a bitmap from one defined format into a different defined format and calls a callback function periodically, if one is specified, to report progress and permit the calling application to terminate the translation.

## Syntax

C++

```
BOOL CMTranslateRBGsExt(
    HCMTRANSFORM    hcmTransform,
    LPVOID          lpSrcBits,
    BMFORMAT        bmInput,
    DWORD           dwWidth,
    DWORD           dwHeight,
    DWORD           dwInputStride,
    LPVOID          lpDestBits,
    BMFORMAT        bmOutput,
    DWORD           dwOutputStride,
    LPBMCALLBACKFN lpfnCallback,
    LPARAM          ulCallbackData
);
```

## Parameters

`hcmTransform`

Specifies the color transform to use.

`lpSrcBits`

Pointer to the bitmap to translate.

`bmInput`

Specifies the input bitmap format.

`dwWidth`

Specifies the number of pixels per scan line in the input bitmap.

`dwHeight`

Specifies the number of scan lines in the input bitmap.

`dwInputStride`

Specifies the number of bytes from the beginning of one scan line to the beginning of the next in the input bitmap. If *dwInputStride* is set to zero, the CMM should assume that scan lines are padded so as to be **DWORD**-aligned.

`lpDestBits`

Points to a destination buffer in which to place the translated bitmap.

`bmOutput`

Specifies the output bitmap format.

`dwOutputStride`

Specifies the number of bytes from the beginning of one scan line to the beginning of the next in the input bitmap. If *dwOutputStride* is set to zero, the CMM should pad scan lines so that they are **DWORD**-aligned.

`lpfnCallback`

Pointer to an application-supplied callback function called periodically by **CMTranslateRBGsExt** to report progress and allow the calling process to cancel the translation. (See [ICMProgressProcCallback](#).)

`ulCallbackData`

Data passed back to the callback function, for example to identify the translation that is reporting progress.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE** and the CMM should call **SetLastError** to set the last error to a valid error value defined in **Winerror.h**.

## Remarks

Every CMM is required to export this function.

When writing into the destination buffer, the CMM should make sure that scan lines are padded to be **DWORD**-aligned.

If the input and output formats are not compatible with the color transform, this function fails.

If both input and output bitmap formats are 3 channel, 4 bytes per pixel as in the case of BM\_xRGBQUADS, the fourth bytes should be preserved and copied to the output buffer.

If the callback function returns zero, processing should be cancelled and **CMTranslateRBGsExt** should return zero to indicate failure; the output buffer may be partially filled.

Note that this function must support in-place translation. That is, whenever the memory footprint of the output is less than or equal to the memory footprint of the input, this function must be able to translate the bitmap colors even if the source and destination buffers are the same.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [Basic color management concepts](#))
- [Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ColorCorrectPalette function (wingdi.h)

Article 02/22/2024

The **ColorCorrectPalette** function corrects the entries of a palette using the WCS 1.0 parameters in the specified device context.

## Syntax

C++

```
BOOL ColorCorrectPalette(  
    HDC      hdc,  
    HPALETTE hPal,  
    DWORD    deFirst,  
    DWORD    num  
) ;
```

## Parameters

hdc

Specifies a device context whose WCS parameters to use.

hPal

Specifies the handle to the palette to be color corrected.

deFirst

Specifies the first entry in the palette to be color corrected.

num

Specifies the number of entries to color correct.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ColorMatchToTarget function (wingdi.h)

Article04/02/2021

The **ColorMatchToTarget** function enables you to preview colors as they would appear on the target device.

## Syntax

C++

```
BOOL ColorMatchToTarget(
    HDC     hdc,
    HDC     hdcTarget,
    DWORD   action
);
```

## Parameters

hdc

Specifies the device context for previewing, generally the screen.

hdcTarget

Specifies the target device context, generally a printer.

action

A constant that can have one of the following values.

Value	Meaning
CS_ENABLE	Map the colors to the target device's color gamut. This enables color proofing. All subsequent draw commands to the DC will render colors as they would appear on the target device.
CS_DISABLE	Disable color proofing.
CS_DELETE_TRANSFORM	If color management is enabled for the target profile, disable it and delete the concatenated transform.

# Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**.

## Remarks

**ColorMatchToTarget** can be used to proof the colors of a color output device on another color output device. Setting the *uiAction* parameter to CS\_ENABLE causes all subsequent drawing commands to the DC to render colors as they would appear on the target device. If *uiAction* is set to CS\_DISABLE, proofing is turned off. However, the current color transform is not deleted from the DC. It is just inactive.

When **ColorMatchToTarget** is called, the color transform for the target device is performed first, and then the transform to the preview device is applied to the results of the first transform. This is used primarily for checking gamut mapping conditions. Before using this function, you must enable WCS for both device contexts.

This function cannot be cascaded. While color mapping to the target is enabled by setting *uiAction* to CS\_ENABLE, application changes to the color space or gamut mapping method are ignored. Those changes then take effect when color mapping to the target is disabled.

**Note** A memory leak will not occur if an application does not delete a transform using CS\_DELETE\_TRANSFORM. The transform will be deleted when either the device context (DC) is closed, or when the application color space is deleted. However if the transform is not going to be used again, or if the application will not be performing any more color matching on the DC, it should explicitly delete the transform to free the memory it occupies.

The *uiAction* parameter should only be set to CS\_DELETE\_TRANSFORM if color management is enabled before the **ColorMatchToTarget** function is called.

## Requirements

Minimum supported client

Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ColorProfileAddDisplayAssociation function (icm.h)

Associates an installed color profile with a specified display in the given scope.

## Syntax

C++

```
HRESULT ColorProfileAddDisplayAssociation(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PCWSTR                     profileName,
    LUID                        targetAdapterID,
    UINT32                      sourceID,
    BOOL                        setAsDefault,
    BOOL                        associateAsAdvancedColor
);
```

## Parameters

scope

Specifies the association as system-wide or the current user.

profileName

Identifies the installed profile to associate.

targetAdapterID

An identifier assigned to the adapter (e.g. GPU) of the target display. See [Remarks](#) for more details.

sourceID

An identifier assigned to the source of the display. See [Remarks](#) for more details.

setAsDefault

Whether or not to set the newly associated profile as the default.

associateAsAdvancedColor

Specifies to which association list the new profile is added.

# Return value

S\_OK for success, or a failure HRESULT value

## Remarks

See [connecting and configuring displays](#) for information on display adapter IDs and source IDs.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Mscms.Lib

## See also

[Connecting and configuring displays](#)

---

Last updated on 10/31/2025

# ColorProfileGetDisplayDefault function (icm.h)

Gets the default color profile for a given display in the specified scope.

## Syntax

C++

```
HRESULT ColorProfileGetDisplayDefault(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    LUID targetAdapterID,
    UINT32 sourceID,
    COLORPROFILETYPE profileType,
    COLORPROFILESUBTYPE profileSubType,
    LPWSTR *profileName
);
```

## Parameters

scope

Specifies the association as system-wide or the current user.

targetAdapterID

An identifier assigned to the adapter (e.g. GPU) of the target display. See [Remarks](#) for more details.

sourceID

An identifier assigned to the source of the display. See [Remarks](#) for more details.

profileType

The type of color profile to return (currently only CPT\_ICC is supported).

profileSubType

The subtype of the color profile to return.

profileName

Receives a pointer to the default color profile name, which must be freed with [LocalFree](#).

## Remarks

See [connecting and configuring displays](#) for information on display adapter IDs and source IDs.

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Mscms.Lib

## See also

[Connecting and configuring displays](#)

---

Last updated on 10/31/2025

# ColorProfileGetDisplayList function (icm.h)

Retrieves the list of profiles associated with a given display in the specified scope.

## Syntax

C++

```
HRESULT ColorProfileGetDisplayList(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    LUID                         targetAdapterID,
    UINT32                        sourceID,
    LPWSTR                         **profileList,
    PDWORD                         profileCount
);
```

## Parameters

scope

Specifies the association as system-wide or the current user.

targetAdapterID

An identifier assigned to the adapter (e.g. GPU) of the target display. See [Remarks](#) for more details.

sourceID

An identifier assigned to the source of the display. See [Remarks](#) for more details.

profileList

Pointer to a buffer where the profile names are placed, must be freed with [LocalFree](#).

profileCount

Receives the number of profiles names copied into profileList.

## Return value

[S\\_OK](#) for success, or a failure [HRESULT](#) value

## Remarks

See [connecting and configuring displays](#) for information on display adapter IDs and source IDs.

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Mscms.Lib

## See also

[Connecting and configuring displays](#)

---

Last updated on 10/31/2025

# ColorProfileGetDisplayUserScope function (icm.h)

Gets the currently selected color profile scope of the provided display - either user or system.

## Syntax

C++

```
HRESULT ColorProfileGetDisplayUserScope(
    LUID targetAdapterID,
    UINT32 sourceID,
    WCS_PROFILE_MANAGEMENT_SCOPE *scope
);
```

## Parameters

`targetAdapterID`

An identifier assigned to the adapter (e.g. GPU) of the target display. See [Remarks](#) for more details.

`sourceID`

An identifier assigned to the source of the display. See [Remarks](#) for more details.

`scope`

Returns the scope of the currently selected color profile - either the current user or system.

## Return value

`S_OK` for success, or a failure `HRESULT` value

## Remarks

See [Connecting and configuring displays](#) for information on display adapter IDs and source IDs.

## Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Mscms.Lib

## See also

[Connecting and configuring displays](#)

---

Last updated on 10/31/2025

# ColorProfileRemoveDisplayAssociation function (icm.h)

Disassociates an installed color profile from a specified display in the given scope.

## Syntax

C++

```
HRESULT ColorProfileRemoveDisplayAssociation(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PCWSTR                 profileName,
    LUID                   targetAdapterID,
    UINT32                 sourceID,
    BOOL                   dissociateAdvancedColor
);
```

## Parameters

scope

Specifies the association as system-wide or the current user.

profileName

Identifies the installed profile to associate.

targetAdapterID

An identifier assigned to the adapter (e.g. GPU) of the target display. See [Remarks](#) for more details.

sourceID

An identifier assigned to the source of the display. See [Remarks](#) for more details.

dissociateAdvancedColor

Specifies to which association list the new profile is added.

## Return value

S\_OK for success, or a failure HRESULT value

## Remarks

See [connecting and configuring displays](#) for information on display adapter IDs and source IDs.

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Mscms.Lib

## See also

[Connecting and configuring displays](#)

---

Last updated on 10/31/2025

# ColorProfileSetDisplayDefaultAssociation function (icm.h)

Sets an installed color profile as the default profile for a specified display in the given scope.

## Syntax

C++

```
HRESULT ColorProfileSetDisplayDefaultAssociation(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PCWSTR                     profileName,
    COLORPROFILETYPE            profileType,
    COLORPROFILESUBTYPE         profileSubType,
    LUID                        targetAdapterID,
    UINT32                      sourceID
);
```

## Parameters

scope

Specifies the association as system-wide or the current user.

profileName

Identifies the installed profile to associate.

profileType

The type of color profile to set as default (currently only CPT\_ICC is supported).

profileSubType

The subtype of the color profile to set as default.

targetAdapterID

An identifier assigned to the adapter (e.g. GPU) of the target display. See [Remarks](#) for more details.

sourceID

An identifier assigned to the source of the display. See [Remarks](#) for more details.

# Return value

S\_OK for success, or a failure HRESULT value

## Remarks

See [connecting and configuring displays](#) for information on display adapter IDs and source IDs.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h
Library	Mscms.Lib

## See also

[Connecting and configuring displays](#)

---

Last updated on 10/31/2025

# ConvertColorNameToIndex function (icm.h)

Article 02/22/2024

Converts color names in a named color space to index numbers in an International Color Consortium (ICC) color profile.

## Syntax

C++

```
BOOL ConvertColorNameToIndex(
    HPROFILE     hProfile,
    PCOLOR_NAME  paColorName,
    PDWORD       paIndex,
    DWORD        dwCount
);
```

## Parameters

`hProfile`

The handle to an ICC named color profile.

`paColorName`

Pointer to an array of color name structures.

`paIndex`

Pointer to an array of **DWORDs** that this function fills with the indices. The indices begin with one, not zero.

`dwCount`

The number of color names to convert.

## Return value

If this function succeeds with the conversion, the return value is **TRUE**.

If the conversion function fails, the return value is **FALSE**.

## Remarks

This function will fail if *hProfile* is not a valid ICC profile.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because named profiles are explicit ICC profile types.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# ConvertIndexToColorName function (icm.h)

Article02/22/2024

Transforms indices in a color space to an array of names in a named color space.

## Syntax

C++

```
BOOL ConvertIndexToColorName(
    HPROFILE     hProfile,
    PDWORD       paIndex,
    PCOLOR_NAME  paColorName,
    DWORD        dwCount
);
```

## Parameters

`hProfile`

The handle to an International Color Consortium (ICC) color space profile.

`paIndex`

Pointer to an array of color-space index numbers. The indices begin with one, not zero.

`paColorName`

Pointer to an array of color name structures.

`dwCount`

The number of indices to convert.

## Return value

If this conversion function succeeds, the return value is **TRUE**.

If this conversion function fails, the return value is **FALSE**.

# Remarks

This function will fail if *hProfile* is not a valid ICC profile.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because named profiles are explicit ICC profile types.

# Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CreateColorSpaceA function (wingdi.h)

Article 11/20/2024

The `CreateColorSpace` function creates a logical [color space](#).

## Syntax

C++

```
HCOLORSPACE CreateColorSpaceA(  
    LPLOGCOLORSPACEA lplcs  
) ;
```

## Parameters

`lplcs`

Pointer to the [LOGCOLORSPACE](#) data structure.

## Return value

If this function succeeds, the return value is a handle that identifies a color space.

If this function fails, the return value is `NULL`.

## Remarks

When the color space is no longer needed, use `DeleteColorSpace` to delete it.

**Windows 95/98/Me:** `CreateColorSpaceW` is supported by the Microsoft Layer for Unicode. To use this, you must add certain files to your application, as outlined in [Microsoft Layer for Unicode on Windows 95/98/Me Systems](#).

### Note

The wingdi.h header defines `CreateColorSpace` as an alias that automatically selects the ANSI or Unicode version of this function based on the definition of the `UNICODE` preprocessor constant. Mixing usage of the encoding-neutral alias with

code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

# Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [DeleteColorSpaceW](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# CreateColorTransformA function (icm.h)

Article 08/23/2022

Creates a color transform that applications can use to perform color management.

## Syntax

C++

```
HTRANSFORM CreateColorTransformA(  
    LPLOGCOLORSPACEA pLogColorSpace,  
    HPROFILE          hDestProfile,  
    HPROFILE          hTargetProfile,  
    DWORD             dwFlags  
) ;
```

## Parameters

`pLogColorSpace`

Pointer to the input [LOGCOLORSPACEA](#).

`hDestProfile`

Handle to the profile of the destination device. The function determines whether the HPROFILE contains International Color Consortium (ICC) or Windows Color System (WCS) profile information.

`hTargetProfile`

Handle to the profile of the target device. The function determines whether the HPROFILE contains ICC or WCS profile information.

`dwFlags`

Specifies flags to used control creation of the transform. See Remarks.

## Return value

If this function succeeds, the return value is a handle to the color transform.

If this function fails, the return value is **NULL**. For extended error information, call **GetLastError**.

## Remarks

If the target profile is **NULL**, the transform goes from the source logical color space to the destination profile. If the target profile is given, the transform goes from the source logical color space to the target profile and then to the destination profile. This allows previewing output meant for the target device on the destination device.

The values in *dwFlags* are intended as hints only. The color management module must determine the best way to use them.

**Windows Vista:** Three new flags have been added that can be used with *dwFlags*:

Flag	Description
<b>PRESERVEBLACK</b>	If this bit is set, the transform engine inserts the appropriate black generation GMMP as the last GMMP in the transform sequence. This flag only works in a pure WCS transform.
<b>SEQUENTIAL_TRANSFORM</b>	If this bit is set, each step in the WCS processing pipeline is performed for every pixel in the image and no optimized color transform is built. This flag only works in a pure WCS transform. <b>Restrictions:</b> A transform created with the <b>SEQUENTIAL_TRANSFORM</b> flag set may only be used in the thread on which it was created and only for one color translation call at a time. COM must be initialized prior to creating the sequential transform and must remain initialized for the lifetime of the transform object.
<b>WCS_ALWAYS</b>	If this bit is set, even all-ICC transforms will use the WCS code path.

### ⓘ Note

**SEQUENTIAL\_TRANSFORM** was inadvertently omitted from the icm.h header in the Windows Vista SDK. If you wish to use the **SEQUENTIAL\_TRANSFORM** flag, define it in your application as follows:`#define SEQUENTIAL_TRANSFORM 0x80800000`

For details, see [CMM Transform Creation Flags](#). All of the flags mentioned there are supported for all types of transforms, except for **FAST\_TRANSLATE**, which only works in a pure ICC-to-ICC transform.

The `CreateColorTransform` function is used outside of a device context. Colors may shift when transforming from a color profile to the same color profile. This is due to precision errors. Therefore, a color transform should not be performed under these circumstances.

The B2Ax tags are required for any profile that is the target of a transform.

WCS transform support for ICC ColorSpace profiles is limited to RGB colorspace profiles. The following ICC profile types cannot be used in a CITE-processed transform, either a mixed WCS/ICC transform or an all-ICC transform with `WCS_ALWAYS` set:

- Non-RGB ColorSpace profiles
- NamedColor profiles
- n-channel profiles (where  $n > 8$ )
- DeviceLink profiles
- Abstract profiles

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CreateColorTransformW function (icm.h)

Article 08/23/2022

Creates a color transform that applications can use to perform color management.

## Syntax

C++

```
HTRANSFORM CreateColorTransformW(
    LPLOGCOLORSPACEW pLogColorSpace,
    HPROFILE          hDestProfile,
    HPROFILE          hTargetProfile,
    DWORD             dwFlags
);
```

## Parameters

`pLogColorSpace`

Pointer to the input [LOGCOLORSPACEA](#).

`hDestProfile`

Handle to the profile of the destination device. The function determines whether the HPROFILE contains International Color Consortium (ICC) or Windows Color System (WCS) profile information.

`hTargetProfile`

Handle to the profile of the target device. The function determines whether the HPROFILE contains ICC or WCS profile information.

`dwFlags`

Specifies flags to used control creation of the transform. See Remarks.

## Return value

If this function succeeds, the return value is a handle to the color transform.

If this function fails, the return value is **NULL**. For extended error information, call **GetLastError**.

## Remarks

If the target profile is **NULL**, the transform goes from the source logical color space to the destination profile. If the target profile is given, the transform goes from the source logical color space to the target profile and then to the destination profile. This allows previewing output meant for the target device on the destination device.

The values in *dwFlags* are intended as hints only. The color management module must determine the best way to use them.

**Windows Vista:** Three new flags have been added that can be used with *dwFlags*:

Flag	Description
<b>PRESERVEBLACK</b>	If this bit is set, the transform engine inserts the appropriate black generation GMMP as the last GMMP in the transform sequence. This flag only works in a pure WCS transform.
<b>SEQUENTIAL_TRANSFORM</b>	If this bit is set, each step in the WCS processing pipeline is performed for every pixel in the image and no optimized color transform is built. This flag only works in a pure WCS transform. <b>Restrictions:</b> A transform created with the <b>SEQUENTIAL_TRANSFORM</b> flag set may only be used in the thread on which it was created and only for one color translation call at a time. COM must be initialized prior to creating the sequential transform and must remain initialized for the lifetime of the transform object.
<b>WCS_ALWAYS</b>	If this bit is set, even all-ICC transforms will use the WCS code path.

### ⓘ Note

**SEQUENTIAL\_TRANSFORM** was inadvertently omitted from the icm.h header in the Windows Vista SDK. If you wish to use the **SEQUENTIAL\_TRANSFORM** flag, define it in your application as follows:`#define SEQUENTIAL_TRANSFORM 0x80800000`

For details, see [CMM Transform Creation Flags](#). All of the flags mentioned there are supported for all types of transforms, except for **FAST\_TRANSLATE**, which only works in a pure ICC-to-ICC transform.

The `CreateColorTransform` function is used outside of a device context. Colors may shift when transforming from a color profile to the same color profile. This is due to precision errors. Therefore, a color transform should not be performed under these circumstances.

The B2Ax tags are required for any profile that is the target of a transform.

WCS transform support for ICC ColorSpace profiles is limited to RGB colorspace profiles. The following ICC profile types cannot be used in a CITE-processed transform, either a mixed WCS/ICC transform or an all-ICC transform with `WCS_ALWAYS` set:

- Non-RGB ColorSpace profiles
- NamedColor profiles
- n-channel profiles (where  $n > 8$ )
- DeviceLink profiles
- Abstract profiles

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CreateDeviceLinkProfile function (icm.h)

Article 10/05/2021

Creates an International Color Consortium (ICC) *device link profile* from a set of color profiles, using the specified intents.

## Syntax

C++

```
BOOL CreateDeviceLinkProfile(
    PHPROFILE hProfile,
    DWORD     nProfiles,
    PDWORD    padwIntent,
    DWORD     nIntents,
    DWORD     dwFlags,
    PBYTE    *pProfileData,
    DWORD     indexPreferredCMM
);
```

## Parameters

`hProfile`

Pointer to an array of handles of the color profiles to be used. The function determines whether the HPROFILEs contain ICC profile information and, if so, it processes them appropriately.

`nProfiles`

Specifies the number of profiles in the array pointed to by *hProfile*.

`padwIntent`

Pointer to an array of **DWORDS** containing the intents to be used. See [Rendering intents](#).

`nIntents`

The number of intents in the array pointed to by *padwIntent*.

`dwFlags`

Specifies flags to used control creation of the transform. For details, see [CMM Transform Creation Flags](#).

`pProfileData`

Pointer to a pointer to a buffer. If successful, this function allocates the buffer, places its address in `*pProfileData`, and fills it with a device link profile. If the function succeeds, the calling application must free the buffer after it is no longer needed.

`indexPreferredCMM`

Specifies the one-based index of the color profile that indicates what color management module (CMM) to use. The application developer may allow Windows to choose the CMM by setting this parameter to INDEX\_DONT\_CARE. See [Using Color Management Modules \(CMM\)](#).

## Return value

If this function succeeds, the return value is a nonzero value.

If this function fails, the return value is zero. For extended error information, call `GetLastError`.

## Remarks

For HPROFILEs that contain WCS profile information, the HPROFILEs are converted into valid ICC profile handles and then these ICC profile handles are used in creating the device link profile.

The first and the last profiles in the array must be device profiles. The other profiles can be color space or abstract profiles.

Each profile's output color space must be the next profile's input color space.

The calling application must free the buffer allocated by this function and pointed to by the `pProfileData` parameter. The `GlobalFree` function should be used to free the buffer.

## Requirements

Minimum supported client

Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [GlobalFree](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CreateMultiProfileTransform function (icm.h)

Article 08/23/2022

Accepts an array of profiles or a single [device link profile](#) and creates a color transform that applications can use to perform color mapping.

## Syntax

C++

```
HTRANSFORM CreateMultiProfileTransform(
    PHPROFILE pahProfiles,
    DWORD      nProfiles,
    PDWORD     padwIntent,
    DWORD      nIntents,
    DWORD      dwFlags,
    DWORD      indexPreferredCMM
);
```

## Parameters

**pahProfiles**

Pointer to an array of handles to the profiles to be used. The function determines whether the HPROFILEs contain International Color Consortium (ICC) or Windows Color System (WCS) profile information and processes them appropriately. When valid WCS profiles are returned by [OpenColorProfileW](#) and [WcsOpenColorProfileW](#), these profile handles contain the combination of DMP, CAMP, and GMMP profiles.

**nProfiles**

Specifies the number of profiles in the array. The maximum is 10.

**padwIntent**

Pointer to an array of intents to use. Each intent is one of the following values:

**INTENT\_PERCEPTUAL**

**INTENT\_SATURATION**

## `INTENT_RELATIVE_COLORIMETRIC`

## `INTENT_ABSOLUTE_COLORIMETRIC`

GMMPs are a generalization of intents. There are two possible sources of intents: the "destination" profile and the intent list parameter to [CreateMultiProfileTransform](#). The term "destination" is not used since all but two of the profiles in the profile list parameter will serve as first destination and then source.

For more information, see [Rendering Intents](#).

### `nIntents`

Specifies the number of elements in the intents array: can either be 1 or the same value as `nProfiles`. For profile arrays that contain any WCS profiles, the first rendering intent is ignored and only `nProfiles` - 1 elements are used for these profile arrays. The maximum number of `nIntents` is 10.

### `dwFlags`

Specifies flags used to control creation of the transform. See Remarks.

### `indexPreferredCMM`

Specifies the one-based index of the color profile that indicates what color management module (CMM) to use. The application developer may allow Windows to choose the CMM by setting this parameter to INDEX\_DONT\_CARE. See [Using Color Management Modules \(CMM\)](#) Third party CMMs are only available for ICC workflows. Profile arrays containing WCS profiles will ignore this flag. It is also ignored when only ICC profiles are used and when the WCS\_ALWAYS flag is used.

## Return value

If this function succeeds, the return value is a handle to the color transform.

If this function fails, the return value is `NULL`. For extended error information, call [GetLastError](#).

## Remarks

If a device link profile is being used, the function will fail if `nProfiles` is not set to 1.

The array of intents specifies how profiles should be combined. The `nth` intent is used for combining the `nth` profile in the array. If only one intent is specified, it is used for the

first profile, and all other profiles are combined using [Match intent](#).

The values in *dwFlags* are intended as hints only. The color management module must determine the best way to use them.

**Windows Vista:** Three new flags have been added that can be used with *dwFlags*:

Flag	Description
PRESERVEBLACK	If this bit is set, the transform engine inserts the appropriate black generation GMMP as the last GMMP in the transform sequence. This flag only works in a pure WCS transform.
SEQUENTIAL_TRANSFORM	If this bit is set, each step in the WCS processing pipeline is performed for every pixel in the image and no optimized color transform is built. This flag only works in a pure WCS transform. <b>Restrictions:</b> A transform created with the SEQUENTIAL_TRANSFORM flag set may only be used in the thread on which it was created and only for one color translation call at a time. COM must be initialized prior to creating the sequential transform and must remain initialized for the lifetime of the transform object.
WCS_ALWAYS	If this bit is set, even all-ICC transforms will use the WCS code path.

#### ① Note

SEQUENTIAL\_TRANSFORM was inadvertently omitted from the icm.h header in the Windows Vista SDK. If you wish to use the SEQUENTIAL\_TRANSFORM flag, define it in your application as follows:

```
#define SEQUENTIAL_TRANSFORM 0x80800000
```

For details, see [CMM Transform Creation Flags](#). All of the flags mentioned there are supported for all types of transforms, except for FAST\_TRANSLATE and USE\_RELATIVE\_COLORIMETRIC, which only work in a pure ICC-to-ICC transform.

The **CreateMultiProfileTransform** function is used outside of a device context. Colors may shift when transforming from a color profile to the same color profile. This is due to precision errors. Therefore, a color transform should not be performed under these circumstances.

We recommend that there be only one GMMP between a source and destination DMP. Gamut boundary descriptions (GBDs) are created from the DMP/CAMP combinations. The subsequent GMMPs use the GDBs prior to them in the processing chain until there

exists a DMP/CAMP GBD next in the sequence to be used. For example, assume a sequence DMP1, CAMP1, GMMP1, GMMP2, GMMP3, DMP2, CAMP2, GMMP4, GMMP5, CAMP3, DMP3. Then GMMP1, GMMP2 use GBD1 as their source and destination. Then GMMP3 uses GBD1 as source and GBD2 as destination. Then GMMP4 uses GBD2 as source and destination. Finally GMMP5 uses GBD2 as source and GBD3 as destination. This assumes no GMMP is identical to one next to it.

For WCS profiles, we recommend that the rendering intents be set to DWORD\_MAX in order to use the GMMP within the WCS profile handle. This is because the array of rendering intents takes precedence over the rendering intents or gamut mapping models specified or contained in the profiles specified by the HPROFILEs. The array of rendering intents references the default GMMP for those rendering intents. Ideally, only one gamut mapping is performed between a source and destination device by setting one or the other GMMP to **NULL** when creating the HPROFILE with WCS profile information. Any legacy application that uses a WCS DMP will invoke a sequence of GMMPs. GDBs are chosen based on DMPs and CAMPs. For intermediate GMMP gamut boundaries, the source and destination GBDs are used.

In summary, if *nIntents* == 1, then the first GMM is set based on the GMMP that is set as default\* for the *padwIntent* value, unless that value is DWORD\_MAX, in which case the embedded GMM information from the second profile is used (The embedded GMM information is either a GMMP or, in the case of an ICC profile, the baseline GMM corresponding to\*\* the intent from the profile header). The remainder of the GMMs are set based on the GMMP that is set as default\* for RelativeColorimetric.

If *nIntents* = *nProfiles* -1, then each GMM is set based on the GMMP that is set as default\* for the value in the *padwIntent* array at the corresponding index, except where *padwIntent* values are DWORD\_MAX. For values in the *padwIntent* array that are DWORD\_MAX, the GMMs at corresponding positions are set based on the embedded GMM information from the second of the two profiles whose gamuts are mapped by the GMM. (Again, the embedded GMM information is either a GMMP or, in the case of an ICC profile, the baseline GMM corresponding to\*\* the intent from the profile header).

If *nIntents* = *nProfiles*, then first intent is ignored and function behaves as it does in the case when *nIntents* = *nProfiles* -1.

Any other combination of *padwIntents* and *nIntents* will return an error.

\* "set as default" means that the default GMMP is queried using **WcsGetDefaultColorProfile** with its *profileManagementScope* parameter set to **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER**. This may return either current-user or system-wide defaults as described in the documentation for **WcsGetDefaultColorProfile**.

\*\* "GMM corresponding to" does not mean "GMM from the GMMP set as default for". Instead it means "a constant association between ICC profile intents and baseline GMM algorithms."

WCS transform support for ICC ColorSpace profiles is limited to RGB colorspace profiles. The following ICC profile types cannot be used in a CITE-processed transform, either a mixed WCS/ICC transform or an all-ICC transform with **WCS\_ALWAYS** set:

- Non-RGB ColorSpace profiles
- NamedColor profiles
- n-channel profiles (where n > 8)
- DeviceLink profiles
- Abstract profiles

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [COLOR Structure\\*\\*](#)
- [DeleteColorTransform](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CreateProfileFromLogColorSpaceA function (icm.h)

Article 02/22/2024

Converts a logical [color space](#) to a [device profile](#).

## Syntax

C++

```
BOOL CreateProfileFromLogColorSpaceA(
    LPLOGCOLORSPACEA pLogColorSpace,
    PBYTE             *pProfile
);
```

## Parameters

`pLogColorSpace`

A pointer to a logical color space structure. See [LOGCOLORSPACEA](#) for details. The `IcsFilename` [0] member of the structure must be set to the `null` character ('\0') or this function call will fail with the return value of `INVALID_PARAMETER`.

`pProfile`

A pointer to a pointer to a buffer where the device profile will be created. This function allocates the buffer and fills it with profile information if it is successful. If not, the pointer is set to `NULL`. The caller is responsible for freeing this buffer when it is no longer needed.

## Return value

If this function succeeds, the return value is `TRUE`.

If this function fails, the return value is `FALSE`.

If the `IcsFilename` [0] member if the [LOGCOLORSPACEA](#) structure pointed to by `pLogColorSpace` is not '\0', this function returns `INVALID_PARAMETER`.

## Remarks

This function can be used with ASCII or Unicode strings. The buffer created by this function must be freed by the caller when it is no longer needed or there will be a memory leak. The [GlobalFree](#) function should be used to free this buffer.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP.

## Requirements

  [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [GlobalFree](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# CreateProfileFromLogColorSpaceW function (icm.h)

Article 02/22/2024

Converts a logical [color space](#) to a [device profile](#).

