

Contents

GDI+

[Security Considerations: GDI+](#)

[About GDI+](#)

[Introduction to GDI+](#)

[Overview of GDI+](#)

[The Three Parts of GDI+](#)

[The Structure of the Class-Based Interface](#)

[What's New In GDI+?](#)

[New Features](#)

[Changes in the Programming Model](#)

[Lines, Curves, and Shapes](#)

[Overview of Vector Graphics](#)

[Pens, Lines, and Rectangles](#)

[Ellipses and Arcs](#)

[Polygons](#)

[Cardinal Splines](#)

[Bezier Splines](#)

[Paths](#)

[Brushes and Filled Shapes](#)

[Open and Closed Curves](#)

[Regions](#)

[Clipping](#)

[Flattening Paths](#)

[Antialiasing with Lines and Curves](#)

[Images, Bitmaps, and Metafiles](#)

[Types of Bitmaps](#)

[Metafiles](#)

[Drawing, Positioning, and Cloning Images](#)

[Cropping and Scaling Images](#)

Coordinate Systems and Transformations

Types of Coordinate Systems

Matrix Representation of Transformations

Global and Local Transformations

Graphics Containers

Using GDI+

Getting Started

Drawing a Line

Drawing a String

Using a Pen to Draw Lines and Shapes

Using a Pen to Draw Lines and Rectangles

Setting Pen Width and Alignment

Drawing a Line with Line Caps

Joining Lines

Drawing a Custom Dashed Line

Drawing a Line Filled with a Texture

Using a Brush to Fill Shapes

Filling a Shape with a Solid Color

Filling a Shape with a Hatch Pattern

Filling a Shape with an Image Texture

Tiling a Shape with an Image

Filling a Shape with a Color Gradient

Using Images, Bitmaps, and Metafiles

Loading and Displaying Bitmaps

Loading and Displaying Metafiles

Recording Metafiles

Cropping and Scaling Images

Rotating, Reflecting, and Skewing Images

Using Interpolation Mode to Control Image Quality During Scaling

Creating Thumbnail Images

Using a Cached Bitmap to Improve Performance

Improving Performance by Avoiding Automatic Scaling

- [Reading and Writing Metadata](#)
- [Using Image Encoders and Decoders](#)
 - [Listing Installed Encoders](#)
 - [Listing Installed Decoders](#)
 - [Retrieving the Class Identifier for an Encoder](#)
 - [Determining the Parameters Supported by an Encoder](#)
 - [Using the EncoderValue Enumeration](#)
 - [Listing Parameters and Values for All Encoders](#)
 - [Converting a BMP Image to a PNG Image](#)
 - [Setting JPEG Compression Level](#)
 - [Lossless transform of a JPEG image](#)
 - [Creating and Saving a Multiple-Frame Image](#)
 - [Copy individual frames from a multiple-frame image](#)
- [Alpha Blending Lines and Fills](#)
 - [Drawing Opaque and Semitransparent Lines](#)
 - [Drawing with Opaque and Semitransparent Brushes](#)
 - [Using Compositing Mode to Control Alpha Blending](#)
 - [Using a Color Matrix to Set Alpha Values in Images](#)
 - [Setting the Alpha Values of Individual Pixels](#)
- [Using Text and Fonts](#)
 - [Constructing Font Families and Fonts](#)
 - [Drawing Text](#)
 - [Formatting Text](#)
 - [Enumerating Installed Fonts](#)
 - [Creating a Private Font Collection](#)
 - [Obtaining Font Metrics](#)
 - [Antialiasing with Text](#)
- [Constructing and Drawing Curves](#)
 - [Drawing Cardinal Splines](#)
 - [Drawing Bezier Splines](#)
- [Filling Shapes with a Gradient Brush](#)
 - [Creating a Linear Gradient](#)

- [Creating a Path Gradient](#)
- [Applying Gamma Correction to a Gradient](#)
- [Constructing and Drawing Paths](#)
 - [Creating Figures from Lines, Curves, and Shapes](#)
 - [Filling Open Figures](#)
- [Using Graphics Containers](#)
 - [The State of a Graphics Object](#)
 - [Nested Graphics Containers](#)
- [Transformations](#)
 - [Using the World Transformation](#)
 - [Why Transformation Order Is Significant](#)
- [Using Regions](#)
 - [Hit Testing with a Region](#)
 - [Clipping with a Region](#)
- [Recoloring](#)
 - [Using a Color Matrix to Transform a Single Color](#)
 - [Translating Colors](#)
 - [Scaling Colors](#)
 - [Rotating Colors](#)
 - [Shearing Colors](#)
 - [Using a Color Remap Table](#)
- [Printing](#)
 - [Sending GDI+ Output to a Printer](#)
 - [Displaying a Print Dialog Box](#)
 - [Optimizing Printing by Providing a Printer Handle](#)
- [GDI+ Reference](#)
 - [DebugEventProc callback](#)
 - [EnumerateMetafileProc callback](#)
 - [ImageAbort callback](#)
 - [NotificationHookProc callback](#)
 - [NotificationUnhookProc callback](#)
- [Classes](#)

[AdjustableArrowCap](#)

[AdjustableArrowCap](#)

[AdjustableArrowCap Methods](#)

[GetHeight](#)

[GetMiddleInset](#)

[GetWidth](#)

[IsFilled](#)

[SetFillState](#)

[SetHeight](#)

[SetMiddleInset](#)

[SetWidth](#)

[Bitmap](#)

[Bitmap Overloaded Constructor](#)

[Bitmap\(IDirectDrawSurface7*\)](#)

[Bitmap\(INT,INT,Graphics*\)](#)

[Bitmap\(BITMAPINFO*,VOID*\)](#)

[Bitmap\(INT,INT,PixelFormat\)](#)

[Bitmap\(HBITMAP,HPalette\)](#)

[Bitmap\(INT,INT,INT,PixelFormat,BYTE*\)](#)

[Bitmap\(WCHAR*,BOOL\)](#)

[Bitmap\(HINSTANCE, WCHAR*\)](#)

[Bitmap\(HICON\)](#)

[Bitmap\(IStream*,BOOL\)](#)

[Bitmap Methods](#)

[Clone Overloaded Method](#)

[ApplyEffect Overloaded Method](#)

[ConvertFormat](#)

[FromBITMAPINFO](#)

[FromDirectDrawSurface7](#)

[FromFile](#)

[FromHBITMAP](#)

[FromHICON](#)

- [FromResource](#)
- [FromStream](#)
- [GetHBITMAP](#)
- [GetHICON](#)
- [GetHistogram](#)
- [GetHistogramSize](#)
- [GetPixel](#)
- [InitializePalette](#)
- [LockBits](#)
- [SetPixel](#)
- [SetResolution](#)
- [UnlockBits](#)
- [BitmapData](#)
- [Blur](#)
 - [Blur](#)
 - [Blur Methods](#)
 - [GetParameters](#)
 - [SetParameters](#)
- [BrightnessContrast](#)
 - [BrightnessContrast](#)
 - [BrightnessContrast Methods](#)
 - [GetParameters](#)
 - [SetParameters](#)
- [Brush](#)
 - [Brush Methods](#)
 - [Clone](#)
 - [GetLastStatus](#)
 - [GetType](#)
- [CachedBitmap](#)
 - [CachedBitmap](#)
 - [GetLastStatus](#)
- [CharacterRange](#)

CharacterRange Overloaded Constructor

CharacterRange()

CharacterRange(INT,INT)

operator=

Color

Color Overloaded Constructor

Color(BYTE,BYTE,BYTE,BYTE)

Color()

Color(BYTE,BYTE,BYTE)

Color(ARGB)

Color Methods

GetA

GetAlpha

GetB

GetBlue

GetG

GetGreen

GetR

GetRed

GetValue

MakeARGB

SetFromCOLORREF

SetValue

ToCOLORREF

ColorBalance

ColorBalance

ColorBalance Methods

GetParameters

SetParameters

ColorCurve

ColorCurve

ColorCurve Methods

- [GetParameters](#)
- [SetParameters](#)
- [ColorLUT](#)
 - [ColorLUT](#)
 - [ColorLUT Methods](#)
 - [GetParameters](#)
 - [SetParameters](#)
- [ColorMatrixEffect](#)
 - [ColorMatrixEffect](#)
 - [ColorMatrixEffect Methods](#)
 - [GetParameters](#)
 - [SetParameters](#)
- [CustomLineCap](#)
 - [CustomLineCap](#)
 - [CustomLineCap Methods](#)
 - [Clone](#)
 - [GetBaseCap](#)
 - [GetBaseInset](#)
 - [GetLastStatus](#)
 - [GetStrokeCaps](#)
 - [GetStrokeJoin](#)
 - [GetWidthScale](#)
 - [SetBaseCap](#)
 - [SetBaseInset](#)
 - [SetStrokeCap](#)
 - [SetStrokeCaps](#)
 - [SetStrokeJoin](#)
 - [SetWidthScale](#)
- [Effect](#)
 - [Effect](#)
 - [Effect Methods](#)
 - [GetAuxData](#)

[GetAuxDataSize](#)

[GetParameterSize](#)

[UseAuxData](#)

[EncoderParameter](#)

[EncoderParameters](#)

[Font](#)

[Font Overloaded Constructor](#)

[Font\(FontFamily*,REAL,INT,Unit\)](#)

[Font\(HDC,HFONT\)](#)

[Font\(HDC,LOGFONTA*\)](#)

[Font\(HDC,LOGFONTW*\)](#)

[Font\(WCHAR*,REAL,INT,Unit,FontCollection*\)](#)

[Font\(HDC\)](#)

[Font Methods](#)

[Clone](#)

[GetFamily](#)

[GetHeight Overloaded Method](#)

[GetLastStatus](#)

[GetLogFontA](#)

[GetLogFontW](#)

[GetSize](#)

[GetStyle](#)

[GetUnit](#)

[IsAvailable](#)

[FontCollection](#)

[FontCollection](#)

[FontCollection Methods](#)

[GetFamilies](#)

[GetFamilyCount](#)

[GetLastStatus](#)

[FontFamily](#)

[FontFamily Overloaded Constructor](#)

[FontFamily\(\)](#)

[FontFamily\(WCHAR*,FontCollection*\)](#)

[FontFamily Methods](#)

[Clone](#)

[GenericMonospace](#)

[GenericSansSerif](#)

[GenericSerif](#)

[GetCellAscent](#)

[GetCellDescent](#)

[GetEmHeight](#)

[GetFamilyName](#)

[GetLastStatus](#)

[GetLineSpacing](#)

[IsAvailable](#)

[IsStyleAvailable](#)

[GdiplusBase](#)

[GdiplusBase Methods](#)

[operator delete](#)

[operator delete\[](#)

[operator new](#)

[operator new\[](#)

[Graphics](#)

[Graphics Overloaded Constructor](#)

[Graphics\(Image*\)](#)

[Graphics\(HDC\)](#)

[Graphics\(HDC,HANDLE\)](#)

[Graphics\(HWND,BOOL\)](#)

[Graphics Methods](#)

[AddMetafileComment](#)

[BeginContainer Overloaded Method](#)

[Clear](#)

[DrawArc Overloaded Method](#)

DrawBezier Overloaded Method
DrawBeziers Overloaded Method
DrawCachedBitmap
DrawClosedCurve Overloaded Method
DrawCurve Overloaded Method
DrawDriverString
DrawEllipse Overloaded Method
DrawImage Overloaded Method
DrawLine Overloaded Method
DrawLines Overloaded Method
DrawPath
DrawPie Overloaded Method
DrawPolygon Overloaded Method
DrawRectangle Overloaded Method
DrawRectangles Overloaded Method
DrawString Overloaded Method
EndContainer
EnumerateMetafile Overloaded Method
ExcludeClip Overloaded Method
FillClosedCurve Overloaded Method
FillEllipse Overloaded Method
FillPath
FillPie Overloaded Method
FillPolygon Overloaded Method
FillRectangle Overloaded Method
FillRectangles Overloaded Method
FillRegion
Flush
FromHDC Overloaded Method
FromHWND
FromImage
GetClip

[GetClipBounds Overloaded Method](#)
[GetCompositingMode](#)
[GetCompositingQuality](#)
[GetDpiX](#)
[GetDpiY](#)
[GetHalftonePalette](#)
[GetHDC](#)
[GetInterpolationMode](#)
[GetLastStatus](#)
[GetNearestColor](#)
[GetPageScale](#)
[GetPageUnit](#)
[GetPixelOffsetMode](#)
[GetRenderingOrigin](#)
[GetSmoothingMode](#)
[GetTextContrast](#)
[GetTextRenderingHint](#)
[GetTransform](#)
[GetVisibleClipBounds Overloaded Method](#)
[IntersectClip Overloaded Method](#)
[IsClipEmpty](#)
[IsVisible Overloaded Method](#)
[IsVisibleClipEmpty](#)
[MeasureCharacterRanges](#)
[MeasureDriverString](#)
[MeasureString Overloaded Method](#)
[MultiplyTransform](#)
[ReleaseHDC](#)
[ResetClip](#)
[ResetTransform](#)
[Restore](#)
[RotateTransform](#)

[Save](#)

[ScaleTransform](#)

[SetAbort](#)

[SetClip Overloaded Method](#)

[SetCompositingMode](#)

[SetCompositingQuality](#)

[SetInterpolationMode](#)

[SetPageScale](#)

[SetPageUnit](#)

[SetPixelOffsetMode](#)

[SetRenderingOrigin](#)

[SetSmoothingMode](#)

[SetTextContrast](#)

[SetTextRenderingHint](#)

[SetTransform](#)

[TransformPoints](#)

[TranslateClip Overloaded Method](#)

[TranslateTransform](#)

[GraphicsPath](#)

[GraphicsPath Overloaded Constructor](#)

[GraphicsPath\(Point*,BYTE*,INT,FillMode\)](#)

[GraphicsPath\(FillMode\)](#)

[GraphicsPath\(PointF*,BYTE*,INT,FillMode\)](#)

[GraphicsPath Methods](#)

[AddArc Overloaded Method](#)

[AddBezier Overloaded Method](#)

[AddBeziers Overloaded Method](#)

[AddClosedCurve Overloaded Method](#)

[AddCurve Overloaded Method](#)

[AddEllipse Overloaded Method](#)

[AddLine Overloaded Method](#)

[AddLines Overloaded Method](#)

AddPath
AddPie Overloaded Method
AddPolygon Overloaded Method
AddRectangle Overloaded Method
AddRectangles Overloaded Method
AddString Overloaded Method
ClearMarkers
Clone
CloseAllFigures
CloseFigure
Flatten
GetBounds Overloaded Method
GetFillMode
GetLastPoint
GetLastStatus
GetPathData
GetPathPoints Overloaded Method
GetPathTypes
GetPointCount
IsOutlineVisible Overloaded Method
IsVisible Overloaded Method
Outline
Reset
Reverse
SetFillMode
SetMarker
StartFigure
Transform
Warp
Widen
GraphicsPathIterator
GraphicsPathIterator

GraphicsPathIterator Methods

[CopyData](#)

[Enumerate](#)

[GetCount](#)

[GetLastStatus](#)

[GetSubpathCount](#)

[HasCurve](#)

[NextMarker Overloaded Method](#)

[NextPathType](#)

[NextSubpath Overloaded Method](#)

[Rewind](#)

HatchBrush

[HatchBrush](#)

[HatchBrush Methods](#)

[GetBackgroundColor](#)

[GetForegroundColor](#)

[GetHatchStyle](#)

HueSaturationLightness

[HueSaturationLightness](#)

[HueSaturationLightness Methods](#)

[GetParameters](#)

[SetParameters](#)

Image

[Image Overloaded Constructor](#)

[Image\(IStream*,BOOL\)](#)

[Image\(WCHAR*,BOOL\)](#)

[Image Methods](#)

[Clone](#)

[FindFirstItem](#)

[FindNextItem](#)

[FromFile](#)

[FromStream](#)

[GetAllPropertyItems](#)
[GetBounds](#)
[GetEncoderParameterList](#)
[GetEncoderParameterListSize](#)
[GetFlags](#)
[GetFrameCount](#)
[GetFrameDimensionsCount](#)
[GetFrameDimensionsList](#)
[GetHeight](#)
[GetHorizontalResolution](#)
[GetItemData](#)
[GetLastStatus](#)
[GetPalette](#)
[GetPaletteSize](#)
[GetPhysicalDimension](#)
[GetPixelFormat](#)
[GetPropertyCount](#)
[GetPropertyIdList](#)
[GetPropertyItem](#)
[GetPropertyItemSize](#)
[GetPropertySize](#)
[GetRawFormat](#)
[GetThumbnailImage](#)
[GetType](#)
[GetVerticalResolution](#)
[GetWidth](#)
[RemovePropertyItem](#)
[RotateFlip](#)
[Save Overloaded Method](#)
[SaveAdd Overloaded Method](#)
[SelectActiveFrame](#)
[SetAbort](#)

[SetPalette](#)
[SetPropertyItem](#)
[ImageAttributes](#)
 [ImageAttributes](#)
[ImageAttributes Methods](#)
 [ClearBrushRemapTable](#)
 [ClearColorKey](#)
 [ClearColorMatrices](#)
 [ClearColorMatrix](#)
 [ClearGamma](#)
 [ClearNoOp](#)
 [ClearOutputChannel](#)
 [ClearOutputChannelColorProfile](#)
 [ClearRemapTable](#)
 [ClearThreshold](#)
 [Clone](#)
 [GetAdjustedPalette](#)
 [GetLastStatus](#)
 [Reset](#)
 [SetBrushRemapTable](#)
 [SetColorKey](#)
 [SetColorMatrices](#)
 [SetColorMatrix](#)
 [SetGamma](#)
 [SetNoOp](#)
 [SetOutputChannel](#)
 [SetOutputChannelColorProfile](#)
 [SetRemapTable](#)
 [SetThreshold](#)
 [SetTolIdentity](#)
 [SetWrapMode](#)

[ImageCodecInfo](#)

[ImageItemData](#)

[InstalledFontCollection](#)

[InstalledFontCollection](#)

[InstalledFontCollection Methods](#)

[Levels](#)

[Levels](#)

[Levels Methods](#)

[GetParameters](#)

[SetParameters](#)

[LinearGradientBrush](#)

[LinearGradientBrush Overloaded Constructor](#)

[LinearGradientBrush\(PointF&,PointF&,Color&,Color&\)](#)

[LinearGradientBrush\(Rect&,Color&,Color&,REAL,BOOL\)](#)

[LinearGradientBrush\(RectF&,Color&,Color&,LinearGradientMode\)](#)

[LinearGradientBrush\(Point&,Point&,Color&,Color&\)](#)

[LinearGradientBrush\(Rect&,Color&,Color&,REAL,BOOL\)](#)

[LinearGradientBrush\(Rect&,Color&,Color&,LinearGradientMode\)](#)

[LinearGradientBrush Methods](#)

[GetBlend](#)

[GetBlendCount](#)

[GetGammaCorrection](#)

[GetInterpolationColorCount](#)

[GetInterpolationColors](#)

[GetLinearColors](#)

[GetRectangle Overloaded Method](#)

[GetTransform](#)

[GetWrapMode](#)

[MultiplyTransform](#)

[ResetTransform](#)

[RotateTransform](#)

[ScaleTransform](#)

[SetBlend](#)

[SetBlendBellShape](#)
[SetBlendTriangularShape](#)
[SetGammaCorrection](#)
[SetInterpolationColors](#)
[SetLinearColors](#)
[SetTransform](#)
[SetWrapMode](#)
[TranslateTransform](#)

Matrix

[Matrix Overloaded Constructor](#)
[Matrix\(RectF&, PointF*\)](#)
[Matrix\(Rect&, Point*\)](#)
[Matrix\(\)](#)
[Matrix\(REAL,REAL,REAL,REAL,REAL,REAL\)](#)

Matrix Methods

[Clone](#)
[Equals](#)
[GetElements](#)
[GetLastStatus](#)
[Invert](#)
[IsIdentity](#)
[IsInvertible](#)
[Multiply](#)
[OffsetX](#)
[OffsetY](#)
[Reset](#)
[Rotate](#)
[RotateAt](#)
[Scale](#)
[SetElements](#)
[Shear](#)
[TransformPoints Overloaded Method](#)

[TransformVectors Overloaded Method](#)

[Translate](#)

[Metafile](#)

[Metafile Overloaded Constructor](#)

[Metafile\(HDC,EmfType,WCHAR*\)](#)

[Metafile\(WCHAR*\)](#)

[Metafile\(HDC,RectF&,MetaFileFrameUnit,EmfType,WCHAR*\)](#)

[Metafile\(HMETAFILE,WmfPlaceableFileHeader*,BOOL\)](#)

[Metafile\(WCHAR*,HDC,Rect&,MetaFileFrameUnit,EmfType,WCHAR*\)](#)

[Metafile\(IStream*,HDC,RectF&,MetafileFrameUnit,EmfType,WCHAR*\)](#)

[Metafile\(IStream*,HDC,EmfType,WCHAR*\)](#)

[Metafile\(IStream*,HDC,Rect&,MetafileFrameUnit,EmfType,WCHAR*\)](#)

[Metafile\(WCHAR*,HDC,RectF&,MetafileFrameUnit,EmfType,WCHAR*\)](#)

[Metafile\(WCHAR*,HDC,EmfType,WCHAR*\)](#)

[Metafile\(IStream*\)](#)

[Metafile\(HENHMETAFILE,BOOL\)](#)

[Metafile\(HDC,Rect&,MetafileFrameUnit,EmfType,WCHAR*\)](#)

[Metafile Methods](#)

[ConvertToEmfPlus Overloaded Method](#)

[EmfToWmfBits](#)

[GetDownLevelRasterizationLimit](#)

[GetHENHMETAFILE](#)

[GetMetafileHeader Overloaded Method](#)

[PlayRecord](#)

[SetDownLevelRasterizationLimit](#)

[MetafileHeader](#)

[MetafileHeader Methods](#)

[GetBounds](#)

[GetDpiX](#)

[GetDpiY](#)

[GetEmfHeader](#)

[GetEmfPlusFlags](#)

[GetMetafileSize](#)

[GetType](#)

[GetVersion](#)

[GetWmfHeader](#)

[IsDisplay](#)

[IsEmf](#)

[IsEmfOrEmfPlus](#)

[IsEmfPlus](#)

[IsEmfPlusDual](#)

[IsEmfPlusOnly](#)

[IsWmf](#)

[IsWmfPlaceable](#)

[PathData](#)

[PathData::PathData constructor](#)

[PathGradientBrush](#)

[PathGradientBrush Overloaded Constructor](#)

[PathGradientBrush\(Point*,INT,WrapMode\)](#)

[PathGradientBrush\(PointF*,INT,WrapMode\)](#)

[PathGradientBrush\(GraphicsPath*\)](#)

[PathGradientBrush Methods](#)

[GetBlendCount](#)

[GetBlend](#)

[GetCenterColor](#)

[GetCenterPoint Overloaded Method](#)

[GetFocusScales](#)

[GetGammaCorrection](#)

[GetGraphicsPath](#)

[GetInterpolationColorCount](#)

[GetInterpolationColors](#)

[GetPointCount](#)

[GetRectangle Overloaded Method](#)

[GetSurroundColorCount](#)

GetSurroundColors
GetTransform
GetWrapMode
MultiplyTransform
ResetTransform
RotateTransform
ScaleTransform
SetBlendBellShape
SetBlendTriangularShape
SetBlend
SetCenterColor
SetCenterPoint Overloaded Method
SetFocusScales
SetGammaCorrection
SetGraphicsPath
SetInterpolationColors
SetSurroundColors
SetTransform
SetWrapMode
TranslateTransform

Pen

Pen Overloaded Constructor
Pen(Brush*,REAL)
Pen(Color&,REAL)

Pen Methods

Clone
GetAlignment
GetBrush
GetColor
GetCompoundArray
GetCompoundArrayCount
GetCustomEndCap

GetCustomStartCap
GetDashCap
GetDashOffset
GetDashPattern
GetDashPatternCount
GetDashStyle
GetEndCap
GetLastStatus
GetLineJoin
GetMiterLimit
GetPenType
GetStartCap
GetTransform
GetWidth
MultiplyTransform
ResetTransform
RotateTransform
ScaleTransform
SetAlignment
SetBrush
SetColor
SetCompoundArray
SetCustomEndCap
SetCustomStartCap
SetDashCap
SetDashOffset
SetDashPattern
SetDashStyle
SetEndCap
SetLineCap
SetLineJoin
SetMiterLimit