## Syntax

C++

```
BOOL CreateProfileFromLogColorSpaceW(
    LPLOGCOLORSPACEW pLogColorSpace,
    PBYTE           *pProfile
);
```

## Parameters

`pLogColorSpace`

A pointer to a logical color space structure. See [LOGCOLORSPACEA](#) for details. The `IcsFilename` [0] member of the structure must be set to the `null` character ('\0') or this function call will fail with the return value of `INVALID_PARAMETER`.

`pProfile`

A pointer to a pointer to a buffer where the device profile will be created. This function allocates the buffer and fills it with profile information if it is successful. If not, the pointer is set to `NULL`. The caller is responsible for freeing this buffer when it is no longer needed.

## Return value

If this function succeeds, the return value is `TRUE`.

If this function fails, the return value is `FALSE`.

If the `IcsFilename` [0] member if the [LOGCOLORSPACEA](#) structure pointed to by `pLogColorSpace` is not '\0', this function returns `INVALID_PARAMETER`.

## Remarks

This function can be used with ASCII or Unicode strings. The buffer created by this function must be freed by the caller when it is no longer needed or there will be a memory leak. The [GlobalFree](#) function should be used to free this buffer.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP.

## Requirements

  [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [GlobalFree](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# DeleteColorSpace function (wingdi.h)

Article02/22/2024

The [DeleteColorSpace](#) function removes and destroys a specified color space.

## Syntax

C++

```
BOOL DeleteColorSpace(  
    HCOLORSPACE hcs  
) ;
```

## Parameters

`hcs`

Specifies the handle to a color space to delete.

## Return value

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- Basic color management concepts
  - Functions
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# DeleteColorTransform function (icm.h)

Article02/22/2024

Deletes a given color transform.

## Syntax

C++

```
BOOL DeleteColorTransform(  
    HTRANSFORM hxform  
) ;
```

## Parameters

`hxform`

Identifies the color transform to delete.

## Return value

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE. For extended error information, call [GetLastError](#).

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
  - [Functions](#)
  - [COLOR structure](#)
  - [CreateMultiProfileTransform](#)
- 

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# DisassociateColorProfileFromDeviceA function (icm.h)

Article07/27/2022

Disassociates a specified color profile with a specified device on a specified computer.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileRemoveDisplayAssociation](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL DisassociateColorProfileFromDeviceA(  
    PCSTR pMachineName,  
    PCSTR pProfileName,  
    PCSTR pDeviceName  
) ;
```

## Parameters

pMachineName

Reserved. Must be **NULL**. This parameter is intended to point to the name of the computer on which to disassociate the specified profile and device. A **NULL** pointer indicates the local computer.

pProfileName

Pointer to the file name of the profile to disassociate.

pDeviceName

Pointer to the name of the device to disassociate.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Remarks

If more than one profile is associated with a device, WCS uses the last one associated as the default. That is, if your application sequentially associates three profiles with a device, WCS will use the last one associated as the default. If your application then calls the **DisassociateColorProfileFromDevice** function to disassociate the third profile (which is the default in this example), the WCS will use the second profile as the default.

If your application disassociates all profiles from a device, WCS uses the sRGB profile as the default.

**DisassociateColorProfileFromDevice** always removes the specified profile from the current user's per-user profile association list for the specified device. Before removing the profile from the list, **DisassociateColorProfileFromDevice** determines whether the user has previously expressed the desire to use a per-user profile association list for the device. If so, then **DisassociateColorProfileFromDevice** simply removes the specified profile from the existing per-user profile association list for the device. If not, then **DisassociateColorProfileFromDevice** creates a new per-user profile association list for the device by copying the system-wide association list for that device. It then removes the specified profile from the per-user list. From that point on, the current user will be using a per-user profile association list for the specified device, as if **WcsSetUsePerUserProfiles** had been called for *pDevice* with the *usePerUserProfiles* parameter set to **TRUE**.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
  - Functions
  - AssociateColorProfileWithDeviceA
- 

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# DisassociateColorProfileFromDeviceW function (icm.h)

Article07/27/2022

Disassociates a specified color profile with a specified device on a specified computer.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileRemoveDisplayAssociation](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL DisassociateColorProfileFromDeviceW(
    PCWSTR pMachineName,
    PCWSTR pProfileName,
    PCWSTR pDeviceName
);
```

## Parameters

pMachineName

Reserved. Must be **NULL**. This parameter is intended to point to the name of the computer on which to disassociate the specified profile and device. A **NULL** pointer indicates the local computer.

pProfileName

Pointer to the file name of the profile to disassociate.

pDeviceName

Pointer to the name of the device to disassociate.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Remarks

If more than one profile is associated with a device, WCS uses the last one associated as the default. That is, if your application sequentially associates three profiles with a device, WCS will use the last one associated as the default. If your application then calls the **DisassociateColorProfileFromDevice** function to disassociate the third profile (which is the default in this example), the WCS will use the second profile as the default.

If your application disassociates all profiles from a device, WCS uses the sRGB profile as the default.

**DisassociateColorProfileFromDevice** always removes the specified profile from the current user's per-user profile association list for the specified device. Before removing the profile from the list, **DisassociateColorProfileFromDevice** determines whether the user has previously expressed the desire to use a per-user profile association list for the device. If so, then **DisassociateColorProfileFromDevice** simply removes the specified profile from the existing per-user profile association list for the device. If not, then **DisassociateColorProfileFromDevice** creates a new per-user profile association list for the device by copying the system-wide association list for that device. It then removes the specified profile from the per-user list. From that point on, the current user will be using a per-user profile association list for the specified device, as if **WcsSetUsePerUserProfiles** had been called for *pDevice* with the *usePerUserProfiles* parameter set to **TRUE**.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
  - Functions
  - AssociateColorProfileWithDeviceW
- 

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# EnumColorProfilesA function (icm.h)

Enumerates all the profiles satisfying the given enumeration criteria.

## Syntax

C++

```
BOOL EnumColorProfilesA(
    PCSTR      pMachineName,
    PENUMTYPEA  pEnumRecord,
    PBYTE       pEnumerationBuffer,
    PDWORD     pdwSizeOfEnumerationBuffer,
    PDWORD     pnProfiles
);
```

## Parameters

`pMachineName`

Reserved. Must be **NULL**. This parameter is intended to point to the name of the computer on which to enumerate profiles. A **NULL** pointer indicates the local computer.

`pEnumRecord`

Pointer to a structure specifying the enumeration criteria.

`pEnumerationBuffer`

Pointer to a buffer in which the profiles are to be enumerated. A `MULTI_SZ` string of profile names satisfying the criteria specified in `*pEnumRecord` will be placed in this buffer.

`pdwSizeOfEnumerationBuffer`

Pointer to a variable containing the size of the buffer pointed to by `pBuffer`. On return, `*pdwSize` contains the size of buffer actually used or needed.

`pnProfiles`

Pointer to a variable that will contain, on return, the number of profile names actually copied to the buffer.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

Several profiles are typically associated with printers, based on the paper and ink types. There is a default profile for each device. For International Color Consortium (ICC) profiles, GDI selects the best one from the ICC-associated profiles when your application creates a device context (DC).

Do not attempt to use **EnumColorProfiles** to determine the default profile for a device. Instead, create a device context for the device and then invoke the [GetICMProfile](#) function. On Windows Vista and Windows 7, the [WcsGetDefaultColorProfile](#) function can also be used to determine a device's default color profile.

If the **dwFields** member of the structure of type **ENUMTYPE** that is pointed to by the *pEnumRecord* parameter is set to **ET\_DEVICENAME**, this function will enumerate all of the color profiles associated with all types of devices attached to the user's computer, regardless of the device class. If the **dwFields** member of the structure pointed to by the *pEnumRecord* parameter is set to **ET\_DEVICENAME** or **ET\_DEVICECLASS** and a device class is specified in the **dwDeviceClass** member of the structure, this function will only enumerate the profiles associated with the specified device class. If the **dwFields** member is set only to **ET\_DEVICECLASS**, the **EnumColorProfiles** function will enumerate all profiles that can be associated with that type of device.

Whenever **EnumColorProfiles** is examining the profiles associated with a specific device, the results depend on whether the user has chosen to use the system-wide list of profiles associated with that device, or his or her own ("per-user") list. Calling [WcsSetUsePerUserProfiles](#) with its *usePerUserProfiles* parameter set to **TRUE** causes future calls to **EnumColorProfiles** to look at only the current user's per-user list of profile associations for the specified device. Calling [WcsSetUsePerUserProfiles](#) with its *usePerUserProfiles* parameter set to **FALSE** causes future calls to **EnumColorProfiles** to look at the system-wide list of profile associations for the specified device. If [WcsSetUsePerUserProfiles](#) has never been called for the current user, **EnumColorProfiles** examines the system-wide list.

Your application can use **EnumColorProfiles** to obtain the size of the buffer in which the profiles are enumerated. It should call the **EnumColorProfiles** function with the *pBuffer* parameter set to **NULL**. When the function returns, the *pdwSize* parameter will contain the required buffer size in bytes. Your program can use that information to allocate the enumeration buffer. It can then invoke **EnumColorProfiles** again with the *pBuffer* parameter set to the address of the buffer.

This function will provide the information for converting WCS-specific DMP information to the legacy EnumType record in enable consistent profile enumeration. The defaults will be the same as ICC if this information is not present.

## Per-user/LUA support

The enumeration is specific to current user. Both system wide and current user device associations are considered. For default profile configuration, current user settings override system wide ones.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [GetICMProfile](#)
- [ENUMTYPEW](#)

---

Last updated on 10/31/2025

# EnumColorProfilesW function (icm.h)

Enumerates all the profiles satisfying the given enumeration criteria.

## Syntax

C++

```
BOOL EnumColorProfilesW(
    PCWSTR      pMachineName,
    PENUMTYPEW   pEnumRecord,
    PBYTE       pEnumerationBuffer,
    PDWORD     pdwSizeOfEnumerationBuffer,
    PDWORD     pnProfiles
);
```

## Parameters

`pMachineName`

Reserved. Must be **NULL**. This parameter is intended to point to the name of the computer on which to enumerate profiles. A **NULL** pointer indicates the local computer.

`pEnumRecord`

Pointer to a structure specifying the enumeration criteria.

`pEnumerationBuffer`

Pointer to a buffer in which the profiles are to be enumerated. A **MULTI\_SZ** string of profile names satisfying the criteria specified in `*pEnumRecord` will be placed in this buffer.

`pdwSizeOfEnumerationBuffer`

Pointer to a variable containing the size of the buffer pointed to by `pBuffer`. On return, `*pdwSize` contains the size of buffer actually used or needed.

`pnProfiles`

Pointer to a variable that will contain, on return, the number of profile names actually copied to the buffer.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

Several profiles are typically associated with printers, based on the paper and ink types. There is a default profile for each device. For International Color Consortium (ICC) profiles, GDI selects the best one from the ICC-associated profiles when your application creates a device context (DC).

Do not attempt to use **EnumColorProfiles** to determine the default profile for a device. Instead, create a device context for the device and then invoke the [GetICMProfile](#) function. On Windows Vista and Windows 7, the [WcsGetDefaultColorProfile](#) function can also be used to determine a device's default color profile.

If the **dwFields** member of the structure of type **ENUMTYPE** that is pointed to by the *pEnumRecord* parameter is set to **ET\_DEVICENAME**, this function will enumerate all of the color profiles associated with all types of devices attached to the user's computer, regardless of the device class. If the **dwFields** member of the structure pointed to by the *pEnumRecord* parameter is set to **ET\_DEVICENAME** or **ET\_DEVICECLASS** and a device class is specified in the **dwDeviceClass** member of the structure, this function will only enumerate the profiles associated with the specified device class. If the **dwFields** member is set only to **ET\_DEVICECLASS**, the **EnumColorProfiles** function will enumerate all profiles that can be associated with that type of device.

Whenever **EnumColorProfiles** is examining the profiles associated with a specific device, the results depend on whether the user has chosen to use the system-wide list of profiles associated with that device, or his or her own ("per-user") list. Calling [WcsSetUsePerUserProfiles](#) with its *usePerUserProfiles* parameter set to **TRUE** causes future calls to **EnumColorProfiles** to look at only the current user's per-user list of profile associations for the specified device. Calling [WcsSetUsePerUserProfiles](#) with its *usePerUserProfiles* parameter set to **FALSE** causes future calls to **EnumColorProfiles** to look at the system-wide list of profile associations for the specified device. If [WcsSetUsePerUserProfiles](#) has never been called for the current user, **EnumColorProfiles** examines the system-wide list.

Your application can use **EnumColorProfiles** to obtain the size of the buffer in which the profiles are enumerated. It should call the **EnumColorProfiles** function with the *pBuffer* parameter set to **NULL**. When the function returns, the *pdwSize* parameter will contain the required buffer size in bytes. Your program can use that information to allocate the enumeration buffer. It can then invoke **EnumColorProfiles** again with the *pBuffer* parameter set to the address of the buffer.

This function will provide the information for converting WCS-specific DMP information to the legacy EnumType record in enable consistent profile enumeration. The defaults will be the same as ICC if this information is not present.

## Per-user/LUA support

The enumeration is specific to current user. Both system wide and current user device associations are considered. For default profile configuration, current user settings override system wide ones.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.Lib

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [GetICMProfile](#)
- [ENUMTYPEW](#)

---

Last updated on 10/31/2025

# EnumICMProfilesA function (wingdi.h)

Article02/09/2023

The **EnumICMProfiles** function enumerates the different output color profiles that the system supports for a given device context.

## Syntax

C++

```
int EnumICMProfilesA(
    HDC         hdc,
    ICMENUMPROCA proc,
    LPARAM      param
);
```

## Parameters

hdc

Specifies the device context.

proc

Specifies the procedure instance address of a callback function defined by the application. (See [EnumICMProfilesProcCallback](#).)

param

Data supplied by the application that is passed to the callback function along with the color profile information.

## Return value

This function returns zero if the application interrupted the enumeration. The return value is -1 if there are no color profiles to enumerate. Otherwise, the return value is the last value returned by the callback function.

## Remarks

The **EnumICMProfiles** function returns a list of profiles that are associated with a device context (DC), and whose settings match those of the DC. It is possible for a device context to contain device profiles that are not associated with particular hardware devices, or device profiles that do not match the settings of the DC. The sRGB profile is an example. The [SetICMProfile](#) function is used to associate these types of profiles with a DC. The [GetICMProfile](#) function can be used to retrieve a profile that is not enumerated by the **EnumICMProfiles** function.

**Windows 95/98/Me:**[EnumICMProfilesW](#) is supported by the Microsoft Layer for Unicode. To use this, you must add certain files to your application, as outlined in [Microsoft Layer for Unicode on Windows 95/98/Me Systems](#).

 **Note**

The wingdi.h header defines **EnumICMProfiles** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [DeleteColorSpaceW](#)
- [ICMENUMPROCA callback function](#)

- [GetICMProfileW](#)
  - [SetICMProfileW](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# GetCMMInfo function (icm.h)

Article02/22/2024

Retrieves various information about the color management module (CMM) that created the specified color transform.

## Syntax

C++

```
DWORD GetCMMInfo(
    HTRANSFORM hColorTransform,
    DWORD       unnamedParam2
);
```

## Parameters

`hColorTransform`

Identifies the transform for which to find CMM information.

`unnamedParam2`

Specifies the information to be retrieved. This parameter can take one of the following constant values.

[ ] Expand table

Value	Meaning
<code>CMM_WIN_VERSION</code>	Retrieves the version of Windows targeted by the color management module (CMM).
<code>CMM_DLL_VERSION</code>	Retrieves the version number of the CMM.
<code>CMM_IDENT</code>	Retrieves the CMM signature registered with the International Color Consortium (ICC).

## Return value

If this function succeeds, the return value is the information specified in *dwInfo*.

If this function fails, the return value is zero.

# Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetColorDirectoryA function (icm.h)

Article02/22/2024

## ⓘ Note

This API may be unavailable in future releases. We encourage new and existing software to use other APIs for color profile interactions. Please refer to the below table for some examples.

  Expand table

Scenario	Mechanism
Enumerating all installed profiles	Use <a href="#">WcsEnumColorProfilesSize</a> and <a href="#">WcsEnumColorProfiles</a> , or <a href="#">EnumColorProfilesA</a>
Installing/Uninstalling color profiles	Use <a href="#">InstallColorProfileA</a> / <a href="#">UninstallColorProfileA</a>
Opening a color profile file directly	Use <a href="#">OpenColorProfileA</a> with dwType=PROFILE_FILENAME in the PROFILE struct parameter. Or use <a href="#">WcsOpenColorProfileA</a> . <a href="#">Icm.h</a> contains many APIs that accept the returned HPROFILE for color profile manipulation

Retrieves the path of the Windows COLOR directory on a specified machine.

## Syntax

C++

```
BOOL GetColorDirectoryA(
    PCSTR pMachineName,
    PSTR pBuffer,
    PDWORD pdwSize
);
```

## Parameters

pMachineName

Reserved; must be **NULL**. This parameter is intended to point to the name of the machine on which the profile is to be installed. A **NULL** pointer indicates the local machine.

#### pBuffer

Points to the buffer in which the color directory path is to be placed.

#### pdwSize

Points to a variable containing the size in bytes of the buffer pointed to by *pBuffer*. On return, the variable contains the size of the buffer actually used or needed.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Remarks

### Per-user/LUA support

Color directory is still system-wide. This function is executable in LUA context.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
  - Functions
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetColorDirectoryW function (icm.h)

Article02/22/2024

## ⓘ Note

This API may be unavailable in future releases. We encourage new and existing software to use other APIs for color profile interactions. Please refer to the below table for some examples.

  Expand table

Scenario	Mechanism
Enumerating all installed profiles	Use <a href="#">WcsEnumColorProfilesSize</a> and <a href="#">WcsEnumColorProfiles</a> , or <a href="#">EnumColorProfilesW</a>
Installing/Uninstalling color profiles	Use <a href="#">InstallColorProfileW</a> / <a href="#">UninstallColorProfileW</a>
Opening a color profile file directly	Use <a href="#">OpenColorProfileW</a> with dwType=PROFILE_FILENAME in the PROFILE struct parameter. Or use <a href="#">WcsOpenColorProfileW</a> . <a href="#">Icm.h</a> contains many APIs that accept the returned HPROFILE for color profile manipulation

Retrieves the path of the Windows COLOR directory on a specified machine.

## Syntax

C++

```
BOOL GetColorDirectoryW(
    PCWSTR pMachineName,
    PWSTR pBuffer,
    PDWORD pdwSize
);
```

## Parameters

pMachineName

Reserved; must be **NULL**. This parameter is intended to point to the name of the machine on which the profile is to be installed. A **NULL** pointer indicates the local machine.

#### pBuffer

Points to the buffer in which the color directory path is to be placed.

#### pdwSize

Points to a variable containing the size in bytes of the buffer pointed to by *pBuffer*. On return, the variable contains the size of the buffer actually used or needed.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Remarks

### Per-user/LUA support

Color directory is still system-wide. This function is executable in LUA context.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
  - Functions
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetColorProfileElement function (icm.h)

Article 10/05/2021

Copies data from a specified tagged profile element of a specified color profile into a buffer.

## Syntax

C++

```
BOOL GetColorProfileElement(
    HPROFILE hProfile,
    TAGTYPE tag,
    DWORD dwOffset,
    PDWORD pcbElement,
    PVOID pElement,
    PBOOL pbReference
);
```

## Parameters

`hProfile`

Specifies a handle to the International Color Consortium (ICC) color profile in question.

`tag`

Identifies the tagged element from which to copy.

`dwOffset`

Specifies the offset from the first byte of the tagged element data at which to begin copying.

`pcbElement`

Pointer to a variable specifying the number of bytes to copy. On return, the variable contains the number of bytes actually copied.

`pElement`

Pointer to a buffer into which the tagged element data is to be copied. The buffer must contain at least as many bytes as are specified by the variable pointed to by `pcbSize`. If

If the *pBuffer* pointer is set to **NULL**, the size of the entire tagged element data in bytes is returned in the memory location pointed to by *pcbSize*, and *dwOffset* is ignored. In this case, the function will return **FALSE**.

#### pbReference

Points to a Boolean value that is set to **TRUE** if more than one tag in the color profile refers to the same data as the specified tag refers to, or **FALSE** if not.

## Return value

If this function succeeds, the return value is nonzero.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

This function will fail if *hProfile* is not a valid International Color Consortium (ICC) profile.

If the *pBuffer* pointer is set to **NULL**, the size of the entire tagged element data in bytes is returned in the variable pointed to by *pcbSize*, and *dwOffset* is ignored.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because profile elements are implicitly associated with, and hard coded to, ICC tag types and there exist many robust XML parsing libraries.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
  - Functions
- 

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# GetColorProfileElementTag function (icm.h)

Article02/22/2024

Retrieves the tag name specified by *dwIndex* in the tag table of a given International Color Consortium (ICC) color profile, where *dwIndex* is a one-based index into that table.

## Syntax

C++

```
BOOL GetColorProfileElementTag(  
    HPROFILE hProfile,  
    DWORD     dwIndex,  
    PTAGTYPE  pTag  
) ;
```

## Parameters

`hProfile`

Specifies a handle to the ICC color profile in question.

`dwIndex`

Specifies the one-based index of the tag to retrieve.

`pTag`

Pointer to a variable in which the tag name is to be placed.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Remarks

This function will fail if *hProfile* is not a valid ICC profile.

**GetColorProfileElementTag** can be used to enumerate all tags in a profile after getting the number of tags in the profile using [GetCountColorProfileElements](#).

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because profile elements are implicitly associated with, and hard coded to, ICC tag types and there exist many robust XML parsing libraries.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetColorProfileFromHandle function (icm.h)

Article02/22/2024

Given a handle to an open color profile, the **GetColorProfileFromHandle** function copies the contents of the profile into a buffer supplied by the application. If the handle is a Windows Color System (WCS) handle, then the DMP is returned and the CAMP and GMMP associated with the HPROFILE are ignored.

## Syntax

C++

```
BOOL GetColorProfileFromHandle(
    HPROFILE hProfile,
    PBYTE    pProfile,
    PDWORD   pcbProfile
);
```

## Parameters

`hProfile`

Handle to an open color profile. The function determines whether the HPROFILE contains ICC or WCS profile information.

`pProfile`

Pointer to buffer to receive raw ICC or DMP profile data. Can be **NULL**. If it is, the size required for the buffer will be stored in the memory location pointed to by *pcbSize*. The buffer can be allocated to the appropriate size, and this function called again with *pBuffer* containing the address of the buffer.

`pcbProfile`

Pointer to a **DWORD** that holds the size of buffer pointed at by *pBuffer*. On return it is filled with size of buffer that was actually used if the function succeeds. If this function is called with *pBuffer* set to **NULL**, this parameter will contain the size of the buffer required.

# Return value

If this function succeeds, the return value is **TRUE**. It returns **FALSE** if the *pBuffer* parameter is **NULL** and the size required for the buffer is copied into *pcbSize*.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetColorProfileHeader function (icm.h)

Article02/22/2024

Retrieves or derives ICC header structure from either ICC color profile or WCS XML profile. Drivers and applications should assume returning **TRUE** only indicates that a properly structured header is returned. Each tag will still need to be validated independently using either legacy ICM2 APIs or XML schema APIs.

## Syntax

C++

```
BOOL GetColorProfileHeader(
    HPROFILE      hProfile,
    PPROFILEHEADER pHeader
);
```

## Parameters

`hProfile`

Specifies a handle to the color profile in question.

`pHeader`

Points to a variable in which the ICC header structure is to be placed.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. This function will fail if an invalid ICC or WCS XML profile is referenced in the `hProfile` parameter. For extended error information, call `GetLastError`.

## Remarks

To determine whether the header is derived from an ICC or DMP profile handle, check the header signature (header bytes 36-39). If the signature is "acsp" (big endian) then an ICC profile was used. If the signature is "cdmp" (big-endian) then a DMP was used.

The distinguishing features that identify a header as having been "synthesized" for a WCS DMP are:

plcmProfileHeader->phSignature = 'pmdc' (little endian = big endian 'cdmp')

plcmProfileHeader->phCMMType = '1scw' (little endian = big endian 'wcs1').

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [PROFILEHEADER](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetColorSpace function (wingdi.h)

Article04/02/2021

The **GetColorSpace** function retrieves the handle to the input color space from a specified device context.

## Syntax

C++

```
HCOLORSPACE GetColorSpace(  
    HDC hdc  
) ;
```

## Parameters

hdc

Specifies a device context that is to have its input color space handle retrieved.

## Return value

If the function succeeds, the return value is the current input color space handle.

If this function fails, the return value is **NULL**.

## Remarks

**GetColorSpace** obtains the handle to the input color space regardless of whether color management is enabled for the device context.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# GetCountColorProfileElements function (icm.h)

Article 02/22/2024

Retrieves the number of tagged elements in a given color profile.

## Syntax

C++

```
BOOL GetCountColorProfileElements(
    HPROFILE hProfile,
    PDWORD    pnElementCount
);
```

## Parameters

`hProfile`

Specifies a handle to the profile in question.

`pnElementCount`

Pointer to a variable in which to place the number of tagged elements in the profile.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

This function will fail if *hProfile* is not a valid ICC profile.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# GetDeviceGammaRamp function (wingdi.h)

Article 09/23/2022

The **GetDeviceGammaRamp** function gets the [gamma ramp](#) on direct color display boards having drivers that support downloadable gamma ramps in hardware.

## ⓘ Important

We strongly recommend that you don't use this API. Use of this API is subject to major limitations. See [SetDeviceGammaRamp](#) for more information.

## Syntax

C++

```
BOOL GetDeviceGammaRamp(
    HDC     hdc,
    LPVOID lpRamp
);
```

## Parameters

(hdc)

Specifies the device context of the direct color display board in question.

(lpRamp)

Points to a buffer where the function can place the current gamma ramp of the color display board. The gamma ramp is specified in three arrays of 256 **WORD** elements each, which contain the mapping between RGB values in the frame buffer and digital-analog-converter (DAC) values. The sequence of the arrays is red, green, blue.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**.

# Example

C++

```
WORD gArray[3][256];
GetDeviceGammaRamp(handle, gArray);
// `handle` is the device context. See GetDC for more details.
// `gArray` will hold the gamma array values in a 2-D array
```

## Remarks

Direct color display modes do not use color lookup tables and are usually 16, 24, or 32 bit. Not all direct color video boards support loadable gamma ramps.

**GetDeviceGammaRamp** succeeds only for devices with drivers that support downloadable gamma ramps in hardware.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# GetICMProfileA function (wingdi.h)

Article11/20/2024

The **GetICMProfile** function retrieves the file name of the current output color profile for a specified device context.

## Syntax

C++

```
BOOL GetICMProfileA(
    HDC     hdc,
    LPDWORD pBufSize,
    LPSTR   pszFilename
);
```

## Parameters

hdc

Specifies a device context from which to retrieve the color profile.

pBufSize

Pointer to a **DWORD** that contains the size of the buffer pointed to by *lpszFilename*. For the ANSI version of this function, the size is in bytes. For the Unicode version, the size is in WCHARs. If this function is successful, on return this parameter contains the size of the buffer actually used. However, if the buffer is not large enough, this function returns **FALSE**. In this case, the **GetLastError()** function returns **ERROR\_INSUFFICIENT\_BUFFER** and the **DWORD** pointed to by this parameter contains the size needed for the *lpszFilename* buffer.

pszFilename

Points to the buffer that receives the path name of the profile.

## Return value

If this function succeeds, the return value is **TRUE**. It also returns **TRUE** if the *lpszFilename* parameter is **NULL** and the size required for the buffer is copied into *lpchName*.

If this function fails, the return value is **FALSE**.

## Remarks

**GetICMProfile** obtains the file name of the current output profile regardless of whether or not color management is enabled for the device context.

Given a device context, **GetICMProfile** will output, through the parameter *lpszFilename*, the path name of the file containing the color profile currently being used by the device context. It will also output, through the parameter *lpcbName*, the length of the string containing the path name.

It is possible that the profile name returned by **GetICMProfile** will not be in the list of profiles returned by **EnumICMProfiles**. The **EnumICMProfiles** function returns all color space profiles that are associated with a device context (DC) whose settings match that of the DC. If the **SetICMProfile** function is used to set the current profile, a profile may be associated with the DC that does not match its settings. For instance, the **SetICMProfile** function can be used to associate the device-independent sRGB profile with a DC. This profile will be used as the current WCS profile for that DC, and calls to **GetICMProfile** will return its file name. However, the profile will not appear in the list of profiles that is returned from **EnumICMProfiles**.

If this function is called before any calls to the **SetICMProfile** function, it can be used to get the default profile for a device context.

**Windows 95/98/Me:** **GetICMProfileW** is supported by the Microsoft Layer for Unicode. To use this, you must add certain files to your application, as outlined in [Microsoft Layer for Unicode on Windows 95/98/Me Systems](#) ↗.

### ⓘ Note

The wingdi.h header defines **GetICMProfile** as an alias that automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [DeleteColorSpaceW](#)
- [ICMENUMPROCA callback function](#)
- [EnumICMProfilesW](#)
- [SetICMProfileW](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

# GetLogColorSpaceA function (wingdi.h)

Article11/20/2024

The **GetLogColorSpace** function retrieves the [color space](#) definition identified by a specified handle.

## Syntax

C++

```
BOOL GetLogColorSpaceA(
    HCOLORSPACE     hColorSpace,
    LPLOGCOLORSPACEA lpBuffer,
    DWORD           nSize
);
```

## Parameters

`hColorSpace`

Specifies the handle to a color space.

`lpBuffer`

Points to a buffer to receive the [LOGCOLORSPACE](#) structure.

`nSize`

Specifies the maximum size of the buffer.

## Return value

If this function succeeds, the return value is TRUE.

If this function fails, the return value is FALSE.

## Remarks

**Windows 95/98/Me:** `GetLogColorSpaceW` is supported by the Microsoft Layer for Unicode. To use this, you must add certain files to your application, as outlined in [Microsoft Layer for Unicode on Windows 95/98/Me Systems](#).

## Note

The wingdi.h header defines GetLogColorSpace as an alias that automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetNamedProfileInfo function (icm.h)

Article02/22/2024

Retrieves information about the International Color Consortium (ICC) named color profile that is specified in the first parameter.

## Syntax

C++

```
BOOL GetNamedProfileInfo(  
    HPROFILE           hProfile,  
    PNAMED_PROFILE_INFO pNamedProfileInfo  
) ;
```

## Parameters

`hProfile`

The handle to the ICC profile from which the information will be retrieved.

`pNamedProfileInfo`

A pointer to a `NAMED_PROFILE_INFO` structure.

## Return value

If this function succeeds, the return value is `TRUE`.

If this function fails, the return value is `FALSE`.

## Remarks

This function will fail if `hProfile` is not a valid ICC profile.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because named profiles are explicit ICC profile types.

## Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Gdi32.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [NAMED\\_PROFILE\\_INFO](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetPS2ColorRenderingDictionary function (icm.h)

Article 02/22/2024

Retrieves the PostScript Level 2 color rendering dictionary from the specified ICC color profile.

## Syntax

C++

```
BOOL GetPS2ColorRenderingDictionary(
    HPROFILE hProfile,
    DWORD     dwIntent,
    PBYTE     pPS2ColorRenderingDictionary,
    PDWORD    pcbPS2ColorRenderingDictionary,
    PBOOL     pbBinary
);
```

## Parameters

`hProfile`

Specifies a handle to the ICC color profile in question.

`dwIntent`

Specifies the desired rendering intent for the color rendering dictionary. Valid values are:

- INTENT\_PERCEPTUAL
- INTENT\_SATURATION
- INTENT\_RELATIVE\_COLORIMETRIC
- INTENT\_ABSOLUTE\_COLORIMETRIC

For more information, see [Rendering intents](#).

`pPS2ColorRenderingDictionary`

Pointer to a buffer in which the color rendering dictionary is to be placed. If the `pBuffer` pointer is set to **NULL**, the required buffer size is returned in `*pcbSize`.

`pcbPS2ColorRenderingDictionary`

Pointer to a variable containing the size of the buffer in bytes. On return, the variable contains the number of bytes actually copied.

#### pbBinary

Pointer to a Boolean variable. If **TRUE**, the color rendering dictionary could be copied in binary form. If **FALSE**, the dictionary will be encoded in ASCII85 form. On return, this Boolean variable indicates whether the dictionary was actually binary (**TRUE**) or ASCII85 (**FALSE**).

## Return value

If this function succeeds, the return value is **TRUE**. It also returns **TRUE** if the *pBuffer* parameter is **NULL** and the size required for the buffer is copied into *pcbSize*.

If this function fails, the return value is **FALSE**.

## Remarks

If the dictionary is not available in the profile, the **GetPS2ColorRenderingDictionary** function builds one using the profile contents. This dictionary can then be used as the operand for the PostScript Level 2 **setcolorrendering** operator.

This method does not support WCS profiles.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
  - Functions
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetPS2ColorRenderingIntent function (icm.h)

Article 02/22/2024

Retrieves the PostScript Level 2 color [rendering intent](#) from an ICC color profile.

## Syntax

C++

```
BOOL GetPS2ColorRenderingIntent(
    HPROFILE hProfile,
    DWORD     dwIntent,
    PBYTE     pBuffer,
    PDWORD    pcbPS2ColorRenderingIntent
);
```

## Parameters

`hProfile`

Specifies a handle to the ICC color profile in question.

`dwIntent`

Specifies the desired rendering intent to retrieve. Valid values are:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering Intents](#).

`pBuffer`

Points to a buffer in which the color rendering intent is to be placed. If the `pBuffer` pointer is set to `NULL`, the buffer size required is returned in `*pcbSize`.

`pcbPS2ColorRenderingIntent`

Points to a variable containing the buffer size in bytes. On return, this variable contains the number of bytes actually copied.

## Return value

If this function succeeds, the return value is **TRUE**. If this function succeeds, the return value is **TRUE**. It also returns **TRUE** if the *pBuffer* parameter is **NULL** and the size required for the buffer is copied into *pcbSize*.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

The rendering intent returned by [GetPS2ColorRenderingIntent](#) can be used as the operand for the PostScript Level 2 *findcolorrendering* operator.

This method does not support WCS profiles.

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GetPS2ColorSpaceArray function (icm.h)

Article 10/05/2021

Retrieves the PostScript Level 2 [color space](#) array from an ICC color profile.

## Syntax

C++

```
BOOL GetPS2ColorSpaceArray(
    HPROFILE hProfile,
    DWORD     dwIntent,
    DWORD     dwCSAType,
    PBYTE     pPS2ColorSpaceArray,
    PDWORD    pcbPS2ColorSpaceArray,
    PBOOL     pbBinary
);
```

## Parameters

`hProfile`

Specifies a handle to the ICC profile from which to retrieve the PostScript Level 2 color space array.

`dwIntent`

Specifies the desired rendering intent for the color space array. This field may take one of the following values:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering Intents](#).

`dwCSAType`

Specifies the type of color space array. See [Color Space Type Identifiers](#).

`pPS2ColorSpaceArray`

Pointer to a buffer in which the color space array is to be placed. If the *pBuffer* pointer is set to **NULL**, the function returns the required size of the buffer in the memory location pointed to by *pcbSize*.

`pcbPS2ColorSpaceArray`

Pointer to a variable containing the size of the buffer in bytes. On return, it contains the number of bytes copied into the buffer.

`pbBinary`

Pointer to a Boolean variable. If set to **TRUE**, the data copied could be binary. If set to **FALSE**, data should be encoded as ASCII85. On return, the memory location pointed to by *pbBinary* indicates whether the data returned actually is binary (**TRUE**) or ASCII85 (**FALSE**).

## Return value

If this function succeeds, the return value is **TRUE**. It also returns **TRUE** if the *pBuffer* parameter is **NULL** and the size required for the buffer is copied into *pcbSize*.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Remarks

If the color space array is not available in the profile, the **GetPS2ColorSpaceArray** function builds a PostScript Level 2 color space array using the profile contents. This array can then be used as the operand for the PostScript Level2 `setcolorspace` operator.

This method does not support WCS profiles.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# GetStandardColorSpaceProfileA function (icm.h)

Article07/27/2022

Retrieves the color profile registered for the specified standard [color space](#).

## Syntax

C++

```
BOOL GetStandardColorSpaceProfileA(
    PCSTR pMachineName,
    DWORD dwSCS,
    PSTR pBuffer,
    PDWORD pcbSize
);
```

## Parameters

pMachineName

Reserved. Must be **NULL**. This parameter is intended to point to the name of the computer on which to get a standard color space profile. A **NULL** pointer indicates the local machine.

dwSCS

Specifies the ID value of the standard color space for which to retrieve the profile. The only valid values for this parameter are **LCS\_sRGB** and **LCS\_WINDOWS\_COLOR\_SPACE**.

pBuffer

Pointer to the buffer in which the name of the profile is to be placed. If **NULL**, the call will return **TRUE** and the required size of the buffer is placed in *pdwSize*.

pcbSize

Pointer to a variable containing the size in bytes of the buffer pointed to by *pProfileName*. On return, the variable contains the size of the buffer actually used or needed.

# Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

If the buffer pointed to by *pProfileName* is to be dynamically allocated by an application, the application can call the [GetStandardColorSpaceProfile](#) function to retrieve the size required for the buffer. If [GetStandardColorSpaceProfile](#) is called with *pProfileName* set to **NULL**, it will return **FALSE** and the **DWORD** pointed at by *pdwSize* will contain the number of bytes needed for the buffer pointed at by *pProfileName*. The application can then allocate the buffer and call [GetStandardColorSpaceProfile](#) again with *pProfileName* set to the address of the buffer.

This function supports Windows Color System (WCS) device model profiles (DMPs) in addition to International Color Consortium (ICC) profiles. It does not support WCS CAMP or GMMP profiles and will return an error if such profiles are used.

### Overview of Windows Vista Specific Functionality

This will support WCS DMPs in addition to ICC profiles. It will not support WCS CAMP or GMMP profiles and will return an error if such profiles are used with this API.

#### *Per-user/LUA support*

This will retrieve the color profile registered for the given standard color space for current user. If there is no such setting for the current user, it retrieves the system wide setting.

This uses [WcsGetDefaultColorProfile](#) with  
`WCS_PROFILE_MANAGEMENT_SCOPE_CURRENT_USER`.

This is executable in LUA context.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [SetStandardColorSpaceProfile](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# GetStandardColorSpaceProfileW function (icm.h)

Article 07/27/2022

Retrieves the color profile registered for the specified standard [color space](#).

## Syntax

C++

```
BOOL GetStandardColorSpaceProfileW(
    PCWSTR pMachineName,
    DWORD dwSCS,
    PWSTR pBuffer,
    PDWORD pcbSize
);
```

## Parameters

`pMachineName`

Reserved. Must be **NULL**. This parameter is intended to point to the name of the computer on which to get a standard color space profile. A **NULL** pointer indicates the local machine.

`dwSCS`

Specifies the ID value of the standard color space for which to retrieve the profile. The only valid values for this parameter are `LCS_sRGB` and `LCS_WINDOWS_COLOR_SPACE`.

`pBuffer`

Pointer to the buffer in which the name of the profile is to be placed. If **NULL**, the call will return **TRUE** and the required size of the buffer is placed in `pdwSize`.

`pcbSize`

Pointer to a variable containing the size in bytes of the buffer pointed to by `pProfileName`. On return, the variable contains the size of the buffer actually used or needed.

# Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

If the buffer pointed to by *pProfileName* is to be dynamically allocated by an application, the application can call the [GetStandardColorSpaceProfile](#) function to retrieve the size required for the buffer. If [GetStandardColorSpaceProfile](#) is called with *pProfileName* set to **NULL**, it will return **FALSE** and the **DWORD** pointed at by *pdwSize* will contain the number of bytes needed for the buffer pointed at by *pProfileName*. The application can then allocate the buffer and call [GetStandardColorSpaceProfile](#) again with *pProfileName* set to the address of the buffer.

This function supports Windows Color System (WCS) device model profiles (DMPs) in addition to International Color Consortium (ICC) profiles. It does not support WCS CAMP or GMMP profiles and will return an error if such profiles are used.

### Overview of Windows Vista Specific Functionality

This will support WCS DMPs in addition to ICC profiles. It will not support WCS CAMP or GMMP profiles and will return an error if such profiles are used with this API.

#### *Per-user/LUA support*

This will retrieve the color profile registered for the given standard color space for current user. If there is no such setting for the current user, it retrieves the system wide setting.

This uses [WcsGetDefaultColorProfile](#) with  
`WCS_PROFILE_MANAGEMENT_SCOPE_CURRENT_USER`.

This is executable in LUA context.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
--------------------------	---

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [SetStandardColorSpaceProfile](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# InstallColorProfileA function (icm.h)

Article02/22/2024

Installs a given profile for use on a specified machine. The profile is also copied to the COLOR directory.

## Syntax

C++

```
BOOL InstallColorProfileA(  
    PCSTR pMachineName,  
    PCSTR pProfileName  
) ;
```

## Parameters

pMachineName

Reserved. Must be **NULL**. This parameter is intended to point to the name of the computer on which the profile is to be installed. A **NULL** pointer indicates the local computer.

pProfileName

Pointer to the fully qualified path name of the profile to install.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# InstallColorProfileW function (icm.h)

Article02/22/2024

Installs a given profile for use on a specified machine. The profile is also copied to the COLOR directory.

## Syntax

C++

```
BOOL InstallColorProfileW(  
    PCWSTR pMachineName,  
    PCWSTR pProfileName  
) ;
```

## Parameters

pMachineName

Reserved. Must be **NULL**. This parameter is intended to point to the name of the computer on which the profile is to be installed. A **NULL** pointer indicates the local computer.

pProfileName

Pointer to the fully qualified path name of the profile to install.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# IsColorProfileTagPresent function (icm.h)

Article02/22/2024

Reports whether a specified International Color Consortium (ICC) tag is present in the specified color profile.

## Syntax

C++

```
BOOL IsColorProfileTagPresent(
    HPROFILE hProfile,
    TAGTYPE tag,
    PBOOL    pbPresent
);
```

## Parameters

`hProfile`

Specifies a handle to the ICC profile in question.

`tag`

Specifies the ICC tag to check.

`pbPresent`

Pointer to a variable that is set to **TRUE** on return if the specified ICC tag is present, or **FALSE** if not.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Remarks

This function will fail if *hProfile* is not a valid ICC profile.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because profile elements are implicitly associated with and hard coded to ICC tag types and there exist many robust XML parsing libraries.

## Requirements

  Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# IsColorProfileValid function (icm.h)

Article 02/22/2024

Allows you to determine whether the specified profile is a valid International Color Consortium (ICC) profile, or a valid Windows Color System (WCS) profile handle that can be used for color management. WCS profile validation doesn't invoke the underlying device models, but instead simply validates against the XML schema and the schema element range limits.

## Syntax

C++

```
BOOL IsColorProfileValid(
    HPROFILE hProfile,
    PBOOL     pbValid
);
```

## Parameters

`hProfile`

Specifies a handle to the profile to be validated. The function determines whether the HPROFILE contains ICC or WCS profile information.

`pbValid`

Pointer to a variable that is set to **TRUE** on return if the operation succeeds and the profile is a valid ICC or WCS profile. If the operation fails or the profile is not valid the variable is **FALSE**.

## Return value

If this function succeeds and the profile is valid, the return value is **TRUE**.

If this function fails (or succeeds and the profile is not valid), the return value is **FALSE**. For extended error information, call **GetLastError**.

## Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# OpenColorProfileA function (icm.h)

Article 07/27/2022

Creates a handle to a specified color profile. The handle can then be used in other profile management functions.

## Syntax

C++

```
HPROFILE OpenColorProfileA(
    PPROFILE pProfile,
    DWORD     dwDesiredAccess,
    DWORD     dwShareMode,
    DWORD     dwCreationMode
);
```

## Parameters

pProfile

Pointer to a color profile structure specifying the profile. The *pProfile* pointer can be freed as soon as the handle is created.

dwDesiredAccess

Specifies how to access the given profile. This parameter must take one of the following constant values.

Value	Meaning
PROFILE_READ	Opens the profile for read access.
PROFILE_READWRITE	Opens the profile for both read and write access. Has no effect for WCS XML profiles.

dwShareMode

Specifies how the profile should be shared, if the profile is contained in a file. A value of zero prevents the profile from being shared at all. The parameter can contain one or both of the following constants (combined by addition or logical OR).

Value	Meaning
FILE_SHARE_READ	Other open operations can be performed on the profile for read access.
FILE_SHARE_WRITE	Other open operations can be performed on the profile for write access. Has no effect for WCS XML profiles.

#### dwCreationMode

Specifies which actions to take on the profile while opening it, if it is contained in a file. This parameter must take one of the following constant values.

Value	Meaning
CREATE_NEW	Creates a new profile. Fails if the profile already exists.
CREATE_ALWAYS	Creates a new profile. Overwrites the profile if it exists.
OPEN_EXISTING	Opens the profile. Fails if it does not exist
OPEN_ALWAYS	Opens the profile if it exists. For ICC profiles, if the profile does not exist, creates the profile. For WCS XML profiles, if the profile does not exist, returns an error.
TRUNCATE_EXISTING	Opens the profile, and truncates it to zero bytes, returning a blank ICC profile. Fails if the profile doesn't exist.

## Return value

If this function succeeds, the return value is the handle of the color profile that is opened. For ICC and WCS profiles, a CAMP and GMMP are provided by the function based on the current default CAMP and GMMP in the registry.

When OpenColorProfile encounters an ICC profile with an embedded WCS profile, and if the dwType member within the Profile structure does not take the value DONT\_USE\_EMBEDDED\_WCS\_PROFILES, it should extract and use the WCS profile(s) contained in this WcsProfilesTag. The HPROFILE returned would be a WCS HPROFILE.

If this function fails, the return value is **NULL**. For extended error information, call **GetLastError**.

## Remarks

If the profile data is not specified using a file name, *dwShareMode* and *dwCreationMode* are ignored.

*dwCreationMode* flags CREATE\_NEW, CREATE\_ALWAYS, and TRUNCATE\_EXISTING, will always return blank ICC HPROFILEs. If other *dwCreationMode* flags are present, InternalOpenColorProfile is called (using the flags as provided by the API) to determine whether the profile is ICC or WCS XML.

Within the ICC code path, an ICC HPROFILE is returned using the requested sharing, access and creation flags as specified in the tables above.

Within the WCS path, the *dwCreationMode* flag OPEN\_ALWAYS will fail if the profile doesn't exist, since WCS profiles cannot be created or edited within the WCS architecture (they must be edited outside of it, using MSXML6). For the same reason, *dwShareMode* flag FILE\_SHARE\_WRITE, and *dwDesiredAccess* flag PROFILE\_READWRITE are ignored within the WCS path.

When the function opens the ICC profile, it will look for a *WcsProfilesTag* and, if there is one, it will extract and use the original WCS profiles contained therein. (See [WcsCreateIccProfile](#).)

An HPROFILE with WCS profile information is derived from a DMP by acquiring the default CAMP and default GMMP from the registry. An HPROFILE is a composition of a DMP, CAMP and GMMP.

Once the handle to the color profile is created, any information used to create that handle can be deleted.

Use the [CloseColorProfile](#) function to close an object handle returned by [OpenColorProfile](#).

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
  - Functions
  - CloseColorProfile
  - PROFILE
- 

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# OpenColorProfileW function (icm.h)

Article 07/27/2022

Creates a handle to a specified color profile. The handle can then be used in other profile management functions.

## Syntax

C++

```
HPROFILE OpenColorProfileW(
    PPROFILE pProfile,
    DWORD     dwDesiredAccess,
    DWORD     dwShareMode,
    DWORD     dwCreationMode
);
```

## Parameters

pProfile

Pointer to a color profile structure specifying the profile. The *pProfile* pointer can be freed as soon as the handle is created.

dwDesiredAccess

Specifies how to access the given profile. This parameter must take one of the following constant values.

Value	Meaning
PROFILE_READ	Opens the profile for read access.
PROFILE_READWRITE	Opens the profile for both read and write access. Has no effect for WCS XML profiles.

dwShareMode

Specifies how the profile should be shared, if the profile is contained in a file. A value of zero prevents the profile from being shared at all. The parameter can contain one or both of the following constants (combined by addition or logical OR).

Value	Meaning
FILE_SHARE_READ	Other open operations can be performed on the profile for read access.
FILE_SHARE_WRITE	Other open operations can be performed on the profile for write access. Has no effect for WCS XML profiles.

#### dwCreationMode

Specifies which actions to take on the profile while opening it, if it is contained in a file. This parameter must take one of the following constant values.

Value	Meaning
CREATE_NEW	Creates a new profile. Fails if the profile already exists.
CREATE_ALWAYS	Creates a new profile. Overwrites the profile if it exists.
OPEN_EXISTING	Opens the profile. Fails if it does not exist
OPEN_ALWAYS	Opens the profile if it exists. For ICC profiles, if the profile does not exist, creates the profile. For WCS XML profiles, if the profile does not exist, returns an error.
TRUNCATE_EXISTING	Opens the profile, and truncates it to zero bytes, returning a blank ICC profile. Fails if the profile doesn't exist.

## Return value

If this function succeeds, the return value is the handle of the color profile that is opened. For ICC and WCS profiles, a CAMP and GMMP are provided by the function based on the current default CAMP and GMMP in the registry.

When OpenColorProfile encounters an ICC profile with an embedded WCS profile, and if the dwType member within the Profile structure does not take the value DONT\_USE\_EMBEDDED\_WCS\_PROFILES, it should extract and use the WCS profile(s) contained in this WcsProfilesTag. The HPROFILE returned would be a WCS HPROFILE.

If this function fails, the return value is **NULL**. For extended error information, call **GetLastError**.

## Remarks

If the profile data is not specified using a file name, *dwShareMode* and *dwCreationMode* are ignored.

*dwCreationMode* flags CREATE\_NEW, CREATE\_ALWAYS, and TRUNCATE\_EXISTING, will always return blank ICC HPROFILEs. If other *dwCreationMode* flags are present, InternalOpenColorProfile is called (using the flags as provided by the API) to determine whether the profile is ICC or WCS XML.

Within the ICC code path, an ICC HPROFILE is returned using the requested sharing, access and creation flags as specified in the tables above.

Within the WCS path, the *dwCreationMode* flag OPEN\_ALWAYS will fail if the profile doesn't exist, since WCS profiles cannot be created or edited within the WCS architecture (they must be edited outside of it, using MSXML6). For the same reason, *dwShareMode* flag FILE\_SHARE\_WRITE, and *dwDesiredAccess* flag PROFILE\_READWRITE are ignored within the WCS path.

When the function opens the ICC profile, it will look for a *WcsProfilesTag* and, if there is one, it will extract and use the original WCS profiles contained therein. (See [WcsCreateIccProfile](#).)

An HPROFILE with WCS profile information is derived from a DMP by acquiring the default CAMP and default GMMP from the registry. An HPROFILE is a composition of a DMP, CAMP and GMMP.

Once the handle to the color profile is created, any information used to create that handle can be deleted.

Use the [CloseColorProfile](#) function to close an object handle returned by [OpenColorProfile](#).

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
  - Functions
  - CloseColorProfile
  - PROFILE
- 

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# RegisterCMMA function (icm.h)

Article02/22/2024

Associates a specified identification value with the specified color management module dynamic link library (CMM DLL). When this ID appears in a color profile, Windows can then locate the corresponding CMM so as to create a transform.

## Syntax

C++

```
BOOL RegisterCMMA(
    PCSTR pMachineName,
    DWORD cmmID,
    PCSTR pCMMdll
);
```

## Parameters

pMachineName

Reserved; must currently be set to **NULL**, until non-local registration is supported. This parameter is intended to point to the name of the machine on which a CMM DLL should be registered. A **NULL** pointer indicates the local machine.

cmmID

Specifies the ID signature of the CMM registered with the International Color Consortium (ICC).

pCMMdll

Pointer to the fully qualified path name of the CMM DLL.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# RegisterCMMW function (icm.h)

Article02/22/2024

Associates a specified identification value with the specified color management module dynamic link library (CMM DLL). When this ID appears in a color profile, Windows can then locate the corresponding CMM so as to create a transform.

## Syntax

C++

```
BOOL RegisterCMMW(
    PCWSTR pMachineName,
    DWORD cmmID,
    PCWSTR pCMMdll
);
```

## Parameters

pMachineName

Reserved; must currently be set to **NULL**, until non-local registration is supported. This parameter is intended to point to the name of the machine on which a CMM DLL should be registered. A **NULL** pointer indicates the local machine.

cmmID

Specifies the ID signature of the CMM registered with the International Color Consortium (ICC).

pCMMdll

Pointer to the fully qualified path name of the CMM DLL.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# SelectCMM function (icm.h)

Article02/22/2024

Allows you to select the preferred color management module (CMM) to use.

## Syntax

C++

```
BOOL SelectCMM(  
    DWORD dwCMMType  
) ;
```

## Parameters

`dwCMMType`

Specifies the signature of the desired CMM as registered with the International Color Consortium (ICC).

**Windows 2000 only:** Setting this parameter to **NULL** causes the WCS system to select the default CMM.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

For **SelectCMM** to succeed, the specified CMM must be registered with the system.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# SetColorProfileElement function (icm.h)

Article 02/22/2024

Sets the element data for a tagged profile element in an ICC color profile.

## Syntax

C++

```
BOOL SetColorProfileElement(
    HPROFILE hProfile,
    TAGTYPE tag,
    DWORD dwOffset,
    PDWORD pcbElement,
    PVOID pElement
);
```

## Parameters

`hProfile`

Specifies a handle to the ICC profile in question.

`tag`

Identifies the tagged element.

`dwOffset`

Specifies the offset from the first byte of the tagged element data at which to start writing.

`pcbElement`

Pointer to a variable containing the number of bytes of data to write. On return, it contains the number of bytes actually written.

`pElement`

Pointer to a buffer containing the data to write to the tagged element in the color profile.

# Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

This function will fail if *hProfile* is not a valid ICC profile.

If the color profile is not opened for read/write permission, this function fails.

If *dwOffset* exceeds the size set for the specified tagged element, this function fails.

If *dwOffset* + *\*pcbSize* is greater than the size of the specified element, this function only writes as many bytes as will fit within the current size of the element.

Any existing data in the specified portion of the tagged element is overwritten when this function succeeds.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because profile elements are implicitly associated with and hard coded to ICC tag types and there exist many robust XML parsing libraries.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)

- Functions
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# SetColorProfileElementReference function (icm.h)

Article02/22/2024

Creates in a specified ICC color profile a new tag that references the same data as an existing tag.

## Syntax

C++

```
BOOL SetColorProfileElementReference(
    HPROFILE hProfile,
    TAGTYPE newTag,
    TAGTYPE refTag
);
```

## Parameters

`hProfile`

Specifies a handle to the ICC color profile in question.

`newTag`

Identifies the new tag to create.

`refTag`

Identifies the existing tag whose data is to be referenced by the new tag.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

This function will fail if *hProfile* is not a valid ICC profile.

If *newTag* already exists or *refTag* does not exist, **SetColorProfileElementReference** fails.

If the color profile was not opened with read/write permission,  
**SetColorProfileElementReference** fails.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because profile elements are implicitly associated with and hard coded to ICC tag types and there exist many robust XML parsing libraries.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# SetColorProfileElementSize function (icm.h)

Article 10/05/2021

Sets the size of a tagged element in an ICC color profile.

## Syntax

C++

```
BOOL SetColorProfileElementSize(
    HPROFILE hProfile,
    TAGTYPE tagType,
    DWORD    pcbElement
);
```

## Parameters

`hProfile`

Specifies a handle to the ICC color profile in question.

`tagType`

Identifies the tagged element.

`pcbElement`

Specifies the size to set the tagged element to. If `cbSize` is zero, this function deletes the specified tagged element. If the tag is a reference, only the tag table entry is deleted, not the data.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

This function will fail if *hProfile* is not a valid ICC profile.

To create a new tagged element in a color profile, use **SetColorProfileElementSize** to set the size, then use **SetColorProfileElement** to set the element value.

If the specified tag already exists in the profile, **SetColorProfileElementSize** changes the size of the element by truncating it or adding zeroes at the end as the case may be.

If the specified tag already exists and is a reference to another tag, **SetColorProfileElementSize** creates a new data area for the tag that is not shared.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because profile elements are implicitly associated with and hard coded to ICC tag types and there exist many robust XML parsing libraries.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [SetColorProfileElement](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# SetColorProfileHeader function (icm.h)

Article02/22/2024

Sets the header data in a specified ICC color profile.

## Syntax

C++

```
BOOL SetColorProfileHeader(
    HPROFILE      hProfile,
    PPROFILEHEADER pHeader
);
```

## Parameters

`hProfile`

Specifies a handle to the ICC color profile in question.

`pHeader`

Pointer to the profile header data to write to the specified profile.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call `GetLastError`.

## Remarks

This function will fail if `hProfile` is not a valid ICC profile.

If the color profile was not opened with read/write permission, `SetColorProfileHeader` fails.

`SetColorProfileHeader` overwrites the current header in the ICC profile.

This function does not support Windows Color System (WCS) profiles CAMP, DMP, and GMMP; because profile elements are implicitly associated with and hard coded to ICC tag types and there exist many robust XML parsing libraries.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [PROFILEHEADER](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# SetColorSpace function (wingdi.h)

Article02/22/2024

The `SetColorSpace` function defines the input [color space](#) for a given device context.

## Syntax

C++

```
HCOLORSPACE SetColorSpace(  
    HDC      hdc,  
    HCOLORSPACE hcs  
)
```

## Parameters

hdc

Specifies the handle to a device context.

hcs

Identifies handle to the color space to set.

## Return value

If this function succeeds, the return value is a handle to the *hColorSpace* being replaced.

If this function fails, the return value is **NULL**.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# SetDeviceGammaRamp function (wingdi.h)

Article04/02/2021

The **SetDeviceGammaRamp** function sets the [gamma ramp](#) on direct color display boards having drivers that support downloadable gamma ramps in hardware.

## Important

We strongly recommend that you don't use this API. Use of this API is subject to major limitations:

- **SetDeviceGammaRamp** implements heuristics to check whether a provided ramp will result in an unreadable screen. If a ramp violates those heuristics, then the function fails silently (that is, it returns **TRUE**, but it doesn't set your ramp). For that reason, you can't expect to use this function to set *just any arbitrary* gamma ramp. In particular, the heuristics prevent ramps that would result in nearly all pixels approaching a single value (such as fullscreen black/white) as this may prevent a user from recovering the screen.
- Because of the function's global nature, any other application on the system could, at any time, overwrite any ramp that you've set. In some cases the operating system itself may reserve the use of this function, causing any existing ramp to be overwritten. The gamma ramp is also reset on most display events (connecting/disconnecting a monitor, resolution changes, etc.). So you can't be certain that any ramp you set is in effect.
- This API has undefined behavior in HDR modes.
- This API has undefined interaction with both built-in and third-party color calibration solutions.

For color calibration, we recommend that you create an International Color Consortium (ICC) profile, and let the OS apply the profile. For advanced original equipment manufacturer (OEM) scenarios, there's a device driver model that you can use to customize color calibration more directly. See the [Windows Color System](#) for information on managing color profiles.

For blue light filtering, Windows now provides built-in support called [Night Light](#). We recommend directing users to this feature.

For color adaptation (for example, adjusting color calibration based on ambient light sensors), Windows now provides built-in support, which we recommend for use by OEMs.

For custom filter effects, there are a variety of built-in accessibility [color filters](#) to help with a range of cases.

## Syntax

C++

```
BOOL SetDeviceGammaRamp(  
    HDC     hdc,  
    LPVOID lpRamp  
)
```

## Parameters

hdc

Specifies the device context of the direct color display board in question.

lpRamp

Pointer to a buffer containing the gamma ramp to be set. The gamma ramp is specified in three arrays of 256 **WORD** elements each, which contain the mapping between RGB values in the frame buffer and digital-analog-converter (*DAC*) values. The sequence of the arrays is red, green, blue. The RGB values must be stored in the most significant bits of each WORD to increase DAC independence.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**.

## Remarks

Direct color display modes do not use color lookup tables and are usually 16, 24, or 32 bit. Not all direct color video boards support loadable gamma ramps. **SetDeviceGammaRamp** succeeds only for devices with drivers that support downloadable gamma ramps in hardware.

 **Note**

This API can take a non-trivial amount of time to execute. It may take as long as 200ms to return on some hardware.

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Target Platform</b>	Windows
<b>Header</b>	wingdi.h
<b>Library</b>	Gdi32.lib
<b>DLL</b>	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# SetICMMode function (wingdi.h)

Article04/02/2021

The **SetICMMode** function causes Image Color Management to be enabled, disabled, or queried on a given device context (DC).

## Syntax

C++

```
int SetICMMode(
    HDC hdc,
    int mode
);
```

## Parameters

hdc

Identifies handle to the device context.

mode

Turns on and off image color management. This parameter can take one of the following constant values.

Value	Meaning
ICM_ON	Turns on color management. Turns off old-style color correction of halftones.
ICM_OFF	Turns off color management. Turns on old-style color correction of halftones.
ICM_QUERY	Queries the current state of color management.
ICM_DONE_OUTSIDEDC	Turns off color management inside DC. Under Windows 2000, also turns off old-style color correction of halftones. Not supported under Windows 95.

## Return value

If this function succeeds, the return value is a nonzero value.

If this function fails, the return value is zero.

If ICM\_QUERY is specified and the function succeeds, the nonzero value returned is ICM\_ON or ICM\_OFF to indicate the current mode.

## Remarks

If the system cannot find an ICC color profile to match the state of the device, **SetICMMODE** fails and returns zero.

Once WCS is enabled for a device context (DC), colors passed into the DC using most Win32 API functions are color matched. The primary exceptions are **BitBlt** and **StretchBlt**. The assumption is that when performing a bit block transfer (blit) from one DC to another, the two DCs are already compatible and need no color correction. If this is not the case, color correction may be performed. Specifically, if a device independent bitmap (DIB) is used as the source for a blit, and the blit is performed into a DC that has WCS enabled, color matching will be performed. If this is not what you want, turn WCS off for the destination DC by calling **SetICMMODE** before calling **BitBlt** or **StretchBlt**.

If the **CreateCompatibleDC** function is used to create a bitmap in a DC, it is possible for the bitmap to be color matched twice, once when it is created and once when a blit is performed. The reason is that a bitmap in a DC created by the **CreateCompatibleDC** function acquires the current brush, pens, and palette of the source DC. However, WCS will be disabled by default for the new DC. If WCS is later enabled for the new DC by using the **SetICMMODE** function, a color correction will be done. To prevent double color corrections through the use of the **CreateCompatibleDC** function, use the **SetICMMODE** function to turn WCS off for the source DC before the **CreateCompatibleDC** function is called.

When a compatible DC is created from a printer's DC (see **CreateCompatibleDC** ), the default is for color matching to always be performed if it is enabled for the printer's DC. The default color profile for the printer is used when a blit is performed into the printer's DC using **SetDIBitsToDevice** or **StretchDIBits**. If this is not what you want, turn WCS off for the printer's DC by calling **SetICMMODE** before calling **SetDIBitsToDevice** or **StretchDIBits**.

Also, when printing to a printer's DC with WCS turned on, the **SetICMMODE** function needs to be called after every call to the **StartPage** function to turn back on WCS. The **StartPage** function calls the **RestoreDC** and **SaveDC** functions, which result in WCS being turned off for the printer's DC.

# Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [BitBlt](#)
- [CreateCompatibleDC](#)
- [SetDIBitsToDevice](#)
- [StartPage](#)
- [StretchBlt](#)
- [StretchDIBits](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# SetICMProfileA function (wingdi.h)

Article 02/22/2024

The **SetICMProfile** function sets a specified color profile as the output profile for a specified device context (DC).