[SetStartCap](#)

[SetTransform](#)

[SetWidth](#)

[Point](#)

[Point Overloaded Constructor](#)

[Point\(\)](#)

[Point\(INT,INT\)](#)

[Point\(Point&\)](#)

[Point\(Size&\)](#)

[Point Methods](#)

[Equals](#)

[operator+\(Point&\)](#)

[operator-\(Point&\)](#)

[PointF](#)

[PointF Overloaded Constructor](#)

[PointF\(REAL,REAL\)](#)

[PointF\(SizeF&\)](#)

[PointF\(\)](#)

[PointF\(PointF&\)](#)

[PointF Methods](#)

[Equals](#)

[operator+\(PointF&\)](#)

[operator-\(PointF&\)](#)

[PrivateFontCollection](#)

[PrivateFontCollection](#)

[PrivateFontCollection Methods](#)

[AddFontFile](#)

[AddMemoryFont](#)

[PropertyItem](#)

[Rect](#)

[Rect Overloaded Constructor](#)

[Rect\(\)](#)

`Rect(Point&,Size&)`

`Rect(INT,INT,INT,INT)`

Rect Methods

`Clone`

`Contains Overloaded Method`

`Equals`

`GetBottom`

`GetBounds`

`GetLeft`

`GetLocation`

`GetRight`

`GetSize`

`GetTop`

`Inflate Overloaded Method`

`Intersect Overloaded Method`

`IntersectsWith`

`IsEmptyArea`

`Offset Overloaded Method`

`Union`

RectF

RectF Overloaded Constructor

`RectF(PointF&,SizeF&)`

`RectF()`

`RectF(REAL,REAL,REAL,REAL)`

RectF Methods

`Clone`

`Contains Overloaded Method`

`Equals`

`GetBottom`

`GetBounds`

`GetLeft`

`GetLocation`

- [GetRight](#)
- [GetSize](#)
- [GetTop](#)
- [Inflate Overloaded Method](#)
- [Intersect Overloaded Method](#)
- [IntersectsWith](#)
- [IsEmptyArea](#)
- [Offset Overloaded Method](#)
- [Union](#)
- [RedEyeCorrection](#)
 - [RedEyeCorrection](#)
 - [RedEyeCorrection Methods](#)
 - [GetParameters](#)
 - [SetParameters](#)
- [Region](#)
 - [Region Overloaded Constructor](#)
 - [Region\(Rect&\)](#)
 - [Region\(\)](#)
 - [Region\(BYTE*,INT\)](#)
 - [Region\(GraphicsPath*\)](#)
 - [Region\(HRGN\)](#)
 - [Region\(RectF&\)](#)
 - [Region Methods](#)
 - [Clone](#)
 - [Complement Overloaded Method](#)
 - [Equals](#)
 - [Exclude Overloaded Method](#)
 - [FromHRGN](#)
 - [GetBounds Overloaded Method](#)
 - [GetData](#)
 - [GetDataSize](#)
 - [GetHRGN](#)

[GetLastStatus](#)
[GetRegionScans Overloaded Method](#)
[GetRegionScansCount](#)
[Intersect Overloaded Method](#)
[IsEmpty](#)
[IsInfinite](#)
[IsVisible Overloaded Method](#)
[MakeEmpty](#)
[MakeInfinite](#)
[Transform](#)
[Translate Overloaded Method](#)
[Union Overloaded Method](#)
[Xor Overloaded Method](#)

[Sharpen](#)
[Sharpen](#)
[Sharpen Methods](#)
 [GetParameters](#)
 [SetParameters](#)

[Size](#)
 [Size Overloaded Constructor](#)
 [Size\(\)](#)
 [Size\(INT,INT\)](#)
 [Size\(Size&\)](#)
[Size Methods](#)
 [Empty](#)
 [Equals](#)
 [operator+](#)
 [operator-](#)

[SizeF](#)
 [SizeF Overloaded Constructor](#)
 [SizeF\(\)](#)
 [SizeF\(REAL,REAL\)](#)

[SizeF\(SizeF&\)](#)

[SizeF Methods](#)

[Empty](#)

[Equals](#)

[operator+](#)

[operator-](#)

[SolidBrush](#)

[SolidBrush](#)

[SolidBrush Methods](#)

[GetColor](#)

[SetColor](#)

[StringFormat](#)

[StringFormat Overloaded Constructor](#)

[StringFormat\(INT,LANGID\)](#)

[StringFormat\(StringFormat*\)](#)

[StringFormat Methods](#)

[Clone](#)

[GenericDefault](#)

[GenericTypographic](#)

[GetAlignment](#)

[GetDigitSubstitutionLanguage](#)

[GetDigitSubstitutionMethod](#)

[GetFormatFlags](#)

[GetHotkeyPrefix](#)

[GetLastStatus](#)

[GetLineAlignment](#)

[GetMeasurableCharacterRangeCount](#)

[GetTabStopCount](#)

[GetTabStops](#)

[GetTrimming](#)

[SetAlignment](#)

[SetDigitSubstitution](#)

[SetFormatFlags](#)

[SetHotkeyPrefix](#)

[SetLineAlignment](#)

[SetMeasurableCharacterRanges](#)

[SetTabStops](#)

[SetTrimming](#)

[TextureBrush](#)

[TextureBrush Overloaded Constructor](#)

[TextureBrush\(Image*,wrapMode,RectF&\)](#)

[TextureBrush\(Image*,Rect&,ImageAttributes*\)](#)

[TextureBrush\(Image*,WrapMode,INT,INT,INT,INT\)](#)

[TextureBrush\(Image*,WrapMode,REAL,REAL,REAL,REAL\)](#)

[TextureBrush\(Image*,RectF&,ImageAttributes*\)](#)

[TextureBrush\(Image*,WrapMode\)](#)

[TextureBrush\(Image*,WrapMode,Rect&\)](#)

[TextureBrush Methods](#)

[GetImage](#)

[GetTransform](#)

[GetWrapMode](#)

[MultiplyTransform](#)

[ResetTransform](#)

[RotateTransform](#)

[ScaleTransform](#)

[SetTransform](#)

[SetWrapMode](#)

[TranslateTransform](#)

[Tint](#)

[Tint](#)

[Tint Methods](#)

[GetParameters](#)

[SetParameters](#)

[Functions](#)

[GdiplusShutdown](#)
[GdiplusStartup](#)
[GetImageDecoders](#)
[GetImageDecodersSize](#)
[GetImageEncoders](#)
[GetImageEncodersSize](#)
[GetPixelFormatSize](#)
[IsAlphaPixelFormat](#)
[IsCanonicalPixelFormat](#)
[IsExtendedPixelFormat](#)
[IsIndexedPixelFormat](#)
[ObjectTypeisValid](#)

Constants

[Image Effect Constants](#)
[Image Encoder Constants](#)
[Image File Format Constants](#)
[Image Frame Dimension Constants](#)
[Image Pixel Format Constants](#)
[Image Property Tag Type Constants](#)
[Image Property Tag Constants](#)
[Property Tags in Numerical Order](#)
[Property Tags in Alphabetical Order](#)
[Property Item Descriptions](#)
[Image File Format Specifications](#)

Enumerations

[BrushType](#)
[ColorAdjustType](#)
[ColorChannelFlags](#)
[ColorMatrixFlags](#)
[CombineMode](#)
[CompositingMode](#)
[CompositingQuality](#)

CoordinateSpace
CurveAdjustments
CurveChannel
DashCap
DashStyle
DebugEventLevel
DitherType
DriverStringOptions
EmfPlusRecordType
EmfToWmfBitsFlags
EmfType
EncoderParameterValue
EncoderValue
FillMode
FlushIntention
FontStyle
GdiplusStartupParams
HatchStyle
HistogramFormat
HotkeyPrefix
ImageCodecFlags
ImageFlags
ImageLockMode
ImageType
InterpolationMode
ItemDataPosition
LinearGradientMode
LineCap
LineJoin
MatrixOrder
MetafileFrameUnit
MetafileType

[ObjectType](#)
[PaletteFlags](#)
[PaletteType](#)
[PathPointType](#)
[PenAlignment](#)
[PenType](#)
[PixelOffsetMode](#)
[RotateFlipType](#)
[SmoothingMode](#)
[Status](#)
[StringAlignment](#)
[StringDigitSubstitute](#)
[StringFormatFlags](#)
[StringTrimming](#)
[TextRenderingHint](#)
[Unit](#)
[WarpMode](#)
[WrapMode](#)

Structures

[BlurParams](#)
[BrightnessContrastParams](#)
[ColorBalanceParams](#)
[ColorCurveParams](#)
[ColorLUTParams](#)
[ColorMap](#)
[ColorMatrix](#)
[ColorPalette](#)
[ENHMETAHEADER3](#)
[GdiplusAbort](#)
[GdiplusStartupInput](#)
[GdiplusStartupInputEx](#)
[GdiplusStartupOutput](#)

[HueSaturationLightnessParams](#)
[LevelsParams](#)
[PWMFRect16](#)
[RedEyeCorrectionParams](#)
[SharpenParams](#)
[TintParams](#)
[WmfPlaceableFileHeader](#)

GDI+ Flat API

[AdjustableArrowCap Functions](#)
[Bitmap Functions](#)
[Brush Functions](#)
[CachedBitmap Functions](#)
[CustomLineCap Functions](#)
[Font Functions](#)
[FontFamilyFunctions](#)
[Graphics Functions](#)
[GraphicsPath Functions](#)
[HatchBrush Functions](#)
[Image Functions](#)
[ImageAttributes Functions](#)
[LinearGradientBrush Functions](#)
[Matrix Functions](#)
[Memory Functions](#)
[Notification Functions](#)
[PathGradientBrush Functions](#)
[PathIterator Functions](#)
[Pen Functions](#)
[Region Functions](#)
[SolidBrush Functions](#)
[String Format Functions](#)
[Text Functions](#)
[Texture Brush Functions](#)

GDI+

11/2/2020 • 2 minutes to read • [Edit Online](#)

Purpose

Windows GDI+ is a class-based API for C/C++ programmers. It enables applications to use graphics and formatted text on both the video display and the printer. Applications based on the Microsoft Win32 API do not access graphics hardware directly. Instead, GDI+ interacts with device drivers on behalf of applications. GDI+ is also supported by Microsoft Win64.

Where applicable

GDI+ functions and classes are not supported for use within a Windows service. Attempting to use these functions and classes from a Windows service may produce unexpected problems, such as diminished service performance and run-time exceptions or errors.

NOTE

When you use the GDI+ API, you must never allow your application to download arbitrary fonts from untrusted sources. The operating system requires elevated privileges to assure that all installed fonts are trusted.

Developer audience

The GDI+ C++ class-based interface is designed for use by C/C++ programmers. Familiarity with the Windows graphical user interface and message-driven architecture is required.

Run-time requirements

GDI+ can be used in all Windows-based applications. GDI+ was introduced in Windows XP and Windows Server 2003. For information about which operating systems are required to use a particular class or method, see the More Information section of the documentation for the class or method.

In this section

TOPIC	DESCRIPTION
Overview	General information about GDI+.
Using	Tasks and examples using GDI+.
Reference	Documentation of GDI+ C++ class-based API.

Related topics

[Windows GDI](#)

[DirectX](#)

[Windows Image Acquisition](#)

OpenGL

Windows Multimedia

Security Considerations: GDI+

11/2/2020 • 5 minutes to read • [Edit Online](#)

This topic provides information about security considerations related to programming with Windows GDI+. This topic doesn't provide all you need to know about security issues—instead, use it as a starting point and reference for this technology area.

- [Verifying the Success of Constructors](#)
- [Allocating Buffers](#)
- [Error Checking](#)
- [Thread Synchronization](#)
- [Related topics](#)

Verifying the Success of Constructors

Many of the GDI+ classes provide a [Image::GetLastStatus](#) method that you can call to determine whether methods invoked on an object are successful. You can also call [Image::GetLastStatus](#) to determine whether a constructor is successful.

The following example shows how to construct an [Image](#) object and call the [Image::GetLastStatus](#) method to determine whether the constructor was successful. The values **Ok** and **InvalidParameter** are elements of the [Status](#) enumeration.

```
Image myImage(L"Climber.jpg");
Status st = myImage.GetLastStatus();

if(Ok == st)
    // The constructor was successful. Use myImage.
else if(InvalidParameter == st)
    // The constructor failed because of an invalid parameter.
else
    // Compare st to other elements of the Status
    // enumeration or do general error processing.
```

Allocating Buffers

Several GDI+ methods return numeric or character data in a buffer that is allocated by the caller. For each of those methods, there is a companion method that gives the size of the required buffer. For example, the [GraphicsPath::GetPathPoints](#) method returns an array of [Point](#) objects. Before you call [GraphicsPath::GetPathPoints](#), you must allocate a buffer large enough to hold that array. You can determine the size of the required buffer by calling the [GraphicsPath::GetPointCount](#) method of a [GraphicsPath](#) object.

The following example shows how to determine the number of points in a [GraphicsPath](#) object, allocate a buffer large enough to hold that many points, and then call [GraphicsPath::GetPathPoints](#) to fill the buffer. Before the code calls [GraphicsPath::GetPathPoints](#), it verifies that the buffer allocation was successful by making sure that the buffer pointer is not **NULL**.

```

GraphicsPath path;
path.AddEllipse(10, 10, 200, 100);

INT count = path.GetPointCount();           // get the size
Point* pointArray = new Point[count];      // allocate the buffer

if(pointArray) // Check for successful allocation.
{
    path.GetPathPoints(pointArray, count); // get the data
    ...
    delete[] pointArray;                // release the buffer
    pointArray = NULL;
}

```

The previous example uses the new operator to allocate a buffer. The new operator was convenient because the buffer was filled with a known number of [Point](#) objects. In some cases, GDI+ writes more into buffer than an array of GDI+ objects. Sometimes a buffer is filled with an array of GDI+ objects along with additional data that is pointed to by members of those objects. For example, the [Image::GetAllPropertyItems](#) method returns an array of [PropertyItem](#) objects, one for each property item (piece of metadata) stored in the image. But [Image::GetAllPropertyItems](#) returns more than just the array of [PropertyItem](#) objects; it appends the array with additional data.

Before you call [Image::GetAllPropertyItems](#), you must allocate a buffer large enough to hold the array of [PropertyItem](#) objects along with the additional data. You can call the [Image::GetPropertySize](#) method of an [Image](#) object to determine the total size of the required buffer.

The following example shows how to create an [Image](#) object and later call the [Image::GetAllPropertyItems](#) method of that [Image](#) object to retrieve all the property items (metadata) stored in the image. The code allocates a buffer based on a size value returned by the [Image::GetPropertySize](#) method. [Image::GetPropertySize](#) also returns a count value that gives the number of property items in the image. Notice that the code does not calculate the buffer size as `count*sizeof(PropertyItem)`. A buffer calculated that way would be too small.

```

UINT count = 0;
UINT size = 0;
Image myImage(L"FakePhoto.jpg");
myImage.GetPropertySize(&size, &count);

// GetAllPropertyItems returns an array of PropertyItem objects
// along with additional data. Allocate a buffer large enough to
// receive the array and the additional data.
PropertyItem* propBuffer =(PropertyItem*)malloc(size);

if(propBuffer)
{
    myImage.GetAllPropertyItems(size, count, propBuffer);
    ...
    free(propBuffer);
    propBuffer = NULL;
}

```

Error Checking

Most of the code examples in the GDI+ documentation do not show error checking. Complete error checking makes a code example much longer and can obscure the point being illustrated by the example. You should not paste examples from the documentation directly into production code; rather, you should enhance the examples by adding your own error checking.

The following example shows one way of implementing error checking with GDI+. Each time a GDI+ object is

constructed, the code checks to see whether the constructor was successful. That check is especially important for the [Image](#) constructor, which relies on reading a file. If all four of the GDI+ objects ([Graphics](#), [GraphicsPath](#), [Image](#), and [TextureBrush](#)) are constructed successfully, the code calls methods on those objects. Each method call is checked for success, and in the event of failure, the remaining method calls are skipped.

```
 Status GdipExample(HDC hdc)
{
    Status status = GenericError;
    INT count = 0;
    Point* points = NULL;

    Graphics graphics(hdc);
    status = graphics.GetLastStatus();
    if(Ok != status)
        return status;

    GraphicsPath path;
    status = path.GetLastStatus();
    if(Ok != status)
        return status;

    Image image(L"MyTexture.bmp");
    status = image.GetLastStatus();
    if(Ok != status)
        return status;

    TextureBrush brush(&image);
    status = brush.GetLastStatus();
    if(Ok != status)
        return status;

    status = path.AddEllipse(10, 10, 200, 100);

    if(Ok == status)
    {
        status = path.AddBezier(40, 130, 200, 130, 200, 200, 60, 200);
    }

    if(Ok == status)
    {
        count = path.GetPointCount();
        status = path.GetLastStatus();
    }

    if(Ok == status)
    {
        points = new Point[count];

        if(NULL == points)
            status = OutOfMemory;
    }

    if(Ok == status)
    {
        status = path.GetPathPoints(points, count);
    }

    if(Ok == status)
    {
        status = graphics.FillPath(&brush, &path);
    }

    if(Ok == status)
    {
        for(int j = 0; j < count; ++j)
        {
            status = graphics.FillEllipse(

```

```
    &brush, points[j].X - 5, points[j].Y - 5, 10, 10);
}

if(points)
{
    delete[] points;
    points = NULL;
}

return status;
}
```

Thread Synchronization

It is possible for more than one thread to have access to a single GDI+ object. However, GDI+ does not provide any automatic synchronization mechanism. So if two threads in your application have a pointer to the same GDI+ object, it is your responsibility to synchronize access to that object.

Some GDI+ methods return **ObjectBusy** if a thread attempts to call a method while another thread is executing a method on the same object. Do not try to synchronize access to an object based on the **ObjectBusy** return value. Instead, each time you access a member or call a method of the object, place the call inside a critical section, or use some other standard synchronization technique.

Related topics

[MSDN Security Developer Center](#)

[Security How-To Resources](#)

[TechNet Security Center](#)

About GDI+

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ is the portion of the Windows XP operating system or Windows Server 2003 operating system that provides two-dimensional vector graphics, imaging, and typography. GDI+ improves on Windows Graphics Device Interface (GDI) (the graphics device interface included with earlier versions of Windows) by adding new features and by optimizing existing features.

The following topics provide information about the GDI+ API with the C++ programming language.

- [Introduction to GDI+](#)
- [What's New In GDI+?](#)
- [Lines, Curves, and Shapes](#)
- [Images, Bitmaps, and Metafiles](#)
- [Coordinate Systems and Transformations](#)
- [Graphics Containers](#)

Introduction to GDI+

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ is a graphics device interface that allows programmers to write device-independent applications. The services of GDI+ are exposed through a set of C++ classes.

- [Overview of GDI+](#)
- [The Three Parts of GDI+](#)
- [The Structure of the Class-Based Interface](#)

Overview of GDI+

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ is the subsystem of the Windows XP operating system or Windows Server 2003 that is responsible for displaying information on screens and printers. GDI+ is an API that is exposed through a set of C++ classes.

As its name suggests, GDI+ is the successor to Windows Graphics Device Interface (GDI), the graphics device interface included with earlier versions of Windows. Windows XP or Windows Server 2003 supports GDI for compatibility with existing applications, but programmers of new applications should use GDI+ for all their graphics needs because GDI+ optimizes many of the capabilities of GDI and also provides additional features.

A graphics device interface, such as GDI+, allows application programmers to display information on a screen or printer without having to be concerned about the details of a particular display device. The application programmer makes calls to methods provided by GDI+ classes and those methods in turn make the appropriate calls to specific device drivers. GDI+ insulates the application from the graphics hardware, and it is this insulation that allows developers to create device-independent applications.

The Three Parts of GDI+

2/22/2020 • 2 minutes to read • [Edit Online](#)

The services of Windows GDI+ fall into the following three broad categories:

- [2-D vector graphics](#)
- [Imaging](#)
- [Typography](#)

2-D vector graphics

Vector graphics involves drawing primitives (such as lines, curves, and figures) that are specified by sets of points on a coordinate system. For example, a straight line can be specified by its two endpoints, and a rectangle can be specified by a point giving the location of its upper-left corner and a pair of numbers giving its width and height. A simple path can be specified by an array of points to be connected by straight lines. A Bézier spline is a sophisticated curve specified by four control points.

GDI+ provides classes that store information about the primitives themselves, classes that store information about how the primitives are to be drawn, and classes that actually do the drawing. For example, the **Rect** class stores the location and size of a rectangle; the **Pen** class stores information about line color, line width, and line style; and the **Graphics** class has methods for drawing lines, rectangles, paths, and other figures. There are also several **Brush** classes that store information about how closed figures and paths are to be filled with colors or patterns.

Imaging

Certain kinds of pictures are difficult or impossible to display with the techniques of vector graphics. For example, the pictures on toolbar buttons and the pictures that appear as icons would be difficult to specify as collections of lines and curves. A high-resolution digital photograph of a crowded baseball stadium would be even more difficult to create with vector techniques. Images of this type are stored as bitmaps, arrays of numbers that represent the colors of individual dots on the screen. Data structures that store information about bitmaps tend to be more complex than those required for vector graphics, so there are several classes in GDI+ devoted to this purpose. An example of such a class is **CachedBitmap**, which is used to store a bitmap in memory for fast access and display.

Typography

Typography is concerned with the display of text in a variety of fonts, sizes, and styles. GDI+ provides an impressive amount of support for this complex task. One of the new features in GDI+ is subpixel antialiasing, which gives text rendered on an LCD screen a smoother appearance.

The Structure of the Class-Based Interface

11/2/2020 • 2 minutes to read • [Edit Online](#)

The C++ interface to Windows GDI+ contains about 40 classes, 50 enumerations, and 6 structures. There are also a few functions that are not members of any class.

You must indicate that the namespace `Gdiplus` is being used before any GDI+ functions are called. The following statement indicates that the `Gdiplus` namespace is being used in the application.

```
using namespace Gdiplus;
```

The `Graphics` class is the core of the GDI+ interface; it is the class that actually draws lines, curves, figures, images, and text.

Many classes work together with the `Graphics` class. For example, the `Graphics::DrawLine` method receives a pointer to a `Pen` object, which holds attributes (color, width, dash style, and the like) of the line to be drawn. The `Graphics::FillRectangle` method can receive a pointer to a `LinearGradientBrush` object, which works with the `Graphics` object to fill a rectangle with a gradually changing color. `Font` and `StringFormat` objects influence the way a `Graphics` object draws text. A `Matrix` object stores and manipulates the world transformation of a `Graphics` object, which is used to rotate, scale, and flip images.

Certain classes serve primarily as structured data types. Some of those classes (for example, `Rect`, `Point`, and `Size`) are for general purposes. Others are for specialized purposes and are considered helper classes. For example, the `BitmapData` class is a helper for the `Bitmap` class, and the `PathData` class is a helper for the `GraphicsPath` class. GDI+ also defines a few structures that are used for organizing data. For example, the `ColorMap` structure holds a pair of `Color` objects that form one entry in a color conversion table.

GDI+ defines several enumerations, which are collections of related constants. For example, the `LineJoin` enumeration contains the elements `LineJoinBevel`, `LineJoinMiter`, and `LineJoinRound`, which specify styles that can be used to join two lines.

GDI+ provides a few functions that are not part of any class. Two of those functions are `GdiplusStartup` and `GdiplusShutdown`. You must call `GdiplusStartup` before you make any other GDI+ calls, and you must call `GdiplusShutdown` when you have finished using GDI+.

What's New In GDI+?

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ is different from Windows Graphics Device Interface (GDI) in a couple of ways. First, GDI+ expands on the features of GDI by providing new capabilities, such as gradient brushes and alpha blending. Second, the programming model has been revised to make graphics programming easier and more flexible.

- [New Features](#)
- [Changes in the Programming Model](#)

New Features

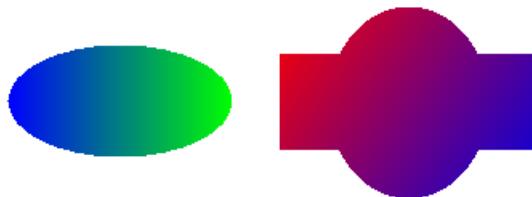
2/22/2020 • 3 minutes to read • [Edit Online](#)

The following sections describe several of the new features in Windows GDI+.

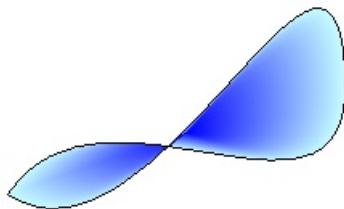
- [Gradient Brushes](#)
- [Cardinal Splines](#)
- [Independent Path Objects](#)
- [Transformations and the Matrix Object](#)
- [Scalable Regions](#)
- [Alpha Blending](#)
- [Support for Multiple Image Formats](#)

Gradient Brushes

GDI+ expands on Windows Graphics Device Interface (GDI) by providing linear gradient and path gradient brushes for filling shapes, paths, and regions. Gradient brushes can also be used to draw lines, curves, and paths. When you fill a shape with a linear gradient brush, the color gradually changes as you move across the shape. For example, suppose you create a horizontal gradient brush by specifying blue at the left edge of a shape and green at the right edge. When you fill that shape with the horizontal gradient brush, it will gradually change from blue to green as you move from its left edge to its right edge. Similarly, a shape filled with a vertical gradient brush will change color as you move from top to bottom. The following illustration shows an ellipse filled with a horizontal gradient brush and a region filled with a diagonal gradient brush.



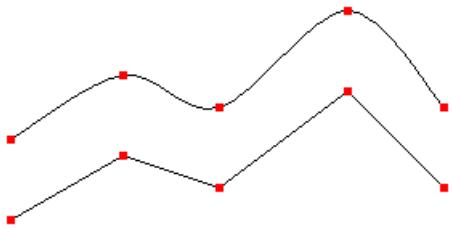
When you fill a shape with a path gradient brush, you have a variety of options for specifying how the colors change as you move from one portion of the shape to another. One option is to have a center color and a boundary color so that the pixels change gradually from one color to the other as you move from the middle of the shape towards the outer edges. The following illustration shows a path (created from a pair of Bézier splines) filled with a path gradient brush.



Cardinal Splines

GDI+ supports cardinal splines, which are not supported in GDI. A cardinal spline is a sequence of individual curves joined to form a larger curve. The spline is specified by an array of points and passes through each point in that array. A cardinal spline passes smoothly (no sharp corners) through each point in the array and thus is more refined than a path created by connecting straight lines. The following illustration shows two paths, one created by

connecting straight lines and one created as a cardinal spline.

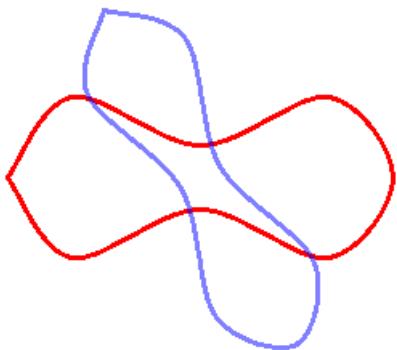


Independent Path Objects

In GDI, a path belongs to a device context, and the path is destroyed as it is drawn. With GDI+, drawing is performed by a [Graphics](#) object, and you can create and maintain several [GraphicsPath](#) objects that are separate from the [Graphics](#) object. A [GraphicsPath](#) object is not destroyed by the drawing action, so you can use the same [GraphicsPath](#) object to draw a path several times.

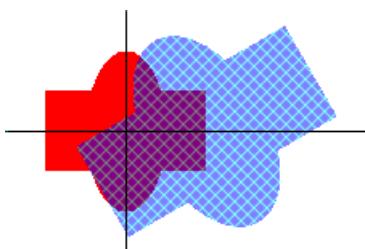
Transformations and the Matrix Object

GDI+ provides the [Matrix](#) object, a powerful tool that makes transformations (rotations, translations, and so on) easy and flexible. A matrix object works in conjunction with the objects that are transformed. For example, a [GraphicsPath](#) object has a [GraphicsPath::Transform](#) method that receives the address of a [Matrix](#) object as an argument. A single 3×3 matrix can store one transformation or a sequence of transformations. The following illustration shows a path before and after a sequence of two transformations (first scale, then rotate).



Scalable Regions

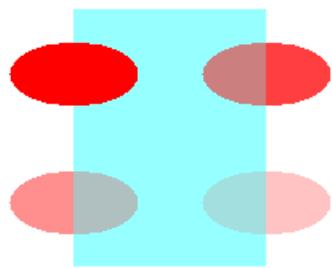
GDI+ expands greatly on GDI with its support for regions. In GDI, regions are stored in device coordinates, and the only transformation that can be applied to a region is a translation. GDI+ stores regions in world coordinates and allows a region to undergo any transformation (scaling, for example) that can be stored in a transformation matrix. The following illustration shows a region before and after a sequence of three transformations: scale, rotate, and translate.



Alpha Blending

Note that in the previous figure, you can see the untransformed region (filled with red) through the transformed

region (filled with a hatch brush). This is made possible by alpha blending, which is supported by GDI+. With alpha blending, you can specify the transparency of a fill color. A transparent color is blended with the background color — the more transparent you make a fill color, the more the background shows through. The following illustration shows four ellipses that are filled with the same color (red) at different transparency levels.



Support for Multiple Image Formats

GDI+ provides the [Image](#), [Bitmap](#), and [Metafile](#) classes, which allow you to load, save and manipulate images in a variety of formats. The following formats are supported:

- BMP
- Graphics Interchange Format (GIF)
- JPEG
- Exif
- PNG
- TIFF
- ICON
- WMF
- EMF

Changes in the Programming Model

11/2/2020 • 8 minutes to read • [Edit Online](#)

The following sections describe several ways that programming with Windows GDI+ is different from programming with Windows Graphics Device Interface (GDI).

- [Device Contexts, Handles, and Graphics Objects](#)
- [Two Ways to Draw a Line](#)
 - [Drawing a line with GDI](#)
 - [Drawing a line with GDI+ and the C++ class interface](#)
- [Pens, Brushes, Paths, Images, and Fonts as Parameters](#)
- [Method Overloading](#)
- [No More Current Position](#)
- [Separate Methods for Draw and Fill](#)
- [Constructing Regions](#)

Device Contexts, Handles, and Graphics Objects

If you have written programs using GDI (the graphics device interface included in previous versions of Windows), you are familiar with the idea of a device context (DC). A device context is a structure used by Windows to store information about the capabilities of a particular display device and attributes that specify how items will be drawn on that device. A device context for a video display is also associated with a particular window on the display. First you obtain a handle to a device context (HDC), and then you pass that handle as an argument to GDI functions that actually do the drawing. You also pass the handle as an argument to GDI functions that obtain or set the attributes of the device context.

When you use GDI+, you don't have to be as concerned with handles and device contexts as you do when you use GDI. You simply create a **Graphics** object and then invoke its methods in the familiar object-oriented style — `myGraphicsObject.DrawLine(parameters)`. The **Graphics** object is at the core of GDI+ just as the device context is at the core of GDI. The device context and the **Graphics** object play similar roles, but there are some fundamental differences between the handle-based programming model used with device contexts (GDI) and the object-oriented model used with **Graphics** objects (GDI+).

The **Graphics** object, like the device context, is associated with a particular window on the screen and contains attributes (for example, smoothing mode and text rendering hint) that specify how items are to be drawn. However, the **Graphics** object is not tied to a pen, brush, path, image, or font as a device context is. For example, in GDI, before you can use a device context to draw a line, you must call **SelectObject** to associate a pen object with the device context. This is referred to as selecting the pen into the device context. All lines drawn in the device context will use that pen until you select a different pen. With GDI+, you pass a **Pen** object as an argument to the **DrawLine** method of the **Graphics** class. You can use a different **Pen** object in each of a series of **DrawLine** calls without having to associate a given **Pen** object with a **Graphics** object.

Two Ways to Draw a Line

The following two examples each draw a red line of width 3 from location (20, 10) to location (200,100). The first example calls GDI, and the second calls GDI+ through the C++ class interface.

- [Drawing a line with GDI](#)
- [Drawing a line with GDI+ and the C++ class interface](#)

Drawing a line with GDI

To draw a line with GDI, you need two objects: a device context and a pen. You get a handle to a device context by calling [BeginPaint](#), and a handle to a pen by calling [CreatePen](#). Next, you call [SelectObject](#) to select the pen into the device context. You set the pen position to (20, 10) by calling [MoveToEx](#) and then draw a line from that pen position to (200, 100) by calling [LineTo](#). Note that [MoveToEx](#) and [LineTo](#) both receive `hdc` as an argument.

```
HDC           hdc;
PAINTSTRUCT  ps;
HPEN          hPen;
HPEN          hPenOld;
hdc = BeginPaint(hWnd, &ps);
hPen = CreatePen(PS_SOLID, 3, RGB(255, 0, 0));
hPenOld = (HPEN)SelectObject(hdc, hPen);
MoveToEx(hdc, 20, 10, NULL);
LineTo(hdc, 200, 100);
SelectObject(hdc, hPenOld);
DeleteObject(hPen);
EndPaint(hWnd, &ps);
```

Drawing a line with GDI+ and the C++ class interface

To draw a line with GDI+ and the C++ class interface, you need a [Graphics](#) object and a [Pen](#) object. Note that you don't ask Windows for handles to these objects. Instead, you use constructors to create an instance of the [Graphics](#) class (a [Graphics](#) object) and an instance of the [Pen](#) class (a [Pen](#) object). Drawing a line involves calling the [Graphics::DrawLine](#) method of the [Graphics](#) class. The first parameter of the [Graphics::DrawLine](#) method is a pointer to your [Pen](#) object. This is a simpler and more flexible scheme than selecting a pen into a device context as shown in the preceding GDI example.

```
HDC           hdc;
PAINTSTRUCT  ps;
Pen*          myPen;
Graphics*     myGraphics;
hdc = BeginPaint(hWnd, &ps);
myPen = new Pen(Color(255, 255, 0, 0), 3);
myGraphics = new Graphics(hdc);
myGraphics->DrawLine(myPen, 20, 10, 200, 100);
delete myGraphics;
delete myPen;
EndPaint(hWnd, &ps);
```

Pens, Brushes, Paths, Images, and Fonts as Parameters

The preceding examples show that [Pen](#) objects can be created and maintained separately from the [Graphics](#) object, which supplies the drawing methods. [Brush](#), [GraphicsPath](#), [Image](#), and [Font](#) objects can also be created and maintained separately from the [Graphics](#) object. Many of the drawing methods provided by the [Graphics](#) class receive a [Brush](#), [GraphicsPath](#), [Image](#), or [Font](#) object as an argument. For example, the address of a [Brush](#) object is passed as an argument to the [FillRectangle](#) method, and the address of a [GraphicsPath](#) object is passed as an argument to the [Graphics::DrawPath](#) method. Similarly, addresses of [Image](#) and [Font](#) objects are passed to the [DrawImage](#) and [DrawString](#) methods. This is in contrast to GDI where you select a brush, path, image, or font into the device context and then pass a handle to the device context as an argument to a drawing function.

Method Overloading

Many of the GDI+ methods are overloaded; that is, several methods share the same name but have different parameter lists. For example, the [DrawLine](#) method of the [Graphics](#) class comes in the following forms:

```

Status DrawLine(IN const Pen* pen,
                IN REAL x1,
                IN REAL y1,
                IN REAL x2,
                IN REAL y2);
Status DrawLine(IN const Pen* pen,
                IN const PointF& pt1,
                IN const PointF& pt2);
Status DrawLine(IN const Pen* pen,
                IN INT x1,
                IN INT y1,
                IN INT x2,
                IN INT y2);

Status DrawLine(IN const Pen* pen,
                IN const Point& pt1,
                IN const Point& pt2);

```

All four of the [DrawLine](#) variations above receive a pointer to a [Pen](#) object, the coordinates of the starting point, and the coordinates of the ending point. The first two variations receive the coordinates as floating point numbers, and the last two variations receive the coordinates as integers. The first and third variations receive the coordinates as a list of four separate numbers, while the second and fourth variations receive the coordinates as a pair of [Point](#) (or [PointF](#)) objects.

No More Current Position

Note that in the [DrawLine](#) methods shown previously both the starting point and the ending point of the line are received as arguments. This is a departure from the GDI scheme where you call [MoveToEx](#) to set the current pen position followed by [LineTo](#) in order to draw a line starting at (x_1, y_1) and ending at (x_2, y_2) . GDI+ as a whole has abandoned the notion of current position.

Separate Methods for Draw and Fill

GDI+ is more flexible than GDI when it comes to drawing the outlines and filling the interiors of shapes like rectangles. GDI has a [Rectangle](#) function that draws the outline and fills the interior of a rectangle all in one step. The outline is drawn with the currently selected pen, and the interior is filled with the currently selected brush.

```

hBrush = CreateHatchBrush(HS_CROSS, RGB(0, 0, 255));
hPen = CreatePen(PS_SOLID, 3, RGB(255, 0, 0));
SelectObject(hdc, hBrush);
SelectObject(hdc, hPen);
Rectangle(hdc, 100, 50, 200, 80);

```

GDI+ has separate methods for drawing the outline and filling the interior of a rectangle. The [DrawRectangle](#) method of the [Graphics](#) class has the address of a [Pen](#) object as one of its parameters, and the [FillRectangle](#) method has the address of a [Brush](#) object as one of its parameters.

```

HatchBrush* myHatchBrush = new HatchBrush(
    HatchStyleCross,
    Color(255, 0, 255, 0),
    Color(255, 0, 0, 255));
Pen* myPen = new Pen(Color(255, 255, 0, 0), 3);
myGraphics.FillRectangle(myHatchBrush, 100, 50, 100, 30);
myGraphics.DrawRectangle(myPen, 100, 50, 100, 30);

```

Note that the [FillRectangle](#) and [DrawRectangle](#) methods in GDI+ receive arguments that specify the rectangle's left edge, top, width, and height. This is in contrast to the GDI [Rectangle](#) function, which takes arguments that specify

the rectangle's left edge, right edge, top, and bottom. Also note that the constructor for the [Color](#) class in GDI+ has four parameters. The last three parameters are the usual red, green, and blue values; the first parameter is the alpha value, which specifies the extent to which the color being drawn is blended with the background color.

Constructing Regions

GDI provides several functions for creating regions: [CreateRectRgn](#), [CreateEllipticRgn](#), [CreateRoundRectRgn](#), [CreatePolygonRgn](#), and [CreatePolyPolygonRgn](#). You might expect the [Region](#) class in GDI+ to have analogous constructors that take rectangles, ellipses, rounded rectangles, and polygons as arguments, but that is not the case. The [Region](#) class in GDI+ provides a constructor that receives a [Rect](#) object reference and another constructor that receives the address of a [GraphicsPath](#) object. If you want to construct a region based on an ellipse, rounded rectangle, or polygon, you can easily do so by creating a [GraphicsPath](#) object (that contains an ellipse, for example) and then passing the address of that [GraphicsPath](#) object to a [Region](#) constructor.

GDI+ makes it easy to form complex regions by combining shapes and paths. The [Region](#) class has [Union](#) and [Intersect](#) methods that you can use to augment an existing region with a path or another region. One nice feature of the GDI+ scheme is that a [GraphicsPath](#) object is not destroyed when it is passed as an argument to a [Region](#) constructor. In GDI, you can convert a path to a region with the [PathToRegion](#) function, but the path is destroyed in the process. Also, a [GraphicsPath](#) object is not destroyed when its address is passed as an argument to a [Union](#) or [Intersect](#) method, so you can use a given path as a building block for several separate regions. This is shown in the following example. Assume that **onePath** is a pointer to a [GraphicsPath](#) object (simple or complex) that has already been initialized.

```
Region region1(rect1);
Region region2(rect2);
region1.Union(onePath);
region2.Intersect(onePath);
```

Lines, Curves, and Shapes

2/22/2020 • 2 minutes to read • [Edit Online](#)

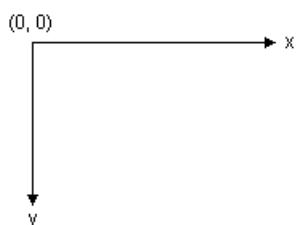
The vector graphics portion of GDI+ is used to draw lines, to draw curves, and to draw and fill shapes.

- [Overview of Vector Graphics](#)
- [Pens, Lines, and Rectangles](#)
- [Ellipses and Arcs](#)
- [Polygons](#)
- [Cardinal Splines](#)
- [Bézier Splines](#)
- [Paths](#)
- [Brushes and Filled Shapes](#)
- [Open and Closed Curves](#)
- [Regions](#)
- [Clipping](#)
- [Flattening Paths](#)
- [Antialiasing with Lines and Curves](#)

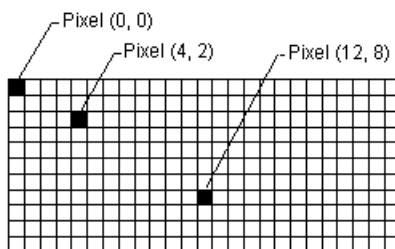
Overview of Vector Graphics

11/2/2020 • 2 minutes to read • [Edit Online](#)

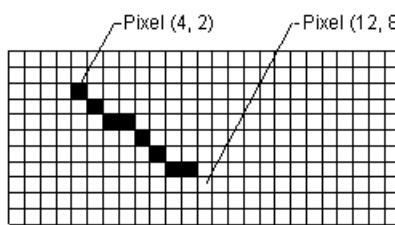
Windows GDI+ draws lines, rectangles, and other figures on a coordinate system. You can choose from a variety of coordinate systems, but the default coordinate system has the origin in the upper left corner with the x-axis pointing to the right and the y-axis pointing down. The unit of measure in the default coordinate system is the pixel.



A computer monitor creates its display on a rectangular array of dots called picture elements or pixels. The number of pixels appearing on the screen varies from one monitor to the next, and the number of pixels appearing on an individual monitor can usually be configured to some extent by the user.



When you use GDI+ to draw a line, rectangle, or curve, you provide certain key information about the item to be drawn. For example, you can specify a line by providing two points, and you can specify a rectangle by providing a point, a height, and a width. GDI+ works in conjunction with the display driver software to determine which pixels must be turned on to show the line, rectangle, or curve. The following illustration shows the pixels that are turned on to display a line from the point (4, 2) to the point (12, 8).



Over time, certain basic building blocks have proven to be the most useful for creating two-dimensional pictures. These building blocks, which are all supported by GDI+, are given in the following list:

- Lines
- Rectangles
- Ellipses
- Arcs
- Polygons
- Cardinal splines
- Bézier splines

The **Graphics** class in GDI+ provides the following methods for drawing the items in the previous list: **DrawLine**, **DrawRectangle**, **DrawEllipse**, **DrawPolygon**, **DrawArc**, **DrawCurve** (for cardinal splines), and **DrawBezier**. Each of these methods is overloaded; that is, each method comes in several variations with different parameter lists. For example, one variation of the **DrawLine** method receives the address of a **Pen** object and four integers, while another variation of the **DrawLine** method receives the address of a **Pen** object and two **Point** object references.

The methods for drawing lines, rectangles, and Bézier splines have plural companion methods that draw several items in a single call: **DrawLines**, **DrawRectangles**, and **DrawBeziers**. Also, the **DrawCurve** method has a companion method, **DrawClosedCurve**, that closes a curve by connecting the ending point of the curve to the starting point.

All the drawing methods of the **Graphics** class work in conjunction with a **Pen** object. Thus, in order to draw anything, you must create at least two objects: a **Graphics** object and a **Pen** object. The **Pen** object stores attributes of the item to be drawn, such as line width and color. The address of the **Pen** object is passed as one of the arguments to the drawing method. For example, one variation of the **DrawRectangle** method receives the address of a **Pen** object and four integers as shown in the following code, which draws a rectangle with a width of 100, a height of 50 and an upper-left corner of (20, 10).

```
myGraphics.DrawRectangle(&myPen, 20, 10, 100, 50);
```

Pens, Lines, and Rectangles

11/2/2020 • 2 minutes to read • [Edit Online](#)

To draw lines with Windows GDI+ you need to create a **Graphics** object and a **Pen** object. The **Graphics** object provides the methods that actually do the drawing, and the **Pen** object stores attributes of the line, such as color, width, and style. Drawing a line is simply a matter of calling the **DrawLine** method of the **Graphics** object. The address of the **Pen** object is passed as one of the arguments to the **DrawLine** method. The following example draws a line from the point (4, 2) to the point (12, 6).

```
myGraphics.DrawLine(&myPen, 4, 2, 12, 6);
```

DrawLine is an overloaded method of the **Graphics** class, so there are several ways you can supply it with arguments. For example, you can construct two **Point** objects and pass references to the **Point** objects as arguments to the **DrawLine** method.

```
Point myStartPoint(4, 2);
Point myEndPoint(12, 6);
myGraphics.DrawLine(&myPen, myStartPoint, myEndPoint);
```

You can specify certain attributes when you construct a **Pen** object. For example, one **Pen** constructor allows you to specify color and width. The following example draws a blue line of width 2 from (0, 0) to (60, 30).

```
Pen myPen(Color(255, 0, 0, 255), 2);
myGraphics.DrawLine(&myPen, 0, 0, 60, 30);
```

The **Pen** object also has attributes, such as dash style, that you can use to specify features of the line. For example, the following example draws a dashed line from (100, 50) to (300, 80).

```
myPen.SetDashStyle(DashStyleDash);
myGraphics.DrawLine(&myPen, 100, 50, 300, 80);
```

You can use various methods of the **Pen** object to set many more attributes of the line. The **Pen::SetStartCap** and **Pen::SetEndCap** methods specify the appearance of the ends of the line; the ends can be flat, square, rounded, triangular, or a custom shape. The **Pen::SetLineJoin** method lets you specify whether connected lines are mitered (joined with sharp corners), beveled, rounded, or clipped. The following illustration shows lines with various cap and join styles.



Drawing rectangles with GDI+ is similar to drawing lines. To draw a rectangle, you need a **Graphics** object and a **Pen** object. The **Graphics** object provides a **DrawRectangle** method, and the **Pen** object stores attributes, such as line width and color. The address of the **Pen** object is passed as one of the arguments to the **DrawRectangle** method. The following example draws a rectangle with its upper-left corner at (100, 50), a width of 80, and a height of 40.

```
myGraphics.DrawRectangle(&myPen, 100, 50, 80, 40);
```

[DrawRectangle](#) is an overloaded method of the [Graphics](#) class, so there are several ways you can supply it with arguments. For example, you can construct a [Rect](#) object and pass a reference to the [Rect](#) object as an argument to the DrawRectangle method.

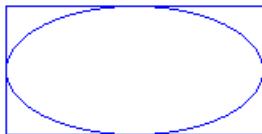
```
Rect myRect(100, 50, 80, 40);
myGraphics.DrawRectangle(&myPen, myRect);
```

A [Rect](#) object has methods for manipulating and gathering information about the rectangle. For example, the [Inflate](#) and [Offset](#) methods change the size and position of the rectangle. The [Rect::IntersectsWith](#) method tells you whether the rectangle intersects another given rectangle, and the [Contains](#) method tells you whether a given point is inside the rectangle.

Ellipses and Arcs

5/22/2020 • 2 minutes to read • [Edit Online](#)

An ellipse is specified by its bounding rectangle. The following illustration shows an ellipse along with its bounding rectangle.