## Syntax

C++

```
BOOL SetICMProfileA(  
    HDC    hdc,  
    LPSTR lpFileName  
) ;
```

## Parameters

hdc

Specifies a device context in which to set the color profile.

lpFileName

Specifies the path name of the color profile to be set.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**.

## Remarks

**SetICMProfile** associates a color profile with a device context. It becomes the output profile for that device context. The color profile does not have to be associated with any particular device. Device-independent profiles such as sRGB can also be used. If the color profile is not associated with a hardware device, it will be returned by [GetICMProfile](#), but not by [EnumICMProfiles](#).

Note that under Windows 95 or later, the PostScript device driver for printers assumes a CMYK color model. Therefore, all PostScript printers must use a CMYK color profile. Windows 2000 does not have this limitation.

**SetICMProfile** supports only RGB profiles in compatible DCs.

**Windows 95/98/Me:** **SetICMProfileW** is supported by the Microsoft Layer for Unicode. To use this, you must add certain files to your application, as outlined in [Microsoft Layer for Unicode on Windows 95/98/Me Systems](#).

 **Note**

The wingdi.h header defines SetICMProfile as an alias that automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [EnumICMProfilesW](#)
- [GetICMProfileW](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# SetStandardColorSpaceProfileA function (icm.h)

Article07/27/2022

Registers a specified profile for a given standard [color space](#). The profile can be queried using [GetStandardColorSpaceProfileW](#).

## Syntax

C++

```
BOOL SetStandardColorSpaceProfileA(  
    PCSTR pMachineName,  
    DWORD dwProfileID,  
    PCSTR pProfilename  
>;
```

## Parameters

`pMachineName`

Reserved. Must be **NULL**. This parameter is intended to point to the name of the machine on which to set a standard color space profile. A **NULL** pointer indicates the local machine.

`dwProfileID`

Specifies the ID value of the standard color space that the given profile represents. This is a custom ID value used to uniquely identify the color space profile within your application.

`pProfilename`

Points to a fully qualified path to the profile file.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

The profile must already be installed on the system before it can be registered for a standard color space.

This function supports Windows Color System (WCS) device model profiles (DMPs) in addition to International Color Consortium (ICC) profiles. It does not support WCS CAMP or GMMP profiles and will return an error if such profiles are used.

### *Per-user/LUA support*

This will register a specified profile for a given standard color space only for current user.

This uses [WcsSetDefaultColorProfile](#) with  
WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER.

This is executable in LUA context if the profile is already installed, fails otherwise with access denied since install is system-wide and requires administrator privileges.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [SetStandardColorSpaceProfileW](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# SetStandardColorSpaceProfileW function (icm.h)

Article07/27/2022

Registers a specified profile for a given standard [color space](#). The profile can be queried using [GetStandardColorSpaceProfileW](#).

## Syntax

C++

```
BOOL SetStandardColorSpaceProfileW(  
    PCWSTR pMachineName,  
    DWORD dwProfileID,  
    PCWSTR pProfileName  
) ;
```

## Parameters

`pMachineName`

Reserved. Must be **NULL**. This parameter is intended to point to the name of the machine on which to set a standard color space profile. A **NULL** pointer indicates the local machine.

`dwProfileID`

Specifies the ID value of the standard color space that the given profile represents. This is a custom ID value used to uniquely identify the color space profile within your application.

`pProfileName`

Points to a fully qualified path to the profile file.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

The profile must already be installed on the system before it can be registered for a standard color space.

This function supports Windows Color System (WCS) device model profiles (DMPs) in addition to International Color Consortium (ICC) profiles. It does not support WCS CAMP or GMMP profiles and will return an error if such profiles are used.

### *Per-user/LUA support*

This will register a specified profile for a given standard color space only for current user.

This uses [WcsSetDefaultColorProfile](#) with  
WCS\_PROFILE MANAGEMENT\_SCOPE\_CURRENT\_USER.

This is executable in LUA context if the profile is already installed, fails otherwise with access denied since install is system-wide and requires administrator privileges.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [SetStandardColorSpaceProfileW](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# SetupColorMatchingA function (icm.h)

Article02/22/2024

Creates a Color Management dialog box that lets the user choose whether to enable color management and, if so, provides control over the color profiles used and over the rendering intent.

## Syntax

C++

```
BOOL SetupColorMatchingA(  
    PCOLORMATCHSETUPA pcms  
) ;
```

## Parameters

pcms

Pointer to a [COLORMATCHSETUPW](#) structure that on entry contains information used to initialize the dialog box.

When [SetupColorMatching](#) returns, if the user clicked the OK button, this structure contains information about the user's selection. Otherwise, if an error occurred or the user canceled the dialog box, the structure is left unchanged.

## Return value

If this function succeeds, the return value is **TRUE** indicating that no errors occurred and the user clicked the OK button.

If this function fails, the return value is **FALSE** indicating that an error occurred or the dialog was canceled. For extended error information, call [GetLastError](#).

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icmui.lib
DLL	Icmui.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# SetupColorMatchingW function (icm.h)

Article02/22/2024

Creates a Color Management dialog box that lets the user choose whether to enable color management and, if so, provides control over the color profiles used and over the rendering intent.

## Syntax

C++

```
BOOL SetupColorMatchingW(  
    PCOLORMATCHSETUPW pcms  
);
```

## Parameters

pcms

Pointer to a **COLORMATCHSETUPW** structure that on entry contains information used to initialize the dialog box.

When **SetupColorMatching** returns, if the user clicked the OK button, this structure contains information about the user's selection. Otherwise, if an error occurred or the user canceled the dialog box, the structure is left unchanged.

## Return value

If this function succeeds, the return value is **TRUE** indicating that no errors occurred and the user clicked the OK button.

If this function fails, the return value is **FALSE** indicating that an error occurred or the dialog was canceled. For extended error information, call **GetLastError**.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Icmui.lib
DLL	Icmui.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# TranslateBitmapBits function (icm.h)

Article 10/05/2021

Translates the colors of a bitmap having a defined format so as to produce another bitmap in a requested format.

## Syntax

C++

```
BOOL TranslateBitmapBits(
    HTRANSFORM    hColorTransform,
    PVOID         pSrcBits,
    BMFORMAT      bmInput,
    DWORD         dwWidth,
    DWORD         dwHeight,
    DWORD         dwInputStride,
    PVOID         pDestBits,
    BMFORMAT      bmOutput,
    DWORD         dwOutputStride,
    PBMCALLBACKFN pfnCallBack,
    LPARAM        ulCallbackData
);
```

## Parameters

`hColorTransform`

Identifies the color transform to use.

`pSrcBits`

Pointer to the bitmap to translate.

`bmInput`

Specifies the format of the input bitmap. Must be set to one of the values of the [BMFORMAT](#) enumerated type.

### Note

This function doesn't support `BM_XYZTRIPLETS` or `BM_YxyTRIPLETS` as inputs.

`dwWidth`

Specifies the number of pixels per scan line in the input bitmap.

`dwHeight`

Specifies the number of scan lines in the input bitmap.

`dwInputStride`

Specifies the number of bytes from the beginning of one scan line to the beginning of the next in the input bitmap; if set to zero, the function assumes that scan lines are padded so as to be **DWORD**-aligned.

`pDestBits`

Pointer to the buffer in which to place the translated bitmap.

`bmOutput`

Specifies the format of the output bitmap. Must be set to one of the values of the **BMFORMAT** enumerated type.

`dwOutputStride`

Specifies the number of bytes from the beginning of one scan line to the beginning of the next in the output bitmap; if set to zero, the function assumes that scan lines should be padded to be **DWORD**-aligned.

`pfnCallBack`

Pointer to a callback function called periodically by **TranslateBitmapBits** to report progress and allow the calling process to cancel the translation. (See [ICMProgressProcCallback](#))

`ulCallbackData`

Data passed back to the callback function, for example, to identify the translation that is reporting progress.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

If the input and output formats are not compatible with the color transform, this function fails.

When either of the floating point BMFORMATs, BM\_32b\_scARGB or BM\_32b\_scRGB are used, the color data being translated should not contain NaN or infinity. NaN and infinity are not considered to represent legitimate color component values, and the result of translating pixels containing NaN or infinity is meaningless in color terms. NaN or infinity values in the color data being processed will be handled silently, and an error will not be returned.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [ICMProgressProcCallback](#)
- [Windows bitmap header structures](#)
- [BMFORMAT](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# TranslateColors function (icm.h)

Article 10/05/2021

Translates an array of colors from the source [color space](#) to the destination color space as defined by a color transform.

## Syntax

C++

```
BOOL TranslateColors(
    HTRANSFORM hColorTransform,
    PCOLOR     paInputColors,
    DWORD      nColors,
    COLORTYPE   ctInput,
    PCOLOR     paOutputColors,
    COLORTYPE   ctOutput
);
```

## Parameters

`hColorTransform`

Identifies the color transform to use.

`paInputColors`

Pointer to an array of *nColors*[COLOR](#) structures to translate.

`nColors`

Contains the number of elements in the arrays pointed to by *paInputColors* and *paOutputColors*.

`ctInput`

Specifies the input color type.

`paOutputColors`

Pointer to an array of *nColors* [COLOR](#) structures that receive the translated colors.

`ctOutput`

Specifies the output color type.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

If the input and the output color types are not compatible with the color transform, this function fails.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# UninstallColorProfileA function (icm.h)

Article 02/22/2024

Removes a specified color profile from a specified computer. Associated files are optionally deleted from the system.

## Syntax

C++

```
BOOL UninstallColorProfileA(  
    PCSTR pMachineName,  
    PCSTR pProfileName,  
    BOOL bDelete  
) ;
```

## Parameters

pMachineName

Reserved. Must be **NULL**. This parameter is intended to point to the name of the machine from which to uninstall the specified profile. A **NULL** pointer indicates the local machine.

pProfileName

Points to the file name of the profile to uninstall.

bDelete

If set to **TRUE**, the function deletes the profile from the COLOR directory. If set to **FALSE**, this function has no effect.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# UninstallColorProfileW function (icm.h)

Article 02/22/2024

Removes a specified color profile from a specified computer. Associated files are optionally deleted from the system.

## Syntax

C++

```
BOOL UninstallColorProfileW(
    PCWSTR pMachineName,
    PCWSTR pProfileName,
    BOOL    bDelete
);
```

## Parameters

pMachineName

Reserved. Must be **NULL**. This parameter is intended to point to the name of the machine from which to uninstall the specified profile. A **NULL** pointer indicates the local machine.

pProfileName

Points to the file name of the profile to uninstall.

bDelete

If set to **TRUE**, the function deletes the profile from the COLOR directory. If set to **FALSE**, this function has no effect.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# UnregisterCMMA function (icm.h)

Article02/22/2024

Dissociates a specified ID value from a given color management module dynamic-link library (CMM DLL).

## Syntax

C++

```
BOOL UnregisterCMMA(
    PCSTR pMachineName,
    DWORD cmmID
);
```

## Parameters

pMachineName

Reserved; must currently be set to **NULL**, until non-local registration is supported. This parameter is intended to point to the name of the computer on which a CMM DLLs registration should be removed. A **NULL** pointer indicates the local computer.

cmmID

Specifies the ID value identifying the CMM whose registration is to be removed. This is the signature of the CMM registered with the International Color Consortium (ICC).

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

If the profile for creating a transform later specifies this ID, the Windows default CMM will be used rather than this CMM DLL.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# UnregisterCMMW function (icm.h)

Article02/22/2024

Dissociates a specified ID value from a given color management module dynamic-link library (CMM DLL).

## Syntax

C++

```
BOOL UnregisterCMMW(
    PCWSTR pMachineName,
    DWORD   cmmID
);
```

## Parameters

pMachineName

Reserved; must currently be set to **NULL**, until non-local registration is supported. This parameter is intended to point to the name of the computer on which a CMM DLLs registration should be removed. A **NULL** pointer indicates the local computer.

cmmID

Specifies the ID value identifying the CMM whose registration is to be removed. This is the signature of the CMM registered with the International Color Consortium (ICC).

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Remarks

If the profile for creating a transform later specifies this ID, the Windows default CMM will be used rather than this CMM DLL.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# WcsAssociateColorProfileWithDevice function (icm.h)

Article02/22/2024

Associates a specified WCS color profile with a specified device.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileAddDisplayAssociation](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL WcsAssociateColorProfileWithDevice(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PCWSTR pProfileName,
    PCWSTR pDeviceName
);
```

## Parameters

scope

A [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#) value that specifies the scope of this profile management operation, which could be system-wide or for the current user.

pProfileName

A pointer to the file name of the profile to associate.

pDeviceName

A pointer to the name of the device with which the profile is to be associated.

## Return value

If this function succeeds, the return value is TRUE.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

The [WCSAssociateColorProfileWithDevice](#) function fails if the profile has not been installed on the computer using the [InstallColorProfileW](#) function.

If the *profileManagementScope* parameter is `WCS_PROFILE_MANAGEMENT_SCOPE_SYSTEM_WIDE`, the profile association is system-wide and applies to all users. If *profileManagementScope* is `WCS_PROFILE_MANAGEMENT_SCOPE_CURRENT_USER`, the association is only for the current user.

This function is executable in Least-Privileged User Account (LUA) context if *profileManagementScope* is `WCS_PROFILE_MANAGEMENT_SCOPE_CURRENT_USER`. Otherwise, administrative privileges are required.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [Windows Color System schemas and algorithms](#)
- [WcsDisassociateColorProfileFromDevice\\*\\*](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# WcsCheckColors function (icm.h)

Article02/22/2024

Determines whether the colors in an array are within the output gamut of a specified WCS color transform.

## Syntax

C++

```
BOOL WcsCheckColors(
    HTRANSFORM    hColorTransform,
    DWORD         nColors,
    DWORD         nInputChannels,
    COLORDATATYPE cdtInput,
    DWORD         cbInput,
    PVOID         pInputData,
    PBYTE         paResult
);
```

## Parameters

`hColorTransform`

A handle to the specified WCS color transform.

`nColors`

The number of elements in the array pointed to by `pInputData` and `paResult`.

`nInputChannels`

The number of channels per element in the array pointed to by `pInputData`.

`cdtInput`

The input COLORDATATYPE color data type.

`cbInput`

The buffer size of `pInputData`.

`pInputData`

A pointer to an array of input colors. Colors in this array correspond to the color space of the source profile. The size of the buffer for this array will be the number of bytes indicated by *cbInput*.

paResult

A pointer to an array of *nColors* bytes that receives the results of the test.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

If the input and the output color data types are not compatible with the color transform, this function will convert the input color data as required.

This function fails if you use an International Color Consortium (ICC) transform is used.

## Requirements

[\[\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [Windows Color System schemas and algorithms](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# WcsCreateIccProfile function (icm.h)

Article 10/05/2021

Converts a WCS profile into an International Color Consortium (ICC) profile.

## Syntax

C++

```
HPROFILE WcsCreateIccProfile(
    HPROFILE hWcsProfile,
    DWORD     dwOptions
);
```

## Parameters

`hWcsProfile`

A handle to the WCS color profile that is converted. See Remarks.

`dwOptions`

A flag value that specifies the profile conversion options.

By default, the original WCS profiles used for the conversion are embedded in the output ICC profile in a Microsoft private tag, *WcsProfilesTag* (with signature "MS000"). This produces an ICC profile that is compatible with ICC software, yet retains the original WCS profile data available to code designed to parse it.

The possible values of this parameter are as follows. Any bits not defined in this list are reserved and should be set to zero:

Value	Description
<code>WCS_DEFAULT</code>	Specifies that the new ICC profile contains the original WCS profile in a private <i>WcsProfilesTag</i> .
<code>WCS_ICCONLY</code>	Specifies that the new ICC profile does not contain either the <i>WcsProfilesTag</i> or the original WCS profile.

## Return value

If this function succeeds, the return value is the handle of the new color profile.

If this function fails, the return value is **NULL**. For extended error information, call [GetLastError](#).

## Remarks

This function can be used with ASCII or Unicode strings.

The **CloseColorProfile** function should be used to close the returned HPROFILE handle when it is no longer needed.

The DMP, CAMP, and GMMP from the HPROFILE are embedded in a private tag within the created ICC profile.

The ICC profile created using this API will have its profile description tag constructed from the ProfileName elements of the WCS profiles according to the following pattern:  
"Created by Microsoft WCS from DMP:[the DMP ProfileName], CAMP:[the CAMP ProfileName], GMMP:[the GMMP ProfileName]"

When WCS encounters this ICC profile (via [OpenColorProfileW](#) or [WcsOpenColorProfileW](#)) it extracts and uses the WCS profile(s) contained in the *WcsProfilesTag*.

The out-of-gamut information in the gamut tags created in WCS uses the perceptual color distance in CIECAM02, which is the mean square root in CIECAM02 Jab space. The distance in legacy ICC profile gamut tags is the mean square root in CIELAB space. It is recommended that you use the CIECAM02 space when it is available, to provide more perceptually accurate distance metrics.

WCS extracts and uses the original WCS profile by means of an XML profile explicitly associated with a device, or an ICC profile that has a *WcsProfilesTag*.

The *WcsProfilesTag* is a Microsoft private ICC profile tag used in profiles created by **WcsCreateIccProfile** to contain the WCS profiles input to **WcsCreateIccProfile**. This tag conforms to ICC profile requirements for profile tags. The non-XML components of the tag must be in "Big-Endian" byte ordering, which is standard for ICC profiles. Further, the tag data must be aligned on a 4-byte boundary (measured from the start of the ICC profile). The structure of the tag is defined by the **WcsProfilesTagType** below. Note that the XML components of the tag, the WCS profiles contained in the *WcsProfileTag*, are left in their native byte ordering, which may be either little-endian or big-endian since XML parsers correctly process either.

The WcsProfilesTag signature is "MS00". This is the tag signature that will appear in the ICC profiles tag table for the WcsProfilesTag.

The **WcsProfilesTagType** Structure has the following structure:

Byte Offset	Content
0-3	The MS10 type signature.
4-7	Reserved, must be set to 0 (ICC tradition).
8-11	Byte offset from the beginning of the tag to the CDMP data.
12-15	Size of the CDMP data in bytes.
16-19	Byte offset from the beginning of the tag to the CAMP data.
20-23	Size of the CAMP data in bytes.
24-27	Byte offset from the beginning of the tag to the GMMP data.
28-31	Byte offset from the beginning of the tag to the GMMP data.
31-n	A sequence of (element size -32) bytes [where element size is the tag size recorded in the ICC profile tag table entry for this tag.]

These are the WCS XML profiles that were used by **WcsCreateIccProfile** to create this ICC profile. The WCS profiles are ordered: the DMP (required) first, followed by the CAMP (if present), followed by the GMMP (if present).

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
  - Functions
  - Windows Color System schemas and algorithms
- 

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# WcsDisassociateColorProfileFromDevice function (icm.h)

Article02/22/2024

Disassociates a specified WCS color profile from a specified device on a computer.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileRemoveDisplayAssociation](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL WcsDisassociateColorProfileFromDevice(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PCWSTR pProfileName,
    PCWSTR pDeviceName
);
```

## Parameters

scope

A [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#) value that specifies the scope of this profile management operation, which could be system-wide or for the current user.

pProfileName

A pointer to the file name of the profile to disassociate.

pDeviceName

A pointer to the name of the device from which to disassociate the profile.

## Return value

If this function succeeds, the return value is TRUE.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

The WCS color profile should be installed. Moreover, you must use the same *profileManagementScope* value as when the device was associated with the profile. See [WcsAssociateColorProfileWithDevice](#).

If *profileManagementScope* is **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_SYSTEM\_WIDE**, the profile disassociation is systemwide and applies to all users. If *profileManagementScope* is **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER**, the disassociation is only for the current user.

If more than one color profile is associated with a device, WCS uses the last associated profile as the default. For example, if your application sequentially associates three profiles with a device, WCS uses the last profile associated as the default. If your application then calls the [WcsDisassociateColorProfileFromDevice](#) function to disassociate the third profile (which is the default in this example), WCS uses the second profile as the default.

If your application disassociates all profiles from a device, WCS uses the sRGB profile as the default.

If *profileManagementScope* is **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER**, this function is executable in Least-Privileged User Account (LUA) context. Otherwise, administrative privileges are required.

## Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
  - [Functions](#)
  - [Windows Color System schemas and algorithms](#)
  - [WcsAssociateColorProfileWithDevice](#)
- 

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# WcsEnumColorProfiles function (icm.h)

Article10/05/2021

Enumerates color profiles associated with any device, in the specified scope.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileGetDisplayList](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL WcsEnumColorProfiles(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PENUMTYPEW pEnumRecord,
    PBYTE pBuffer,
    DWORD dwSize,
    PDWORD pnProfiles
);
```

## Parameters

scope

A [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#) value specifying the scope of this profile management operation.

pEnumRecord

A pointer to a structure specifying the enumeration criteria.

pBuffer

A pointer to a buffer in which the profile names are to be enumerated. The **WcsEnumColorProfiles** function places, in this buffer, a MULTI\_SZ string that consists of profile names that satisfy the criteria specified in *\*pEnumRecord*.

dwSize

A variable that contains the size, in bytes, of the buffer that is pointed to by *pBuffer*. See Remarks.

#### pnProfiles

An optional pointer to a variable that receives the number of profile names that are copied to the buffer to which *pBuffer* points. Can be **NULL** if this information is not needed.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

Use the [WcsEnumColorProfilesSize](#) function to retrieve the value for the *dwSize* parameter, which is the size, in bytes, of the buffer pointed to by the *pBuffer* parameter.

If the *profileManagementScope* parameter is **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_SYSTEM\_WIDE**, only system-wide associations of profiles to the device are considered. If *profileManagementScope* is **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER**, only per-user associations for the current user are considered. If [WcsSetUsePerUserProfiles](#) has never been called for this user, or if [WcsSetUsePerUserProfiles](#) was most recently called for this user with its *usePerUserProfiles* parameter set to **FALSE**, then [WCSEnumColorProfiles](#) returns an empty list.

If **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER** (the current user setting) is present, it overrides the system-wide default for the *profileManagementScope* parameter.

This function is executable in Least-Privileged User Account (LUA) context.

## Requirements

Minimum supported client

Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [Windows Color System schemas and algorithms](#)
- [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#)
- [WcsEnumColorProfilesSize](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WcsEnumColorProfilesSize function (icm.h)

Article02/22/2024

Returns the size, in bytes, of the buffer that is required by the [WcsEnumColorProfiles](#) function to enumerate color profiles.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileGetDisplayList](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL WcsEnumColorProfilesSize(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PENUMTYPEW                 pEnumRecord,
    PDWORD                      pdwSize
);
```

## Parameters

scope

A [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#) value that specifies the scope of the profile management operation that is performed by this function.

pEnumRecord

A pointer to a structure that specifies the enumeration criteria.

pdwSize

A pointer to a variable that receives the size of the buffer that is required to receive all enumerated profile names. This value is used by the *dwSize* parameter of the [WcsEnumColorProfiles](#) function.

# Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

This function is executable in Least-Privileged User Account (LUA) context.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [Windows Color System schemas and algorithms](#)
- [WcsEnumColorProfiles](#)
- [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#)

---

## Feedback

Was this page helpful?



Yes



No

# WcsGetCalibrationManagementState function (icm.h)

Article02/22/2024

Determines whether system management of the display calibration state is enabled.

## Syntax

C++

```
BOOL WcsGetCalibrationManagementState(
    BOOL *pbEnabled
);
```

## Parameters

pbEnabled

**TRUE** if system management of the display calibration state is enabled; otherwise **FALSE**.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**.

## Requirements

  Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib

Requirement	Value
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# WcsGetDefaultColorProfile function (icm.h)

Article10/05/2021

Retrieves the default color profile for a device, or for a device-independent default if the device is not specified.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileGetDisplayDefault](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL WcsGetDefaultColorProfile(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PCWSTR pDeviceName,
    COLORPROFILETYPE cptColorProfileType,
    COLORPROFILESUBTYPE cpstColorProfileSubType,
    DWORD dwProfileID,
    DWORD cbProfileName,
    LPWSTR pProfileName
);
```

## Parameters

scope

A [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#) value specifying the scope of this profile management operation.

pDeviceName

A pointer to the name of the device for which the default color profile is obtained. If **NULL**, a device-independent default profile is obtained.

cptColorProfileType

A [COLORPROFILETYPE](#) value specifying the color profile type.

`cpstColorProfileSubType`

A [COLORPROFILESUBTYPE](#) value specifying the color profile subtype.

`dwProfileID`

The ID of the color space that the color profile represents.

`cbProfileName`

The buffer size, in bytes, of the buffer that is pointed to by *pProfileName*.

`pProfileName`

A pointer to a buffer to receive the name of the color profile. The size of the buffer, in bytes, will be the indicated by *cbProfileName*.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

Use the [WcsGetDefaultColorProfileSize](#) function to obtain the required size of the buffer that is pointed to by the *pProfileName* parameter.

If **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER** is present, it overrides the system-wide default for *profileManagementScope*.

This function is executable in Least-Privileged User Account (LUA) context.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Windows Color System schemas and algorithms](#)
- [Functions](#)
- [COLORPROFILESUBTYPE](#)
- [COLORPROFILETYPE](#)
- [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#)
- [WcsGetDefaultColorProfileSize](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WcsGetDefaultColorProfileSize function (icm.h)

Article02/22/2024

Returns the size, in bytes, of the default color profile name (including the **NUL** terminator), for a device.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileGetDisplayDefault](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL WcsGetDefaultColorProfileSize(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PCWSTR pDeviceName,
    COLORPROFILETYPE cptColorProfileType,
    COLORPROFILESUBTYPE cpstColorProfileSubType,
    DWORD dwProfileID,
    PDWORD pcbProfileName
);
```

## Parameters

scope

A [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#) value that specifies the scope of this profile management operation.

pDeviceName

A pointer to the name of the device for which the default color profile is to be obtained. If **NULL**, a device-independent default profile will be used.

cptColorProfileType

A [COLORPROFILETYPE](#) value specifying the color profile type.

`cpstColorProfileSubType`

A [COLORPROFILESUBTYPE](#) value specifying the color profile subtype.

`dwProfileID`

The ID of the color space that the color profile represents.

`pcbProfileName`

A pointer to a location that receives the size, in bytes, of the path name of the default color profile, including the **NULL** terminator.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

Use this function to return the required size of the buffer that is pointed to by the *pProfileName* parameter in the [WcsGetDefaultColorProfile](#) function.

This function is executable in Least-Privileged User Account (LUA) context.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
  - [Windows Color System schemas and algorithms](#)
  - [Functions](#)
  - [COLORPROFILESUBTYPE](#)
  - [COLORPROFILETYPE](#)
  - [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#)
  - [WcsGetDefaultColorProfile](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# WcsGetDefaultRenderingIntent function (icm.h)

Article 02/22/2024

Retrieves the default rendering intent in the specified profile management scope.

## Syntax

C++

```
BOOL WcsGetDefaultRenderingIntent(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PDWORD pdwRenderingIntent
);
```

## Parameters

scope

The profile management scope for this operation, which can be system-wide or the current user only.

pdwRenderingIntent

A pointer to the variable that will hold the rendering intent.

For more information, see [Rendering intents](#).

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

This function does not revert to the system-wide scope if you do not set the per-user default rendering intent. Instead, it fails, which allows the calling process to distinguish

between the per-user and the system-wide setting. If the per-user rendering intent cannot be retrieved, call this function again with system-wide.

# Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Windows Color System schemas and algorithms](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# WcsGetDefaultRenderingIntent function (icm.h)

Article 02/22/2024

Retrieves the default rendering intent in the specified profile management scope.

## Syntax

C++

```
BOOL WcsGetDefaultRenderingIntent(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PDWORD pdwRenderingIntent
);
```

## Parameters

scope

The profile management scope for this operation, which can be system-wide or the current user only.

pdwRenderingIntent

A pointer to the variable that will hold the rendering intent.

For more information, see [Rendering intents](#).

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

This function does not revert to the system-wide scope if you do not set the per-user default rendering intent. Instead, it fails, which allows the calling process to distinguish

between the per-user and the system-wide setting. If the per-user rendering intent cannot be retrieved, call this function again with system-wide.

# Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Windows Color System schemas and algorithms](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# WcsOpenColorProfileA function (icm.h)

Article07/27/2022

Creates a handle to a specified color profile.

## Syntax

C++

```
HPROFILE WcsOpenColorProfileA(  
    PPROFILE pCDMPPProfile,  
    PPROFILE pCAMPProfile,  
    PPROFILE pGMMPProfile,  
    DWORD    dwDesireAccess,  
    DWORD    dwShareMode,  
    DWORD    dwCreationMode,  
    DWORD    dwFlags  
) ;
```

## Parameters

pCDMPPProfile

Pointer to a WCS DMP or an ICC color profile structure specifying the profile. You can free the *pCDMPPProfile* pointer after you create the handle. If the profile is ICC and its **dwType** member is set to **DONT\_USE\_EMBEDDED\_WCS\_PROFILES**, **WcsOpenColorProfile** ignores any embedded WCS profile within the ICC profile.

pCAMPProfile

A pointer to a profile structure that specifies a WCS color appearance model profile (CAMP). You can free the *pCAMPProfile* pointer after you create the handle. If **NULL**, the default CAMP is used, and the current user setting, **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER**, is used while querying the default CAMP.

pGMMPProfile

A pointer to a profile structure that specifies a WCS gamut map model profile (GMMP). You can free the *pGMMPProfile* pointer after you create the handle. If **NULL**, the default GMMP for the default rendering intent is used, and the current user setting,

`WCS_PROFILE_MANAGEMENT_SCOPE_CURRENT_USER`, is used while querying the default GMMP. For a description of rendering intents, see [Rendering Intents](#).

#### `dwDesireAccess`

A flag value that specifies how to access the specified color profile. This parameter must take one of the following values:

<b>Value</b>	<b>Description</b>
<code>PROFILE_READ</code>	Specifies that the color profile opens for read-only access.
<code>PROFILE_READWRITE</code>	Specifies that the color profile opens for both read and write access. The value of this flag is ignored if the profile is a WCS profile.

#### `dwShareMode`

A flag value that specifies actions to take while opening a color profile contained in a file. This parameter must take one of the following values, which are defined in *winnt.h*:

<b>Value</b>	<b>Description</b>
<code>FILE_SHARE_READ</code>	Specifies that you can perform other open (for read access) operations on the profile.
<code>FILE_SHARE_WRITE</code>	Specifies that you can perform other open (for write access) operations on the profile. This flag value is ignored when a WCS profile is opened.

#### `dwCreationMode`

A flag value that specifies the actions to take while opening a color profile if it is contained in a file. This parameter must take one of the following values, which are defined in *winbase.h*:

<b>Value</b>	<b>Description</b>
<code>CREATE_NEW</code>	Specifies that a new profile is created. This function fails if the profile already exists.
<code>CREATE_ALWAYS</code>	Specifies that a new profile is created. If a profile already exists, it is overwritten.
<code>OPEN_EXISTING</code>	Specifies that the profile is opened. This function fails if the profile does not exist.

Value	Description
OPEN_ALWAYS	Specifies that the profile is to be opened if an International Color Consortium (ICC) file exists. If an ICC profile does not exist, WCS creates a new ICC profile. The function will fail for WCS profiles if this flag is set and a WCS profile does not exist.
TRUNCATE_EXISTING	Specifies that the profile is to be opened and truncated to zero bytes. The function fails if the profile does not exist.

### dwFlags

A flag value that specifies whether to use the embedded WCS profile. This parameter has no effect unless *pCDMPProfile* specifies an ICC profile that contains an embedded WCS profile.

This parameter takes one of the following values:

Value	Description
0	Specifies that the embedded WCS profile will be used and the ICC profile specified by <i>pCDMPPProfile</i> will be ignored.
DONT_USE_EMBEDDED_WCS_PROFILES	Specifies that the ICC profile specified by <i>pCDMPPProfile</i> will be used and the embedded WCS profile will be ignored.