To draw an ellipse, you need a **Graphics** object and a **Pen** object. The **Graphics** object provides the **DrawEllipse** method, and the **Pen** object stores attributes of the ellipse, such as line width and color. The address of the **Pen** object is passed as one of the arguments to the **DrawEllipse** method. The remaining arguments passed to the **DrawEllipse** method specify the bounding rectangle for the ellipse. The following example draws an ellipse; the bounding rectangle has a width of 160, a height of 80, and an upper-left corner of (100, 50).

```
myGraphics.DrawEllipse(&myPen, 100, 50, 160, 80);
```

DrawEllipse is an overloaded method of the **Graphics** class, so there are several ways you can supply it with arguments. For example, you can construct a **Rect** object and pass a reference to the **Rect** object as an argument to the **DrawEllipse** method.

```
Rect myRect(100, 50, 160, 80);
myGraphics.DrawEllipse(&myPen, myRect);
```

An arc is a portion of an ellipse. To draw an arc, you call the **DrawArc** method of the **Graphics** class. The parameters of the **DrawArc** method are the same as the parameters of the **DrawEllipse** method, except that **DrawArc** requires a starting angle and sweep angle. The following example draws an arc with a starting angle of 30 degrees and a sweep angle of 180 degrees.

```
myGraphics.DrawArc(&myPen, 100, 50, 160, 80, 30, 180);
```

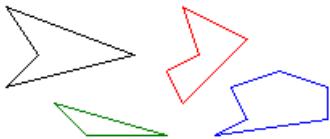
The following illustration shows the arc, the ellipse, and the bounding rectangle.



Polygons

11/2/2020 • 2 minutes to read • [Edit Online](#)

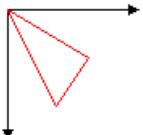
A polygon is a closed figure with three or more straight sides. For example, a triangle is a polygon with three sides, a rectangle is a polygon with four sides, and a pentagon is a polygon with five sides. The following illustration shows several polygons.



To draw a polygon, you need a **Graphics** object, a **Pen** object, and an array of **Point** (or **PointF**) objects. The **Graphics** object provides the **DrawPolygon** method. The **Pen** object stores attributes of the polygon, such as line width and color, and the array of **Point** objects stores the points to be connected by straight lines. The addresses of the **Pen** object and the array of **Point** objects are passed as arguments to the **DrawPolygon** method. The following example draws a three-sided polygon. Note that there are only three points in **myPointArray**: (0, 0), (50, 30), and (30, 60). The **DrawPolygon** method automatically closes the polygon by drawing a line from (30, 60) back to the starting point (0, 0);

```
Point myPointArray[] =  
    {Point(0, 0), Point(50, 30), Point(30, 60)};  
myGraphics.DrawPolygon(&myPen, myPointArray, 3);
```

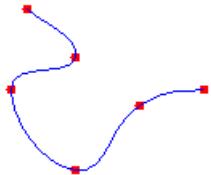
The following illustration shows the polygon.



Cardinal Splines

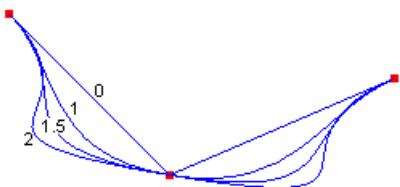
11/2/2020 • 2 minutes to read • [Edit Online](#)

A cardinal spline is a sequence of individual curves joined to form a larger curve. The spline is specified by an array of points and a tension parameter. A cardinal spline passes smoothly through each point in the array; there are no sharp corners and no abrupt changes in the tightness of the curve. The following illustration shows a set of points and a cardinal spline that passes through each point in the set.



A physical spline is a thin piece of wood or other flexible material. Before the advent of mathematical splines, designers used physical splines to draw curves. A designer would place the spline on a piece of paper and anchor it to a given set of points. The designer could then create a curve by drawing along the spline with a pencil. A given set of points could yield a variety of curves, depending on the properties of the physical spline. For example, a spline with a high resistance to bending would produce a different curve than an extremely flexible spline.

The formulas for mathematical splines are based on the properties of flexible rods, so the curves produced by mathematical splines are similar to the curves that were once produced by physical splines. Just as physical splines of different tension will produce different curves through a given set of points, mathematical splines with different values for the tension parameter will produce different curves through a given set of points. The following illustration shows four cardinal splines passing through the same set of points. The tension is shown for each spline. Note that a tension of 0 corresponds to infinite physical tension, forcing the curve to take the shortest way (straight lines) between points. A tension of 1 corresponds to no physical tension, allowing the spline to take the path of least total bend. With tension values greater than 1, the curve behaves like a compressed spring, pushed to take a longer path.



Note that the four splines in the preceding figure share the same tangent line at the starting point. The tangent is the line drawn from the starting point to the next point along the curve. Likewise, the shared tangent at the ending point is the line drawn from the ending point to the previous point on the curve.

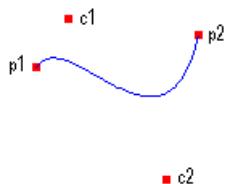
To draw a cardinal spline, you need a **Graphics** object, a **Pen** object, and an array of **Point** objects. The **Graphics** object provides the **DrawCurve** method, which draws the spline, and the **Pen** object stores attributes of the spline, such as line width and color. The array of **Point** objects stores the points that the curve will pass through. The following example draws a cardinal spline that passes through the points in *myPointArray*. The third parameter is the tension.

```
myGraphics.DrawCurve(&myPen, myPointArray, 3, 1.5f);
```


Bezier Splines

11/2/2020 • 2 minutes to read • [Edit Online](#)

A Bézier spline is a curve specified by four points: two end points (p_1 and p_2) and two control points (c_1 and c_2). The curve begins at p_1 and ends at p_2 . The curve doesn't pass through the control points, but the control points act as magnets, pulling the curve in certain directions and influencing the way the curve bends. The following illustration shows a Bézier curve along with its endpoints and control points.

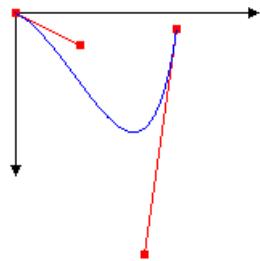


Note that the curve starts at p_1 and moves toward the control point c_1 . The tangent line to the curve at p_1 is the line drawn from p_1 to c_1 . Also note that the tangent line at the endpoint p_2 is the line drawn from c_2 to p_2 .

To draw a Bézier spline, you need a **Graphics** object and a **Pen** object. The **Graphics** object provides the **DrawBezier** method, and the **Pen** object stores attributes of the curve, such as line width and color. The address of the **Pen** object is passed as one of the arguments to the **DrawBezier** method. The remaining arguments passed to the **DrawBezier** method are the endpoints and the control points. The following example draws a Bézier spline with starting point (0, 0), control points (40, 20) and (80, 150), and ending point (100, 10).

```
myGraphics.DrawBezier(&myPen, 0, 0, 40, 20, 80, 150, 100, 10);
```

The following illustration shows the curve, the control points, and two tangent lines.



Bézier splines were originally developed by Pierre Bézier for design in the automotive industry. They have since proven to be very useful in many types of computer-aided design and are also used to define the outlines of fonts. Bézier splines can yield a wide variety of shapes, some of which are shown in the following illustration.



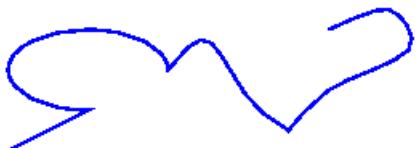
Paths (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

Paths are formed by combining lines, rectangles, and simple curves. Recall from the [Overview of Vector Graphics](#) that the following basic building blocks have proven to be the most useful for drawing pictures.

- Lines
- Rectangles
- Ellipses
- Arcs
- Polygons
- Cardinal splines
- Bézier splines

In Windows GDI+, the **GraphicsPath** object allows you to collect a sequence of these building blocks into a single unit. The entire sequence of lines, rectangles, polygons, and curves can then be drawn with one call to the **Graphics::DrawPath** method of the **Graphics** class. The following illustration shows a path created by combining a line, an arc, a Bézier spline, and a cardinal spline.



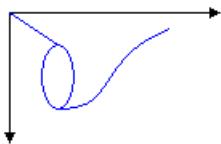
The **GraphicsPath** class provides the following methods for creating a sequence of items to be drawn: [AddLine](#), [AddRectangle](#), [AddEllipse](#), [AddArc](#), [AddPolygon](#), [AddCurve](#) (for cardinal splines), and [AddBezier](#). Each of these methods is overloaded; that is, each method comes in several variations with different parameter lists. For example, one variation of the AddLine method receives four integers, and another variation of the AddLine method receives two **Point** objects.

The methods for adding lines, rectangles, and Bézier splines to a path have plural companion methods that add several items to the path in a single call: [AddLines](#), [AddRectangles](#), and [AddBeziers](#). Also, the [AddCurve](#) method has a companion method, [AddClosedCurve](#), that adds a closed curve to the path.

To draw a path, you need a **Graphics** object, a **Pen** object, and a **GraphicsPath** object. The **Graphics** object provides the **Graphics::DrawPath** method, and the **Pen** object stores attributes of the path, such as line width and color. The **GraphicsPath** object stores the sequence of lines, rectangles, and curves that make up the path. The addresses of the **Pen** object and the **GraphicsPath** object are passed as arguments to the **Graphics::DrawPath** method. The following example draws a path that consists of a line, an ellipse, and a Bézier spline.

```
myGraphicsPath.AddLine(0, 0, 30, 20);
myGraphicsPath.AddEllipse(20, 20, 20, 40);
myGraphicsPath.AddBezier(30, 60, 70, 60, 50, 30, 100, 10);
myGraphics.DrawPath(&myPen, &myGraphicsPath);
```

The following illustration shows the path.



In addition to adding lines, rectangles, and curves to a path, you can add paths to a path. This allows you to combine existing paths to form large, complex paths. The following code adds `graphicsPath1` and `graphicsPath2` to `myGraphicsPath`. The second parameter of the `GraphicsPath::AddPath` method specifies whether the added path is connected to the existing path.

```
myGraphicsPath.AddPath(&graphicsPath1, FALSE);
myGraphicsPath.AddPath(&graphicsPath2, TRUE);
```

There are two other items you can add to a path: strings and pies. A pie is a portion of the interior of an ellipse. The following example creates a path from an arc, a cardinal spline, a string, and a pie.

```
myGraphicsPath.AddArc(0, 0, 30, 20, -90, 180);
myGraphicsPath.AddCurve(myPointArray, 3);
myGraphicsPath.AddString(L"a string in a path", 18, &myFontFamily,
    0, 24, myPointF, &myStringFormat);
myGraphicsPath.AddPie(230, 10, 40, 40, 40, 110);
myGraphics.DrawPath(&myPen, &myGraphicsPath);
```

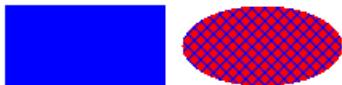
The following illustration shows the path. Note that a path does not have to be connected; the arc, cardinal spline, string, and pie are separated.



Brushes and Filled Shapes

11/2/2020 • 2 minutes to read • [Edit Online](#)

A closed figure such as a rectangle or an ellipse consists of an outline and an interior. The outline is drawn with a **Pen** object and the interior is filled with a **Brush** object. Windows GDI+ provides several brush classes for filling the interiors of closed figures: **SolidBrush**, **HatchBrush**, **TextureBrush**, **LinearGradientBrush**, and **PathGradientBrush**. All these classes inherit from the **Brush** class. The following illustration shows a rectangle filled with a solid brush and an ellipse filled with a hatch brush.



- [Solid Brushes](#)
- [Hatch Brushes](#)
- [Texture Brushes](#)
- [Gradient Brushes](#)

Solid Brushes

To fill a closed shape, you need a **Graphics** object and a **Brush** object. The **Graphics** object provides methods, such as **FillRectangle** and **FillEllipse**, and the **Brush** object stores attributes of the fill, such as color and pattern. The address of the **Brush** object is passed as one of the arguments to the fill method. The following example fills an ellipse with a solid red color.

```
SolidBrush mySolidBrush(Color(255, 255, 0, 0));
myGraphics.FillEllipse(&mySolidBrush, 0, 0, 60, 40);
```

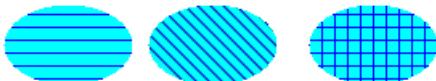
Note that in the preceding example, the brush is of type **SolidBrush**, which inherits from **Brush**.

Hatch Brushes

When you fill a shape with a hatch brush, you specify a foreground color, a background color, and a hatch style. The foreground color is the color of the hatching.

```
HatchBrush myHatchBrush(
    HatchStyleVertical,
    Color(255, 0, 0, 255),
    Color(255, 0, 255, 0));
```

GDI+ provides more than 50 hatch styles, specified in **HatchStyle**. The three styles shown in the following illustration are Horizontal, ForwardDiagonal, and Cross.



Texture Brushes

With a texture brush, you can fill a shape with a pattern stored in a bitmap. For example, suppose the following picture is stored in a disk file named MyTexture.bmp.



The following example fills an ellipse by repeating the picture stored in MyTexture.bmp.

```
Image myImage(L"MyTexture.bmp");
TextureBrush myTextureBrush(&myImage);
myGraphics.FillEllipse(&myTextureBrush, 0, 0, 100, 50);
```

The following illustration shows the filled ellipse.



Gradient Brushes

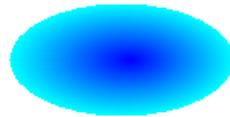
You can use a gradient brush to fill a shape with a color that changes gradually from one part of the shape to another. For example, a horizontal gradient brush will change color as you move from the left side of a figure to the right side. The following example fills an ellipse with a horizontal gradient brush that changes from blue to green as you move from the left side of the ellipse to the right side.

```
LinearGradientBrush myLinearGradientBrush(
    myRect,
    Color(255, 0, 0, 255),
    Color(255, 0, 255, 0),
    LinearGradientModeHorizontal);
myGraphics.FillEllipse(&myLinearGradientBrush, myRect);
```

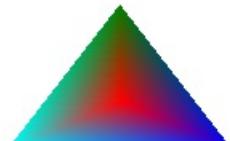
The following illustration shows the filled ellipse.



A path gradient brush can be configured to change color as you move from the center of a figure toward the boundary.



Path gradient brushes are quite flexible. The gradient brush used to fill the triangle in the following illustration changes gradually from red at the center to each of three different colors at the vertices.



Open and Closed Curves

11/2/2020 • 2 minutes to read • [Edit Online](#)

The following illustration shows two curves: one open and one closed.



Closed curves have an interior and therefore can be filled with a brush. The **Graphics** class in Windows GDI+ provides the following methods for filling closed figures and curves: **FillRectangle**, **FillEllipse**, **FillPie**, **FillPolygon**, **FillClosedCurve**, **Graphics::FillPath**, and **Graphics::FillRegion**. Whenever you call one of these methods, you must pass the address of one of the specific brush types (**SolidBrush**, **HatchBrush**, **TextureBrush**, **LinearGradientBrush**, or **PathGradientBrush**) as an argument.

The **FillPie** method is a companion to the **DrawArc** method. Just as the **DrawArc** method draws a portion of the outline of an ellipse, the **FillPie** method fills a portion of the interior of an ellipse. The following example draws an arc and fills the corresponding portion of the interior of the ellipse.

```
myGraphics.FillPie(&mySolidBrush, 0, 0, 140, 70, 0, 120);
myGraphics.DrawArc(&myPen, 0, 0, 140, 70, 0, 120);
```

The following illustration shows the arc and the filled pie.



The **FillClosedCurve** method is a companion to the **DrawClosedCurve** method. Both methods automatically close the curve by connecting the ending point to the starting point. The following example draws a curve that passes through (0, 0), (60, 20), and (40, 50). Then, the curve is automatically closed by connecting (40, 50) to the starting point (0, 0), and the interior is filled with a solid color.

```
Point myPointArray[] =
{Point(10, 10), Point(60, 20), Point(40, 50)};
myGraphics.DrawClosedCurve(&myPen, myPointArray, 3);
myGraphics.FillClosedCurve(&mySolidBrush, myPointArray, 3, FillModeAlternate)
```

A path can consist of several figures (subpaths). The **Graphics::FillPath** method fills the interior of each figure. If a figure is not closed, the **Graphics::FillPath** method fills the area that would be enclosed if the figure were closed. The following example draws and fills a path that consists of an arc, a cardinal spline, a string, and a pie.

```
myGraphics.FillPath(&mySolidBrush, &myGraphicsPath);
myGraphics.DrawPath(&myPen, &myGraphicsPath);
```

The following illustration shows the path before and after it is filled with a solid brush. Note that the text in the string is outlined, but not filled, by the **Graphics::DrawPath** method. It is the **Graphics::FillPath** method that paints the interiors of the characters in the string.

 a string in a path 

 a **string** in a path 

Regions (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

A region is a portion of the display surface. Regions can be simple (a single rectangle) or complex (a combination of polygons and closed curves). The following illustration shows two regions: one constructed from a rectangle, and the other constructed from a path.



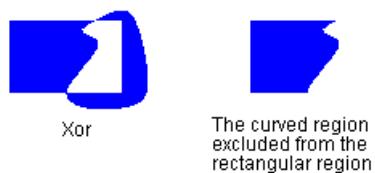
Regions are often used for clipping and hit testing. Clipping involves restricting drawing to a certain region of the screen, usually the portion of the screen that needs to be updated. Hit testing involves checking to see whether the cursor is in a certain region of the screen when a mouse button is pressed.

You can construct a region from a rectangle or from a path. You can also create complex regions by combining existing regions. The [Region](#) class provides the following methods for combining regions: [Intersect](#), [Union](#), [Xor](#), [Exclude](#), and [Region::Complement](#).

The intersection of two regions is the set of all points belonging to both regions. The union is the set of all points belonging to one or the other or both regions. The complement of a region is the set of all points that are not in the region. The following illustration shows the intersection and union of the two regions in the previous figure.



The [Xor](#) method, applied to a pair of regions, produces a region that contains all points that belong to one region or the other, but not both. The [Exclude](#) method, applied to a pair of regions, produces a region that contains all points in the first region that are not in the second region. The following illustration shows the regions that result from applying the Xor and Exclude methods to the two regions shown at the beginning of this topic.



To fill a region, you need a [Graphics](#) object, a [Brush](#) object, and a [Region](#) object. The [Graphics](#) object provides the [Graphics::FillRegion](#) method, and the [Brush](#) object stores attributes of the fill, such as color or pattern. The following example fills a region with a solid color.

```
myGraphics.FillRegion(&mySolidBrush, &myRegion);
```

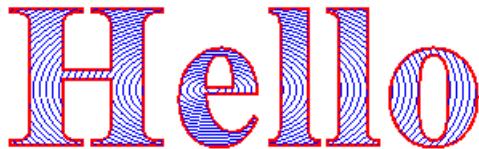
Clipping (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

Clipping involves restricting drawing to a certain region. The following illustration shows the string "Hello" clipped to a heart-shaped region.



Regions can be constructed from paths, and paths can contain the outlines of strings, so you can use outlined text for clipping. The following illustration shows a set of concentric ellipses clipped to the interior of a string of text.



To draw with clipping, create a [Graphics](#) object, call its [SetClip](#) method, and then call the drawing methods of that same [Graphics](#) object. The following example draws a line that is clipped to a rectangular region.

```
Region myRegion(Rect(20, 30, 100, 50));
myGraphics.DrawRectangle(&myPen, 20, 30, 100, 50);
myGraphics.SetClip(&myRegion, CombineModeReplace);
myGraphics.DrawLine(&myPen, 0, 0, 200, 200);
```

The following illustration shows the rectangular region along with the clipped line.



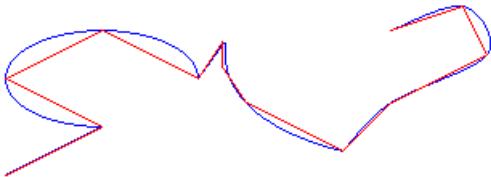
Flattening Paths

2/22/2020 • 2 minutes to read • [Edit Online](#)

A **GraphicsPath** object stores a sequence of lines and Bézier splines. You can add several types of curves (ellipses, arcs, cardinal splines) to a path, but each curve is converted to a Bézier spline before it is stored in the path.

Flattening a path consists of converting each Bézier spline in the path to a sequence of straight lines.

To flatten a path, call the **GraphicsPath::Flatten** method of a **GraphicsPath** object. The **GraphicsPath::Flatten** method receives a flatness argument that specifies the maximum distance between the flattened path and the original path. The following illustration shows a path before and after flattening.



Antialiasing with Lines and Curves

2/22/2020 • 2 minutes to read • [Edit Online](#)

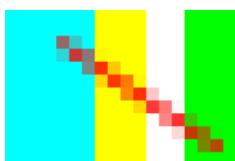
When you use Windows GDI+ to draw a line, you provide the starting point and ending point of the line, but you don't have to provide any information about the individual pixels on the line. GDI+ works in conjunction with the display driver software to determine which pixels will be turned on to show the line on a particular display device.

Consider a straight red line that goes from the point (4, 2) to the point (16, 10). Assume the coordinate system has its origin in the upper-left corner and that the unit of measure is the pixel. Also assume that the x-axis points to the right and the y-axis points down. The following illustration shows an enlarged view of the red line drawn on a multicolored background.

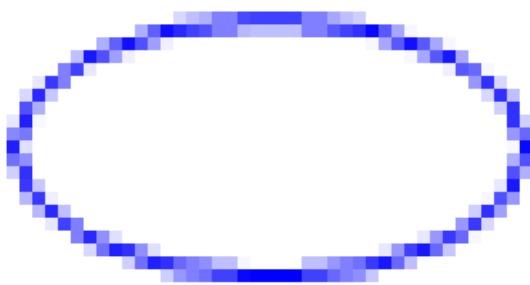


Note that the red pixels used to render the line are opaque. There are no partially transparent pixels involved in displaying the line. This type of line rendering gives the line a jagged appearance, and the line looks a bit like a staircase. This technique of representing a line with a staircase is called aliasing; the staircase is an alias for the theoretical line.

A more sophisticated technique for rendering a line involves using partially transparent pixels along with pure red pixels. Pixels are set to pure red or to some blend of red and the background color depending on how close they are to the line. This type of rendering is called antialiasing and results in a line that the human eye perceives as more smooth. The following illustration shows how certain pixels are blended with the background to produce an antialiased line.



Antialiasing (smoothing) can also be applied to curves. The following illustration shows an enlarged view of a smoothed ellipse.



The following illustration shows the same ellipse in its actual size, once without antialiasing and once with antialiasing.



Without antialiasing With antialiasing

To draw lines and curves that use antialiasing, create a **Graphics** object and pass *SmoothingModeAntiAlias* to its

Graphics::SetSmoothingMode method. Then call one of the drawing methods of that same **Graphics** object.

```
myGraphics.SetSmoothingMode(SmoothingModeAntiAlias);  
myGraphics.DrawLine(&myPen, 0, 0, 12, 8);
```

SmoothingModeAntiAlias is an element of the **SmoothingMode** enumeration.

Images, Bitmaps, and Metafiles

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides the **Image** class for working with raster images (bitmaps) and vector images (metafiles).

The **Bitmap** class and the **Metafile** class both inherit from the **Image** class. The **Bitmap** class expands on the capabilities of the **Image** class by providing additional methods for loading, saving, and manipulating raster images. The **Metafile** class expands on the capabilities of the **Image** class by providing additional methods for recording and examining vector images.

- [Types of Bitmaps](#)
- [Metafiles](#)
- [Drawing, Positioning, and Cloning Images](#)
- [Cropping and Scaling Images](#)

Types of Bitmaps

2/22/2020 • 6 minutes to read • [Edit Online](#)

A bitmap is an array of bits that specifies the color of each pixel in a rectangular array of pixels. The number of bits devoted to an individual pixel determines the number of colors that can be assigned to that pixel. For example, if each pixel is represented by 4 bits, then a given pixel can be assigned one of 16 different colors ($2^4 = 16$). The following table shows a few examples of the number of colors that can be assigned to a pixel represented by a given number of bits.

BITS PER PIXEL	NUMBER OF COLORS THAT CAN BE ASSIGNED TO A PIXEL
1	$2^1 = 2$
2	$2^2 = 4$
4	$2^4 = 16$
8	$2^8 = 256$
16	$2^{16} = 65,536$
24	$2^{24} = 16,777,216$

Disk files that store bitmaps usually contain one or more information blocks that store information such as number of bits per pixel, number of pixels in each row, and number of rows in the array. Such a file might also contain a color table (sometimes called a color palette). A color table maps numbers in the bitmap to specific colors. The following illustration shows an enlarged image along with its bitmap and color table. Each pixel is represented by a 4-bit number, so there are $2^4 = 16$ colors in the color table. Each color in the table is represented by a 24-bit number: 8 bits for red, 8 bits for green, and 8 bits for blue. The numbers are shown in hexadecimal (base 16) form: A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.

3 3 3 3 3 3 3 3 3 0 1 4 1 4 1 4 0 0 4 1 4 1 4 1 0 0 5 5 5 5 5 5 0 0 5 5 5 5 5 5 0 0 1 4 1 4 1 4 0 0 4 1 4 1 4 1 0 2 2 2 2 2 2 2 2 2		<table><tbody><tr><td>0 000000</td><td>█</td></tr><tr><td>1 FF0000</td><td>█</td></tr><tr><td>2 00FF00</td><td>█</td></tr><tr><td>3 0000FF</td><td>█</td></tr><tr><td>4 FFFFFF</td><td>█</td></tr><tr><td>5 FFFF00</td><td>█</td></tr><tr><td>6 FF00FF</td><td>█</td></tr><tr><td>7 00FFFF</td><td>█</td></tr><tr><td>8 FF0080</td><td>█</td></tr><tr><td>9 FF8040</td><td>█</td></tr><tr><td>A 804000</td><td>█</td></tr><tr><td>B 008080</td><td>█</td></tr><tr><td>C 800000</td><td>█</td></tr><tr><td>D 800080</td><td>█</td></tr><tr><td>E 8080FF</td><td>█</td></tr></tbody></table>	0 000000	█	1 FF0000	█	2 00FF00	█	3 0000FF	█	4 FFFFFF	█	5 FFFF00	█	6 FF00FF	█	7 00FFFF	█	8 FF0080	█	9 FF8040	█	A 804000	█	B 008080	█	C 800000	█	D 800080	█	E 8080FF	█
0 000000	█																															
1 FF0000	█																															
2 00FF00	█																															
3 0000FF	█																															
4 FFFFFF	█																															
5 FFFF00	█																															
6 FF00FF	█																															
7 00FFFF	█																															
8 FF0080	█																															
9 FF8040	█																															
A 804000	█																															
B 008080	█																															
C 800000	█																															
D 800080	█																															
E 8080FF	█																															

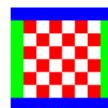
Look at the pixel in row 3, column 5 of the image. The corresponding number in the bitmap is 1. The color table tells us that 1 represents the color red, so the pixel is red. All the entries in the top row of the bitmap are 3. The color table tells us that 3 represents blue, so all the pixels in the top row of the image are blue.

NOTE

Some bitmaps are stored in bottom-up format; the numbers in the first row of the bitmap correspond to the pixels in the bottom row of the image.

A bitmap that stores indexes into a color table is called a *palette-indexed* bitmap. Some bitmaps have no need for a color table. For example, if a bitmap uses 24 bits per pixel, that bitmap can store the colors themselves rather than indexes into a color table. The following illustration shows a bitmap that stores colors directly (24 bits per pixel) rather than using a color table. The illustration also shows an enlarged view of the corresponding image. In the bitmap, FFFFFF represents white, FF0000 represents red, 00FF00 represents green, and 0000FF represents blue.

```
0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF  
00FF00 FF0000 FFFFFF FF0000 FFFFFF FF0000 FFFF00  
00FF00 FFFFFF FF0000 FFFFFF FF0000 FFFF00 00FF00  
00FF00 FF0000 FFFFFF FF0000 FFFFFF FF0000 FFFF00  
00FF00 FFFFFF FF0000 FFFFFF FF0000 FFFF00 00FF00  
00FF00 FF0000 FFFFFF FF0000 FFFF00 FF0000 00FF00  
00FF00 FF0000 FFFFFF FF0000 FFFF00 FF0000 FFFF00  
00FF00 FFFFFF FF0000 FFFF00 FF0000 FFFF00 00FF00  
0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
```



Graphics File Formats

There are many standard formats for saving bitmaps in files. Windows GDI+ supports the graphics file formats described in the following paragraphs.

Bitmap (BMP)

BMP is a standard format used by Windows to store device-independent and application-independent images. The number of bits per pixel (1, 4, 8, 15, 24, 32, or 64) for a given BMP file is specified in a file header. BMP files with 24 bits per pixel are common.

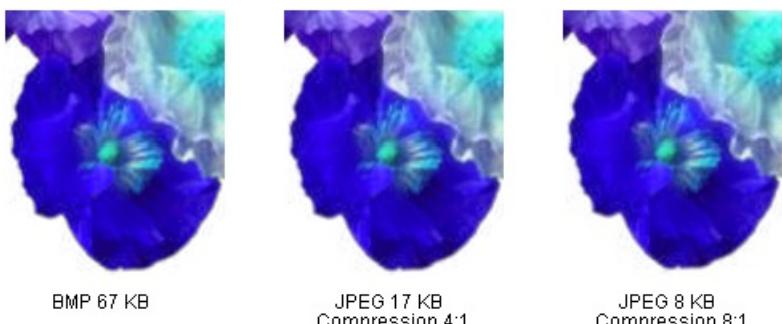
Graphics Interchange Format (GIF)

GIF is a common format for images that appear on Web pages. GIFs work well for line drawings, pictures with blocks of solid color, and pictures with sharp boundaries between colors. GIFs are compressed, but no information is lost in the compression process; a decompressed image is exactly the same as the original. One color in a GIF can be designated as transparent, so that the image will have the background color of any Web page that displays it. A sequence of GIF images can be stored in a single file to form an animated GIF. GIFs store at most 8 bits per pixel, so they are limited to 256 colors.

Joint Photographic Experts Group (JPEG)

JPEG is a compression scheme that works well for natural scenes, such as scanned photographs. Some information is lost in the compression process, but often the loss is imperceptible to the human eye. Color JPEG images store 24 bits per pixel, so they are capable of displaying more than 16 million colors. There is also a grayscale JPEG format that stores 8 bits per pixel. JPEGs do not support transparency or animation.

The level of compression in JPEG images is configurable, but higher compression levels (smaller files) result in more loss of information. A 20:1 compression ratio often produces an image that the human eye finds difficult to distinguish from the original. The following illustration shows a BMP image and two JPEG images that were compressed from that BMP image. The first JPEG has a compression ratio of 4:1 and the second JPEG has a compression ratio of about 8:1.



JPEG compression does not work well for line drawings, blocks of solid color, and sharp boundaries. The following illustration shows a BMP along with two JPEGs and a GIF. The JPEGs and the GIF were compressed from the BMP. The compression ratio is 4:1 for the GIF, 4:1 for the smaller JPEG, and 8:3 for the larger JPEG. Note that the GIF maintains the sharp boundaries along the lines, but the JPEGs tends to blur the boundaries.



JPEG is a compression scheme, not a file format. JPEG File Interchange Format (JFIF) is a file format commonly used for storing and transferring images that have been compressed according to the JPEG scheme. JFIF files displayed by Web browsers use the .jpg extension.

Exchangeable Image File (Exif)

Exif is a file format used for photographs captured by digital cameras. An Exif file contains an image that is compressed according to the JPEG specification. An Exif file also contains information about the photograph (date taken, shutter speed, exposure time, and so on) and information about the camera (manufacturer, model, and so on).

Portable Network Graphics (PNG)

The PNG format retains many of the advantages of the GIF format but also provides capabilities beyond those of GIF. Like GIF files, PNG files are compressed with no loss of information. PNG files can store colors with 8, 24, or 48 bits per pixel and gray scales with 1, 2, 4, 8, or 16 bits per pixel. In contrast, GIF files can use only 1, 2, 4, or 8 bits per pixel. A PNG file can also store an alpha value for each pixel, which specifies the degree to which the color of that pixel is blended with the background color.

PNG improves on GIF in its ability to progressively display an image; that is, to display better and better approximations of the image as it arrives over a network connection. PNG files can contain gamma correction and color correction information so that the images can be accurately rendered on a variety of display devices.

Tag Image File Format (TIFF)

TIFF is a flexible and extendable format that is supported by a wide variety of platforms and image-processing applications. TIFF files can store images with an arbitrary number of bits per pixel and can employ a variety of compression algorithms. Several images can be stored in a single, multiple-page TIFF file. Information related to the image (scanner make, host computer, type of compression, orientation, samples per pixel, and so on) can be stored in the file and arranged through the use of tags. The TIFF format can be extended as needed by the approval and addition of new tags.

Metafiles (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides the **Metafile** class so that you can record and display metafiles. A metafile, also called a vector image, is an image that is stored as a sequence of drawing commands and settings. The commands and settings recorded in a **Metafile** object can be stored in memory or saved to a file or stream.

GDI+ can display metafiles that have been stored in the following formats:

- Windows Metafile Format (WMF)
- Enhanced Metafile (EMF)
- EMF+

GDI+ can record metafiles in the EMF and EMF+ formats, but not in the WMF format.

EMF+ is an extension to EMF that allows GDI+ records to be stored. There are two variations on the EMF+ format: EMF+ Only and EMF+ Dual. EMF+ Only metafiles contain only GDI+ records. Such metafiles can be displayed by GDI+ but not by Windows Graphics Device Interface (GDI). EMF+ Dual metafiles contain GDI+ and GDI records. Each GDI+ record in an EMF+ Dual metafile is paired with an alternate GDI record. Such metafiles can be displayed by GDI+ or by GDI.

The following example records one setting and one drawing command in a disk file. Note that the example creates a **Graphics** object and that the constructor for the **Graphics** object receives the address of a **Metafile** object as an argument.

```
myMetafile = new Metafile(L"MyDiskFile.emf", hdc);
myGraphics = new Graphics(myMetafile);
myPen = new Pen(Color(255, 0, 0, 200));
myGraphics->SetSmoothingMode(SmoothingModeAntiAlias);
myGraphics->DrawLine(myPen, 0, 0, 60, 40);
delete myGraphics;
delete myPen;
delete myMetafile;
```

As the preceding example shows, the **Graphics** class is the key to recording instructions and settings in a **Metafile** object. Any call made to a method of a **Graphics** object can be recorded in a **Metafile** object. Likewise, you can set any property of a **Graphics** object and record that setting in a **Metafile** object. The recording ends when the **Graphics** object is deleted or goes out of scope.

The following example displays the metafile created in the preceding example. The metafile is displayed with its upper-left corner at (100, 100).

```
Graphics myGraphics(hdc);
Image myImage(L"MyDiskFile.emf");
myGraphics.DrawImage(&myImage, 100, 100);
```

The following example records several property settings (clipping region, world transformation, and smoothing mode) in a **Metafile** object. Then the code records several drawing instructions. The instructions and settings are saved in a disk file.

```
myMetafile = new Metafile(L"MyDiskFile2.emf", hdc);
myGraphics = new Graphics(myMetafile);
myGraphics->SetSmoothingMode(SmoothingModeAntiAlias);
myGraphics->RotateTransform(30);

// Create an elliptical clipping region.
GraphicsPath myPath;
myPath.AddEllipse(0, 0, 200, 100);
Region myRegion(&myPath);
myGraphics->SetClip(&myRegion);

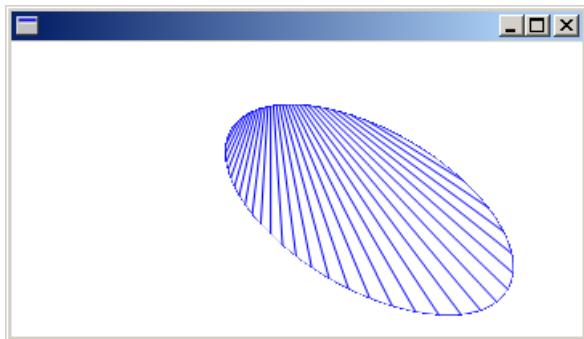
Pen myPen(Color(255, 0, 0, 255));
myGraphics->DrawPath(&myPen, &myPath);

for(INT j = 0; j <= 300; j += 10)
{
    myGraphics->DrawLine(&myPen, 0, 0, 300 - j, j);
}
delete myGraphics;
delete myMetafile;
```

The following example displays the metafile image created in the preceding example.

```
myGraphics = new Graphics(hdc);
myMetafile = new Metafile(L"MyDiskFile.emf");
myGraphics->DrawImage(myMetafile, 10, 10);
```

The following illustration shows the output of the preceding code. Note the antialiasing, the elliptical clipping region, and the 30-degree rotation.



Drawing, Positioning, and Cloning Images

11/2/2020 • 2 minutes to read • [Edit Online](#)

You can use the **Image** class to load and display raster images (bitmaps) and vector images (metafiles). To display an image, you need a **Graphics** object and an **Image** object. The **Graphics** object provides the **Graphics::DrawImage** method, which receives the address of the **Image** object as an argument.

The following example constructs an **Image** object from the file Climber.jpg and then displays the image. The destination point for the upper-left corner of the image, (10, 10), is specified in the second and third parameters of the **Graphics::DrawImage** method.

```
Image myImage(L"Climber.jpg");
myGraphics.DrawImage(&myImage, 10, 10);
```

The preceding code, along with a particular file, Climber.jpg, produced the following output.



You can construct **Image** objects from a variety of graphics file formats: BMP, GIF, JPEG, Exif, PNG, TIFF, WMF, EMF, and ICON.

The following example constructs **Image** objects from a variety of file types and then displays the images.

```
Image myBMP(L"SpaceCadet.bmp");
Image myEMF(L"Metafile1.emf");
Image myGIF(L"Soda.gif");
Image myJPEG(L"Mango.jpg");
Image myPNG(L"Flowers.png");
Image myTIFF(L"MS.tif");

myGraphics.DrawImage(&myBMP, 10, 10);
myGraphics.DrawImage(&myEMF, 220, 10);
myGraphics.DrawImage(&myGIF, 320, 10);
myGraphics.DrawImage(&myJPEG, 380, 10);
myGraphics.DrawImage(&myPNG, 150, 200);
myGraphics.DrawImage(&myTIFF, 300, 200);
```

The **Image** class provides a **Image::Clone** method that you can use to make a copy of an existing **Image**, **Metafile**, or **Bitmap** object. The **Clone** method is overloaded in the **Bitmap** class, and one of the variations has a source-rectangle parameter that you can use to specify the portion of the original image that you want to copy. The following example creates a **Bitmap** object by cloning the top half of an existing **Bitmap** object. Then both images are displayed.

```
Bitmap* originalBitmap = new Bitmap(L"Spiral.png");
RectF sourceRect(
    0.0f,
    0.0f,
    (REAL)(originalBitmap->GetWidth()),
    (REAL)(originalBitmap->GetHeight()/2.0f);

Bitmap* secondBitmap = originalBitmap->Clone(sourceRect, PixelFormatDontCare);

myGraphics.DrawImage(originalBitmap, 10, 10);
myGraphics.DrawImage(secondBitmap, 100, 10);
```

The preceding code, along with a particular file, Spiral.png, produced the following output.



About cropping and scaling GDI+ images

11/2/2020 • 2 minutes to read • [Edit Online](#)

You can use the [DrawImage](#) method of the [Graphics](#) class to draw and position images. [DrawImage](#) is an overloaded method, so there are several ways you can supply it with arguments. One variation of the [Graphics::DrawImage](#) method receives the address of an [Image](#) object and a reference to a [Rect](#) object. The rectangle specifies the destination for the drawing operation; that is, it specifies the rectangle in which the image will be drawn. If the size of the destination rectangle is different from the size of the original image, the image is scaled to fit the destination rectangle. The following example draws the same image three times: once with no scaling, once with an expansion, and once with a compression.

```
Bitmap myBitmap(L"Spiral.png");
Rect expansionRect(80, 10, 2 * myBitmap.GetWidth(), myBitmap.GetHeight());
Rect compressionRect(210, 10, myBitmap.GetWidth() / 2,
    myBitmap.GetHeight() / 2);

myGraphics.DrawImage(&myBitmap, 10, 10);
myGraphics.DrawImage(&myBitmap, expansionRect);
myGraphics.DrawImage(&myBitmap, compressionRect);
```

The preceding code, along with a particular file, Spiral.png, produced the following output.



Some variations of the [Graphics::DrawImage](#) method have a source-rectangle parameter as well as a destination-rectangle parameter. The source rectangle specifies the portion of the original image that will be drawn. The destination rectangle specifies where that portion of the image will be drawn. If the size of the destination rectangle is different from the size of the source rectangle, the image is scaled to fit the destination rectangle.

The following example constructs a [Bitmap](#) object from the file Runner.jpg. The entire image is drawn with no scaling at (0, 0). Then a small portion of the image is drawn twice: once with a compression and once with an expansion.

```
Bitmap myBitmap(L"Runner.jpg");

// The rectangle (in myBitmap) with upper-left corner (80, 70),
// width 80, and height 45, encloses one of the runner's hands.

// Small destination rectangle for compressed hand.
Rect destRect1(200, 10, 20, 16);

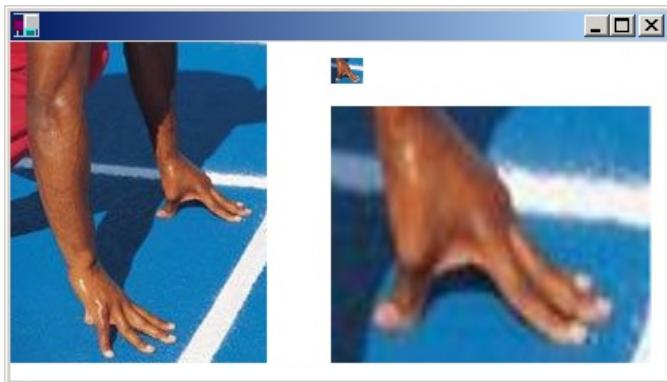
// Large destination rectangle for expanded hand.
Rect destRect2(200, 40, 200, 160);

// Draw the original image at (0, 0).
myGraphics.DrawImage(&myBitmap, 0, 0);

// Draw the compressed hand.
myGraphics.DrawImage(
    &myBitmap, destRect1, 80, 70, 80, 45, UnitPixel);

// Draw the expanded hand.
myGraphics.DrawImage(
    &myBitmap, destRect2, 80, 70, 80, 45, UnitPixel);
```

The following illustration shows the unscaled image, and the compressed and expanded image portions.



Coordinate Systems and Transformations

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides a world transformation and a page transformation so that you can transform (rotate, scale, translate, and so on) the items you draw. The two transformations also allow you to work in a variety of coordinate systems.

- [Types of Coordinate Systems](#)
- [Matrix Representation of Transformations](#)
- [Global and Local Transformations](#)

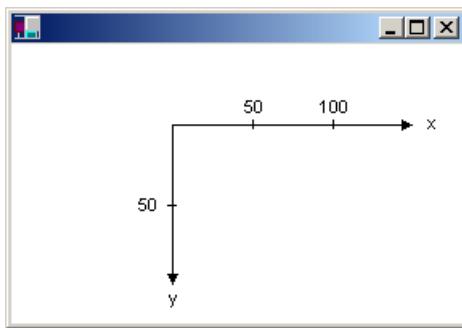
Types of Coordinate Systems

11/2/2020 • 3 minutes to read • [Edit Online](#)

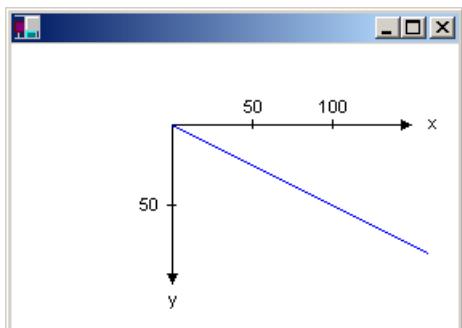
Windows GDI+ uses three coordinate spaces: world, page, and device. When you make the call

`myGraphics.DrawLine(&myPen, 0, 0, 160, 80)`, the points that you pass to the **Graphics::DrawLine** method — (0, 0) and (160, 80) — are in the world coordinate space. Before GDI+ can draw the line on the screen, the coordinates pass through a sequence of transformations. One transformation converts world coordinates to page coordinates, and another transformation converts page coordinates to device coordinates.

Suppose you want to work with a coordinate system that has its origin in the body of the client area rather than the upper-left corner. Say, for example, that you want the origin to be 100 pixels from the left edge of the client area and 50 pixels from the top of the client area. The following illustration shows such a coordinate system.



When you make the call `myGraphics.DrawLine(&myPen, 0, 0, 160, 80)`, you get the line shown in the following illustration.



The coordinates of the endpoints of your line in the three coordinate spaces are as follows:

World	(0, 0) to (160, 80)
Page	(100, 50) to (260, 130)
Device	(100, 50) to (260, 130)

Note that the page coordinate space has its origin at the upper-left corner of the client area; this will always be the case. Also note that because the unit of measure is the pixel, the device coordinates are the same as the page coordinates. If you set the unit of measure to something other than pixels (for example, inches), then the device coordinates will be different from the page coordinates.

The transformation that maps world coordinates to page coordinates is called the *world transformation* and is

maintained by a **Graphics** object. In the previous example, the world transformation is a translation 100 units in the x direction and 50 units in the y direction. The following example sets the world transformation of a **Graphics** object and then uses that **Graphics** object to draw the line shown in the previous figure.

```
myGraphics.TranslateTransform(100.0f, 50.0f);  
  
myGraphics.DrawLine(&myPen, 0, 0, 160, 80);
```

The transformation that maps page coordinates to device coordinates is called the *page transformation*. The **Graphics** class provides four methods for manipulating and inspecting the page transformation: **Graphics::SetPageUnit**, **Graphics::GetPageUnit**, **Graphics::SetPageScale**, and **Graphics::GetPageScale**. The **Graphics** class also provides two methods, **Graphics::GetDpiX** and **Graphics::GetDpiY**, for examining the horizontal and vertical dots per inch of the display device.

You can use the **Graphics::SetPageUnit** method of the **Graphics** class to specify a unit of measure. The following example draws a line from (0, 0) to (2, 1) where the point (2, 1) is 2 inches to the right and 1 inch down from the point (0, 0).

```
myGraphics.SetPageUnit(UnitInch);  
  
myGraphics.DrawLine(&myPen, 0, 0, 2, 1);
```

[!Note] If you don't specify a pen width when you construct your pen, the previous example will draw a line that is one inch wide. You can specify the pen width in the second argument to the **Pen** constructor:

```
Pen myPen(Color(255, 0, 0, 0), 1/myGraphics.GetDpiX()); .
```

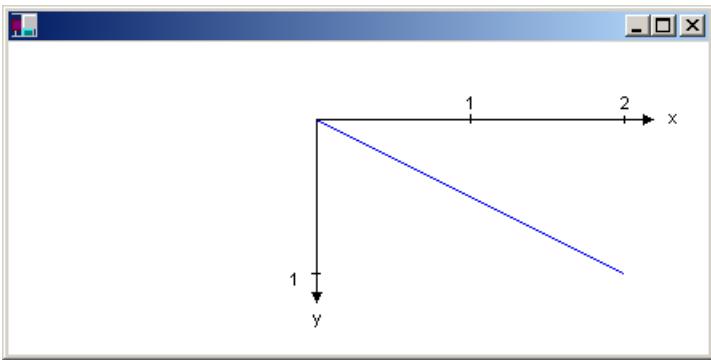
If we assume that the display device has 96 dots per inch in the horizontal direction and 96 dots per inch in the vertical direction, the endpoints of the line in the previous example have the following coordinates in the three coordinate spaces:

World	(0, 0) to (2, 1)
Page	(0, 0) to (2, 1)
Device	(0, 0, to (192, 96)

You can combine the world and page transformations to achieve a variety of effects. For example, suppose you want to use inches as the unit of measure and you want the origin of your coordinate system to be 2 inches from the left edge of the client area and 1/2 inch from the top of the client area. The following example sets the world and page transformations of a **Graphics** object and then draws a line from (0, 0) to (2, 1).

```
myGraphics.TranslateTransform(2.0f, 0.5f);  
myGraphics.SetPageUnit(UnitInch);  
myGraphics.DrawLine(&myPen, 0, 0, 2, 1);
```

The following illustration shows the line and coordinate system.