## Return value

If this function succeeds, the return value is the handle of the color profile that is opened.

If this function fails, the return value is **NULL**.

## Remarks

This API will take a set of DMP, CAMP, and GMMP and return a WCS profile handle. **NULL** values for GMMP are valid. A **NULL** value for CAMP will be replaced with the default CAMP value.

This API will also accept ICC profiles. Using an ICC profile does not guarantee processing by the WCS CITE engine. The WCS engine will only be used if it is passed at least one WCS profile. Pure ICC workflows will be consistent with legacy behavior.

You can use the handle that this function returns in other color profile management functions.

The *dwCreationMode* flags CREATE\_NEW, CREATE\_ALWAYS, and TRUNCATE\_EXISTING will always return blank ICC HPROFILEs. If other *dwCreationMode* flags are present, the function will determine whether the profile is ICC or WCS XML.

Within the ICC code path, an ICC HPROFILE is returned using the requested sharing, access, and creation flags as specified in the tables above.

Within the WCS path, the *dwCreationMode* flag OPEN\_ALWAYS will fail if the profile does not exist, because WCS profiles cannot be created or edited within the WCS architecture (they must be edited outside of it, using MSXML6). For the same reason, *dwShareMode* flag FILE\_SHARE\_WRITE, and the *dwDesiredAccess* flag PROFILE\_READWRITE are ignored within the WCS path.

Once the handle to the color profile is created, any information used to create that handle can be deleted.

Use the [CloseColorProfile](#) function to close an object handle that is returned by [WcsOpenColorProfile](#).

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Windows Color System schemas and algorithms](#)
- [Functions](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WcsOpenColorProfileW function (icm.h)

Article07/27/2022

Creates a handle to a specified color profile.

## Syntax

C++

```
HPROFILE WcsOpenColorProfileW(
    PPROFILE pCDMPPProfile,
    PPROFILE pCAMPProfile,
    PPROFILE pGMMPProfile,
    DWORD     dwDesireAccess,
    DWORD     dwShareMode,
    DWORD     dwCreationMode,
    DWORD     dwFlags
);
```

## Parameters

`pCDMPPProfile`

Pointer to a WCS DMP or an ICC color profile structure specifying the profile. You can free the *pCDMPPProfile* pointer after you create the handle. If the profile is ICC and its **dwType** member is set to **DONT\_USE\_EMBEDDED\_WCS\_PROFILES**, **WcsOpenColorProfile** ignores any embedded WCS profile within the ICC profile.

`pCAMPProfile`

A pointer to a profile structure that specifies a WCS color appearance model profile (CAMP). You can free the *pCAMPProfile* pointer after you create the handle. If **NULL**, the default CAMP is used, and the current user setting, **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER**, is used while querying the default CAMP.

`pGMMPProfile`

A pointer to a profile structure that specifies a WCS gamut map model profile (GMMP). You can free the *pGMMPProfile* pointer after you create the handle. If **NULL**, the default GMMP for the default rendering intent is used, and the current user setting,

`WCS_PROFILE_MANAGEMENT_SCOPE_CURRENT_USER`, is used while querying the default GMMP. For a description of rendering intents, see [Rendering Intents](#).

#### `dwDesireAccess`

A flag value that specifies how to access the specified color profile. This parameter must take one of the following values:

<b>Value</b>	<b>Description</b>
<code>PROFILE_READ</code>	Specifies that the color profile opens for read-only access.
<code>PROFILE_READWRITE</code>	Specifies that the color profile opens for both read and write access. The value of this flag is ignored if the profile is a WCS profile.

#### `dwShareMode`

A flag value that specifies actions to take while opening a color profile contained in a file. This parameter must take one of the following values, which are defined in *winnt.h*:

<b>Value</b>	<b>Description</b>
<code>FILE_SHARE_READ</code>	Specifies that you can perform other open (for read access) operations on the profile.
<code>FILE_SHARE_WRITE</code>	Specifies that you can perform other open (for write access) operations on the profile. This flag value is ignored when a WCS profile is opened.

#### `dwCreationMode`

A flag value that specifies the actions to take while opening a color profile if it is contained in a file. This parameter must take one of the following values, which are defined in *winbase.h*:

<b>Value</b>	<b>Description</b>
<code>CREATE_NEW</code>	Specifies that a new profile is created. This function fails if the profile already exists.
<code>CREATE_ALWAYS</code>	Specifies that a new profile is created. If a profile already exists, it is overwritten.
<code>OPEN_EXISTING</code>	Specifies that the profile is opened. This function fails if the profile does not exist.

Value	Description
OPEN_ALWAYS	Specifies that the profile is to be opened if an International Color Consortium (ICC) file exists. If an ICC profile does not exist, WCS creates a new ICC profile. The function will fail for WCS profiles if this flag is set and a WCS profile does not exist.
TRUNCATE_EXISTING	Specifies that the profile is to be opened and truncated to zero bytes. The function fails if the profile does not exist.

### dwFlags

A flag value that specifies whether to use the embedded WCS profile. This parameter has no effect unless *pCDMPProfile* specifies an ICC profile that contains an embedded WCS profile.

This parameter takes one of the following values:

Value	Description
0	Specifies that the embedded WCS profile will be used and the ICC profile specified by <i>pCDMPPProfile</i> will be ignored.
DONT_USE_EMBEDDED_WCS_PROFILES	Specifies that the ICC profile specified by <i>pCDMPPProfile</i> will be used and the embedded WCS profile will be ignored.

## Return value

If this function succeeds, the return value is the handle of the color profile that is opened.

If this function fails, the return value is **NULL**.

## Remarks

This API will take a set of DMP, CAMP, and GMMP and return a WCS profile handle. **NULL** values for GMMP are valid. A **NULL** value for CAMP will be replaced with the default CAMP value.

This API will also accept ICC profiles. Using an ICC profile does not guarantee processing by the WCS CITE engine. The WCS engine will only be used if it is passed at least one WCS profile. Pure ICC workflows will be consistent with legacy behavior.

You can use the handle that this function returns in other color profile management functions.

The *dwCreationMode* flags CREATE\_NEW, CREATE\_ALWAYS, and TRUNCATE\_EXISTING will always return blank ICC HPROFILEs. If other *dwCreationMode* flags are present, the function will determine whether the profile is ICC or WCS XML.

Within the ICC code path, an ICC HPROFILE is returned using the requested sharing, access, and creation flags as specified in the tables above.

Within the WCS path, the *dwCreationMode* flag OPEN\_ALWAYS will fail if the profile does not exist, because WCS profiles cannot be created or edited within the WCS architecture (they must be edited outside of it, using MSXML6). For the same reason, *dwShareMode* flag FILE\_SHARE\_WRITE, and the *dwDesiredAccess* flag PROFILE\_READWRITE are ignored within the WCS path.

Once the handle to the color profile is created, any information used to create that handle can be deleted.

Use the [CloseColorProfile](#) function to close an object handle that is returned by [WcsOpenColorProfile](#).

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Windows Color System schemas and algorithms](#)
- [Functions](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WcsSetCalibrationManagementState function (icm.h)

Article02/22/2024

Enables or disables system management of the display calibration state.

## Syntax

C++

```
BOOL WcsSetCalibrationManagementState(
    BOOL bIsEnabled
);
```

## Parameters

bIsEnabled

**TRUE** to enable system management of the display calibration state. **FALSE** to disable system management of the display calibration state.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**.

## Remarks

This function must be called with elevated permissions for it to succeed.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# WcsSetDefaultColorProfile function (icm.h)

Article10/05/2021

Sets the default color profile name for the specified profile type in the specified profile management scope.

## ⓘ Note

This API does not support "advanced color" profiles for HDR monitors. Use [ColorProfileSetDisplayDefaultAssociation](#) for managing advanced color profiles.

## Syntax

C++

```
BOOL WcsSetDefaultColorProfile(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    PCWSTR pDeviceName,
    COLORPROFILETYPE cptColorProfileType,
    COLORPROFILESUBTYPE cpstColorProfileSubType,
    DWORD dwProfileID,
    LPCWSTR pProfileName
);
```

## Parameters

scope

A [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#) value that specifies the scope of this profile management operation.

pDeviceName

A pointer to the name of the device for which the default color profile is to be set. If **NULL**, a device-independent default profile is used.

cptColorProfileType

A [COLORPROFILETYPE](#) value that specifies the color profile type.

`cpstColorProfileSubType`

A [COLORPROFILESUBTYPE](#) value that specifies the color profile subtype.

`dwProfileID`

The ID of the color space that the color profile represents. This is a custom ID value used to uniquely identify the color space profile within your application.

`pProfileName`

A pointer to a buffer that holds the name of the color profile. See Remarks.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

If the *pProfileName* parameter is **NULL** and the *profileManagementScope* parameter is **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER**, subsequent calls to **WcsSetDefaultColorProfile** will return the system-wide default profile.

If *profileManagementScope* is **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER**, this function is executable in Least-Privileged User Account (LUA) context. Otherwise, administrative privileges are required. The specified profile must already be installed, but it may be not yet associated with the specified device in the specified profile management scope.

If *profileManagementScope* is **WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER**, this function will not correctly function if launched from system context and not a User Account.

When **WcsSetDefaultColorProfile** is called to set a device model profile DMP as the default profile for the RGB or custom working space, only a DMP profile that is of type **RGBVirtualDevice**, **LCD**, or **CRT** is valid; all others are invalid.

When **WcsSetDefaultColorProfile** is called to set an International Color Consortium (ICC) profile as the default profile for the RGB or custom working space, only an ICC profile with class "spac" or "disp", and "RGB" color space is valid; all others are invalid.

See notes on valid profile type/subtype combinations.

# Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Windows Color System schemas and algorithms](#)
- [Functions](#)
- [COLORPROFILESUBTYPE](#)
- [COLORPROFILETYPE](#)
- [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#)
- [WcsGetDefaultColorProfileSize](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# WcsSetDefaultRenderingIntent function (icm.h)

Article02/22/2024

Sets the default rendering intent in the specified profile management scope.

## Syntax

C++

```
BOOL WcsSetDefaultRenderingIntent(
    WCS_PROFILE_MANAGEMENT_SCOPE scope,
    DWORD dwRenderingIntent
);
```

## Parameters

scope

The profile management scope for this operation, which can be system-wide or the current user only.

dwRenderingIntent

The rendering intent. It can be set to one of the following values:

INTENT\_PERCEPTUAL

INTENT\_RELATIVE\_COLORIMETRIC

INTENT\_SATURATION

INTENT\_ABSOLUTE\_COLORIMETRIC

DWORD\_MAX

If *dwRenderingIntent* is DWORD\_MAX and *scope* is WCS\_PROFILE\_MANAGEMENT\_SCOPE\_CURRENT\_USER, the default rendering intent for the current user reverts to the system-wide default.

For more information, see [Rendering intents](#).

# Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Windows Color System schemas and algorithms](#)
- [Functions](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# WcsSetUsePerUserProfiles function (icm.h)

Article 10/05/2021

Enables a user to specify whether or not to use a per-user profile association list for the specified device.

## Syntax

C++

```
BOOL WcsSetUsePerUserProfiles(
    LPCWSTR pDeviceName,
    DWORD    dwDeviceClass,
    BOOL     usePerUserProfiles
);
```

## Parameters

pDeviceName

A pointer to a string that contains the user-friendly name of the device.

dwDeviceClass

A flag value that specifies the class of the device. This parameter must take one of the following values:

Value	Description
CLASS_MONITOR	Specifies a display device.
CLASS_PRINTER	Specifies a printer.
CLASS_SCANNER	Specifies an image capture device.

usePerUserProfiles

A Boolean value that is **TRUE** if the user wants to use a per-user profile association list for the specified device; otherwise **FALSE**.

# Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call [GetLastError](#).

## Remarks

If *usePerUserProfiles* is **TRUE**, and the user is not already using a per-user profile association list for *pDeviceName*, then the per-user profile association list is initialized by making a copy of the system-wide profile association list for the same device. From then on, changes to the system-wide list are not included in the per-user list.

This function is executable in Least-Privileged User Account (LUA) context.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- [Basic color management concepts](#)
- [Windows Color System schemas and algorithms](#)
- [Functions](#)
- [WcsGetUsePerUserProfiles](#)

---

## Feedback

Was this page helpful?



Yes



No

Get help at Microsoft Q&A

# WcsTranslateColors function (icm.h)

Article02/22/2024

Translates an array of colors from the source color space to the destination color space as defined by a color transform.

## Syntax

C++

```
BOOL WcsTranslateColors(
    HTRANSFORM    hColorTransform,
    DWORD         nColors,
    DWORD         nInputChannels,
    COLORDATATYPE cdtInput,
    DWORD         cbInput,
    PVOID         pInputData,
    DWORD         nOutputChannels,
    COLORDATATYPE cdtOutput,
    DWORD         cbOutput,
    PVOID         pOutputData
);
```

## Parameters

`hColorTransform`

A handle for the WCS color transform.

`nColors`

The number of elements in the array to which `pInputData` and `pOutputData` point.

`nInputChannels`

The number of channels per element in the array to which `pInputData` points.

`cdtInput`

The input **COLORDATATYPE** color data type.

`cbInput`

The buffer size, in bytes, of `pInputData`.

`pInputData`

A pointer to an array of input colors. The size of the buffer for this array, in bytes, is the **DWORD** value of *cbInput*.

`nOutputChannels`

The number of channels per element in the array to which *pOutputData* points.

`cdtOutput`

The **COLORDATATYPE** output that specified the color data type.

`cbOutput`

The buffer size, in bytes, of *pOutputData*.

`pOutputData`

A pointer to an array of colors that receives the results of the color translation. The size of the buffer for this array, in bytes, is the **DWORD** value of *cbOutput*.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**. For extended error information, call **GetLastError**.

## Remarks

If the input and the output color data types are not compatible with the color transform, this function fails. This function will fail if an ICC transform is used.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Requirement	Value
Header	icm.h
Library	Mscms.lib
DLL	Mscms.dll

## See also

- Basic color management concepts
- Windows Color System schemas and algorithms
- Functions

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# Obsolete WCS Functions

Article • 12/30/2021

This section delineates functions that are obsolete in WCS 1.0. They are currently kept for backward compatibility. However, they will be removed from the API in later releases of WCS.

Function	Description
<a href="#">UpdateICMRegKey</a>	<i>Obsolete:</i> manages color profiles and CMMs.

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# UpdateICMRegKeyA function (wingdi.h)

Article 02/09/2023

*(Obsolete; retained for backward compatibility)*

The **UpdateICMRegKey** function manages color profiles and Color Management Modules in the system.

## Syntax

C++

```
BOOL UpdateICMRegKeyA(
    DWORD reserved,
    LPSTR lpszCMID,
    LPSTR lpszFileName,
    UINT command
);
```

## Parameters

`reserved`

Reserved, must be set to zero.

`lpszCMID`

Points to a string that specifies the ICC profile identifier for the color management DLL to use with the profile.

`lpszFileName`

Points to a fully qualified ICC color profile file name or to a **DEVMODE** structure.

`command`

Specifies a function to execute. It can have one of the following values.

Value	Meaning
<code>ICM_ADDPROFILE</code>	Installs the ICC profile in the system.
<code>ICM_DELETEPROFILE</code>	Uninstalls the ICC profile from the system, but does not

	delete the file.
<b>ICM_QUERYPROFILE</b>	Determines whether the profile is already installed in the system.
<b>ICM_SETDEFAULTPROFILE</b>	Makes the profile first among equals.
<b>ICM_REGISTERICMATCHER</b>	Registers a CMM in the system. The <i>pszFileName</i> parameter points to a fully qualified path for the CMM DLL. The <i>lpszCMID</i> parameter points to a <b>DWORD</b> identifying the CMM.
<b>ICM_UNREGISTERICMATCHER</b>	Unregisters the CMM from the system. The <i>lpszCMID</i> parameter points to a <b>DWORD</b> identifying the CMM.
<b>ICM_QUERYMATCH</b>	Determines whether a profile exists based on the <b>DEVMODE</b> structure pointed to by the <i>pszFileName</i> parameter.

## Return value

If this function succeeds, the return value is **TRUE**.

If this function fails, the return value is **FALSE**.

## Remarks

Not all parameters are used by all functions. The *nCommand* parameter specifies the function to execute.

This function is retained for backward compatibility and may be removed in future versions of ICM.

**Windows 95/98/Me:** **UpdateICMRegKeyW** is supported by the Microsoft Layer for Unicode. To use this, you must add certain files to your application, as outlined in [Microsoft Layer for Unicode on Windows 95/98/Me Systems](#) ↗.

### ⓘ Note

The wingdi.h header defines **UpdateICMRegKey** as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the **UNICODE** preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

# Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h
Library	Gdi32.lib
DLL	Gdi32.dll

## See also

- [Basic color management concepts](#)
- [Obsolete WCS functions](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WCS 1.0 enumerations

Article • 12/30/2021

This section of the reference contains an alphabetical listing of the most important enumerations that are used by WCS 1.0:

- [BMFORMAT](#)
- [COLORDATATYPE](#)
- [COLORPROFILESUBTYPE](#)
- [COLORPROFILETYPE](#)
- [COLORTYPE](#)
- [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# BMFORMAT enumeration (icm.h)

Article03/13/2023

The values of the **BMFORMAT** enumerated type are used by several WCS functions to indicate the format that particular bitmaps are in.

## Syntax

C++

```
typedef enum {
    BM_x555RGB = 0x0000,
    BM_x555XYZ = 0x0101,
    BM_x555Yxy,
    BM_x555Lab,
    BM_x555G3CH,
    BM_RGBTRIPLETS = 0x0002,
    BM_BGRTRIPLETS = 0x0004,
    BM_XYZTRIPLETS = 0x0201,
    BM_YxyTRIPLETS,
    BM_LabTRIPLETS,
    BM_G3CHTRIPLETS,
    BM_5CHANNEL,
    BM_6CHANNEL,
    BM_7CHANNEL,
    BM_8CHANNEL,
    BM_GRAY,
    BM_XRGBQUADS = 0x0008,
    BM_XBGRQUADS = 0x0010,
    BM_XG3CHQUADS = 0x0304,
    BM_KYMCQUADS,
    BM_CMYKQUADS = 0x0020,
    BM_10b_RGB = 0x0009,
    BM_10b_XYZ = 0x0401,
    BM_10b_Yxy,
    BM_10b_Lab,
    BM_10b_G3CH,
    BM_NAMED_INDEX,
    BM_16b_RGB = 0x000A,
    BM_16b_XYZ = 0x0501,
    BM_16b_Yxy,
    BM_16b_Lab,
    BM_16b_G3CH,
    BM_16b_GRAY,
    BM_565RGB = 0x0001,
    BM_32b_scRGB = 0x0601,
    BM_32b_scARGB = 0x0602,
    BM_S2DOT13FIXED_scRGB = 0x0603,
    BM_S2DOT13FIXED_scARGB = 0x0604,
    BM_R10G10B10A2 = 0x0701,
    BM_R10G10B10A2_XR = 0x0702,
    BM_R16G16B16A16_FLOAT = 0x0703
} BMFORMAT;
```

## Constants

**BM\_x555RGB**

Value: 0x0000

16 bits per pixel. RGB color space. 5 bits per channel. The most significant bit is ignored.

**BM\_x555XYZ**

Value: 0x0101

16 bits per pixel. CIE device-independent XYZ color space. 5 bits per channel. The most significant bit is ignored.

**BM\_x555Yxy**

16 bits per pixel. Yxy color space. 5 bits per channel. The most significant bit is ignored.

**BM\_x555Lab**

16 bits per pixel. L\*a\*b color space. 5 bits per channel. The most significant bit is ignored.

**BM\_x555G3CH**

16 bits per pixel. G3CH color space. 5 bits per channel. The most significant bit is ignored.

**BM\_RGBTRIPLETS**

Value: 0x0002

24 bits per pixel maximum. For three channel colors, such as Red,Green,Blue, the total size is 24 bits per pixel. For single channel colors, such as gray, the total size is 8 bits per pixel.

**BM\_BGRTRIPLETS**

Value: 0x0004

24 bits per pixel maximum. For three channel colors, such as Red,Green,Blue, the total size is 24 bits per pixel. For single channel colors, such as gray, the total size is 8 bits per pixel.

**BM\_XYZTRIPLETS**

Value: 0x0201

24 bits per pixel maximum. For three channel, X, Y and Z values, the total size is 24 bits per pixel. For single channel gray scale, the total size is 8 bits per pixel.

**① Note**

The TranslateBitmapBits function does not support BM\_XYZTRIPLETS as an input.

**BM\_YxyTRIPLETS**

24 bits per pixel maximum. For three channel, Y, x and y values, the total size is 24 bits per pixel. For single channel gray scale, the total size is 8 bits per pixel.

**① Note**

The TranslateBitmapBits function does not support BM\_YxyTRIPLETS as an input.

**BM\_LabTRIPLETS**

24 bits per pixel maximum. For three channel, L, a and b values, the total size is 24 bits per pixel. For single channel gray scale, the total size is 8 bits per pixel.

**BM\_G3CHTRIPLETS**

24 bits per pixel maximum. For three channel values, the total size is 24 bits per pixel. For single channel gray scale, the total size is 8 bits per pixel.

**BM\_5CHANNEL**

40 bits per pixel. 8 bits apiece are used for each channel.

**BM\_6CHANNEL**

48 bits per pixel. 8 bits apiece are used for each channel.

**BM\_7CHANNEL**

56 bits per pixel. 8 bits apiece are used for each channel.

**BM\_8CHANNEL**

64 bits per pixel. 8 bits apiece are used for each channel.

**BM\_GRAY**

32 bits per pixel. Only the 8 bit gray-scale value is used.

**BM\_xRGBQUADS**

Value: 0x0008

32 bits per pixel. 8 bits are used for each color channel. The most significant byte is ignored.

**BM\_xBGRQUADS**

Value: 0x0010

32 bits per pixel. 8 bits are used for each color channel. The most significant byte is ignored.

**BM\_xG3CHQUADS**

Value: 0x0304

32 bits per pixel. 8 bits are used for each color channel. The most significant byte is ignored.

**BM\_KYMCQUADS**

32 bits per pixel. 8 bits are used for each color channel.

**BM\_CMYKQUADS**

Value: 0x0020

32 bits per pixel. 8 bits are used for each color channel.

<p><b>BM_10b_RGB</b> Value: 0x0009 32 bits per pixel. 10 bits are used for each color channel. The 2 most significant bits are ignored.</p>
<p><b>BM_10b_XYZ</b> Value: 0x0401 32 bits per pixel. 10 bits are used for each color channel. The 2 most significant bits are ignored.</p>
<p><b>BM_10b_Yxy</b> 32 bits per pixel. 10 bits are used for each color channel. The 2 most significant bits are ignored.</p>
<p><b>BM_10b_Lab</b> 32 bits per pixel. 10 bits are used for each color channel. The 2 most significant bits are ignored.</p>
<p><b>BM_10b_G3CH</b> 32 bits per pixel. 10 bits are used for each color channel. The 2 most significant bits are ignored.</p>
<p><b>BM_NAMED_INDEX</b> 32 bits per pixel. Named color indices. Index numbering begins at 1.</p>
<p><b>BM_16b_RGB</b> Value: 0x000A 48 bits per pixel. Each channel uses 16 bits.</p>
<p><b>BM_16b_XYZ</b> Value: 0x0501 48 bits per pixel. Each channel uses 16 bits.</p>
<p><b>BM_16b_Yxy</b> 48 bits per pixel. Each channel uses 16 bits.</p>
<p><b>BM_16b_Lab</b> 48 bits per pixel. Each channel uses 16 bits.</p>
<p><b>BM_16b_G3CH</b> 48 bits per pixel. Each channel uses 16 bits.</p>
<p><b>BM_16b_GRAY</b> 16 bits per pixel.</p>
<p><b>BM_565RGB</b> Value: 0x0001 16 bits per pixel. 5 bits are used for red, 6 for green, and 5 for blue.</p>
<p><b>BM_32b_scRGB</b> Value: 0x0601 96 bits per pixel, 32 bit per channel IEEE floating point.</p>
<p><b>BM_32b_scARGB</b> Value: 0x0602 128 bits per pixel, 32 bit per channel IEEE floating point.</p>
<p><b>BM_S2DOT13FIXED_scRGB</b> Value: 0x0603 48 bits per pixel, Fixed point integer ranging from -4 to +4 with a sign bit and 2 bit exponent and 13 bit mantissa.</p>
<p><b>BM_S2DOT13FIXED_scARGB</b> Value: 0x0604 64 bits per pixel, Fixed point integer ranging from -4 to +4 with a sign bit and 2 bit exponent and 13 bit mantissa.</p>
<p><b>BM_R10G10B10A2</b> Value: 0x0701 32 bits per pixel. 10 bits are used for each color channel. The two most significant bits are alpha.</p>
<p><b>BM_R10G10B10A2_XR</b> Value: 0x0702 32 bits per pixel. 10 bits are used for each color channel. The 10 bits of each color channel are 2.8 fixed point with a -0.75 bias, giving a range of [-0.76 .. 1.25]. This range corresponds to [-0.5 .. 1.5] in a gamma = 1 space. The two most significant bits are preserved for alpha.</p>
<p>This uses an extended range (XR) sRGB color space. It has the same RGB primaries, white point, and gamma as sRGB.</p>
<p><b>BM_R16G16B16A16_FLOAT</b> Value: 0x0703 64 bits per pixel. Each channel is a 16-bit float. The last WORD is alpha.</p>

## Remarks

### Table of bitmap formats

The follow table shows, for each of the formats, the number of bits per pixel, the number of channels, the order of the channels, and the bit-by-bit structure of each byte. You might have to scroll to the right to see all the columns of the table.

Format	Bits Per Pixel	Number of Channels	Channel Ordering	Byte 0	Byte 1	Byte 2
BM_GRAY	8	1		K7K6K5K4K3K2K1K0		
BM_565RGB	16	3	BGR	G2G1G0B4B3B2B1B0	R4R3R2R1R0G5G4G3	
BM_x555RGB	16	3	BGR	G2G1G0B4B3B2B1B0	xR4R3R2R1R0G4G3	
BM_x555XYZ	16	3	ZYX	Y2Y1Y0Z4Z3Z2Z1Z0	xX4X3X2X1X0Y4Y3	
BM_x555Yxy	16	3	yxY	x2x1x0y4y3y2y1y0	xY4Y3Y2Y1Y0x4x3	
BM_x555Lab	16	3	baL	a2a1a0b4b3b2b1b0	xL4L3L2L1L0a4a3	
BM_x555G3CH	16	3	123	xC14C13C12C11C10C24C23	C22C21C20C34C33C32C31C30	
BM_16b_GRAY	16	1	K	K7K6K5K4K3K2K1K0	K15K14K13K12K11K10K9K8	
BM_RGBTRIPLETS	24	3	BGR	B7B6B5B4B3B2B1B0	G7G6G5G4G3G2G1G0	R7R6R5R4R3R2R1R0
BM_BGRTRIPLETS	24	3	RGB	R7R6R5R4R3R2R1R0	G7G6G5G4G3G2G1G0	B7B6B5B4B3B2B1B0
BM_XYZTRIPLETS	24	3	XYZ	X7X6X5X4X3X2X1X0	Y7Y6Y5Y4Y3Y2Y1Y0	Z7Z6Z5Z4Z3Z2Z1Z0
BM_YxyTRIPLETS	24	3	Yxy	Y7Y6Y5Y4Y3Y2Y1Y0	x7x6x5x4x3x2x1x0	y7y6y5y4y3y2y1y0
BM_LabTRIPLETS	24	3	Lab	L7L6L5L4L3L2L1L0	a7a6a5a4a3a2a1a0	b7b6b5b4b3b2b1b0
BM_G3CHTRIPLETS	24	3	123	C17C16C15C14C13C12C11C10	C27C26C25C24C23C22C21C20	C37C36C35C34C33C32C31C
BM_xRGBQUADS	32	3	BGRx	B7B6B5B4B3B2B1B0	G7G6G5G4G3G2G1G0	R7R6R5R4R3R2R1R0
BM_xBGRQUADS	32	3	RGBx	R7R6R5R4R3R2R1R0	G7G6G5G4G3G2G1G0	B7B6B5B4B3B2B1B0
BM_xG3CHQUADS	32	3	123x	C17C16C15C14C13C12C11C10	C27C26C25C24C23C22C21C20	C37C36C35C34C33C32C31C
BM_CMYKQUADS	32	4	KYMC	K7K6K5K4K3K2K1K0	Y7Y6Y5Y4Y3Y2Y1Y0	M7M6M5M4M3M2M1M0
BM_KYMCQUADS	32	4	CMYK	C7C6C5C4C3C2C1C0	M7M6M5M4M3M2M1M0	Y7Y6Y5Y4Y3Y2Y1Y0
BM_10b_RGB	32	3	BGR	B7B6B5B4B3B2B1B0	G5G4G3G2G1G0B9B8	R3R2R1R0G9G8G7G6
BM_10b_XYZ	32	3	ZYX	Z7Z6Z5Z4Z3Z2Z1Z0	Y5Y4Y3Y2Y1Y0Z9Z8	X3X2X1X0Y9Y8Y7Y6
BM_10b_Yxy	32	3	yxY	y7y6y5y4y3y2y1y0	x5x4x3x2x1x0y9y8	Y3Y2Y1Y0x9x8x7x6
BM_10b_Lab	32	3	baL	b7b6b5b4b3b2b1b0	a5a4a3a2a1a0b9b8	L3L2L1L0a9a8a7a6
BM_10b_G3CH	32	3	321	C37C36C35C34C33C32C31C30	C25C24C23C22C21C20C39C38	C13C12C11C10C29C28C27C
BM_NAMED_INDEX	32			n7n6n5n4n3n2n1n0	n15n14n13n12n11n10n9n8	n23n22n21n20n19n18n17n1
BM_5CHANNEL	40	5	12345	C17C16C15C14C13C12C11C10	C27C26C25C24C23C22C21C20	C37C36C35C34C33C32C31C
BM_6CHANNEL	48	6	123456	C17C16C15C14C13C12C11C10	C27C26C25C24C23C22C21C20	C37C36C35C34C33C32C31C
BM_16b_RGB	48	3	RGB	R7R6R5R4R3R2R1R0	R15R14R13R12R11R10R9R8	G7G6G5G4G3G2G1G0
BM_16b_XYZ	48	3	XYZ	X7X6X5X4X3X2X1X0	X15X14X13X12X11X10X9X8	Y7Y6Y5Y4Y3Y2Y1Y0
BM_16b_Lab	48	3	Lab	L7L6L5L4L3L2L1L0	L15L14L13L12L11L10L9L8	a7a6a5a4a3a2a1a0
BM_16b_G3CH	48	3	321	C37C36C35C34C33C32C31C30	C315C314C313C312C311C310C39C38	C27C26C25C24C23C22C21C
BM_16b_Yxy	48	3	Yxy	Y7Y6Y5Y4Y3Y2Y1Y0	Y15Y14Y13Y12Y11Y10Y9Y8	x7x6x5x4x3x2x1x0
BM_7CHANNEL	56	7	1234567	C17C16C15C14C13C12C11C10	C27C26C25C24C23C22C21C20	C37C36C35C34C33C32C31C
BM_8CHANNEL	64	8	12345678	C17C16C15C14C13C12C11C10	C27C26C25C24C23C22C21C20	C37C36C35C34C33C32C31C

Format	Bits Per Pixel	Number of Channels	Channel Ordering	Byte 0	Byte 1	Byte 2
BM_32b_scRGB	96	3	BGR			
BM_32b_scARGB	128	3	BGRA			
BM_S2DOT13FIXED_scRGB	48	3	BGR			
BM_S2DOT13FIXED_scARGB	64	3	BGRA			
BM_R10G10B10A2	32	3	ABGR	A7A6B5B4B3B2B1B0	B7B6B5B4G3G2G1G0	G7G6G5G4G3G2R1R0
BM_R10G10B10A2_XR	32	3	ABGR	A7A6B5B4B3B2B1B0	B7B6B5B4G3G2G1G0	G7G6G5G4G3G2R1R0
BM_R16G16B16A16_FLOAT	64	3	RGBA	R7R6R5R4R3R2R1R0	R7R6R5R4R3R2R1R0	G7G6G5G4G3G2G1G0

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## Feedback

Was this page helpful? [!\[\]\(fc7c5c3102eb0841f8e858aed9e67f8b\_img.jpg\) Yes](#) [!\[\]\(ad973a3e152ec485b0568025ccecacc2\_img.jpg\) No](#)

[Get help at Microsoft Q&A](#)

# COLORDATATYPE enumeration (icm.h)

Article 02/22/2024

Used by WCS functions to indicate the data type of vector content.