If we assume that the display device has 96 dots per inch in the horizontal direction and 96 dots per inch in the vertical direction, the endpoints of the line in the previous example have the following coordinates in the three coordinate spaces:

World	(0, 0) to (2, 1)
Page	(2, 0.5) to (4, 1.5)
Device	(192, 48) to (384, 144)

Matrix Representation of Transformations

2/22/2020 • 5 minutes to read • [Edit Online](#)

An $m \times n$ matrix is a set of numbers arranged in m rows and n columns. The following illustration shows several matrices.

$$\begin{bmatrix} 3 & 1 & 4 \\ 2 & 5 & 0 \end{bmatrix}_{2 \times 3} \quad \begin{bmatrix} 1 & 3 \\ 2 & 8 \\ 0 & 4 \\ 5 & 6 \end{bmatrix}_{4 \times 2} \quad \begin{bmatrix} 1.6 & 0.2 & 1.0 \end{bmatrix}_{1 \times 3}$$

$$\begin{bmatrix} 2 & 0 \\ 0 & 3.5 \end{bmatrix}_{2 \times 2} \quad \begin{bmatrix} 5 \\ 3 \end{bmatrix}_{2 \times 1} \quad \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 40 & 20 & 1 \end{bmatrix}_{3 \times 3}$$

You can add two matrices of the same size by adding individual elements. The following illustration shows two examples of matrix addition.

$$\begin{bmatrix} 5 & 4 \end{bmatrix} + \begin{bmatrix} 20 & 30 \end{bmatrix} = \begin{bmatrix} 25 & 34 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 1 & 3 \end{bmatrix} + \begin{bmatrix} 2 & 4 \\ 1 & 5 \\ 0 & 6 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 1 & 7 \\ 1 & 9 \end{bmatrix}$$

An $m \times n$ matrix can be multiplied by an $n \times p$ matrix, and the result is an $m \times p$ matrix. The number of columns in the first matrix must be the same as the number of rows in the second matrix. For example, a 4×2 matrix can be multiplied by a 2×3 matrix to produce a 4×3 matrix.

Points in the plane and rows and columns of a matrix can be thought of as vectors. For example, $(2, 5)$ is a vector with two components, and $(3, 7, 1)$ is a vector with three components. The dot product of two vectors is defined as follows:

$$(a, b) \cdot (c, d) = ac + bd$$

$$(a, b, c) \cdot (d, e, f) = ad + be + cf$$

For example, the dot product of $(2, 3)$ and $(5, 4)$ is $(2)(5) + (3)(4) = 22$. The dot product of $(2, 5, 1)$ and $(4, 3, 1)$ is $(2)(4) + (5)(3) + (1)(1) = 24$. Note that the dot product of two vectors is a number, not another vector. Also note that you can calculate the dot product only if the two vectors have the same number of components.

Let $A(i, j)$ be the entry in matrix A in the **ith** row and the **jth** column. For example $A(3, 2)$ is the entry in matrix A in the **3rd** row and the **2nd** column. Suppose A, B, and C are matrices, and $AB = C$. The entries of C are calculated as follows:

$$C(i, j) = (\text{row } i \text{ of } A) \cdot (\text{column } j \text{ of } B)$$

The following illustration shows several examples of matrix multiplication.

$$\begin{bmatrix} 2 & 3 \\ 1 \times 2 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 2 \times 1 \end{bmatrix} = \begin{bmatrix} 16 \\ 1 \times 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 \times 3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 4 \\ 5 & 1 \\ 3 \times 2 \end{bmatrix} = \begin{bmatrix} 17 & 14 \\ 1 \times 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 5 & 1 \\ 4 & 3 & 1 \\ 2 \times 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 2 & 3 & 1 \\ 3 \times 3 \end{bmatrix} = \begin{bmatrix} 4 & 13 & 1 \\ 6 & 9 & 1 \\ 2 \times 3 \end{bmatrix}$$

If you think of a point in the plane as a 1×2 matrix, you can transform that point by multiplying it by a 2×2 matrix. The following illustration shows several transformations applied to the point $(2, 1)$.

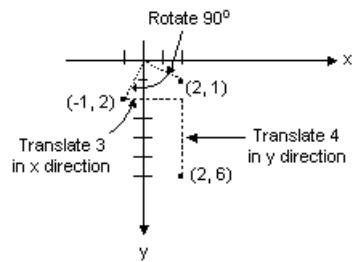
Scale $\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 1 \end{bmatrix}$

Rotate 90° $\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 2 \end{bmatrix}$

Reflect across x-axis $\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 2 & -1 \end{bmatrix}$

All the transformations shown in the previous figure are linear transformations. Certain other transformations, such as translation, are not linear, and cannot be expressed as multiplication by a 2×2 matrix. Suppose you want to start with the point $(2, 1)$, rotate it 90° degrees, translate it 3 units in the x direction, and translate it 4 units in the y direction. You can accomplish this by performing a matrix multiplication followed by a matrix addition.

$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 6 \end{bmatrix}$$



A linear transformation (multiplication by a 2×2 matrix) followed by a translation (addition of a 1×2 matrix) is called an affine transformation. An alternative to storing an affine transformation in a pair of matrices (one for the linear part and one for the translation) is to store the entire transformation in a 3×3 matrix. To make this work, a point in the plane must be stored in a 1×3 matrix with a dummy 3rd coordinate. The usual technique is to make all 3rd coordinates equal to 1. For example, the point $(2, 1)$ is represented by the matrix $[2 \ 1 \ 1]$. The following illustration shows an affine transformation (rotate 90 degrees; translate 3 units in the x direction, 4 units in the y direction) expressed as multiplication by a single 3×3 matrix.

$$\begin{bmatrix} 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 3 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 6 & 1 \end{bmatrix}$$

In the previous example, the point (2, 1) is mapped to the point (2, 6). Note that the third column of the 3×3 matrix contains the numbers 0, 0, 1. This will always be the case for the 3×3 matrix of an affine transformation. The important numbers are the six numbers in columns 1 and 2. The upper-left 2×2 portion of the matrix represents the linear part of the transformation, and the first two entries in the 3rd row represent the translation.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \\ 3 & 4 & 1 \end{bmatrix}$$

In Windows GDI+ you can store an affine transformation in a [Matrix](#) object. Because the third column of a matrix that represents an affine transformation is always (0, 0, 1), you specify only the six numbers in the first two columns when you construct a [Matrix](#) object. The statement

`Matrix myMatrix(0.0f, 1.0f, -1.0f, 0.0f, 3.0f, 4.0f);` constructs the matrix shown in the previous figure.

Composite Transformations

A composite transformation is a sequence of transformations, one followed by the other. Consider the matrices and transformations in the following list:

- Matrix A Rotate 90 degrees
- Matrix B Scale by a factor of 2 in the x direction
- Matrix C Translate 3 units in the y direction

If you start with the point (2, 1) — represented by the matrix [2 1 1] — and multiply by A, then B, then C, the point (2,1) will undergo the three transformations in the order listed.

$$[2 \ 1 \ 1]ABC = [-2 \ 5 \ 1]$$

Rather than store the three parts of the composite transformation in three separate matrices, you can multiply A, B, and C together to get a single 3×3 matrix that stores the entire composite transformation. Suppose $ABC = D$. Then a point multiplied by D gives the same result as a point multiplied by A, then B, then C.

$$[2 \ 1 \ 1]D = [-2 \ 5 \ 1]$$

The following illustration shows the matrices A, B, C, and D.

$$\begin{array}{c} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{bmatrix} \\ A \qquad B \qquad C \end{array} = \begin{bmatrix} 0 & 1 & 0 \\ -2 & 0 & 0 \\ 0 & 3 & 1 \end{bmatrix} \quad D$$

The fact that the matrix of a composite transformation can be formed by multiplying the individual transformation matrices means that any sequence of affine transformations can be stored in a single [Matrix](#) object.

NOTE

The order of a composite transformation is important. In general, rotate, then scale, then translate is not the same as scale, then rotate, then translate. Similarly, the order of matrix multiplication is important. In general, ABC is not the same as BAC .

The [Matrix](#) class provides several methods for building a composite transformation: [Matrix::Multiply](#), [Matrix::Rotate](#), [Matrix::RotateAt](#), [Matrix::Scale](#), [Matrix::Shear](#), and [Matrix::Translate](#). The following example creates the matrix of a composite transformation that first rotates 30 degrees, then scales by a factor of 2 in the y direction, and then translates 5 units in the x direction.

```
Matrix myMatrix;  
myMatrix.Rotate(30.0f);  
myMatrix.Scale(1.0f, 2.0f, MatrixOrderAppend);  
myMatrix.Translate(5.0f, 0.0f, MatrixOrderAppend);
```

The following illustration shows the matrix.

$$\begin{bmatrix} \cos 30^\circ & 2\sin 30^\circ & 0 \\ -\sin 30^\circ & 2\cos 30^\circ & 0 \\ 5 & 0 & 1 \end{bmatrix} \approx \begin{bmatrix} 0.866 & 1.0 & 0 \\ -0.5 & 1.73 & 0 \\ 5 & 0 & 1 \end{bmatrix}$$

Global and Local Transformations

11/2/2020 • 2 minutes to read • [Edit Online](#)

A global transformation is a transformation that applies to every item drawn by a given **Graphics** object. To create a global transformation, construct a **Graphics** object, and then call its **Graphics::SetTransform** method. The **Graphics::SetTransform** method manipulates a **Matrix** object that is associated with the **Graphics** object. The transformation stored in that **Matrix** object is called the *world transformation*. The world transformation can be a simple affine transformation or a complex sequence of affine transformations, but regardless of its complexity, the world transformation is stored in a single **Matrix** object.

The **Graphics** class provides several methods for building up a composite world transformation: **Graphics::MultiplyTransform**, **Graphics::RotateTransform**, **Graphics::ScaleTransform**, and **Graphics::TranslateTransform**. The following example draws an ellipse twice: once before creating a world transformation and once after. The transformation first scales by a factor of 0.5 in the y direction, then translates 50 units in the x direction, and then rotates 30 degrees.

```
myGraphics.DrawEllipse(&myPen, 0, 0, 100, 50);
myGraphics.ScaleTransform(1.0f, 0.5f);
myGraphics.TranslateTransform(50.0f, 0.0f, MatrixOrderAppend);
myGraphics.RotateTransform(30.0f, MatrixOrderAppend);
myGraphics.DrawEllipse(&myPen, 0, 0, 100, 50);
```

The following illustration shows the original ellipse and the transformed ellipse.



NOTE

In the previous example, the ellipse is rotated about the origin of the coordinate system, which is at the upper-left corner of the client area. This produces a different result than rotating the ellipse about its own center.

A local transformation is a transformation that applies to a specific item to be drawn. For example, a **GraphicsPath** object has a **GraphicsPath::Transform** method that allows you to transform the data points of that path. The following example draws a rectangle with no transformation and a path with a rotation transformation. (Assume that there is no world transformation.)

```
Matrix myMatrix;
myMatrix.Rotate(45.0f);
myGraphicsPath.Transform(&myMatrix);
myGraphics.DrawRectangle(&myPen, 10, 10, 100, 50);
myGraphics.DrawPath(&myPen, &myGraphicsPath);
```

You can combine the world transformation with local transformations to achieve a variety of results. For example,

you can use the world transformation to revise the coordinate system and use local transformations to rotate and scale objects drawn on the new coordinate system.

Suppose you want a coordinate system that has its origin 200 pixels from the left edge of the client area and 150 pixels from the top of the client area. Furthermore, assume that you want the unit of measure to be the pixel, with the x-axis pointing to the right and the y-axis pointing up. The default coordinate system has the y-axis pointing down, so you need to perform a reflection across the horizontal axis. The following illustration shows the matrix of such a reflection.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Next, assume you need to perform a translation 200 units to the right and 150 units down.

The following example establishes the coordinate system just described by setting the world transformation of a [Graphics](#) object.

```
Matrix myMatrix(1.0f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f);
myGraphics.SetTransform(&myMatrix);
myGraphics.TranslateTransform(200.0f, 150.0f, MatrixOrderAppend);
```

The following code (placed after the code in the preceding example) creates a path that consists of a single rectangle with its lower-left corner at the origin of the new coordinate system. The rectangle is filled once with no local transformation and once with a local transformation. The local transformation consists of a horizontal scaling by a factor of 2 followed by a 30-degree rotation.

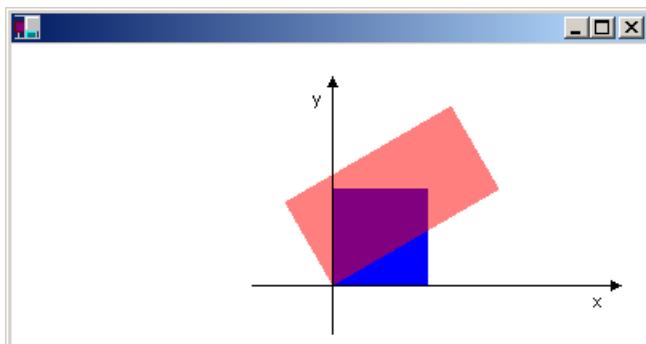
```
// Create the path.
GraphicsPath myGraphicsPath;
Rect myRect(0, 0, 60, 60);
myGraphicsPath.AddRectangle(myRect);

// Fill the path on the new coordinate system.
// No local transformation
myGraphics.FillPath(&mySolidBrush1, &myGraphicsPath);

// Transform the path.
Matrix myPathMatrix;
myPathMatrix.Scale(2, 1);
myPathMatrix.Rotate(30, MatrixOrderAppend);
myGraphicsPath.Transform(&myPathMatrix);

// Fill the transformed path on the new coordinate system.
myGraphics.FillPath(&mySolidBrush2, &myGraphicsPath);
```

The following illustration shows the new coordinate system and the two rectangles.



Graphics Containers

11/2/2020 • 5 minutes to read • [Edit Online](#)

Graphics state — clipping region, transformations, and quality settings — is stored in a **Graphics** object. Windows GDI+ allows you to temporarily replace or augment part of the state in a **Graphics** object by using a container. You start a container by calling the **Graphics::BeginContainer** method of a **Graphics** object, and you end a container by calling the **Graphics::EndContainer** method. In between **Graphics::BeginContainer** and **Graphics::EndContainer**, any state changes you make to the **Graphics** object belong to the container and do not overwrite the existing state of the **Graphics** object.

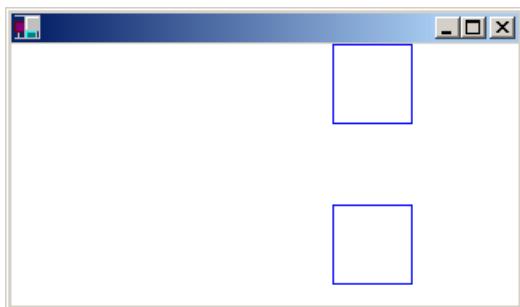
The following example creates a container within a **Graphics** object. The world transformation of the **Graphics** object is a translation 200 units to the right, and the world transformation of the container is a translation 100 units down.

```
myGraphics.TranslateTransform(200.0f, 0.0f);

myGraphicsContainer = myGraphics.BeginContainer();
    myGraphics.TranslateTransform(0.0f, 100.0f);
    myGraphics.DrawRectangle(&myPen, 0, 0, 50, 50);
myGraphics.EndContainer(myGraphicsContainer);

myGraphics.DrawRectangle(&myPen, 0, 0, 50, 50);
```

Note that in the previous example, the statement `myGraphics.DrawRectangle(&myPen, 0, 0, 50, 50)` made in between the calls to **Graphics::BeginContainer** and **Graphics::EndContainer** produces a different rectangle than the same statement made after the call to **Graphics::EndContainer**. Only the horizontal translation applies to the **DrawRectangle** call made outside of the container. Both transformations — the horizontal translation of 200 units and the vertical translation of 100 units — apply to the **Graphics::DrawRectangle** call made inside the container. The following illustration shows the two rectangles.



Containers can be nested within containers. The following example creates a container within a **Graphics** object and another container within the first container. The world transformation of the **Graphics** object is a translation 100 units in the x direction and 80 units in the y direction. The world transformation of the first container is a 30-degree rotation. The world transformation of the second container is a scaling by a factor of 2 in the x direction. A call to the **Graphics::DrawEllipse** method is made inside the second container.

```

myGraphics.TranslateTransform(100.0f, 80.0f, MatrixOrderAppend);

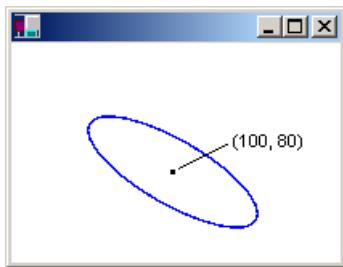
container1 = myGraphics.BeginContainer();
myGraphics.RotateTransform(30.0f, MatrixOrderAppend);

container2 = myGraphics.BeginContainer();
myGraphics.ScaleTransform(2.0f, 1.0f);
myGraphics.DrawEllipse(&myPen, -30, -20, 60, 40);
myGraphics.EndContainer(container2);

myGraphics.EndContainer(container1);

```

The following illustration shows the ellipse.



Note that all three transformations apply to the [Graphics::DrawEllipse](#) call made in the second (innermost) container. Also note the order of the transformations: first scale, then rotate, then translate. The innermost transformation is applied first, and the outermost transformation is applied last.

Any property of a [Graphics](#) object can be set inside a container (in between calls to [Graphics::BeginContainer](#) and [Graphics::EndContainer](#)). For example, a clipping region can be set inside a container. Any drawing done inside the container will be restricted to the clipping region of that container and will also be restricted to the clipping regions of any outer containers and the clipping region of the [Graphics](#) object itself.

The properties discussed so far — the world transformation and the clipping region — are combined by nested containers. Other properties are temporarily replaced by a nested container. For example, if you set the smoothing mode to `SmoothingModeAntiAlias` within a container, any drawing methods called inside that container will use the antialias smoothing mode, but drawing methods called after [Graphics::EndContainer](#) will use the smoothing mode that was in place before the call to [Graphics::BeginContainer](#).

For another example of combining the world transformations of a [Graphics](#) object and a container, suppose you want to draw an eye and place it at various locations on a sequence of faces. The following example draws an eye centered at the origin of the coordinate system.

```

void DrawEye(Graphics* pGraphics)
{
    GraphicsContainer eyeContainer;

    eyeContainer = pGraphics->BeginContainer();

    Pen myBlackPen(Color(255, 0, 0, 0));
    SolidBrush myGreenBrush(Color(255, 0, 128, 0));
    SolidBrush myBlackBrush(Color(255, 0, 0, 0));

    GraphicsPath myTopPath;
    myTopPath.AddEllipse(-30, -50, 60, 60);

    GraphicsPath myBottomPath;
    myBottomPath.AddEllipse(-30, -10, 60, 60);

    Region myTopRegion(&myTopPath);
    Region myBottomRegion(&myBottomPath);

    // Draw the outline of the eye.
    // The outline of the eye consists of two ellipses.
    // The top ellipse is clipped by the bottom ellipse, and
    // the bottom ellipse is clipped by the top ellipse.
    pGraphics->SetClip(&myTopRegion);
    pGraphics->DrawPath(&myBlackPen, &myBottomPath);
    pGraphics->SetClip(&myBottomRegion);
    pGraphics->DrawPath(&myBlackPen, &myTopPath);

    // Fill the iris.
    // The iris is clipped by the bottom ellipse.
    pGraphics->FillEllipse(&myGreenBrush, -10, -15, 20, 22);

    // Fill the pupil.
    pGraphics->FillEllipse(&myBlackBrush, -3, -7, 6, 9);

    pGraphics->EndContainer(eyeContainer);
}

```

The following illustration shows the eye and the coordinate axes.



The `DrawEye` function, defined in the previous example receives the address of a `Graphics` object and immediately creates a container within that `Graphics` object. This container insulates any code that calls the `DrawEye` function from property settings made during the execution of the `DrawEye` function. For example, code in the `DrawEye` function sets the clipping region of the `Graphics` object, but when `DrawEye` returns control to the calling routine, the clipping region will be as it was before the call to `DrawEye`.

The following example draws three ellipses (faces), each with an eye inside.

```

// Draw an ellipse with center at (100, 100).
myGraphics.TranslateTransform(100.0f, 100.0f);
myGraphics.DrawEllipse(&myBlackPen, -40, -60, 80, 120);

// Draw the eye at the center of the ellipse.
DrawEye(&myGraphics);

// Draw an ellipse with center at 200, 100.
myGraphics.TranslateTransform(100.0f, 0.0f, MatrixOrderAppend);
myGraphics.DrawEllipse(&myBlackPen, -40, -60, 80, 120);

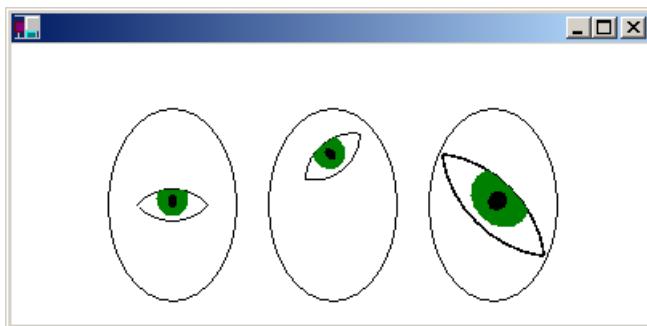
// Rotate the eye 40 degrees, and draw it 30 units above
// the center of the ellipse.
myGraphicsContainer = myGraphics.BeginContainer();
    myGraphics.RotateTransform(-40.0f);
    myGraphics.TranslateTransform(0.0f, -30.0f, MatrixOrderAppend);
    DrawEye(&myGraphics);
myGraphics.EndContainer(myGraphicsContainer);

// Draw a ellipse with center at (300.0f, 100.0f).
myGraphics.TranslateTransform(100.0f, 0.0f, MatrixOrderAppend);
myGraphics.DrawEllipse(&myBlackPen, -40, -60, 80, 120);

// Stretch and rotate the eye, and draw it at the
// center of the ellipse.
myGraphicsContainer = myGraphics.BeginContainer();
    myGraphics.ScaleTransform(2.0f, 1.5f);
    myGraphics.RotateTransform(45.0f, MatrixOrderAppend);
    DrawEye(&myGraphics);
myGraphics.EndContainer(myGraphicsContainer);

```

The following illustration shows the three ellipses.



In the previous example, all ellipses are drawn with the call `DrawEllipse(&myBlackPen, -40, -60, 80, 120)`, which draws an ellipse centered at the origin of the coordinate system. The ellipses are moved away from the upper-left corner of the client area by setting the world transformation of the `Graphics` object. The statement causes the first ellipse to be centered at (100, 100). The statement causes the center of the second ellipse to be 100 units to the right of the center of the first ellipse. Likewise, the center of the third ellipse is 100 units to the right of the center of the second ellipse.

The containers in the previous example are used to transform the eye relative to the center of a given ellipse. The first eye is drawn at the center of the ellipse with no transformation, so the `DrawEye` call is not inside a container. The second eye is rotated 40 degrees and drawn 30 units above the center of the ellipse, so the `DrawEye` function and the methods that set the transformation are called inside a container. The third eye is stretched and rotated and drawn at the center of the ellipse. As with the second eye, the `DrawEye` function and the methods that set the transformation are called inside a container.

Using GDI+

2/22/2020 • 2 minutes to read • [Edit Online](#)

The following topics describe how to use the GDI+ API with the C++ programming language:

- [Getting Started](#)
- [Using a Pen to Draw Lines and Shapes](#)
- [Using a Brush to Fill Shapes](#)
- [Using Images, Bitmaps, and Metafiles](#)
- [Using Image Encoders and Decoders](#)
- [Alpha Blending Lines and Fills](#)
- [Using Text and Fonts](#)
- [Constructing and Drawing Curves](#)
- [Filling Shapes with a Gradient Brush](#)
- [Constructing and Drawing Paths](#)
- [Using Graphics Containers](#)
- [Transformations](#)
- [Using Regions](#)
- [Recoloring](#)
- [Printing](#)

Getting started with GDI+

2/22/2020 • 2 minutes to read • [Edit Online](#)

This section shows how to get started using Windows GDI+ in a standard C++ Windows application. Drawing lines and strings are two of the simplest tasks you can perform in GDI+. The following topics discuss these two tasks:

[Drawing a Line](#)

[Drawing a String](#)

Drawing a Line

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic demonstrates how to draw a line using GDI Plus.