## Syntax

C++

```
typedef enum {
    COLOR_BYTE = 1,
    COLOR_WORD,
    COLOR_FLOAT,
    COLOR_S2DOT13FIXED,
    COLOR_10b_R10G10B10A2,
    COLOR_10b_R10G10B10A2_XR,
    COLOR_FLOAT16
} COLORDATATYPE;
```

## Constants

[] [Expand table](#)

**COLOR\_BYTE**

Value: 1

Color data is stored as 8 bits per channel, with a value from 0 to 255, inclusive.

**COLOR\_WORD**

Color data is stored as 16 bits per channel, with a value from 0 to 65535, inclusive.

**COLOR\_FLOAT**

Color data is stored as 32 bits value per channel, as defined by the IEEE 32-bit floating point standard.

**COLOR\_S2DOT13FIXED**

Color data is stored as 16 bits per channel, with a fixed range of -4 to +4, inclusive. A signed format is used, with 1 bit for the sign, 2 bits for the integer portion, and 13 bits for the fractional portion.

**COLOR\_10b\_R10G10B10A2**

Color data is stored as 10 bits per channel. The two most significant bits are alpha.

#### COLOR\_10b\_R10G10B10A2\_XR

Color data is stored as 10 bits per channel, 32 bits per pixel. The 10 bits of each color channel are 2.8 fixed point with a -0.75 bias, giving a range of [-0.76 .. 1.25]. This range corresponds to [-0.5 .. 1.5] in a gamma = 1 space. The two most significant bits are preserved for alpha.

This uses an extended range (XR) sRGB color space. It has the same RGB primaries, white point, and gamma as sRGB.

#### COLOR\_FLOAT16

Color data is stored as 16 bits value per channel, as defined by the IEEE 16-bit floating point standard.

## Remarks

The PCOLORDATATYPE and LPCOLORDATATYPE data types are defined as pointers to the COLORDATATYPE enumeration:

```
typedef COLORDATATYPE *PCOLORDATATYPE, *LPCOLORDATATYPE;
```

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	icm.h

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# COLORPROFILESUBTYPE enumeration (icm.h)

Article 02/22/2024

Specifies the subtype of the color profile.

## Syntax

C++

```
typedef enum {
    CPST_PERCEPTUAL,
    CPST_RELATIVE_COLORIMETRIC,
    CPST_SATURATION,
    CPST_ABSOLUTE_COLORIMETRIC,
    CPST_NONE,
    CPST_RGB_WORKING_SPACE,
    CPST_CUSTOM_WORKING_SPACE,
    CPST_STANDARD_DISPLAY_COLOR_MODE,
    CPST_EXTENDED_DISPLAY_COLOR_MODE
} COLORPROFILESUBTYPE;
```

## Constants

[+] Expand table

<b>CPST_PERCEPTUAL</b> A perceptual rendering intent for gamut map model profiles (GMMPs) defined in WCS.
<b>CPST_RELATIVE_COLORIMETRIC</b> A relative colorimetric rendering intent for GMMPs defined in WCS.
<b>CPST_SATURATION</b> A saturation rendering intent for GMMPs defined in WCS.
<b>CPST_ABSOLUTE_COLORIMETRIC</b> An absolute colorimetric rendering intent for GMMPs defined in WCS.
<b>CPST_NONE</b> The color profile subtype is not applicable to the selected color profile type.

<code>CPST_RGB_WORKING_SPACE</code>	The RGB color working space for International Color Consortium (ICC) profiles or device model profiles (DMPs) defined in WCS.
<code>CPST_CUSTOM_WORKING_SPACE</code>	A custom color working space.
<code>CPST_STANDARD_DISPLAY_COLOR_MODE</code>	TBD
<code>CPST_EXTENDED_DISPLAY_COLOR_MODE</code>	TBD

## Remarks

For a description of rendering intents, see [Rendering Intents](#).

The PCOLORPROFILESUBTYPE and LPCOLORPROFILESUBTYPE data types are defined as pointers to the **COLORPROFILESUBTYPE** enumeration:

```
typedef COLORPROFILESUBTYPE *PCOLORPROFILESUBTYPE, *LPCOLORPROFILESUBTYPE;
```

The valid profile type/subtype combinations are

\${ROWSpan3}\$ COLORPROFILETYPE  
 \${REMOVE}\$  
 Valid COLORPROFILESUBTYPE

Valid COLORPROFILESUBTYPE

\${ROWSpan3}\$ Notes  
 \${REMOVE}\$  
 Valid COLORPROFILESUBTYPE

default for a device

global default

Intended Usage

Intended Usage

CPT\_ICC

CPST\_NONE

Get/Set default ICC profile associated with a device

CPST\_RGBWorkingSpace or CPST\_CustomWorkingSpace

Get/Set ICC profile as global RGB or custom working space

CPT\_DMP

CPST\_NONE

Get/Set default DMP profile associated with a device

CPST\_RGBWorkingSpace or CPST\_CustomWorkingSpace

Get/Set DMP as global RGB or custom working space

CPT\_CAMP

CPST\_NONE

Get/Set default CAMP profile associated with a device

CPST\_NONE

Get/Set CAMP profile as global color appearance profile

CPT\_GMMP

CPST\_NONE

Get/Set default GMMP profile associated with a device

CPST\_Perceptual or

CPST\_Absolute\_colorimetric or

CPST\_Relative\_colorimetric or

CPTS\_Saturation

Get/Set GMMP as global gamut map model profile for a specific rendering intent as described by that subtype to be used in CreateMultiProfileTransform API when resolving the rendering intent array in WCS transform.

COLORPROFILESUBTYPE Global default can be or'd with WCS\_DEFAULT to set this GMMP as the global default for use in OpenColorProfile or WcsOpenColorProfile where GMMP is NULL.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	icm.h

## See also

- [COLORPROFILETYPE](#)
- [WcsSetDefaultColorProfile](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# COLORPROFILETYPE enumeration (icm.h)

Article 02/22/2024

Specifies the type of color profile.

## Syntax

C++

```
typedef enum {
    CPT_ICC,
    CPT_DMP,
    CPT_CAMP,
    CPT_GMMP
} COLORPROFILETYPE;
```

## Constants

[ ] [Expand table](#)

**CPT\_ICC**

An International Color Consortium (ICC) profile. If you specify this value, only the CPST\_RGB\_WORKING\_SPACE and CPST\_CUSTOM\_WORKING\_SPACE values of [COLORPROFILESUBTYPE](#) are valid.

**CPT\_DMP**

A device model profile (DMP) defined in WCS. If you specify this value, only the CPST\_RGB\_WORKING\_SPACE and CPST\_CUSTOM\_WORKING\_SPACE values of [COLORPROFILESUBTYPE](#) are valid.

**CPT\_CAMP**

A color appearance model profile (CAMP) defined in WCS. If you specify this value, only the CPST\_NONE value of [COLORPROFILESUBTYPE](#) is valid.

**CPT\_GMMP**

Specifies a WCS gamut map model profile (GMMP). If this value is specified, only the CPST\_PERCEPTUAL, CPST\_SATURATION, CPST\_RELATIVE\_COLORIMETRIC, and CPST\_ABSOLUTE\_COLORIMETRIC values of [COLORPROFILESUBTYPE](#) are valid. Any of these values may optionally be combined (in a bitwise **OR** operation) with CPST\_DEFAULT.

# Remarks

The PCOLORPROFILETYPE and LPCOLORPROFILETYPE data types are defined as pointers to this enumeration:

```
typedef COLORPROFILETYPE *PCOLORPROFILETYPE, *LPCOLORPROFILETYPE;
```

# Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	icm.h

## See also

[COLORPROFILESUBTYPE](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# COLORTYPE enumeration (icm.h)

Article 02/22/2024

The values of the **COLORTYPE** enumeration are used by several WCS functions. Variables of type **COLOR** are defined in the color spaces enumerated by the **COLORTYPE** enumeration.

## Syntax

C++

```
typedef enum {
    COLOR_GRAY = 1,
    COLOR_RGB,
    COLOR_XYZ,
    COLOR_Yxy,
    COLOR_Lab,
    COLOR_3_CHANNEL,
    COLOR_CMYK,
    COLOR_5_CHANNEL,
    COLOR_6_CHANNEL,
    COLOR_7_CHANNEL,
    COLOR_8_CHANNEL,
    COLOR_NAMED
} COLORTYPE;
```

## Constants

[ ] Expand table

**COLOR\_GRAY**

Value: 1

The **COLOR** is in the GRAYCOLOR color space.

**COLOR\_RGB**

The **COLOR** is in the RGBCOLOR color space.

**COLOR\_XYZ**

The **COLOR** is in the XYZCOLOR color space.

**COLOR\_Yxy**

The **COLOR** is in the YxyCOLOR color space.

#### COLOR\_Lab

The COLOR is in the LabCOLOR color space.

#### COLOR\_3\_CHANNEL

The COLOR is in the GENERIC3CHANNEL color space.

#### COLOR\_CMYK

The COLOR is in the CMYKCOLOR color space.

#### COLOR\_5\_CHANNEL

The COLOR is in a five channel color space.

#### COLOR\_6\_CHANNEL

The COLOR is in a six channel color space.

#### COLOR\_7\_CHANNEL

The COLOR is in a seven channel color space.

#### COLOR\_8\_CHANNEL

The COLOR is in an eight channel color space.

#### COLOR\_NAMED

The COLOR is in a named color space.

## Remarks

In addition to managing the common two, three, and four channel color spaces, WCS 1.0 is able to perform color management with device profiles that contain five through eight [color channels](#). It is also able to use named color spaces. When five, six, seven, or eight color channels are used, the provider of the device profile is free to determine what the color channels represent. The same is true of named color spaces. WCS 1.0 is able to manage these color spaces as long as there is a mapping in the device profile that maps the channels or the name space into the [PCS](#). The device profile must also contain a mapping from the PCS into the five, size, seven, or eight channel space, or into the named color space.

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# WCS\_PROFILE\_MANAGEMENT\_SCOPE enumeration (icm.h)

Article02/22/2024

Specifies the scope of a profile management operation, such as associating a profile with a device.

## Syntax

C++

```
typedef enum {
    WCS_PROFILE_MANAGEMENT_SCOPE_SYSTEM_WIDE,
    WCS_PROFILE_MANAGEMENT_SCOPE_CURRENT_USER
} WCS_PROFILE_MANAGEMENT_SCOPE;
```

## Constants

[ ] Expand table

`WCS_PROFILE_MANAGEMENT_SCOPE_SYSTEM_WIDE`

Indicates that the profile management operation affects all users.

`WCS_PROFILE_MANAGEMENT_SCOPE_CURRENT_USER`

Indicates that the profile management operation affects only the current user.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	icm.h

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# Interfaces for the Windows Color System

Article • 12/30/2021

Lists the most important interfaces that the Windows Color System uses:

- [IDeviceModelPlugIn](#)
- [IGamutMapModelPlugIn](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn interface (wcsplugin.h)

Article 02/16/2023

Describes the methods that are defined for the **IDeviceModelPlugIn** Component Object Model (COM) interface.

- [ColorimetricToDeviceColors](#)
- [DeviceToColorimetricColors](#)
- [DeviceToColorimetricColorsWithBlack](#)
- [GetGamutBoundaryMesh](#)
- [GetGamutBoundaryMeshSize](#)
- [GetNeutralAxis](#)
- [GetNeutralAxisSize](#)
- [GetNumChannels](#)
- [GetPrimarySamples](#)
- [Initialize](#)
- [SetTransformDeviceModelInfo](#)

## Inheritance

The **IDeviceModelPlugIn** interface inherits from the **IUnknown** interface.

## Methods

The **IDeviceModelPlugIn** interface has these methods.

<a href="#">IDeviceModelPlugIn::ColorimetricToDeviceColors</a>
Returns the appropriate XYZ colors in response to the specified number of colors, channels, device colors and the proprietary plug-in algorithms. ( <b>IDeviceModelPlugIn::ColorimetricToDeviceColors</b> )

<a href="#">IDeviceModelPlugIn::ColorimetricToDeviceColorsWithBlack</a>
Returns the appropriate device colors in response to the incoming number of colors, channels, black information, Commission Internationale l'Eclairge XYZ (CIEXYZ) colors and the proprietary plug-in algorithms.

## [IDeviceModelPlugIn::DeviceToColorimetricColors](#)

Returns the appropriate XYZ colors in response to the specified number of colors, channels, device colors and the proprietary plug-in algorithms. (IDeviceModelPlugIn.DeviceToColorimetricColors)

## [IDeviceModelPlugIn::GetGamutBoundaryMesh](#)

Returns the triangular mesh from the plug-in. This function is used to compute the GamutBoundaryDescription.

## [IDeviceModelPlugIn::GetGamutBoundaryMeshSize](#)

Returns the required data structure sizes for the GetGamutBoundaryMesh function.

## [IDeviceModelPlugIn::GetNeutralAxis](#)

The IDeviceModelPlugIn::GetNeutralAxis return the XYZ colorimetry of sample points along the device's neutral axis.

## [IDeviceModelPlugIn::GetNeutralAxisSize](#)

The IDeviceModelPlugIn::GetNeutralAxisSize function returns the number of data points along the neutral axis that are returned by the GetNeutralAxis function.

## [IDeviceModelPlugIn::GetNumChannels](#)

Returns the number of device channels in the parameter pNumChannels.

## [IDeviceModelPlugIn::GetPrimarySamples](#)

Returns the measurement color for the primary sample.

## [IDeviceModelPlugIn::Initialize](#)

Takes a pointer to a Stream that contains the whole device model plug-in as input, and initializes any internal parameters required by the plug-in.

## [IDeviceModelPlugIn::SetTransformDeviceModellInfo](#)

Provides the plug-in with parameters to determine where in the transform sequence the specific plug-in occurs.

# Requirements

Target Platform	Windows
-----------------	---------

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::ColorimetricToDeviceColors method (wcsplugin.h)

Article 02/22/2024

Returns the appropriate XYZ colors in response to the specified number of colors, channels, device colors and the proprietary plug-in algorithms.

## Syntax

C++

```
HRESULT ColorimetricToDeviceColors(
    [in]  UINT          cColors,
    [in]  UINT          cChannels,
    [in]  const XYZColorF *pXYZColors,
    [out] FLOAT         *pDeviceValues
);
```

## Parameters

[in] cColors

The number of colors in the *pXYZColors* and *pDeviceValues* arrays.

[in] cChannels

The number of color channels in the *pDeviceValues* arrays.

[in] pXYZColors

The pointer to the array of incoming XYZColors to convert to device colors.

[out] pDeviceValues

The pointer to an array that contains the resulting outgoing device color values.

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL.

# Remarks

If *cColors* or *cChannels* is zero, the return value is E\_FAIL. If there are invalid or out of gamut colors (which may occur if there has been prior processing of the gamut map model), then the return value is E\_FAIL.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugIn](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::ColorimetricToDeviceColorsWithBlack method (wcsplugin.h)

Article 02/22/2024

Returns the appropriate device colors in response to the incoming number of colors, channels, black information, Commission Internationale l'Eclairge XYZ (CIEXYZ) colors and the proprietary plug-in algorithms.

## Syntax

C++

```
HRESULT ColorimetricToDeviceColorsWithBlack(
    [in]  UINT           cColors,
    [in]  UINT           cChannels,
    [out] const XYZColorF *pXYZColors,
    [in]  const BlackInformation *pBlackInformation,
    [in]  FLOAT          *pDeviceValues
);
```

## Parameters

[in] cColors

The number of colors in the *pXYZColors* and *pDeviceValues* arrays.

[in] cChannels

The number of color channels in the *pDeviceValues* arrays.

[out] pXYZColors

A pointer to the array of outgoing [XYZColorF](#) structures.

[in] pBlackInformation

A pointer to the [BlackInformation](#).

[in] pDeviceValues

A pointer to the array of incoming device colors that are to be converted to `XYZColorF` structures.

## Return value

If this function succeeds, the return value is `S_OK`.

If this function fails, the return value is `E_FAIL`. For extended error information, call `GetLastError`.

## Remarks

If `cColors` or `cChannels` is zero, the return value is `E_FAIL`.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	<code>wcsplugin.h</code>

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugIn](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::DeviceToColorimetricColors method (wcsplugin.h)

Article 02/22/2024

Returns the appropriate XYZ colors in response to the specified number of colors, channels, device colors and the proprietary plug-in algorithms.

## Syntax

C++

```
HRESULT DeviceToColorimetricColors(  
    [in]  UINT      cColors,  
    [in]  UINT      cChannels,  
    [in]  const FLOAT *pDeviceValues,  
    [out] XYZColorF *pXYZColors  
) ;
```

## Parameters

[in] cColors

The number of colors in the *pXYZColors* and *pDeviceValues* arrays.

[in] cChannels

The number of color channels in the *pDeviceValues* arrays.

[in] pDeviceValues

A pointer to the array of outgoing XYZColors.

[out] pXYZColors

A pointer to the array of incoming device colors to convert to XYZColors.

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL. For extended error information, call [GetLastError](#).

## Remarks

If *cColors* or *cChannels* is zero, the return value is E\_FAIL.

## Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugIn](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::GetGamutBoundaryMesh method (wcsplugin.h)

Article 10/13/2021

Returns the triangular mesh from the plug-in. This function is used to compute the GamutBoundaryDescription.

## Syntax

C++

```
HRESULT GetGamutBoundaryMesh(
    [in]  UINT          cChannels,
    [in]  UINT          cVertices,
    [in]  UINT          cTriangles,
    [out] FLOAT         *pVertices,
    [out] GamutShellTriangle *pTriangles
);
```

## Parameters

[in] cChannels

The number of channels.

[in] cVertices

The number of vertices.

[in] cTriangles

The number of triangles.

[out] pVertices

A pointer to the array of vertices in the plug-in model gamut shell.

[out] pTriangles

A pointer to the array of triangles in the plug-in model gamut shell.

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL.

## Remarks

This function is called by the [CreateMultiProfileTransform](#) function.

## Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugIn](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::GetGamutBoundaryMeshSize method (wcsplugin.h)

Article 02/22/2024

Returns the required data structure sizes for the [GetGamutBoundaryMesh](#) function.

## Syntax

C++

```
HRESULT GetGamutBoundaryMeshSize(
    [out] UINT *pNumVertices,
    [out] UINT *pNumTriangles
);
```

## Parameters

`[out] pNumVertices`

The required number of vertices.

`[out] pNumTriangles`

The required number of triangles.

## Return value

If this function succeeds, the return value is S\_OK.

If the plug-in device model wants WCS to compute its gamut boundary mesh, the return value is S\_FALSE.

If this function fails, the return value is E\_FAIL.

## Remarks

This function is called by the [CreateMultiProfileTransform](#) function.

## Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugIn](#)

---

## Feedback

Was this page helpful?

[!\[\]\(46bde1e8a10e446aa02ed956dc88d6fa\_img.jpg\) Yes](#)[!\[\]\(a94fa1b983362304ad31a2fa90ae94b7\_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::GetNeutralAxis method (wcsplugin.h)

Article 10/13/2021

The [IDeviceModelPlugIn::GetNeutralAxis](#) return the XYZ colorimetry of sample points along the device's neutral axis.

## Syntax

C++

```
HRESULT GetNeutralAxis(
    [in]  UINT      cColors,
    [out] XYZColorF *pXYZColors
);
```

## Parameters

[in] cColors

The number of points that are returned.

[out] pXYZColors

A pointer to an array of [XYZColorF](#) structures.

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL.

## Remarks

You should define "neutral axis" in a way that is appropriate for your device. Usually, it is points made by the device's gray values. This might be R=G=B, or C=M=Y=0 and any value of K. For some devices, the most pleasing gray may be one that uses a different combination of colorants, such as M=Y=0 and C=K. The plug-in is responsible for

determining the colorimetry of a sampling of the neutral axis values and returning them. The sampling may be as sparse as two points (white and black) or as dense as desired.

There is no requirement that the samples be uniformly spaced in any color space.

## Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugIn](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::GetNeutralAxisSize method (wcsplugin.h)

Article 02/22/2024

The [IDeviceModelPlugIn::GetNeutralAxisSize](#) function returns the number of data points along the neutral axis that are returned by the [GetNeutralAxis](#) function. It is provided so that a Color Management Module (CMM) can allocate an appropriately sized buffer.

## Syntax

C++

```
HRESULT GetNeutralAxisSize(  
    [out] UINT *pcColors  
) ;
```

## Parameters

[out] pcColors

The number of points that will be returned by a call to [GetNeutralAxis](#). Minimum is 2 (black and white).

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL.

## Requirements

  Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugIn](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::GetNumChannels method (wcsplugin.h)

Article 02/22/2024

Returns the number of device channels in the parameter *pNumChannels*.

## Syntax

C++

```
HRESULT GetNumChannels(  
    [out] UINT *pNumChannels  
);
```

## Parameters

[out] *pNumChannels*

A pointer to an unsigned integer representing the number of color channels for your device.

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- Basic color management concepts
  - Functions
  - IDeviceModelPlugIn
- 

## Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::GetPrimarySamples method (wcsplugin.h)

Article02/22/2024

Returns the measurement color for the primary sample.

## Syntax

C++

```
HRESULT GetPrimarySamples(
    [out] PrimaryXYZColors *pPrimaryColor
);
```

## Parameters

[out] pPrimaryColor

The primary color type, which is determined by using the hue circle order. If the plugin device model does not natively support primaries for red, yellow, green, cyan, blue, magenta, black and white, it must still return virtual primary data.

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugIn](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::Initialize method (wcsplugin.h)

Article02/22/2024

Takes a pointer to a Stream that contains the whole device model plug-in as input, and initializes any internal parameters required by the plug-in.

## Syntax

C++

```
HRESULT Initialize(  
    [in] BSTR bstrXml,  
    [in] UINT cNumModels,  
    [in] UINT iModelPosition  
);
```

## Parameters

[in] `bstrXml`

A string that contains the BSTR XML device model plug-in profile. This parameter stores data as little-endian Unicode XML; however, it may have no leading bytes to tag it as such. Also, the encoding keyword in the XML may not reflect that this is formatted as little-endian Unicode. Furthermore, due to the action of the MSXML engine, the BSTR XML file is processed and might not have exactly the same contents as the original XML file.

[in] `cNumModels`

The number of total models in the transform sequence.

[in] `iModelPosition`

The one-based model position of the other device model in the workflow of `uiNumModels` as provided in the `Initialize` function.

## Return value

If this function succeeds, the return value is `S_OK`.

If this function fails, the return value is E\_FAIL.

## Remarks

If this function is called more than once, subsequent calls release any allocated memory and reinitialize according to the new *bstrXml* parameter.

## Requirements

  Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugIn](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# IDeviceModelPlugIn::SetTransformDeviceModelInfo method (wcsplugin.h)

Article 02/22/2024

Provides the plug-in with parameters to determine where in the transform sequence the specific plug-in occurs.

## Syntax

C++

```
HRESULT SetTransformDeviceModelInfo(
    [in] UINT             iModelPosition,
    [in] IDeviceModelPlugIn *pIDeviceModelOther
);
```

## Parameters

[in] iModelPosition

The one-based model position of the other device model in the workflow of *uiNumModels*, as provided in the [Initialize](#) function.

[in] pIDeviceModelOther

A pointer to a **IDeviceModelPlugIn** interface that contains the other device model in the transform sequence.

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL.

## Remarks

This function is called by the [CreateMultiProfileTransform](#) function, which is responsible for calling **AddRef** and **Release** as appropriate. The function enables plug-in device

models to exchange information in a proprietary manner by accessing proprietary plug-in interface functions.

This function will fail if the other device model is a baseline device model, because such models are not plugins and thus inter-plugin communication is not supported.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [IDeviceModelPlugin](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# **IGamutMapModelPlugIn interface (wcsplugin.h)**

Article 02/16/2023

Describes the methods that are defined for the **IGamutMapModelPlugIn** Component Object Model (COM) interface.

- [IGamutMapModelPlugIn::Initialize](#)
- [IGamutMapModelPlugIn::SourceToDestinationAppearanceColors](#)

## **Inheritance**

The **IGamutMapModelPlugIn** interface inherits from the **IUnknown** interface.

## **Methods**

The **IGamutMapModelPlugIn** interface has these methods.

<a href="#">IGamutMapModelPlugIn::Initialize</a>
Initializes a gamut map model profile (GMMP) by using the specified source and destination gamut boundary descriptions and optional source and destination device model plug-ins.
<a href="#">IGamutMapModelPlugIn::SourceToDestinationAppearanceColors</a>
Returns the appropriate gamut-mapped appearance colors in response to the specified number of colors and the CIEJCh colors.

## **Requirements**

Target Platform	Windows
Header	wcsplugin.h

## **Feedback**

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# IGamutMapModelPlugIn::Initialize method (wcsplugin.h)

Article 02/22/2024

Initializes a gamut map model profile (GMMP) by using the specified source and destination gamut boundary descriptions and optional source and destination device model plug-ins.

## Syntax

C++

```
HRESULT Initialize(  
    [in] BSTR                 bstrXml,  
    [in] IDeviceModelPlugIn   *pSrcPlugIn,  
    [in] IDeviceModelPlugIn   *pDestPlugIn,  
    [in] GamutBoundaryDescription *pSrcGBD,  
    [in] GamutBoundaryDescription *pDestGBD  
);
```

## Parameters

[in] bstrXml

A string that contains the BSTR XML GMMP profile. This is little-endian Unicode XML, but without the leading bytes to tag it as such. Also, the encoding keyword in the XML may not reflect this format.

[in] pSrcPlugIn

A pointer to a source [IDeviceModelPlugIn](#). If **NULL**, it indicates the source device model profile is not a plug-in profile.

[in] pDestPlugIn

A pointer to a destination [IDeviceModelPlugIn](#). If **NULL**, it indicates the destination device model profile is not a plug-in profile.

[in] pSrcGBD

A pointer to a source [GamutBoundaryDescription](#).

[in] pDestGBD

A pointer to a destination [GamutBoundaryDescription](#).

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL.

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [IGamutMapModelPlugIn](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# IGamutMapModelPlugIn::SourceToDestinationAppearanceColors method (wcsplugin.h)

Article 02/22/2024

Returns the appropriate gamut-mapped appearance colors in response to the specified number of colors and the [CIEJCh](#) colors.

## Syntax

C++

```
HRESULT SourceToDestinationAppearanceColors(
    [in]  UINT          cColors,
    [in]  const JChColorF *pInputColors,
    [out] JChColorF      *pOutputColors
);
```

## Parameters

[in] cColors

The number of colors in the *pXYZColors* and *pDeviceValues* arrays.

[in] pInputColors

A pointer to the array of incoming colors to be gamut mapped.

[out] pOutputColors

A pointer to the array of outgoing colors.

## Return value

If this function succeeds, the return value is S\_OK.

If this function fails, the return value is E\_FAIL.

## Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [IGamutMapModelPlugIn](#)

---

## Feedback

Was this page helpful?

[!\[\]\(64bbb989650f9ef653726ee181d5325a\_img.jpg\) Yes](#)[!\[\]\(7abfb3dddb280a359a46407739c44f77\_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# WCS 1.0 structures

Article • 06/09/2022

This section of the reference contains a listing of the most important structures that are used by WCS 1.0. These are listed as follows:

- [BlackInformation](#)
- [CIEXYZ](#)
- [CIEXYZTRIPLE](#)
- [COLORMATCHSETUPW](#)
- [ENUMTYPEW](#)
- [GamutBoundaryDescription](#)
- [Gamut Map Model Color Structures ↗](#)
- [GamutShell](#)
- [GamutShellTriangle](#)
- [ICM Color Structures](#)
- [LOGCOLORSPACE](#)
- [NAMED\\_PROFILE\\_INFO](#)
- [PrimaryJabColors](#)
- [PrimaryXYZColors](#)
- [PROFILE](#)
- [PROFILEHEADER](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# BlackInformation structure (wcsplugin.h)

Article 10/05/2021

Contains information for device models that have a black color channel.

## Syntax

C++

```
typedef struct _BlackInformation {
    BOOL    fBlackOnly;
    FLOAT   blackWeight;
} BlackInformation;
```

## Members

fBlackOnly

blackWeight

A value between 0.0 and 1.0 that indicates the relative amount of black to use in the output. A value of 0.0 means that no black is used; a value of 1.0 means that the maximum amount of black is used.

## Remarks

If the source device does not support a black channel, then WCS sets **bBlackOnly** to FALSE.

If **bBlackOnly** is TRUE, then WCS generates an output device control value where, at most, the black channel will be non-zero. This only happens if the **BlackPreservation** flag was set in WCS. Note that in such cases, the device model may not be providing the closest colorimetric match to the supplied value.

Black preservation is only performed when both the source and destination devices support a black channel. If black is being preserved with these devices, then for each source device control value, where all channels other than the black channel are zero, the **bBlackOnly** flag is TRUE. Note that this means that a value where all channels are zero will also set **bBlackOnly** to TRUE.

**blackWeight** gives us information about the device control values used in the source device.

- For source devices with a black channel, **blackWeight** is set to the black value.
- For source devices without a black channel, the black weight is computed using a combination of *color purity* and *relative lightness*. *Color purity* is defined as  $(\text{maxColorant} - \text{minColorant})/\text{maxColorant}$

*Relative lightness* is defined as  $(\text{the lightness of the color in appearance space} - \text{minimum lightness of destination device}) / (\text{maximum lightness of destination device} - \text{minimum lightness of destination device})$

For RGB devices,  $\text{blackWeight} = (1 - \text{colorPurity}) * (1 - \text{relativeLightness})$

For CMYK devices,  $\text{blackWeight} = \text{colorPurity} * (1 - \text{relativeLightness})$

WCS is responsible for initializing the **BlackInformation** structure.

If **bBlackOnly** is **FALSE**, then the baseline device models for devices with a black channel will use the **blackWeight** to guide the creation of a colorimetrically appropriate output pixel value. For CMYK devices, **blackWeight** provides WCS's initial estimation of a K value and it searches for C, M, and Y values that will lead to the correct colorimetry. If it does not find a match, it adjusts the K value and searches again.

You can set plug-ins to either support or ignore the **BlackInformation**.

## Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# CIEXYZ structure (wingdi.h)

Article 02/22/2024

The **CIEXYZ** structure contains the *x*, *y*, and *z* coordinates of a specific color in a specified color space.

## Syntax

C++

```
typedef struct tagCIEXYZ {
    FXPT2DOT30 ciexyzX;
    FXPT2DOT30 ciexyzY;
    FXPT2DOT30 ciexyzZ;
} CIEXYZ;
```

## Members

`ciexyzX`

The *x* coordinate in fix point (2.30).

`ciexyzY`

The *y* coordinate in fix point (2.30).

`ciexyzZ`

The *z* coordinate in fix point (2.30).

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h

## See also

- Basic color management concepts
  - Structures
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CIEXYZTRIPLE structure (wingdi.h)

Article02/22/2024

The **CIEXYZTRIPLE** structure contains the *x*, *y*, and *z* coordinates of the three colors that correspond to the red, green, and blue endpoints for a specified logical color space.