To draw a line in Windows GDI+ you need a **Graphics** object, a **Pen** object, and a **Color** object. The **Graphics** object provides the **DrawLine** method, and the **Pen** object holds attributes of the line, such as color and width. The address of the **Pen** object is passed as an argument to the **DrawLine** method.

The following program, which draws a line from (0, 0) to (200, 100), consists of three functions: **WinMain**, **WndProc**, and **OnPaint**. The **WinMain** and **WndProc** functions provide the fundamental code common to most Windows applications. There is no GDI+ code in the **WndProc** function. The **WinMain** function has a small amount of GDI+ code, namely the required calls to **GdiplusStartup** and **GdiplusShutdown**. The GDI+ code that actually creates a **Graphics** object and draws a line is in the **OnPaint** function.

The **OnPaint** function receives a handle to a device context and passes that handle to a **Graphics** constructor. The argument passed to the **Pen** constructor is a reference to a **Color** object. The four numbers passed to the color constructor represent the alpha, red, green, and blue components of the color. The alpha component determines the transparency of the color; 0 is fully transparent and 255 is fully opaque. The four numbers passed to the **DrawLine** method represent the starting point (0, 0) and the ending point (200, 100) of the line.

```
#include <stdafx.h>
#include <windows.h>
#include <objidl.h>
#include <gdiplus.h>
using namespace Gdiplus;
#pragma comment (lib,"Gdiplus.lib")

VOID OnPaint(HDC hdc)
{
    Graphics graphics(hdc);
    Pen      pen(Color(255, 0, 0, 255));
    graphics.DrawLine(&pen, 0, 0, 200, 100);
}

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, PSTR, INT iCmdShow)
{
    HWND          hWnd;
    MSG           msg;
    WNDCLASS     wndClass;
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR     gdiplusToken;

    // Initialize GDI+.
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    wndClass.style      = CS_HREDRAW | CS_VREDRAW;
    wndClass.lpfnWndProc = WndProc;
    wndClass.cbClsExtra = 0;
    wndClass.cbWndExtra = 0;
    wndClass.hInstance  = hInstance;
    wndClass.hIcon      = LoadIcon(NULL, IDI_APPLICATION);
    wndClass.hCursor    = LoadCursor(NULL, IDC_ARROW);
    wndClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndClass.lpszMenuName = NULL;
    wndClass.lpszClassName = TEXT("GettingStarted");
```

```

RegisterClass(&wndClass);

hWnd = CreateWindow(
    TEXT("GettingStarted"), // window class name
    TEXT("Getting Started"), // window caption
    WS_OVERLAPPEDWINDOW, // window style
    CW_USEDEFAULT, // initial x position
    CW_USEDEFAULT, // initial y position
    CW_USEDEFAULT, // initial x size
    CW_USEDEFAULT, // initial y size
    NULL, // parent window handle
    NULL, // window menu handle
    hInstance, // program instance handle
    NULL); // creation parameters

ShowWindow(hWnd, iCmdShow);
UpdateWindow(hWnd);

while( GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}

GdipplusShutdown(gdipplusToken);
return msg.wParam;
} // WinMain

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
    WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;

    switch(message)
    {
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
        OnPaint(hdc);
        EndPaint(hWnd, &ps);
        return 0;
    case WM_DESTROY:
        PostQuitMessage(0);
        return 0;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
} // WndProc

```

Note the call to [GdipplusStartup](#) in the **WinMain** function. The first parameter of the **GdipplusStartup** function is the address of a **ULONG_PTR**. **GdipplusStartup** fills that variable with a token that is later passed to the [GdipplusShutdown](#) function. The second parameter of the **GdipplusStartup** function is the address of a [GdipplusStartupInput](#) structure. The preceding code relies on the default **GdipplusStartupInput** constructor to set the structure members appropriately.

Drawing a String

11/2/2020 • 2 minutes to read • [Edit Online](#)

The topic [Drawing a Line](#) shows how to write a Windows application that uses Windows GDI+ to draw a line. To draw a string, replace the **OnPaint** function shown in that topic with the following **OnPaint** function:

```
VOID OnPaint(HDC hdc)
{
    Graphics    graphics(hdc);
    SolidBrush  brush(Color(255, 0, 0, 255));
    FontFamily  fontFamily(L"Times New Roman");
    Font        font(&fontFamily, 24, FontStyleRegular, UnitPixel);
    PointF      pointF(10.0f, 20.0f);

    graphics.DrawString(L"Hello World!", -1, &font, pointF, &brush);
}
```

The previous code creates several GDI+ objects. The **Graphics** object provides the **DrawString** method, which does the actual drawing. The **SolidBrush** object specifies the color of the string.

The **FontFamily** constructor receives a single, string argument that identifies the font family. The address of the **FontFamily** object is the first argument passed to the **Font** constructor. The second argument passed to the **Font** constructor specifies the font size, and the third argument specifies the style. The value **FontStyleRegular** is a member of the **FontStyle** enumeration, which is declared in **Gdiplusenums.h**. The last argument to the **Font** constructor indicates that the size of the font (24 in this case) is measured in pixels. The value **UnitPixel** is a member of the **Unit** enumeration.

The first argument passed to the **DrawString** method is the address of a wide-character string. The second argument, **-1**, specifies that the string is null terminated. (If the string is not null terminated, the second argument should specify the number of wide characters in the string.) The third argument is the address of the **Font** object. The fourth argument is a reference to a **PointF** object that specifies the location where the string will be drawn. The last argument is the address of the **Brush** object, which specifies the color of the string.

Using a Pen to Draw Lines and Shapes

11/2/2020 • 2 minutes to read • [Edit Online](#)

The **Graphics** class provides a variety of drawing methods including those shown in the following list:

- [DrawLine Methods](#)
- [DrawRectangle Methods](#)
- [DrawEllipse Methods](#)
- [DrawArc Methods](#)
- [Graphics::DrawPath](#)
- [DrawCurve Methods](#)
- [DrawBezier Methods](#)

One of the arguments that you pass to such a drawing method is the address of a **Pen** object.

The following topics cover the use of pens in more detail:

- [Using a Pen to Draw Lines and Rectangles](#)
- [Setting Pen Width and Alignment](#)
- [Drawing a Line with Line Caps](#)
- [Joining Lines](#)
- [Drawing a Custom Dashed Line](#)
- [Drawing a Line Filled with a Texture](#)

Using a Pen to Draw Lines and Rectangles

11/2/2020 • 2 minutes to read • [Edit Online](#)

To draw lines and rectangles, you need a **Graphics** object and a **Pen** object. The **Graphics** object provides the **DrawLine** method, and the **Pen** object stores features of the line, such as color and width.

The following example draws a line from (20, 10) to (300, 100). Assume **graphics** is an existing **Graphics** object.

```
Pen pen(Color(255, 0, 0, 0));
graphics.DrawLine(&pen, 20, 10, 300, 100);
```

The first statement of code uses the **Pen** class constructor to create a black pen. The one argument passed to the **Pen** constructor is a **Color** object. The values used to construct the **Color** object — (255, 0, 0, 0) — correspond to the alpha, red, green, and blue components of the color. These values define an opaque black pen.

The following example draws a rectangle with its upper-left corner at (10, 10). The rectangle has a width of 100 and a height of 50. The second argument passed to the **Pen** constructor indicates that the pen width is 5 pixels.

```
Pen blackPen(Color(255, 0, 0, 0), 5);
stat = graphics.DrawRectangle(&blackPen, 10, 10, 100, 50);
```

When the rectangle is drawn, the pen is centered on the rectangle's boundary. Because the pen width is 5, the sides of the rectangle are drawn 5 pixels wide, such that 1 pixel is drawn on the boundary itself, 2 pixels are drawn on the inside, and 2 pixels are drawn on the outside. For more details on pen alignment, see [Setting Pen Width and Alignment](#).

The following illustration shows the resulting rectangle. The dotted lines show where the rectangle would have been drawn if the pen width had been one pixel. The enlarged view of the upper-left corner of the rectangle shows that the thick black lines are centered on those dotted lines.



Setting Pen Width and Alignment

2/22/2020 • 2 minutes to read • [Edit Online](#)

When you create a [Pen](#) object, you can supply the pen width as one of the arguments to the constructor. You can also change the pen width by using the [Pen::SetWidth](#) method.

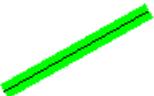
A theoretical line has a width of zero. When you draw a line, the pixels are centered on the theoretical line. The following example draws a specified line twice: once with a black pen of width 1 and once with a green pen of width 10.

```
Pen blackPen(Color(255, 0, 0, 0), 1);
Pen greenPen(Color(255, 0, 255, 0), 10);
stat = greenPen.SetAlignment(PenAlignmentCenter);

// Draw the line with the wide green pen.
stat = graphics.DrawLine(&greenPen, 10, 100, 100, 50);

// Draw the same line with the thin black pen.
stat = graphics.DrawLine(&blackPen, 10, 100, 100, 50);
```

The following illustration shows the output of the preceding code. The green pixels and the black pixels are centered on the theoretical line.



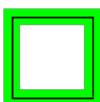
The following example draws a specified rectangle twice: once with a black pen of width 1 and once with a green pen of width 10. The code passes the value [PenAlignmentCenter](#) (an element of the [PenAlignment](#) enumeration) to the [Pen::SetAlignment](#) method to specify that the pixels drawn with the green pen are centered on the boundary of the rectangle.

```
Pen blackPen(Color(255, 0, 0, 0), 1);
Pen greenPen(Color(255, 0, 255, 0), 10);
stat = greenPen.SetAlignment(PenAlignmentCenter);

// Draw the rectangle with the wide green pen.
stat = graphics.DrawRectangle(&greenPen, 10, 100, 50, 50);

// Draw the same rectangle with the thin black pen.
stat = graphics.DrawRectangle(&blackPen, 10, 100, 50, 50);
```

The following illustration shows the output of the preceding code. The green pixels are centered on the theoretical rectangle, which is represented by the black pixels.



You can change the green pen's alignment by modifying the third statement in the preceding example as follows:

```
stat = greenPen.SetAlignment(PenAlignmentInset);
```

Now the pixels in the wide green line appear on the inside of the rectangle as shown in the following illustration.



Drawing a Line with Line Caps

2/22/2020 • 2 minutes to read • [Edit Online](#)

You can draw the start or end of a line in one of several shapes called line caps. Windows GDI+ supports several line caps, such as round, square, diamond, and arrowhead.

You can specify line caps for the start of a line (start cap), the end of a line (end cap), or the dashes of a dashed line (dash cap).

The following example draws a line with an arrowhead at one end and a round cap at the other end:

```
Pen pen(Color(255, 0, 0, 255), 8);
stat = pen.SetStartCap(LineCapArrowAnchor);
stat = pen.SetEndCap(LineCapRoundAnchor);
stat = graphics.DrawLine(&pen, 20, 175, 300, 175);
```

The following illustration shows the resulting line.



`LineCapArrowAnchor` and `LineCapRoundAnchor` are elements of the [LineCap](#) enumeration.

Joining Lines

2/22/2020 • 2 minutes to read • [Edit Online](#)

A line join is the common area that is formed by two lines whose ends meet or overlap. Windows GDI+ provides four line join styles: miter, bevel, round, and miter clipped. Line join style is a property of the [Pen](#) class. When you specify a line join style for a pen and then use that pen to draw a path, the specified join style is applied to all the connected lines in the path.

You can specify the line join style by using the [Pen::SetLineJoin](#) method of the [Pen](#) class. The following example demonstrates a beveled line join between a horizontal line and a vertical line:

```
GraphicsPath path;
Pen penJoin(Color(255, 0, 0, 255), 8);

path.StartFigure();
path.AddLine(Point(50, 200), Point(100, 200));
path.AddLine(Point(100, 200), Point(100, 250));

penJoin.SetLineJoin(LineJoinBevel);
graphics.DrawPath(&penJoin, &path);
```

The following illustration shows the resulting beveled line join.



In the preceding example, the value ([LineJoinBevel](#)) passed to the [Pen::SetLineJoin](#) method is an element of the [LineJoin](#) enumeration.

Drawing a Custom Dashed Line

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides several dash styles that are listed in the [DashStyle](#) enumeration. If those standard dash styles don't suit your needs, you can create a custom dash pattern.

To draw a custom dashed line, put the lengths of the dashes and spaces in an array and pass the address of the array as an argument to the [Pen::SetDashPattern](#) method of a [Pen](#) object. The following example draws a custom dashed line based on the array {5, 2, 15, 4}. If you multiply the elements of the array by the pen width of 5, you get {25, 10, 75, 20}. The displayed dashes alternate in length between 25 and 75, and the spaces alternate in length between 10 and 20.

```
REAL dashValues[4] = {5, 2, 15, 4};  
Pen blackPen(Color(255, 0, 0, 0), 5);  
blackPen.SetDashPattern(dashValues, 4);  
stat = graphics.DrawLine(&blackPen, Point(5, 5), Point(405, 5));
```

The following illustration shows the resulting dashed line. Note that the final dash has to be shorter than 25 units so that the line can end at (405, 5).



Drawing a Line Filled with a Texture

11/2/2020 • 2 minutes to read • [Edit Online](#)

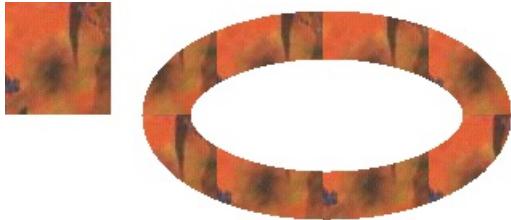
Instead of drawing a line or curve with a solid color, you can draw with a texture. To draw lines and curves with a texture, create a **TextureBrush** object, and pass the address of that **TextureBrush** object to a **Pen** constructor. The image associated with the texture brush is used to tile the plane (invisibly), and when the pen draws a line or curve, the stroke of the pen uncovers certain pixels of the tiled texture.

The following example creates an **Image** object from the file `Texture1.jpg`. That image is used to construct a **TextureBrush** object, and the **TextureBrush** object is used to construct a **Pen** object. The call to **Graphics::DrawImage** draws the image with its upper-left corner at (0, 0). The call to **Graphics::DrawEllipse** uses the **Pen** object to draw a textured ellipse.

```
Image      image(L"Texture1.jpg");
TextureBrush tBrush(&image);
Pen        texturedPen(&tBrush, 30);

graphics.DrawImage(&image, 0, 0, image.GetWidth(), image.GetHeight());
graphics.DrawEllipse(&texturedPen, 100, 20, 200, 100);
```

The following illustration shows the image and the textured ellipse.



Using a Brush to Fill Shapes

2/22/2020 • 2 minutes to read • [Edit Online](#)

A Windows GDI+[Brush](#) object is used to fill the interior of a closed shape. GDI+ defines several fill styles: solid color, hatch pattern, image texture, and color gradient.

The following topics cover the use of brushes in more detail:

- [Filling a Shape with a Solid Color](#)
- [Filling a Shape with a Hatch Pattern](#)
- [Filling a Shape with an Image Texture](#)
- [Tiling a Shape with an Image](#)
- [Filling a Shape with a Color Gradient](#)

Filling a Shape with a Solid Color

11/2/2020 • 2 minutes to read • [Edit Online](#)

To fill a shape with a solid color, create a **SolidBrush** object, and then pass the address of that **SolidBrush** object as an argument to one of the fill methods of the **Graphics** class. The following example shows how to fill an ellipse with the color red:

```
SolidBrush solidBrush(Color(255, 255, 0, 0));
stat = graphics.FillEllipse(&solidBrush, 0, 0, 100, 60);
```

In the preceding example, the **SolidBrush** constructor takes a **Color** object reference as its only argument. The values used by the **Color** constructor represent the alpha, red, green, and blue components of the color. Each of these values must be in the range 0 through 255. The first 255 indicates that the color is fully opaque, and the second 255 indicates that the red component is at full intensity. The two zeros indicate that the green and blue components both have an intensity of 0.

The four numbers (0, 0, 100, 60) passed to the **Graphics::FillEllipse** method specify the location and size of the bounding rectangle for the ellipse. The rectangle has an upper-left corner of (0, 0), a width of 100, and a height of 60.

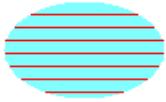
Filling a Shape with a Hatch Pattern

2/22/2020 • 2 minutes to read • [Edit Online](#)

A hatch pattern is made from two colors: one for the background and one for the lines that form the pattern over the background. To fill a closed shape with a hatch pattern, use a [HatchBrush](#) object. The following example demonstrates how to fill an ellipse with a hatch pattern:

```
HatchBrush hBrush(HatchStyleHorizontal, Color(255, 255, 0, 0),
    Color(255, 128, 255, 255));
stat = graphics.FillEllipse(&hBrush, 0, 0, 100, 60);
```

The following illustration shows the filled ellipse.



The [HatchBrush](#) constructor takes three arguments: the hatch style, the color of the hatch line, and the color of the background. The hatch style argument can be any element of the [HatchStyle](#) enumeration. There are more than fifty elements in the [HatchStyle](#) enumeration; a few of those elements are shown in the following list:

- [HatchStyleHorizontal](#)
- [HatchStyleVertical](#)
- [HatchStyleForwardDiagonal](#)
- [HatchStyleBackwardDiagonal](#)
- [HatchStyleCross](#)
- [HatchStyleDiagonalCross](#)

Filling a Shape with an Image Texture

2/22/2020 • 2 minutes to read • [Edit Online](#)

You can fill a closed shape with a texture by using the **Image** class and the **TextureBrush** class.

The following example fills an ellipse with an image. The code constructs an **Image** object, and then passes the address of that **Image** object as an argument to a **TextureBrush** constructor. The third code statement scales the image, and the fourth statement fills the ellipse with repeated copies of the scaled image:

```
Image image(L"ImageFile.jpg");
TextureBrush tBrush(&image);
stat = tBrush.SetTransform(&Matrix(75.0/640.0, 0.0f, 0.0f,
    75.0/480.0, 0.0f, 0.0f));
stat = graphics.FillEllipse(&tBrush,Rect(0, 150, 150, 250));
```

In the preceding code example, the **TextureBrush::SetTransform** method sets the transformation that is applied to the image before it is drawn. Assume that the original image has a width of 640 pixels and a height of 480 pixels. The transform shrinks the image to 75×75 , by setting the horizontal and vertical scaling values.

NOTE

In the preceding example, the image size is 75×75 , and the ellipse size is 150×250 . Because the image is smaller than the ellipse it is filling, the ellipse is tiled with the image. Tiling means that the image is repeated horizontally and vertically until the boundary of the shape is reached. For more information on tiling, see [Tiling a Shape with an Image](#).

Tiling a Shape with an Image

2/22/2020 • 2 minutes to read • [Edit Online](#)

Just as tiles can be placed next to each other to cover a floor, rectangular images can be placed next to each other to fill (tile) a shape. To tile the interior of a shape, use a texture brush. When you construct a **TextureBrush** object, one of the arguments you pass to the constructor is the address of an **Image** object. When you use the texture brush to paint the interior of a shape, the shape is filled with repeated copies of this image.

The wrap mode property of the **TextureBrush** object determines how the image is oriented as it is repeated in a rectangular grid. You can make all the tiles in the grid have the same orientation, or you can make the image flip from one grid position to the next. The flipping can be horizontal, vertical, or both. The following examples demonstrate tiling with different types of flipping.

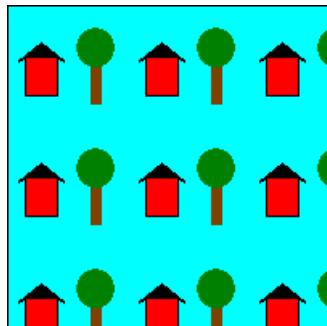
Tiling an Image

This example uses the following 75×75 image to tile a 200×200 rectangle:



```
Image image(L"HouseAndTree.png");
TextureBrush tBrush(&image);
Pen blackPen(Color(255, 0, 0, 0));
stat = graphics.FillRectangle(&tBrush, Rect(0, 0, 200, 200));
stat = graphics.DrawRectangle(&blackPen, Rect(0, 0, 200, 200));
```

The following illustration shows how the rectangle is tiled with the image. Note that all tiles have the same orientation; there is no flipping.

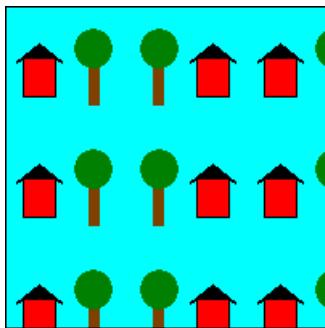


Flipping an Image Horizontally While Tiling

This example uses a 75×75 image to fill a 200×200 rectangle. The wrap mode is set to flip the image horizontally.

```
Image image(L"HouseAndTree.png");
TextureBrush tBrush(&image);
Pen blackPen(Color(255, 0, 0, 0));
stat = tBrush.SetWrapMode(WrapModeTileFlipX);
stat = graphics.FillRectangle(&tBrush, Rect(0, 0, 200, 200));
stat = graphics.DrawRectangle(&blackPen, Rect(0, 0, 200, 200));
```

The following illustration shows how the rectangle is tiled with the image. Note that as you move from one tile to the next in a given row, the image is flipped horizontally.

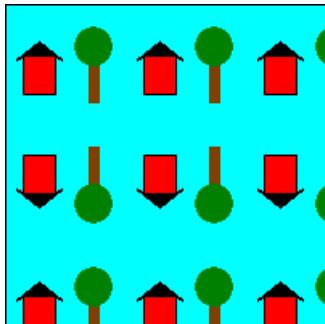


Flipping an Image Vertically While Tiling

This example uses a 75×75 image to fill a 200×200 rectangle. The wrap mode is set to flip the image vertically.

```
Image image(L"HouseAndTree.png");
TextureBrush tBrush(&image);
Pen blackPen(Color(255, 0, 0, 0));
stat = tBrush.SetWrapMode(WrapModeTileFlipY);
stat = graphics.FillRectangle(&tBrush, Rect(0, 0, 200, 200));
stat = graphics.DrawRectangle(&blackPen, Rect(0, 0, 200, 200));
```

The following illustration shows how the rectangle is tiled with the image. Note that as you move from one tile to the next in a given column, the image is flipped vertically.

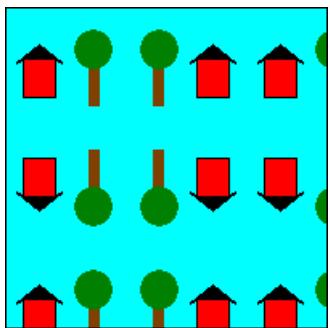


Flipping an Image Horizontally and Vertically While Tiling

This example uses a 75×75 image to tile a 200×200 rectangle. The wrap mode is set to flip the image both horizontally and vertically.

```
Image image(L"HouseAndTree.png");
TextureBrush tBrush(&image);
Pen blackPen(Color(255, 0, 0, 0));
stat = tBrush.SetWrapMode(WrapModeTileFlipXY);
stat = graphics.FillRectangle(&tBrush, Rect(0, 0, 200, 200));
stat = graphics.DrawRectangle(&blackPen, Rect(0, 0, 200, 200));
```

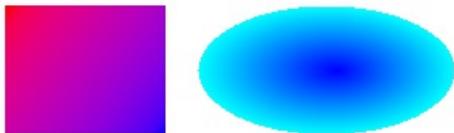
The following illustration shows how the rectangle is tiled by the image. Note that as you move from one tile to the next in a given row, the image is flipped horizontally, and as you move from one tile to the next in a given column, the image is flipped vertically.



Filling a Shape with a Color Gradient

2/22/2020 • 2 minutes to read • [Edit Online](#)

You can fill a shape with a gradually changing color by using a gradient brush. Windows GDI+ provides linear gradient brushes and path gradient brushes. The following illustration shows a rectangle filled with a linear gradient brush and an ellipse filled with a path gradient brush.



For more information about gradient brushes, see [Filling Shapes With a Gradient Brush](#).

Using Images, Bitmaps, and Metafiles

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides the [Image](#) class for working with raster images (bitmaps) and vector images (metafiles). The [Bitmap](#) class and the [Metafile](#) class both inherit from the [Image](#) class. The [Bitmap](#) class expands on the capabilities of the [Image](#) class by providing additional methods for loading, saving, and manipulating raster images. The [Metafile](#) class expands on the capabilities of the [Image](#) class by providing additional methods for recording and examining vector images.