## Syntax

C++

```
typedef struct tagCIEXYZTRIPLE {  
    CIEXYZ ciexyzRed;  
    CIEXYZ ciexyzGreen;  
    CIEXYZ ciexyzBlue;  
} CIEXYZTRIPLE;
```

## Members

`ciexyzRed`

The xyz coordinates of red endpoint.

`ciexyzGreen`

The xyz coordinates of green endpoint.

`ciexyzBlue`

The xyz coordinates of blue endpoint.

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	wingdi.h

## See also

- Basic color management concepts
  - Structures
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# CMYKCOLOR structure (icm.h)

Article02/22/2024

Description of the CMYKCOLOR structure.

## Syntax

C++

```
struct CMYKCOLOR {
    WORD cyan;
    WORD magenta;
    WORD yellow;
    WORD black;
};
```

## Members

cyan

TBD

magenta

TBD

yellow

TBD

black

TBD

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]

Requirement	Value
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# COLOR union (icm.h)

Article02/22/2024

Description of the COLOR union.

## Syntax

C++

```
typedef union tagCOLOR {
    struct GRAYCOLOR      gray;
    struct RGBCOLOR       rgb;
    struct CMYKCOLOR      cmyk;
    struct XYZCOLOR       XYZ;
    struct YxyCOLOR       Yxy;
    struct LabCOLOR       Lab;
    struct GENERIC3CHANNEL gen3ch;
    struct NAMEDCOLOR     named;
    struct HiFiCOLOR      hifi;
    struct {
        DWORD reserved1;
        VOID  *reserved2;
    };
} COLOR;
```

## Members

gray

TBD

rgb

TBD

cmyk

TBD

XYZ

TBD

Yxy

TBD

Lab

TBD

gen3ch

TBD

named

TBD

hifi

TBD

reserved1

TBD

reserved2

TBD

## Remarks

A variable of type COLOR may be accessed as any of the supported color space colors by accessing the appropriate member of the union. For instance, given the following variable declaration:

```
COLOR aColor;
```

the red, green and blue values could be set in the following manner:

```
aColor.rgb.red=100;
```

```
aColor.rgb.green=50;
```

```
aColor.rgb.blue=2;
```

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# COLORMATCHSETUPW structure (icm.h)

Article07/27/2022

The **COLORMATCHSETUP** structure contains information that the [SetupColorMatchingW](#) function uses to initialize the **ColorManagement** dialog box.

After the user closes the dialog box, [SetupColorMatching](#) returns information about the user's selection in this structure.

## Syntax

C++

```
typedef struct _tagCOLORMATCHSETUPW {
    DWORD          dwSize;
    DWORD          dwVersion;
    DWORD          dwFlags;
    HWND           hwndOwner;
    PCWSTR         pSourceName;
    PCWSTR         pDisplayName;
    PCWSTR         pPrinterName;
    DWORD          dwRenderIntent;
    DWORD          dwProofingIntent;
    PWSTR          pMonitorProfile;
    DWORD          ccMonitorProfile;
    PWSTR          pPrinterProfile;
    DWORD          ccPrinterProfile;
    PWSTR          pTargetProfile;
    DWORD          ccTargetProfile;
    DLGPROC        lpfnHook;
    LPARAM         lParam;
    PCMSCALLBACKW lpfnApplyCallback;
    LPARAM         lParamApplyCallback;
} COLORMATCHSETUPW, *PCOLORMATCHSETUPW, *LPCOLORMATCHSETUPW;
```

## Members

**dwSize**

Size of the structure. Should be set to **sizeof ( COLORMATCHSETUP )**.

**dwVersion**

Version of the **COLORMATCHSETUP** structure. This should be set to **COLOR\_MATCH\_VERSION**.

## **dwFlags**

A set of bit flags used to initialize the dialog box. If set to 0 on entry, all controls assume their default states.

When the dialog box returns, these flags are set to indicate the user's input.

This member can be set using a combination of the following flags.

<b>Flag</b>	<b>Meaning</b>
CMS_DISABLEICM	If set on entry, this flag indicates that the "Enable Color Management" check box is cleared, disabling all other controls. If set on exit, it means that the user does not wish color management performed.
CMS_ENABLEPROOFING	If set on entry, this flag indicates that the Proofing controls are to be enabled, and the Proofing check box is checked. If set on exit, it means that the user wishes to perform color management for a different target device than the selected printer.
CMS_SETRENDERINTENT	If set on entry, this flag indicates that the <b>dwRenderIntent</b> member contains the value to use to initialize the Rendering Intent control. Otherwise, the control defaults to Picture rendering. This flag is set on exit if WCS is enabled.
CMS_SETPROOFINTENT	Ignored unless CMS_ENABLEPROOFING is also set. If set on entry, and CMS_ENABLEPROOFING is also set, this flag indicates that the <b>dwProofingIntent</b> member is to be used to initialize the Target Rendering Intent control. Otherwise, the control defaults to Picture rendering. This flag is set on exit if proofing is enabled.
CMS_SETMONITORPROFILE	If set on entry, this flag indicates that the color management profile named in the <b>pMonitorProfile</b> member is to be the initial selection in the monitor profile control. If the specified profile is not associated with the monitor, this flag is ignored, and the default profile for the monitor is used.
CMS_SETPRINTERPROFILE	If set on entry, this flag indicates that the color management profile named in the <b>pPrinterProfile</b> member is to be the initial selection in the printer profile control. If the specified profile is not associated with the printer, this flag is ignored, and the default profile for the printer is used.
CMS_SETTARGETPROFILE	If set on entry, this flag indicates that the color profile named in the <b>pTargetProfile</b> member is to be the initial selection in the target profile control. If the specified profile is not installed, this flag is ignored, and the default profile for the printer is used. If the printer has no default profile, then the first profile in alphabetical order will be displayed.

Flag	Meaning
CMS_USEHOOK	This flag specifies that the <i>lpfnHook</i> member contains the address of a hook procedure, and the <i>IParam</i> member contains a value to be passed to the hook procedure when the WM_INITDIALOG message is sent.
CMS_MONITOROVERFLOW	This flag is set on exit if color management is to be enabled and the buffer size given in <i>ccMonitorProfile</i> is insufficient for the selected profile name. <b>GetLastError</b> returns ERROR_INSUFFICIENT_BUFFER in such a case.
CMS_PRINTEROVERFLOW	This flag is set on exit if color management is to be enabled and the buffer size given in <i>ccPrinterProfile</i> is insufficient for the selected profile name. <b>GetLastError</b> returns ERROR_INSUFFICIENT_BUFFER in such a case.
CMS_TARGETOVERFLOW	This flag is set on exit if proofing is to be enabled and the buffer size given in <i>ccTargetProfile</i> is insufficient for the selected profile name. <b>GetLastError</b> returns ERROR_INSUFFICIENT_BUFFER in such a case.
CMS_USEAPPLYCALLBACK	If set on entry, this flag indicates that the <b>SetupColorMatching</b> function should call the function <b>PCMSCALLBACKW</b> . The address of the callback function is contained in <i>lpfnApplyCallback</i> .
CMS_USEDESCRIPTION	If set on entry, this flag instructs the <b>SetupColorMatching</b> function to retrieve the profile description contained in the profile description tags (See ICC Profile Format Specification v3.4). It will insert them into the <b>Monitor Profile</b> , <b>Printer Profile</b> , <b>Emulated Device Profile</b> edit boxes in the <b>Color Management</b> common dialog box.

**hwndOwner**

The window handle to the owner of the dialog box, or **NULL** if the dialog box has no owner.

**pSourceName**

Pointer to an application-specified string which describes the source profile of the item for which color management is to be performed. If this is **NULL**, the Image Source control displays the name of the Windows default color profile.

**pDisplayName**

Points to a string naming the monitor to be used for color management. If this is not the name of a valid monitor, the first enumerated monitor is used.

`pPrinterName`

Points to a string naming the printer on which the image is to be rendered. If this is not a valid printer name, the default printer is used and named in the dialog.

`dwRenderIntent`

The type of color management desired. Valid values are:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering intents](#).

`dwProofingIntent`

The type of color management desired for the proofed image. Valid values are:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering intents](#).

`pMonitorProfile`

Pointer to a buffer in which to place the name of the user-selected monitor profile. If the CMS\_SETMONITORPROFILE flag is used, this flag can also be used to select a profile other than the monitor default when the dialog is first displayed.

`ccMonitorProfile`

The size of the buffer pointed to by the `pMonitorProfile` member, in characters. If the buffer is not large enough to hold the selected name, the name is truncated to this size, and `ERROR_INSUFFICIENT_BUFFER` is returned. A buffer of `MAX_PATH` size always works.

`pPrinterProfile`

Points to a buffer in which to place the name of the user-selected printer profile. If the CMS\_SETPRINTERPROFILE flag is used, this flag can also be used to select a profile other than the printer default when the dialog is first displayed.

#### `ccPrinterProfile`

The size of the buffer pointed to by the **pPrinterProfile** member, in characters. If the buffer is not large enough to hold the selected name, the name is truncated to this size, and ERROR\_INSUFFICIENT\_BUFFER is returned. A buffer of MAX\_PATH size always works.

#### `pTargetProfile`

Points to a buffer in which to place the name of the user-selected target profile for proofing. If the CMS\_SETTARGETPROFILE flag is used, this flag can also be used to select a profile other than the printer default when the dialog is first displayed.

#### `ccTargetProfile`

The size of the buffer pointed to by the **pTargetProfile** member, in characters. If the buffer is not large enough to hold the selected name, the name is truncated to this size, and ERROR\_INSUFFICIENT\_BUFFER is returned. A buffer of MAX\_PATH size always works.

#### `lpfnHook`

If the CMS\_USEHOOK flag is set, this member is the address of a dialog procedure (see [DialogProc](#)) that can filter or handle messages for the dialog. The hook procedure receives no messages issued before WM\_INITDIALOG. It is called on the WM\_INITDIALOG message after the system-provided dialog procedure has processed the message. On all other messages, the hook procedure receives the message before the system-provided procedure. If the hook procedure returns **TRUE** to these messages, the system-provided procedure is not called.

The hook procedure may call the [EndDialog](#) function.

#### `lParam`

If the CMS\_USEHOOK flag is set, this member is passed to the application-provided hook procedure as the *lParam* parameter when the WM\_INITDIALOG message is processed.

#### `lpfnApplyCallback`

Contains a pointer to a callback function that is invoked when the **Apply** button of the Color Management dialog box is selected. If no callback function is provided, this member should be set to **NULL**. See [PCMSCALLBACKW](#).

## `IParamApplyCallback`

Contains a value that will be passed to the function `ApplyCallbackFunction` through its `IParam` parameter. The meaning and content of the value is specified by the application.

# Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [A common dialog for color management](#)
- [DialogProc ↗](#)
- [EndDialog ↗](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# COLORMATCHSETUPW structure (icm.h)

Article 07/27/2022

The **COLORMATCHSETUP** structure contains information that the [SetupColorMatchingW](#) function uses to initialize the **ColorManagement** dialog box.

After the user closes the dialog box, [SetupColorMatching](#) returns information about the user's selection in this structure.

## Syntax

C++

```
typedef struct _tagCOLORMATCHSETUPW {
    DWORD          dwSize;
    DWORD          dwVersion;
    DWORD          dwFlags;
    HWND           hwndOwner;
    PCWSTR         pSourceName;
    PCWSTR         pDisplayName;
    PCWSTR         pPrinterName;
    DWORD          dwRenderIntent;
    DWORD          dwProofingIntent;
    PWSTR          pMonitorProfile;
    DWORD          ccMonitorProfile;
    PWSTR          pPrinterProfile;
    DWORD          ccPrinterProfile;
    PWSTR          pTargetProfile;
    DWORD          ccTargetProfile;
    DLGPROC        lpfnHook;
    LPARAM         lParam;
    PCMSCALLBACKW lpfnApplyCallback;
    LPARAM         lParamApplyCallback;
} COLORMATCHSETUPW, *PCOLORMATCHSETUPW, *LPCOLORMATCHSETUPW;
```

## Members

**dwSize**

Size of the structure. Should be set to **sizeof ( COLORMATCHSETUP )**.

**dwVersion**

Version of the **COLORMATCHSETUP** structure. This should be set to **COLOR\_MATCH\_VERSION**.

## **dwFlags**

A set of bit flags used to initialize the dialog box. If set to 0 on entry, all controls assume their default states.

When the dialog box returns, these flags are set to indicate the user's input.

This member can be set using a combination of the following flags.

<b>Flag</b>	<b>Meaning</b>
CMS_DISABLEICM	If set on entry, this flag indicates that the "Enable Color Management" check box is cleared, disabling all other controls. If set on exit, it means that the user does not wish color management performed.
CMS_ENABLEPROOFING	If set on entry, this flag indicates that the Proofing controls are to be enabled, and the Proofing check box is checked. If set on exit, it means that the user wishes to perform color management for a different target device than the selected printer.
CMS_SETRENDERINTENT	If set on entry, this flag indicates that the <b>dwRenderIntent</b> member contains the value to use to initialize the Rendering Intent control. Otherwise, the control defaults to Picture rendering. This flag is set on exit if WCS is enabled.
CMS_SETPROOFINTENT	Ignored unless CMS_ENABLEPROOFING is also set. If set on entry, and CMS_ENABLEPROOFING is also set, this flag indicates that the <b>dwProofingIntent</b> member is to be used to initialize the Target Rendering Intent control. Otherwise, the control defaults to Picture rendering. This flag is set on exit if proofing is enabled.
CMS_SETMONITORPROFILE	If set on entry, this flag indicates that the color management profile named in the <b>pMonitorProfile</b> member is to be the initial selection in the monitor profile control. If the specified profile is not associated with the monitor, this flag is ignored, and the default profile for the monitor is used.
CMS_SETPRINTERPROFILE	If set on entry, this flag indicates that the color management profile named in the <b>pPrinterProfile</b> member is to be the initial selection in the printer profile control. If the specified profile is not associated with the printer, this flag is ignored, and the default profile for the printer is used.
CMS_SETTARGETPROFILE	If set on entry, this flag indicates that the color profile named in the <b>pTargetProfile</b> member is to be the initial selection in the target profile control. If the specified profile is not installed, this flag is ignored, and the default profile for the printer is used. If the printer has no default profile, then the first profile in alphabetical order will be displayed.

Flag	Meaning
CMS_USEHOOK	This flag specifies that the <i>lpfnHook</i> member contains the address of a hook procedure, and the <i>IParam</i> member contains a value to be passed to the hook procedure when the WM_INITDIALOG message is sent.
CMS_MONITOROVERFLOW	This flag is set on exit if color management is to be enabled and the buffer size given in <i>ccMonitorProfile</i> is insufficient for the selected profile name. <b>GetLastError</b> returns ERROR_INSUFFICIENT_BUFFER in such a case.
CMS_PRINTEROVERFLOW	This flag is set on exit if color management is to be enabled and the buffer size given in <i>ccPrinterProfile</i> is insufficient for the selected profile name. <b>GetLastError</b> returns ERROR_INSUFFICIENT_BUFFER in such a case.
CMS_TARGETOVERFLOW	This flag is set on exit if proofing is to be enabled and the buffer size given in <i>ccTargetProfile</i> is insufficient for the selected profile name. <b>GetLastError</b> returns ERROR_INSUFFICIENT_BUFFER in such a case.
CMS_USEAPPLYCALLBACK	If set on entry, this flag indicates that the <b>SetupColorMatching</b> function should call the function <b>PCMSCALLBACKW</b> . The address of the callback function is contained in <i>lpfnApplyCallback</i> .
CMS_USEDESCRIPTION	If set on entry, this flag instructs the <b>SetupColorMatching</b> function to retrieve the profile description contained in the profile description tags (See ICC Profile Format Specification v3.4). It will insert them into the <b>Monitor Profile</b> , <b>Printer Profile</b> , <b>Emulated Device Profile</b> edit boxes in the <b>Color Management</b> common dialog box.

**hwndOwner**

The window handle to the owner of the dialog box, or **NULL** if the dialog box has no owner.

**pSourceName**

Pointer to an application-specified string which describes the source profile of the item for which color management is to be performed. If this is **NULL**, the Image Source control displays the name of the Windows default color profile.

**pDisplayName**

Points to a string naming the monitor to be used for color management. If this is not the name of a valid monitor, the first enumerated monitor is used.

`pPrinterName`

Points to a string naming the printer on which the image is to be rendered. If this is not a valid printer name, the default printer is used and named in the dialog.

`dwRenderIntent`

The type of color management desired. Valid values are:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering intents](#).

`dwProofingIntent`

The type of color management desired for the proofed image. Valid values are:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering intents](#).

`pMonitorProfile`

Pointer to a buffer in which to place the name of the user-selected monitor profile. If the CMS\_SETMONITORPROFILE flag is used, this flag can also be used to select a profile other than the monitor default when the dialog is first displayed.

`ccMonitorProfile`

The size of the buffer pointed to by the `pMonitorProfile` member, in characters. If the buffer is not large enough to hold the selected name, the name is truncated to this size, and `ERROR_INSUFFICIENT_BUFFER` is returned. A buffer of `MAX_PATH` size always works.

`pPrinterProfile`

Points to a buffer in which to place the name of the user-selected printer profile. If the CMS\_SETPRINTERPROFILE flag is used, this flag can also be used to select a profile other than the printer default when the dialog is first displayed.

#### `ccPrinterProfile`

The size of the buffer pointed to by the **pPrinterProfile** member, in characters. If the buffer is not large enough to hold the selected name, the name is truncated to this size, and ERROR\_INSUFFICIENT\_BUFFER is returned. A buffer of MAX\_PATH size always works.

#### `pTargetProfile`

Points to a buffer in which to place the name of the user-selected target profile for proofing. If the CMS\_SETTARGETPROFILE flag is used, this flag can also be used to select a profile other than the printer default when the dialog is first displayed.

#### `ccTargetProfile`

The size of the buffer pointed to by the **pTargetProfile** member, in characters. If the buffer is not large enough to hold the selected name, the name is truncated to this size, and ERROR\_INSUFFICIENT\_BUFFER is returned. A buffer of MAX\_PATH size always works.

#### `lpfnHook`

If the CMS\_USEHOOK flag is set, this member is the address of a dialog procedure (see [DialogProc](#)) that can filter or handle messages for the dialog. The hook procedure receives no messages issued before WM\_INITDIALOG. It is called on the WM\_INITDIALOG message after the system-provided dialog procedure has processed the message. On all other messages, the hook procedure receives the message before the system-provided procedure. If the hook procedure returns **TRUE** to these messages, the system-provided procedure is not called.

The hook procedure may call the [EndDialog](#) function.

#### `lParam`

If the CMS\_USEHOOK flag is set, this member is passed to the application-provided hook procedure as the *lParam* parameter when the WM\_INITDIALOG message is processed.

#### `lpfnApplyCallback`

Contains a pointer to a callback function that is invoked when the **Apply** button of the Color Management dialog box is selected. If no callback function is provided, this member should be set to **NULL**. See [PCMSCALLBACKW](#).

## `IParamApplyCallback`

Contains a value that will be passed to the function `ApplyCallbackFunction` through its `IParam` parameter. The meaning and content of the value is specified by the application.

# Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [A common dialog for color management](#)
- [DialogProc ↗](#)
- [EndDialog ↗](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ENUMTYPEA structure (icm.h)

Article09/01/2022

Contains information that defines the profile enumeration constraints.

## Syntax

C++

```
typedef struct tagENUMTYPEA {
    DWORD dwSize;
    DWORD dwVersion;
    DWORD dwFields;
    PCSTR pDeviceName;
    DWORD dwMediaType;
    DWORD dwDitheringMode;
    DWORD dwResolution[2];
    DWORD dwCMMType;
    DWORD dwClass;
    DWORD dwDataColorSpace;
    DWORD dwConnectionSpace;
    DWORD dwSignature;
    DWORD dwPlatform;
    DWORD dwProfileFlags;
    DWORD dwManufacturer;
    DWORD dwModel;
    DWORD dwAttributes[2];
    DWORD dwRenderingIntent;
    DWORD dwCreator;
    DWORD dwDeviceClass;
} ENUMTYPEA, *PENUMTYPEA, *LPENUMTYPEA;
```

## Members

`dwSize`

The size of this structure in bytes.

`dwVersion`

The version number of the **ENUMTYPE** structure. Should be set to **ENUM\_TYPE\_VERSION**.

`dwFields`

Indicates which fields in this structure are being used. Can be set to any combination of the following constant values.

ET\_DEVICENAME

ET\_MEDIATYPE

ET\_DITHERMODE

ET\_RESOLUTION

ET\_CMMTYPE

ET\_CLASS

ET\_DATACOLORSPACE

ET\_CONNECTIONSPACE

ET\_SIGNATURE

ET\_PLATFORM

ET\_PROFILEFLAGS

ET\_MANUFACTURER

ET\_MODEL

ET\_ATTRIBUTES

ET\_RENDERINGINTENT

ET\_CREATOR

ET\_DEVICECLASS

pDeviceName

User friendly name of the device.

dwMediaType

Indicates which type of media is associated with the profile, such as a printer or screen.

dwDitheringMode

Indicates the style of dithering that will be used when an image is displayed.

### **dwResolution[2]**

The horizontal (x) and vertical (y) resolution in pixels of the device on which the image will be displayed. The x resolution is stored in **dwResolution[0]**, and the y resolution is kept in **dwResolution[1]**.

### **dwCMMType**

The identification number of the CMM that is used in the profile. Identification numbers are registered with the ICC.

### **dwClass**

Indicates the profile class. For a description of profile classes, see [Using Device Profiles with WCS](#). A profile class may have any of the following values.

<b>Profile Class</b>	<b>Signature</b>
Input Device Profile	CLASS_SCANNER
Display Device Profile	CLASS_MONITOR
Output Device Profile	CLASS_PRINTER
Device Link Profile	CLASS_LINK
Color Space Conversion Profile	CLASS_COLORSPACE
Abstract Profile	CLASS_ABSTRACT
Named Color Profile	CLASS_NAMED
Color Appearance Model Profile	CLASS_CAMP
Color Gamut Map Model Profile	CLASS_GMMP

### **dwDataColorSpace**

A signature value that indicates the color space in which the profile data is defined. Can be any value from the [Color Space Constants](#).

### **dwConnectionSpace**

A signature value that indicates the color space in which the profile connection space (PCS) is defined. Can be any of the following values.

<b>Profile Class</b>	<b>Signature</b>
----------------------	------------------

<b>Profile Class</b>	<b>Signature</b>
XYZ	SPACE_XYZ
Lab	SPACE_Lab

When the dwClass member is set to CLASS\_LINK, the PCS is taken from the dwDataColorSpace member.

#### `dwSignature`

Reserved for internal use.

#### `dwPlatform`

The primary platform for which the profile was created. The member can be set to any of the following values.

<b>Platform</b>	<b>Value</b>
Apple Computer, Inc.	'APPL'
Microsoft Corp.	'MSFT'
Silicon Graphics, Inc.	'SGI'
Sun Microsystems, Inc.	'SUNW'
Taligent	'TGNT'

#### `dwProfileFlags`

Bit flags containing hints that the CMM uses to interpret the profile data and can be set to one of the following values.

<b>Constant</b>	<b>Meaning</b>
FLAG_EMBEDDEDPROFILE	The profile is embedded in a bitmap file.
FLAG_DEPENDENTONDATA	The profile can't be used independently of the embedded color data. Used for profiles that are embedded in bitmap files.

#### `dwManufacturer`

The identification number of the device profile manufacturer. All manufacturer identification numbers are registered with the ICC.

#### `dwModel`

The device manufacturer's device model number. All model identification numbers are registered with the ICC.

`dwAttributes[2]`

Attributes of profile that can be any of the following values.

<b>Constant</b>	<b>Meaning</b>
ATTRIB_TRANSPARENCY	Turns transparency on. If this flag is not used, the attribute is reflective by default.
ATTRIB_MATTE	Turns matte display on. If this flag is not used, the attribute is glossy by default.

`dwRenderingIntent`

The profile rendering intent that can be set to one of the following values:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering intents](#).

`dwCreator`

Signature of the software that created the profile. Signatures are registered with the ICC.

`dwDeviceClass`

Indicates the device class. A device class may have one of the following values.

<b>Profile Class</b>	<b>Signature</b>
Input Device Profile	CLASS_SCANNER
Display Device Profile	CLASS_MONITOR
Output Device Profile	CLASS_PRINTER

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	icm.h

## See also

- [Further information](#)
- [Using device profiles with WCS](#)
- [Rendering intents](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# ENUMTYPEW structure (icm.h)

Article09/01/2022

Contains information that defines the profile enumeration constraints.

## Syntax

C++

```
typedef struct tagENUMTYPEW {
    DWORD  dwSize;
    DWORD  dwVersion;
    DWORD  dwFields;
    PCWSTR pDeviceName;
    DWORD  dwMediaType;
    DWORD  dwDitheringMode;
    DWORD  dwResolution[2];
    DWORD  dwCMMType;
    DWORD  dwClass;
    DWORD  dwDataColorSpace;
    DWORD  dwConnectionSpace;
    DWORD  dwSignature;
    DWORD  dwPlatform;
    DWORD  dwProfileFlags;
    DWORD  dwManufacturer;
    DWORD  dwModel;
    DWORD  dwAttributes[2];
    DWORD  dwRenderingIntent;
    DWORD  dwCreator;
    DWORD  dwDeviceClass;
} ENUMTYPEW, *PENUMTYPEW, *LPENUMTYPEW;
```

## Members

`dwSize`

The size of this structure in bytes.

`dwVersion`

The version number of the **ENUMTYPE** structure. Should be set to **ENUM\_TYPE\_VERSION**.

`dwFields`

Indicates which fields in this structure are being used. Can be set to any combination of the following constant values.

ET\_DEVICENAME

ET\_MEDIATYPE

ET\_DITHERMODE

ET\_RESOLUTION

ET\_CMMTYPE

ET\_CLASS

ET\_DATACOLORSPACE

ET\_CONNECTIONSPACE

ET\_SIGNATURE

ET\_PLATFORM

ET\_PROFILEFLAGS

ET\_MANUFACTURER

ET\_MODEL

ET\_ATTRIBUTES

ET\_RENDERINGINTENT

ET\_CREATOR

ET\_DEVICECLASS

pDeviceName

User friendly name of the device.

dwMediaType

Indicates which type of media is associated with the profile, such as a printer or screen.

dwDitheringMode

Indicates the style of dithering that will be used when an image is displayed.

### **dwResolution[2]**

The horizontal (x) and vertical (y) resolution in pixels of the device on which the image will be displayed. The x resolution is stored in **dwResolution[0]**, and the y resolution is kept in **dwResolution[1]**.

### **dwCMMType**

The identification number of the CMM that is used in the profile. Identification numbers are registered with the ICC.

### **dwClass**

Indicates the profile class. For a description of profile classes, see [Using Device Profiles with WCS](#). A profile class may have any of the following values.

<b>Profile Class</b>	<b>Signature</b>
Input Device Profile	CLASS_SCANNER
Display Device Profile	CLASS_MONITOR
Output Device Profile	CLASS_PRINTER
Device Link Profile	CLASS_LINK
Color Space Conversion Profile	CLASS_COLORSPACE
Abstract Profile	CLASS_ABSTRACT
Named Color Profile	CLASS_NAMED
Color Appearance Model Profile	CLASS_CAMP
Color Gamut Map Model Profile	CLASS_GMMP

### **dwDataColorSpace**

A signature value that indicates the color space in which the profile data is defined. Can be any value from the [Color Space Constants](#).

### **dwConnectionSpace**

A signature value that indicates the color space in which the profile connection space (PCS) is defined. Can be any of the following values.

<b>Profile Class</b>	<b>Signature</b>
----------------------	------------------

<b>Profile Class</b>	<b>Signature</b>
XYZ	SPACE_XYZ
Lab	SPACE_Lab

When the dwClass member is set to CLASS\_LINK, the PCS is taken from the dwDataColorSpace member.

#### `dwSignature`

Reserved for internal use.

#### `dwPlatform`

The primary platform for which the profile was created. The member can be set to any of the following values.

<b>Platform</b>	<b>Value</b>
Apple Computer, Inc.	'APPL'
Microsoft Corp.	'MSFT'
Silicon Graphics, Inc.	'SGI'
Sun Microsystems, Inc.	'SUNW'
Taligent	'TGNT'

#### `dwProfileFlags`

Bit flags containing hints that the CMM uses to interpret the profile data and can be set to one of the following values.

<b>Constant</b>	<b>Meaning</b>
FLAG_EMBEDDEDPROFILE	The profile is embedded in a bitmap file.
FLAG_DEPENDENTONDATA	The profile can't be used independently of the embedded color data. Used for profiles that are embedded in bitmap files.

#### `dwManufacturer`

The identification number of the device profile manufacturer. All manufacturer identification numbers are registered with the ICC.

#### `dwModel`

The device manufacturer's device model number. All model identification numbers are registered with the ICC.

`dwAttributes[2]`

Attributes of profile that can be any of the following values.

<b>Constant</b>	<b>Meaning</b>
ATTRIB_TRANSPARENCY	Turns transparency on. If this flag is not used, the attribute is reflective by default.
ATTRIB_MATTE	Turns matte display on. If this flag is not used, the attribute is glossy by default.

`dwRenderingIntent`

The profile rendering intent that can be set to one of the following values:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering intents](#).

`dwCreator`

Signature of the software that created the profile. Signatures are registered with the ICC.

`dwDeviceClass`

Indicates the device class. A device class may have one of the following values.

<b>Profile Class</b>	<b>Signature</b>
Input Device Profile	CLASS_SCANNER
Display Device Profile	CLASS_MONITOR
Output Device Profile	CLASS_PRINTER

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	icm.h

## See also

- [Further information](#)
- [Using device profiles with WCS](#)
- [Rendering intents](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# GamutBoundaryDescription structure (wcsplugin.h)

Article 02/22/2024

Defines a gamut boundary.

## Syntax

C++

```
typedef struct _GamutBoundaryDescription {
    PrimaryJabColors *pPrimaries;
    UINT             cNeutralSamples;
    JabColorF        *pNeutralSamples;
    GamutShell       *pReferenceShell;
    GamutShell       *pPlausibleShell;
    GamutShell       *pPossibleShell;
} GamutBoundaryDescription;
```

## Members

pPrimaries

cNeutralSamples

The number of neutral samples.

pNeutralSamples

pReferenceShell

pPlausibleShell

pPossibleShell

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Structures](#)
- [Windows Color System schemas and algorithms](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GamutShell structure (wcsplugin.h)

Article 02/22/2024

Contains information that defines a gamut shell, which is represented by a list of indexed triangles. The vertex buffer contains the vertices data.

## Syntax

C++

```
typedef struct _GamutShell {
    FLOAT          JMin;
    FLOAT          JMax;
    UINT           cVertices;
    UINT           cTriangles;
    JabColorF      *pVertices;
    GamutShellTriangle *pTriangles;
} GamutShell;
```

## Members

JMin

The minimum lightness of the shell.

JMax

The maximum lightness of the shell.

cVertices

The number of vertices in the shell.

cTriangles

The number of triangles in the shell.

pVertices

pTriangles

## Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Structures](#)
- [Windows Color System schemas and algorithms](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# GamutShellTriangle structure (wcsplugin.h)

Article02/22/2024

Contains three vertex indices for accessing a vertex buffer.

## Syntax

C++

```
typedef struct _GamutShellTriangle {
    UINT aVertexIndex[3];
} GamutShellTriangle;
```

## Members

aVertexIndex[3]

An array of three vertex indices that are used for accessing a vertex buffer.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Structures](#)
- [Windows Color System schemas and algorithms](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GENERIC3CHANNEL structure (icm.h)

Article02/22/2024

TBD

## Syntax

C++

```
struct GENERIC3CHANNEL {
    WORD ch1;
    WORD ch2;
    WORD ch3;
};
```

## Members

ch1

TBD

ch2

TBD

ch3

TBD

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# GRAYCOLOR structure (icm.h)

Article02/22/2024

Description of the GRAYCOLOR structure.

## Syntax

C++

```
struct GRAYCOLOR {
    WORD gray;
};
```

## Members

gray

TBD

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# HiFiCOLOR structure (icm.h)

Article02/22/2024

Description of the HiFiCOLOR structure.

## Syntax

C++

```
struct HiFiCOLOR {
    BYTE channel[MAX_COLOR_CHANNELS];
};
```

## Members

channel[MAX\_COLOR\_CHANNELS]

TBD

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# JabColorF structure (wcsplugin.h)

Article 02/22/2024

TBD

## Syntax

C++

```
typedef struct _JabColorF {
    FLOAT J;
    FLOAT a;
    FLOAT b;
} JabColorF;
```

## Members

J

TBD

a

TBD

b

TBD

## Requirements

[+] Expand table

Requirement	Value
Header	wcsplugin.h

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# JChColorF structure (wcsplugin.h)

Article 02/22/2024

TBD

## Syntax

C++

```
typedef struct _JChColorF {
    FLOAT J;
    FLOAT C;
    FLOAT h;
} JChColorF;
```

## Members

J

TBD

C

TBD

h

TBD

## Requirements

[+] Expand table

Requirement	Value
Header	wcsplugin.h

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# LabCOLOR structure (icm.h)

Article02/22/2024

TBD

## Syntax

C++

```
struct LabCOLOR {
    WORD L;
    WORD a;
    WORD b;
};
```

## Members

L

TBD

a

TBD

b

TBD

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# LOGCOLORSPACEA structure (wingdi.h)

Article09/01/2022

The **LOGCOLORSPACE** structure contains information that defines a logical [color space](#).

## Syntax

C++

```
typedef struct tagLOGCOLORSPACEA {
    DWORD         lcsSignature;
    DWORD         lcsVersion;
    DWORD         lcsSize;
    LCSCSTYPE     lcsCSType;
    LCGAMUTMATCH lcsIntent;
    CIEXYZTRIPLE lcsEndpoints;
    DWORD         lcsGammaRed;
    DWORD         lcsGammaGreen;
    DWORD         lcsGammaBlue;
    CHAR          lcsFilename[MAX_PATH];
} LOGCOLORSPACEA, *LPLOGCOLORSPACEA;
```

## Members

**lcsSignature**

Color space signature. At present, this member should always be set to **LCS\_SIGNATURE**.

**lcsVersion**

Version number; must be 0x400.

**lcsSize**

Size of this structure, in bytes.

**lcsCSType**

Color space type. The member can be one of the following values.

Value	Meaning
<b>LCS_CALIBRATED_RGB</b>	Color values are calibrated RGB values. The values are translated using the endpoints specified by the <b>lcsEndpoints</b> member before being passed to the device.

LCS_sRGB	Color values are sRGB values.
LCS_WINDOWS_COLOR_SPACE	Color values are Windows default color space color values.

If LCS\_CALIBRATED\_RGB is not specified, the **lcsEndpoints** member is ignored.

#### **lcsIntent**

The gamut mapping method. This member can be one of the following values.

<b>Value</b>	<b>Intent</b>	<b>ICC Name</b>	<b>Meaning</b>
LCS_GM_ABS_	Match	Absolute Colorimetric	Maintain the white point. Match the colors to their nearest color in the destination gamut.
COLORIMETRIC			
LCS_GM_	Graphic	Saturation	Maintain saturation. Used for business charts and other situations in which undithered colors are required.
BUSINESS			
LCS_GM_	Proof	Relative Colorimetric	Maintain colorimetric match. Used for graphic designs and named colors.
GRAPHICS			
LCS_GM_	Picture	Perceptual	Maintain contrast. Used for photographs and natural images.
IMAGES			

#### **lcsEndpoints**

Red, green, blue endpoints.

#### **lcsGammaRed**

Scale of the red coordinate.

#### **lcsGammaGreen**

Scale of the green coordinate.

#### **lcsGammaBlue**

Scale of the blue coordinate.

#### **lcsFilename[MAX\_PATH]**

A null-terminated string that names a color profile file. This member is typically set to zero, but may be used to set the color space to be exactly as specified by the color profile. This is useful for devices that input color values for a specific printer, or when using an installable image color matcher. If a color profile is specified, all other members of this structure should be set to reasonable values, even if the values are not completely accurate.

## Remarks

Like palettes, but unlike pens and brushes, a pointer must be passed when creating a LogColorSpace.

If the **IcsCSType** member is set to LCS\_sRGB or LCS\_WINDOWS\_COLOR\_SPACE, the other members of this structure are ignored, and WCS uses the sRGB color space. The **IcsEndpoints**, **IcsGammaRed**, **IcsGammaGreen**, and **IcsGammaBlue** members are used to describe the logical color space. The **IcsEndpoints** member is a **CIEXYZTRIPLE** that contains the x, y, and z values of the color space's RGB endpoint.

The required DWORD bit format for the **IcsGammaRed**, **IcsGammaGreen**, and **IcsGammaBlue** is an 8.8 fixed point integer left-shifted by 8 bits. This means 8 integer bits are followed by 8 fraction bits. Taking the bit shift into account, the required format of the 32-bit DWORD is:

00000000nnnnnnnnfffffff00000000

Whenever the **IcsFilename** member contains a file name and the **IcsCSType** member is set to LCS\_CALIBRATED\_RGB, WCS ignores the other members of this structure. It uses the color space in the file as the color space to which this **LOGCOLORSPACE** structure refers.

The relation between tri-stimulus values X,Y,Z and chromaticity values x,y,z is as follows:

$$x = X/(X+Y+Z)$$

$$y = Y/(X+Y+Z)$$

$$z = Z/(X+Y+Z)$$

If the **IcsCSType** member is set to LCS\_sRGB or LCS\_WINDOWS\_COLOR\_SPACE, the other members of this structure are ignored, and ICM uses the sRGB color space. Applications should still initialize the rest of the structure since CreateProfileFromLogColorSpace ignores **IcsCSType** member and uses **IcsEndpoints**, **IcsGammaRed**, **IcsGammaGreen**, **IcsGammaBlue** members to create a profile, which may not be initialized in case of LCS\_sRGB or LCS\_WINDOWS\_COLOR\_SPACE color spaces.

## Note

The wingdi.h header defines LOGCOLORSPACE as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see [Conventions for Function Prototypes](#).

## Requirements

<b>Minimum supported client</b>	Windows 2000 Professional [desktop apps only]
<b>Minimum supported server</b>	Windows 2000 Server [desktop apps only]
<b>Header</b>	wingdi.h

## See also

[BITMAPV4HEADER](#)

[BITMAPV5HEADER](#)

[CMYK](#)

[RGB](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# NAMEDCOLOR structure (icm.h)

Article02/22/2024

TBD

## Syntax

C++

```
struct NAMEDCOLOR {
    DWORD dwIndex;
};
```

## Members

`dwIndex`

TBD

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# NAMED\_PROFILE\_INFO structure (icm.h)

Article 02/22/2024

The NAMED\_PROFILE\_INFO structure is used to store information about a named color profile.

## Syntax

C++

```
typedef struct tagNAMED_PROFILE_INFO {
    DWORD      dwFlags;
    DWORD      dwCount;
    DWORD      dwCountDevCoordinates;
    COLOR_NAME szPrefix;
    COLOR_NAME szSuffix;
} NAMED_PROFILE_INFO;
```

## Members

`dwFlags`

Not currently used by the default CMM.

`dwCount`

Total number of named colors in the profile.

`dwCountDevCoordinates`

Total number of device coordinates for each named color.

`szPrefix`

Pointer to a string containing the prefix for each color name.

`szSuffix`

Pointer to a string containing the suffix for each color name.

## Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# PrimaryJabColors structure (wcsplugin.h)

Article 02/22/2024

This structure contains eight primary colors in Jab coordinates.

## Syntax

C++

```
typedef struct _PrimaryJabColors {
    JabColorF red;
    JabColorF yellow;
    JabColorF green;
    JabColorF cyan;
    JabColorF blue;
    JabColorF magenta;
    JabColorF black;
    JabColorF white;
} PrimaryJabColors;
```

## Members

red

Red primary.

yellow

Yellow primary.

green

Green primary.

cyan

Cyan primary.

blue

Blue primary.

magenta

Magenta primary.

black

Black primary.

white

White primary.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# PrimaryXYZColors structure (wcsplugin.h)

Article 02/22/2024

This structure contains eight primary colors in XYZ coordinates.

## Syntax

C++

```
typedef struct _PrimaryXYZColors {
    XYZColorF red;
    XYZColorF yellow;
    XYZColorF green;
    XYZColorF cyan;
    XYZColorF blue;
    XYZColorF magenta;
    XYZColorF black;
    XYZColorF white;
} PrimaryXYZColors;
```

## Members

red

Red primary.

yellow

Yellow primary.

green

Green primary.

cyan

Cyan primary.

blue

Blue primary.

magenta

Magenta primary.

black

Black primary.

white

White primary.

## Requirements

[ ] Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	wcsplugin.h

## See also

- [Basic color management concepts](#)
- [Structures](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# PROFILE structure (icm.h)

Article02/22/2024

Contains information that defines a color profile. See [Using device profiles with WCS](#) for more information.

## Syntax

C++

```
typedef struct tagPROFILE {
    DWORD dwType;
    PVOID pProfileData;
    DWORD cbDataSize;
} PROFILE;
```

## Members

**dwType**

Must be set to one of the following values.

[+] [Expand table](#)

Value	Meaning
PROFILE_FILENAME	Indicates that the <b>pProfileData</b> member contains a null-terminated string that holds the name of a device profile file.
PROFILE_MEMBUFFER	Indicates that the <b>pProfileData</b> member contains a pointer to a device profile in a memory buffer.

**pProfileData**

The contents of this member is indicated by the **dwTYPE** member. It will either be a pointer to a null-terminated string containing the file name of the device profile, or it will be a pointer to a buffer in memory containing the device profile data.

**cbDataSize**

The size in bytes of the data buffer pointed to by the **pProfileData** member.

# Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [Using device profiles with WCS](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

# PROFILEHEADER structure (icm.h)

Article09/01/2022

Contains information that describes the contents of a device profile file. This header occurs at the beginning of a device profile file.

## Syntax

C++

```
typedef struct tagPROFILEHEADER {
    DWORD    phSize;
    DWORD    phCMMType;
    DWORD    phVersion;
    DWORD    phClass;
    DWORD    phDataColorSpace;
    DWORD    phConnectionSpace;
    DWORD    phDateTime[3];
    DWORD    phSignature;
    DWORD    phPlatform;
    DWORD    phProfileFlags;
    DWORD    phManufacturer;
    DWORD    phModel;
    DWORD    phAttributes[2];
    DWORD    phRenderingIntent;
    CIEXYZ  phIlluminant;
    DWORD    phCreator;
    BYTE     phReserved[44];
} PROFILEHEADER;
```

## Members

`phSize`

The size of the profile in bytes.

`phCMMType`

The identification number of the CMM that is used in the profile. Identification numbers are registered with the ICC.

`phVersion`

The version number of the profile. The version number is determined by the ICC. The current major version number is 02h. The current minor version number is 10h. The

major and minor version numbers are in binary coded decimal (BCD). They must be stored in the following format.

Byte Number	Contents
0	Major version number in BCD.
1	Minor version number in the most significant nibble of this byte. Bug fix version number in the least significant nibble.
2	Reserved. Must be set to 0.
3	Reserved. Must be set to 0.

#### phClass

Indicates the profile class. For a description of profile classes, see [Using Device Profiles with WCS](#). A profile class may have any of the following values.

Profile Class	Signature
Input Device Profile	CLASS_SCANNER
Display Device Profile	CLASS_MONITOR
Output Device Profile	CLASS_PRINTER
Device Link Profile	CLASS_LINK
Color Space Conversion Profile	CLASS_COLORSPACE
Abstract Profile	CLASS_ABSTRACT
Named Color Profile	CLASS_NAMED
Color Appearance Model Profile	CLASS_CAMP
Color Gamut Map Model Profile	CLASS_GMMP

#### phDataColorSpace

A signature value that indicates the color space in which the profile data is defined. The member can be any of value from the [Color Space Constants](#).

#### phConnectionSpace

A signature value that indicates the color space in which the profile connection space (PCS) is defined. The member can be any of the following values.

<b>Profile Class</b>	<b>Signature</b>
XYZ	SPACE_XYZ
Lab	SPACE_Lab

When the **phClass** member is set to CLASS\_LINK, the PCS is taken from the **phDataColorSpace** member.

**phDateTime[3]**

The date and time that the profile was created.

**phSignature**

Reserved for internal use.

**phPlatform**

The primary platform for which the profile was created. The primary platform can be set to any of the following values.

<b>Platform</b>	<b>Value</b>
Apple Computer, Inc.	'APPL'
Microsoft Corp.	'MSFT'
Silicon Graphics, Inc.	'SGI'
Sun Microsystems, Inc.	'SUNW'
Taligent	'TGNT'

**phProfileFlags**

Bit flags containing hints that the CMM uses to interpret the profile data. The member can be set to the following values.

<b>Constant</b>	<b>Meaning</b>
FLAG_EMBEDDEDPROFILE	The profile is embedded in a bitmap file.

Constant	Meaning
FLAG_DEPENDENTONDATA	The profile can't be used independently of the embedded color data. Used for profiles that are embedded in bitmap files.

`phManufacturer`

The identification number of the device profile manufacturer. All manufacturer identification numbers are registered with the ICC.

`phModel`

The device manufacturer's device model number. All model identification numbers are registered with the ICC.

`phAttributes[2]`

Attributes of profile. The profile attributes can be any of the following values.

Constant	Meaning
ATTRIB_TRANSPARENCY	Turns transparency on. If this flag is not used, the attribute is reflective by default.
ATTRIB_MATTE	Turns matte display on. If this flag is not used, the attribute is glossy by default.

`phRenderingIntent`

The profile rendering intent. The member can be set to one of the following values:

`INTENT_PERCEPTUAL`

`INTENT_SATURATION`

`INTENT_RELATIVE_COLORIMETRIC`

`INTENT_ABSOLUTE_COLORIMETRIC`

For more information, see [Rendering intents](#).

`phIlluminant`

Profile illuminant.

phCreator

Signature of the software that created the profile. Signatures are registered with the ICC.

phReserved[44]

Reserved.

## Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

## See also

- [Further information](#)
- [Using device profiles with WCS](#)
- [Rendering intents](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# RGBCOLOR structure (icm.h)

Article02/22/2024

TBD

## Syntax

C++

```
struct RGBCOLOR {
    WORD red;
    WORD green;
    WORD blue;
};
```

## Members

red

TBD

green

TBD

blue

TBD

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# XYZCOLOR structure (icm.h)

Article02/22/2024

TBD

## Syntax

C++

```
struct XYZCOLOR {
    WORD X;
    WORD Y;
    WORD Z;
};
```

## Members

X

TBD

Y

TBD

Z

TBD

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# XYZColorF structure (wcsplugin.h)

Article 02/22/2024

TBD

## Syntax

C++

```
typedef struct _XYZColorF {
    FLOAT X;
    FLOAT Y;
    FLOAT Z;
} XYZColorF;
```

## Members

X

TBD

Y

TBD

Z

TBD

## Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	wcsplugin.h

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

# YxyCOLOR structure (icm.h)

Article02/22/2024

TBD

## Syntax

C++

```
struct YxyCOLOR {
    WORD Y;
    WORD x;
    WORD y;
};
```

## Members

Y

TBD

x

TBD

y

TBD

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	icm.h

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

# Macros for CMYK Values and Colors

Article • 12/30/2021

The following macros are useful in manipulating CMYK values.

Macro	Description
<a href="#">CMYK</a>	Builds a CMYK value from individual cyan, magenta, yellow, and black values.
<a href="#">GetCValue</a>	Retrieves the cyan color value from a CMYK color value.
<a href="#">GetKValue</a>	Retrieves the black color value from a CMYK color value.
<a href="#">GetMValue</a>	Retrieves the magenta value from a CMYK color value.
<a href="#">GetYValue</a>	Retrieves the yellow color value from a CMYK color value.

The following macros are used with color.

Macro	Description
<a href="#">GetBValue</a>	Gets an RGB blue value.
<a href="#">GetGValue</a>	Gets an RGB green value.
<a href="#">GetRValue</a>	Gets an RGB red value.
<a href="#">PALETTEINDEX</a> ↴	Accepts an index to a logical-color palette entry and returns a palette-entry specifier.
<a href="#">PALETERGB</a>	Accepts three values that represent the relative intensities of red, green, and blue and returns a palette-relative red, green, blue (RGB) specifier.
<a href="#">RGB</a>	Selects a red, green, blue (RGB) color based on the arguments supplied and the color capabilities of the output device.

## Feedback

Was this page helpful?  Yes  No

Get help at Microsoft Q&A

# CMYK macro (wingdi.h)

The **CMYK** macro creates a CMYK color value by combining the specified cyan, magenta, yellow, and black values.

## Syntax

C++

```
COLORREF CMYK(  
    c,  
    m,  
    y,  
    k  
) ;
```

## Parameters

c

The cyan value for the color to be created.

m

The magenta value for the color to be created.

y

The yellow value for the color to be created.

k

The black value for the color to be created.

## Return value

Type: **COLORREF**

A CMYK color value.

## Requirements

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h

## See also

- [Basic color management concepts](#)
- [Functions](#)
- [GetCValue](#)
- [GetKValue](#)
- [GetMValue](#)
- [GetYValue](#)
- [Macros for CMYK values and colors](#)

---

Last updated on 07/09/2025

# GetCValue macro (wingdi.h)

The `GetCValue` macro retrieves the cyan color value from a CMYK color value.

## Syntax

C++

```
BYTE GetCValue(  
    cmyk  
)
```

## Parameters

cmyk

CMYK color value from which the cyan color value will be retrieved.

## Return value

Type: `BYTE`

The cyan color value from a CMYK color value.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h

## See also

- [Basic color management concepts](#)
- [Functions](#)

- [CMYK](#)
  - [GetKValue](#)
  - [GetMValue](#)
  - [GetYValue](#)
  - [Macros for CMYK values and colors](#)
- 

Last updated on 07/09/2025

# GetKValue macro (wingdi.h)

The `GetKValue` macro retrieves the black color value from a CMYK color value.

## Syntax

C++

```
BYTE GetKValue(  
    cmyk  
)
```

## Parameters

cmyk

CMYK color value from which the black color value will be retrieved.

## Return value

Type: `BYTE`

The black color value from a CMYK color value.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h

## See also

- [Basic color management concepts](#)
- [Functions](#)

- [CMYK](#)
  - [GetCValue](#)
  - [GetMValue](#)
  - [GetYValue](#)
  - [Macros for CMYK values and colors](#)
- 

Last updated on 07/09/2025

# GetMValue macro (wingdi.h)

The **GetMValue** macro retrieves the magenta color value from a CMYK color value.

## Syntax

C++

```
BYTE GetMValue(  
    cmyk  
)
```

## Parameters

cmyk

CMYK color value from which the magenta color value will be retrieved.

## Return value

Type: **BYTE**

The magenta color value from a CMYK color value.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h

## See also

- [Basic color management concepts](#)
- [Functions](#)

- [CMYK](#)
  - [GetCValue](#)
  - [GetKValue](#)
  - [GetYValue](#)
  - [Macros for CMYK values and colors](#)
- 

Last updated on 07/09/2025

# GetYValue macro (wingdi.h)

The `GetYValue` macro retrieves the yellow color value from a CMYK color value.

## Syntax

C++

```
BYTE GetYValue(  
    cmyk  
)
```

## Parameters

cmyk

CMYK color value from which the yellow color value will be retrieved.

## Return value

Type: `BYTE`

The yellow color value from a CMYK color value.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	wingdi.h

## See also

- [Basic color management concepts](#)
- [Functions](#)

- [CMYK](#)
  - [GetCValue](#)
  - [GetKValue](#)
  - [Macros for CMYK values and colors](#)
- 

Last updated on 07/09/2025

# WCS Constants

Article • 12/30/2021

This section contains information on constants and flag values that used by WCS.

- [Color Space Constants](#)
- [CMM Transform Creation Flags](#)
- [Color Space Type Identifiers](#)
- [Common Color Messages](#)
- [Error Codes Specific To WCS](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Color Space Constants

Article • 12/30/2021

A signature value that indicates the color space in which the profile data is defined. Can be any of the following values.

Profile Class	Signature
XYZ	SPACE_XYZ
Lab	SPACE_Lab
Luv	SPACE_Luv
YCbCr	SPACE_YCbCr
Yxy	SPACE_Yxy
RGB	SPACE_RGB
Gray scale	SPACE_GRAY
HSV	SPACE_HSV
HLS	SPACE_HLS
CMYK	SPACE_CMYK
CMY	SPACE_CMY
Generic 2 channel	SPACE_2_CHANNEL
Generic 3 channel	SPACE_3_CHANNEL
Generic 4 channel	SPACE_4_CHANNEL
Generic 5 channel	SPACE_5_CHANNEL
Generic 6 channel	SPACE_6_CHANNEL
Generic 7 channel	SPACE_7_CHANNEL
Generic 8 channel	SPACE_8_CHANNEL

## Related topics

[Basic color management concepts](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# CMM Transform Creation Flags

Article • 12/30/2021

CMMs use transform creation flags as hints for how to create a color transform. It is up to the CMM to determine how best to use these flags.

All of the functions that use these flags pass or receive flag values through a parameter called *dwFlags*. The high-order **WORD** of *dwFlags* should be set to a value from the following table.

Constant	Description
ENABLE_GAMUT_CHECKING	Use this transform for gamut checking.
USE_RELATIVE_COLORIMETRIC	Do not preserve the white point. If the output gamut does not support a given color, use the nearest supported color. See Rendering Intents.
FAST_TRANSLATE	Look up color only. Do not interpolate the color.
PRESERVEBLACK	Inserts the appropriate black generation GMMP as the last GMMP in the transform sequence
WCS_ALWAYS	Use the WCS code path even for ICC transforms.
SEQUENTIAL_TRANSFORM	Transform creation flag for creating a sequential (non-optimized) color transform.

The low-order **WORD** can have one of the following constant values.

Constant	Description
PROOF_MODE	Transform will be used to preview the image. Low image quality.
NORMAL_MODE	Transform will be used for normal image display. Average image quality.
BEST_MODE	Transform will be used for the display of the highest-quality image possible on the target device.

Moving from PROOF\_MODE to BEST\_MODE, output quality generally improves and transform speed declines.

# Related topics

[Basic color management concepts](#)

[ICM Constants](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Color Space Type Identifiers

Article • 12/30/2021

These constants specify the type of a Postscript 2 color space array. The following values are valid color space array types.

## **CSA\_A**

Get a CIEBasedA color space array from the monochrome profile.

## **CSA\_GRAY**

Get a CIEBasedA color space array from the monochrome profile.

## **CSA\_ABC**

Get a CIEBasedABC color space array from the RGB or L<sup>\*</sup>a<sup>\*</sup>b profile.

## **CSA\_DEF**

Get a CIEBasedDEF color space array from the RGB or L<sup>\*</sup>a<sup>\*</sup>b profile.

## **CSA\_RGB**

Get a CIEBasedDEF color space array followed by a CIEBasedABC color space array from the RGB profile. On execution, if the printer doesn't support CIEBasedDEF color spaces, the function uses the CIEBasedABC definition.

## **CSA\_Lab**

Gets a CIEBasedABC color space array from the L<sup>\*</sup>a<sup>\*</sup>b profile.

## **CSA\_DEFG**

Get a CIEBasedDEFG color space array from the CMYK profile.

## **CSA\_CMYK**

Get a CIEBasedCMYK color space array from the CMYK profile.

## See also

[Basic color management concepts](#)

[ICM Constants](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Common Color Messages

Article • 12/30/2021

The following messages are used with color.

- [WM\\_PALETTECHANGED](#)
- [WM\\_PALETTEISCHANGING](#)
- [WM\\_QUERYNEWPALETTE](#)
- [WM\\_SYSCOLORCHANGE](#)

## Related topics

[Basic color management concepts](#)

[ICM Constants](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Error Codes Specific To WCS

Article • 12/30/2021

The following is a list of error codes that are specifically related to the use of WCS 1.0. This does not mean that WCS functions will return only these error codes. It means that the codes in the table below are returned only by WCS functions. They may also return other Win32 error codes that are documented elsewhere in the Win32 API of the Platform SDK.

## ERROR\_DELETING\_ICM\_XFORM

The specified color transform could not be deleted.

## ERROR\_DUPLICATE\_TAG

The specified color profile element tag is already present in the color profile.

## ERROR\_ICM\_NOT\_ENABLED

Image Color Management is not currently enabled.

## ERROR\_INVALID\_CMM

The specified file name does not refer to a valid color management module.

## ERROR\_INVALID\_COLORSPACE

The specified color space is invalid.

## ERROR\_INVALID\_PROFILE

The specified file name does not refer to a valid color profile.

## ERROR\_INVALID\_TRANSFORM

The color transform that was specified is not a valid color transform.

## ERROR\_PROFILE\_NOT\_ASSOCIATED\_WITH\_DEVICE

The specified color profile is not associated with any device.

## ERROR\_PROFILE\_NOT\_FOUND

The specified color profile file name was not found. The file name or path is incorrect.

## ERROR\_TAG\_NOT\_FOUND

The specified color profile element tag was not found in the color profile.

## ERROR\_TAG\_NOT\_PRESENT

A color profile element tag that is required for the function to succeed was not passed to the function.

## Related topics

[Basic color management concepts](#)

[WCS Constants](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WCS Registry Keys

Article • 12/30/2021

WCS uses registry keys to signal that certain color profile events have occurred. Apps should query these registry keys for updated system color profile state.

## Active color profile changed

Apps may want to respond to color profile change events for a monitor device; this ensures that they always have accurate color information for their target, even if the user or another app has changed the active profile for the device.

## Desktop applications

Desktop apps should listen for changes to the registry to determine when a color profile associations has changed using [RegNotifyChangeKeyValue](#). An app should register both for per-user profile association changes, and for system-wide changes.

[RegNotifyChangeKeyValue](#) should be initialized with an HKEY provided by [RegOpenKeyEx](#). [RegOpenKeyEx](#) should be initialized using the following registry tree locations:

Per-user profile associations	HKEY_CURRENT_USER SOFTWARE\Microsoft\Windows NT\CurrentVersion\ICM\ProfileAssociations\Display\{4d36e96e-e325-11ce-bfc1-08002be10318}
System-wide profile associations	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4d36e96e-e325-11ce-bfc1-08002be10318}

When the app is notified of a registry key change, it should first query whether per-user or system-wide associations are being used by calling [WcsGetUsePerUserProfiles](#). It should then call [WcsGetDefaultColorProfile](#) with the right [WCS\\_PROFILE\\_MANAGEMENT\\_SCOPE](#) value to obtain the new active color profile for the monitor. Note that not all registry key changes will correspond to an actual change in the currently active color profile; the app must check whether the profile returned by [WcsGetDefaultColorProfile](#) has actually changed.

## Universal Windows (UWP) apps

Universal Windows Apps do not have access to the above registry keys. Instead, they should register a handler for the [DisplayInformation.ColorProfileChanged](#) event. This event fires whenever the active color profile for the monitor on which the application is running has changed. ColorProfileChanged takes into account whether per-user or system-wide profile associations are being used; this information is abstracted from UWP apps.

When responding to the ColorProfileChanged event, the app should query the currently active profile using [DisplayInformation.GetColorProfileAsync](#).

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# WCS 1.0 Glossary

A B C D G H L P R S T W

---

Last updated on 03/11/2025

# A

Article • 12/30/2021

## additive primary colors

Colors that can be added to black to mix a new color. The additive primary colors are red, green, and blue.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# B

Article • 12/30/2021

## brightness

The brightness of a color refers to the intensity of light that is reflected or transmitted by an image.

---

## Feedback

Was this page helpful?

 Yes	 No
---	--

[Get help at Microsoft Q&A](#)

# C

Article • 12/30/2021

## CAMP

The Color Appearance Model Profile Format is an XML-based file format that contains viewing conditions required to describe the relationship between the DMP CIEXYZ values and the CIEJab values.

## CMYK

The Cyan, Magenta, Yellow, and black color space. It is often implemented on printers.

## colorant

Something, especially a pigment or an ink, that colors or modifies the hue of something else.

## color channel

A component of a color space. For instance, an RGB color space has red, green, and blue color channels.

## color conversion

The process of converting colors from one color space to another.

## Color Management Module (CMM)

A code module that uses device profiles to perform color conversion and color mapping.

## color mapping

See color matching.

# color matching

Matching a converted color to its visually closest color in the destination color space.

## color model

See color space.

## color space

A mapping of color components onto a geometric coordinate system in three dimensions.

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# D

Article • 12/30/2021

## DAC

Digital to analog converter.

## device link profile

A file that contains a concatenation of color conversions that are used often.

## device profile

A file that contains information on how to characterize the color capabilities of a device. ICC device profiles convert colors from the gamut of a device into the PCS. It also holds the information necessary for conversion of colors from the PCS into the device's gamut. WCS XML device profiles provide the measurement data and algorithmic references to convert between the native device space and CIEXYZ.

## DMP

The Color Device Model Profile Format is an XML-based file format that contains measurements required to describe the relationship between native device color space values and CIEXYZ values.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# G

Article • 12/30/2021

## gamma correction

The native physical tone characteristics of a device are often different on different types of devices. The gamma correction factor compensates for these differences.

## gamma curve

A collection of gamma corrections over the entire range of colors. When plotted on a graph, they form a curve.

## gamma ramp

See gamma curve.

## gamut

The set of colors from a color space that a device can produce.

## GMMP

The Color Gamut Map Model Profile Format is an XML-based file format that controls the mapping of colors between source and destination device gamuts.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# H

Article • 12/30/2021

## hard proofing

Previewing or proofing an image by printing a hard copy of it.

## HLS

A color space often used by artists. Its components are Hue, Lightness, and Saturation (chroma).

## HSV

A color space often used by artists. Its components are Hue, Saturation (chroma), and Value (lightness).

## hue

A pure color. A hue is a specific position on the visible portion of the electromagnetic spectrum.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# L

Article • 12/30/2021

## lightness

One color channel in the HLS color space. It is a measure of the relative brightness of the color. Increasing lightness produces tints of a hue. Decreasing lightness produces shades of a hue.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# P

Article • 12/30/2021

## primary colors

The three essential colors in a color space that can be mixed to produce all other colors.

## Profile Connection Space (PCS)

A device independent color space used for color conversions and color matching.

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# R

Article • 12/30/2021

## rendering intent

Approaches to rendering color from one color space to another. The rendering intents used in ICC profiles are defined by the International Color Consortium (ICC).

## RGB

The Red, Green, and Blue color space.

## RGB Triple

A quantity that contains RGB color space values.

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# S

Article • 12/30/2021

## saturation (chroma)

Generally chroma is the amount of gray that is mixed into a hue. Zero chroma produces a shade of gray. Full chroma produces a pure hue.

## shade

A shade of a color is made by adding black to its hue.

## soft proofing

Previewing or proofing an image on a video display.

## sRGB

An international standard RGB color space proposed by Microsoft and Hewlett-Packard that is device independent. This standard has been approved as a formal international standard by the Multimedia Systems and EquipmentColour Measurement and ManagementPart 2-1: Colour ManagementDefault RGB Colour Space. It is available directly from the IEC at <https://www.iec.ch/>.

## subtractive primary colors

Colors that can be subtracted from white to mix a new color. The subtractive primary colors are cyan, yellow, and magenta.

---

## Feedback

Was this page helpful?

 Yes	 No
---	--

Get help at Microsoft Q&A

# T

Article • 12/30/2021

## tint

A tint of a color is created by mixing a hue with white.

## tone

A tone of a color is obtained by mixing its hue with gray.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# W

Article • 12/30/2021

## white point

The color value used as the reference to which the user adapts. Brighter values are possible in high dynamic range gamuts.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)