The following topics cover the [Image](#), [Bitmap](#), and [Metafile](#) classes in more detail:

- [Loading and Displaying Bitmaps](#)
- [Loading and Displaying Metafiles](#)
- [Recording Metafiles](#)
- [Cropping and Scaling Images](#)
- [Rotating, Reflecting, and Skewing Images](#)
- [Using Interpolation Mode to Control Image Quality During Scaling](#)
- [Creating Thumbnail Images](#)
- [Using a Cached Bitmap to Improve Performance](#)
- [Improving Performance by Avoiding Automatic Scaling](#)
- [Reading and Writing Metadata](#)

Loading and Displaying Bitmaps

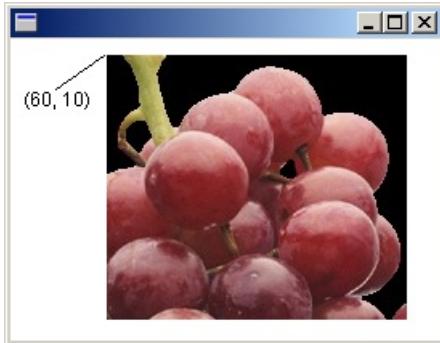
2/22/2020 • 2 minutes to read • [Edit Online](#)

To display a raster image (bitmap) on the screen, you need an **Image** object and a **Graphics** object. Pass the name of a file (or a pointer to a stream) to an **Image** constructor. After you have created an **Image** object, pass the address of that **Image** object to the **DrawImage** method of a **Graphics** object.

The following example creates an **Image** object from a JPEG file and then draws the image with its upper-left corner at (60, 10):

```
Image image(L"Grapes.jpg");
graphics.DrawImage(&image, 60, 10);
```

The following illustration shows the image drawn at the specified location.



The **Image** class provides basic methods for loading and displaying raster images and vector images. The **Bitmap** class, which inherits from the **Image** class, provides more specialized methods for loading, displaying, and manipulating raster images. For example, you can construct a **Bitmap** object from an icon handle (HICON).

The following example obtains a handle to an icon and then uses that handle to construct a **Bitmap** object. The code displays the icon by passing the address of the **Bitmap** object to the **DrawImage** method of a **Graphics** object.

```
HICON hIcon = LoadIcon(NULL, IDI_APPLICATION);
Bitmap bitmap(hIcon);
graphics.DrawImage(&bitmap, 10, 10);
```

Loading and Displaying Metafiles

2/22/2020 • 2 minutes to read • [Edit Online](#)

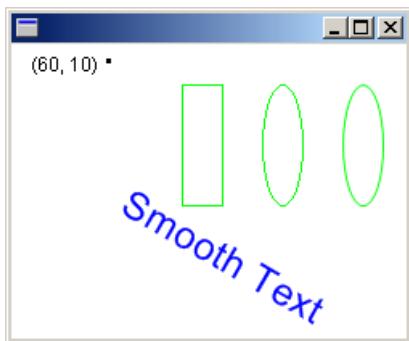
The **Image** class provides basic methods for loading and displaying raster images and vector images. The **Metafile** class, which inherits from the **Image** class, provides more specialized methods for recording, displaying, and examining vector images.

To display a vector image (metafile) on the screen, you need an **Image** object and a **Graphics** object. Pass the name of a file (or a pointer to a stream) to an **Image** constructor. After you have created an **Image** object, pass the address of that **Image** object to the **DrawImage** method of a **Graphics** object.

The following example creates an **Image** object from an EMF (enhanced metafile) file and then draws the image with its upper-left corner at (60, 10):

```
Image image(L"SampleMetafile.emf");
graphics.DrawImage(&image, 60, 10);
```

The following illustration shows the image drawn at the specified location.



Recording Metafiles

2/22/2020 • 2 minutes to read • [Edit Online](#)

The **Metafile** class, which inherits from the **Image** class, allows you to record a sequence of drawing commands. The recorded commands can be stored in memory, saved to a file, or saved to a stream. Metafiles can contain vector graphics, raster images, and text.

The following example creates a **Metafile** object. The code uses the **Metafile** object to record a sequence of graphics commands and then saves the recorded commands in a file named SampleMetafile.emf. Note that the **Metafile** constructor receives a device context handle, and the **Graphics** constructor receives the address of the **Metafile** object. The recording stops (and the recorded commands are saved to the file) when the **Graphics** object goes out of scope. The last two lines of code display the metafile by creating a new **Graphics** object and passing the address of the **Metafile** object to the **DrawImage** method of that **Graphics** object. Note that the code uses the same **Metafile** object to record and to display (play back) the metafile.

```
Metafile metafile(L"SampleMetafile.emf", hdc);
{
    Graphics graphics(&metafile);
    Pen greenPen(Color(255, 0, 255, 0));
    SolidBrush solidBrush(Color(255, 0, 0, 255));

    // Add a rectangle and an ellipse to the metafile.
    graphics.DrawRectangle(&greenPen, Rect(50, 10, 25, 75));
    graphics.DrawEllipse(&greenPen, Rect(100, 10, 25, 75));

    // Add an ellipse (drawn with antialiasing) to the metafile.
    graphics.SetSmoothingMode(SmoothingModeHighQuality);
    graphics.DrawEllipse(&greenPen, Rect(150, 10, 25, 75));

    // Add some text (drawn with antialiasing) to the metafile.
    FontFamily fontFamily(L"Arial");
    Font font(&fontFamily, 24, FontStyleRegular, UnitPixel);

    graphics.SetTextRenderingHint(TextRenderingHintAntiAlias);
    graphics.RotateTransform(30.0f);
    graphics.DrawString(L"Smooth Text", 11, &font,
        PointF(50.0f, 50.0f), &solidBrush);
} // End of recording metafile.

// Play back the metafile.
Graphics playbackGraphics(hdc);
playbackGraphics.DrawImage(&metafile, 200, 100);
```

NOTE

To record a metafile, you must construct a **Graphics** object based on a **Metafile** object. The recording of the metafile ends when that **Graphics** object is deleted or goes out of scope.

A metafile contains its own graphics state, which is defined by the **Graphics** object used to record the metafile. Any properties of the **Graphics** object (clip region, world transformation, smoothing mode, and the like) that you set while recording the metafile will be stored in the metafile. When you display the metafile, the drawing will be done according to those stored properties.

In the following example, assume that the smoothing mode was set to SmoothingModeNormal during the recording of the metafile. Even though the smoothing mode of the **Graphics** object used for playback is set to SmoothingModeHighQuality, the metafile will be played according to the SmoothingModeNormal setting. It is the smoothing mode set during the recording that is important, not the smoothing mode set prior to playback.

```
graphics.SetSmoothingMode(SmoothingModeHighQuality);
graphics.DrawImage(&meta, 0, 0);
```

Cropping and scaling GDI+ images

2/22/2020 • 2 minutes to read • [Edit Online](#)

The **Graphics** class provides several **DrawImage** methods, some of which have source and destination rectangle parameters that you can use to crop and scale images.

The following example constructs an **Image** object from the file Apple.gif. The code draws the entire apple image in its original size. The code then calls the **DrawImage** method of a **Graphics** object to draw a portion of the apple image in a destination rectangle that is larger than the original apple image.

The **DrawImage** method determines which portion of the apple to draw by looking at the source rectangle, which is specified by the third, fourth, fifth, and sixth arguments. In this case, the apple is cropped to 75 percent of its width and 75 percent of its height.

The **DrawImage** method determines where to draw the cropped apple and how big to make the cropped apple by looking at the destination rectangle, which is specified by the second argument. In this case, the destination rectangle is 30 percent wider and 30 percent taller than the original image.

```
Image image(L"Apple.gif");
UINT width = image.GetWidth();
UINT height = image.GetHeight();
// Make the destination rectangle 30 percent wider and
// 30 percent taller than the original image.
// Put the upper-left corner of the destination
// rectangle at (150, 20).
Rect destinationRect(150, 20, 1.3 * width, 1.3 * height);
// Draw the image unaltered with its upper-left corner at (0, 0).
graphics.DrawImage(&image, 0, 0);
// Draw a portion of the image. Scale that portion of the image
// so that it fills the destination rectangle.
graphics.DrawImage(
    &image,
    destinationRect,
    0, 0,           // upper-left corner of source rectangle
    0.75 * width,   // width of source rectangle
    0.75 * height,  // height of source rectangle
    UnitPixel);
```

The following illustration shows the original apple and the scaled, cropped apple.



Rotating, Reflecting, and Skewing Images

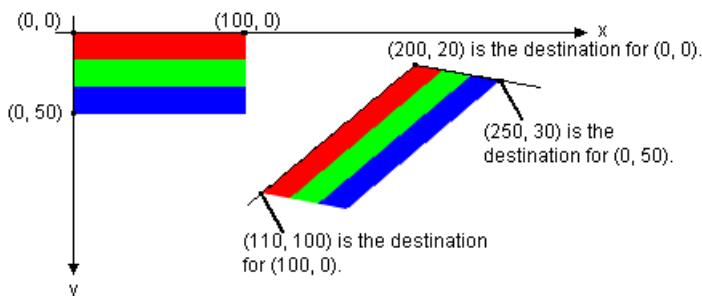
2/22/2020 • 2 minutes to read • [Edit Online](#)

You can rotate, reflect, and skew an image by specifying destination points for the upper-left, upper-right, and lower-left corners of the original image. The three destination points determine an affine transformation that maps the original rectangular image to a parallelogram. (The lower-right corner of the original image is mapped to the fourth corner of the parallelogram, which is calculated from the three specified destination points.)

For example, suppose the original image is a rectangle with upper-left corner at (0, 0), upper-right corner at (100, 0), and lower-left corner at (0, 50). Now suppose we map those three points to destination points as follows.

ORIGINAL POINT	DESTINATION POINT
Upper-left (0, 0)	(200, 20)
Upper-right (100, 0)	(110, 100)
Lower-left (0, 50)	(250, 30)

The following illustration shows the original image and the image mapped to the parallelogram. The original image has been skewed, reflected, rotated, and translated. The x-axis along the top edge of the original image is mapped to the line that runs through (200, 20) and (110, 100). The y-axis along the left edge of the original image is mapped to the line that runs through (200, 20) and (250, 30).



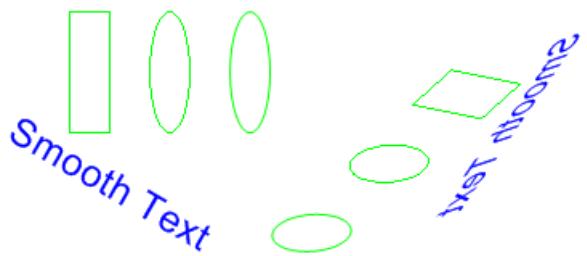
The following example produces the images shown in the preceding illustration.

```
Point destinationPoints[] = {  
    Point(200, 20), // destination for upper-left point of original  
    Point(110, 100), // destination for upper-right point of original  
    Point(250, 30)}; // destination for lower-left point of original  
  
Image image(L"Stripes.bmp");  
// Draw the image unaltered with its upper-left corner at (0, 0).  
graphics.DrawImage(&image, 0, 0);  
// Draw the image mapped to the parallelogram.  
graphics.DrawImage(&image, destinationPoints, 3);
```

The following illustration shows a similar transformation applied to a photographic image.



The following illustration shows a similar transformation applied to a metafile.



Using Interpolation Mode to Control Image Quality During Scaling

2/22/2020 • 2 minutes to read • [Edit Online](#)

The interpolation mode of a [Graphics](#) object influences the way Windows GDI+ scales (stretches and shrinks) images. The [InterpolationMode](#) enumeration in `Gdiplusenums.h` defines several interpolation modes, some of which are shown in the following list:

- `InterpolationModeNearestNeighbor`
- `InterpolationModeBilinear`
- `InterpolationModeHighQualityBilinear`
- `InterpolationModeBicubic`
- `InterpolationModeHighQualityBicubic`

To stretch an image, each pixel in the original image must be mapped to a group of pixels in the larger image. To shrink an image, groups of pixels in the original image must be mapped to single pixels in the smaller image. The effectiveness of the algorithms that perform these mappings determines the quality of a scaled image. Algorithms that produce higher-quality scaled images tend to require more processing time. In the preceding list, `InterpolationModeNearestNeighbor` is the lowest-quality mode and `InterpolationModeHighQualityBicubic` is the highest-quality mode.

To set the interpolation mode, pass one of the members of the [InterpolationMode](#) enumeration to the `SetInterpolationMode` method of a [Graphics](#) object.

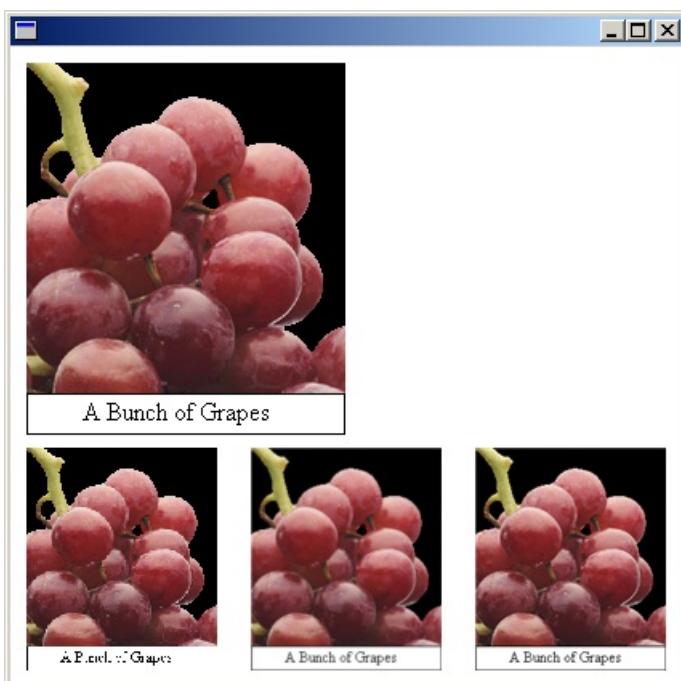
The following example draws an image and then shrinks the image with three different interpolation modes:

```

Image image(L"GrapeBunch.bmp");
UINT width = image.GetWidth();
UINT height = image.GetHeight();
// Draw the image with no shrinking or stretching.
graphics.DrawImage(
    &image,
    Rect(10, 10, width, height), // destination rectangle
    0, 0, // upper-left corner of source rectangle
    width, // width of source rectangle
    height, // height of source rectangle
    UnitPixel);
// Shrink the image using low-quality interpolation.
graphics.SetInterpolationMode(InterpolationModeNearestNeighbor);
graphics.DrawImage(
    &image,
    Rect(10, 250, 0.6*width, 0.6*height), // destination rectangle
    0, 0, // upper-left corner of source rectangle
    width, // width of source rectangle
    height, // height of source rectangle
    UnitPixel);
// Shrink the image using medium-quality interpolation.
graphics.SetInterpolationMode(InterpolationModeHighQualityBilinear);
graphics.DrawImage(
    &image,
    Rect(150, 250, 0.6 * width, 0.6 * height), // destination rectangle
    0, 0, // upper-left corner of source rectangle
    width, // width of source rectangle
    height, // height of source rectangle
    UnitPixel);
// Shrink the image using high-quality interpolation.
graphics.SetInterpolationMode(InterpolationModeHighQualityBicubic);
graphics.DrawImage(
    &image,
    Rect(290, 250, 0.6 * width, 0.6 * height), // destination rectangle
    0, 0, // upper-left corner of source rectangle
    width, // width of source rectangle
    height, // height of source rectangle
    UnitPixel);

```

The following illustration shows the original image and the three smaller images.



Creating Thumbnail Images

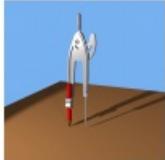
2/22/2020 • 2 minutes to read • [Edit Online](#)

A thumbnail image is a small version of an image. You can create a thumbnail image by calling the **GetThumbnailImage** method of an **Image** object.

The following example constructs an **Image** object from the file `Compass.bmp`. The original image has a width of 640 pixels and a height of 479 pixels. The code creates a thumbnail image that has a width of 100 pixels and a height of 100 pixels.

```
Image image(L"Compass.bmp");
Image* pThumbnail = image.GetThumbnailImage(100, 100, NULL, NULL);
graphics.DrawImage(pThumbnail, 10, 10,
    pThumbnail->GetWidth(), pThumbnail->GetHeight());
```

The following illustration shows the thumbnail image.



Using a Cached Bitmap to Improve Performance

2/22/2020 • 2 minutes to read • [Edit Online](#)

Image and **Bitmap** objects store images in a device-independent format. A **CachedBitmap** object stores an image in the format of the current display device. Rendering an image stored in a **CachedBitmap** object is fast because no processing time is spent converting the image to the format required by the display device.

The following example creates a **Bitmap** object and a **CachedBitmap** object from the file Texture.jpg. The **Bitmap** and the **CachedBitmap** are each drawn 30,000 times. If you run the code, you will see that the **CachedBitmap** images are drawn substantially faster than the **Bitmap** images.

```
Bitmap      bitmap(L"Texture.jpg");
UINT       width = bitmap.GetWidth();
UINT       height = bitmap.GetHeight();
CachedBitmap cBitmap(&bitmap, &graphics);
int        j, k;
for(j = 0; j < 300; j += 10)
    for(k = 0; k < 1000; ++k)
        graphics.DrawImage(&bitmap, j, j / 2, width, height);
for(j = 0; j < 300; j += 10)
    for(k = 0; k < 1000; ++k)
        graphics.DrawCachedBitmap(&cBitmap, j, 150 + j / 2 );
```

NOTE

A **CachedBitmap** object matches the format of the display device at the time the **CachedBitmap** object was constructed. If the user of your program changes the display settings, your code should construct a new **CachedBitmap** object. The **DrawImage** method will fail if you pass it a **CachedBitmap** object that was created prior to a change in the display format.

Improving Performance by Avoiding Automatic Scaling

2/22/2020 • 2 minutes to read • [Edit Online](#)

If you pass only the upper-left corner of an image to the **DrawImage** method, Windows GDI+ might scale the image, which would decrease performance.

The following call to the **DrawImage** method specifies an upper-left corner of (50, 30) but does not specify a destination rectangle:

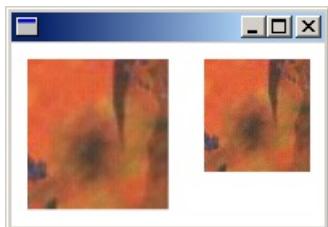
```
graphics.DrawImage(&image, 50, 30); // upper-left corner at (50, 30)
```

Although this is the easiest version of the **DrawImage** method in terms of the number of required arguments, it is not necessarily the most efficient. If the number of dots per inch on the current display device is different than the number of dots per inch on the device where the image was created, GDI+ scales the image so that its physical size on the current display device is as close as possible to its physical size on the device where it was created.

If you want to prevent such scaling, pass the width and height of a destination rectangle to the **DrawImage** method. The following example draws the same image twice. In the first case, the width and height of the destination rectangle are not specified, and the image is automatically scaled. In the second case, the width and height (measured in pixels) of the destination rectangle are specified to be the same as the width and height of the original image.

```
Image image(L"Texture.jpg");
graphics.DrawImage(&image, 10, 10);
graphics.DrawImage(&image, 120, 10, image.GetWidth(), image.GetHeight());
```

The following illustration shows the image rendered twice.



Reading and Writing Metadata

11/2/2020 • 6 minutes to read • [Edit Online](#)

Some image files contain metadata that you can read to determine features of the image. For example, a digital photograph might contain metadata that you can read to determine the make and model of the camera used to capture the image. With Windows GDI+, you can read existing metadata, and you can also write new metadata to image files.

GDI+ provides a uniform way of storing and retrieving metadata from image files in various formats. In GDI+, a piece of metadata is called a *property item*. You can store and retrieve metadata by calling the **SetPropertyItem** and **GetPropertyItem** methods of the **Image** class, and you don't have to be concerned about the details of how a particular file format stores that metadata.

GDI+ currently supports metadata for the TIFF, JPEG, Exif, and PNG file formats. The Exif format, which specifies how to store images captured by digital still cameras, is built on top of the TIFF and JPEG formats. Exif uses the TIFF format for uncompressed pixel data and the JPEG format for compressed pixel data.

GDI+ defines a set of property tags that identify property items. Certain tags are general-purpose; that is, they are supported by all of the file formats mentioned in the preceding paragraph. Other tags are special-purpose and apply only to certain formats. If you try to save a property item to a file that does not support that property item, GDI+ ignores the request. More specifically, the **Image::SetPropertyItem** method returns **PropertyNotSupported**.

You can determine the property items that are stored in an image file by calling **Image::GetPropertyIdList**. If you try to retrieve a property item that is not in the file, GDI+ ignores the request. More specifically, the **Image::GetPropertyItem** method returns **PropertyNotFound**.

Reading Metadata from a File

The following console application calls the **GetPropertySize** method of an **Image** object to determine how many pieces of metadata are in the file *FakePhoto.jpg*.

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;
INT main()
{
    // Initialize <tla rid="tla_gdiplus"/>.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
    UINT size = 0;
    UINT count = 0;
    Bitmap* bitmap = new Bitmap(L"FakePhoto.jpg");
    bitmap->GetPropertySize(&size, &count);
    printf("There are %d pieces of metadata in the file.\n", count);
    printf("The total size of the metadata is %d bytes.\n", size);

    delete bitmap;
    GdiplusShutdown(gdiplusToken);
    return 0;
}
```

The preceding code, along with a particular file, *FakePhoto.jpg*, produced the following output:

```
There are 7 pieces of metadata in the file.  
The total size of the metadata is 436 bytes.
```

GDI+ stores an individual piece of metadata in a [PropertyItem](#) object. You can call the [GetAllPropertyItems](#) method of the [Image](#) class to retrieve all the metadata from a file. The [GetAllPropertyItems](#) method returns an array of [PropertyItem](#) objects. Before you call [GetAllPropertyItems](#), you must allocate a buffer large enough to receive that array. You can call the [GetPropertySize](#) method of the [Image](#) class to get the size (in bytes) of the required buffer.

A [PropertyItem](#) object has the following four public members:

id	A tag that identifies the metadata item. The values that can be assigned to id (PropertyTagImageTitle , PropertyTagEquipMake , PropertyTagExifExposureTime , and the like) are defined in Gdipplusimaging.h .
length	The length, in bytes, of the array of values pointed to by the value data member. Note that if the type data member is set to PropertyTagTypeASCII , then the length data member is the length of a null-terminated character string, including the NULL terminator.
type	The data type of the values in the array pointed to by the value data member. Constants (PropertyTagTypeByte , PropertyTagTypeASCII , and the like) that represent various data types are described in Image Property Tag Type Constants .
value	A pointer to an array of values.

The following console application reads and displays the seven pieces of metadata in the file `FakePhoto.jpg`. The main function relies on the helper function `.PropertyTypeFromWORD`, which is shown following the main function.

```
#include <windows.h>  
#include <gdiplus.h>  
#include <strsafe.h>  
using namespace Gdiplus;  
  
INT main()  
{  
    // Initialize GDI+  
    GdiplusStartupInput gdiplusStartupInput;  
    ULONG_PTR gdiplusToken;  
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);  
  
    UINT size = 0;  
    UINT count = 0;  
  
    #define MAX_PROPRTYPE_SIZE 30  
    WCHAR str.PropertyType[MAX_PROPRTYPE_SIZE] = L"";  
  
    Bitmap* bitmap = new Bitmap(L"FakePhoto.jpg");  
  
    bitmap->GetPropertySize(&size, &count);  
    printf("There are %d pieces of metadata in the file.\n\n", count);  
  
    // GetAllPropertyItems returns an array of PropertyItem objects.
```

```

// Allocate a buffer large enough to receive that array.
PropertyItem* pPropBuffer = (PropertyItem*)malloc(size);

// Get the array of PropertyItem objects.
bitmap->GetAllPropertyItems(size, count, pPropBuffer);

// For each PropertyItem in the array, display the id, type, and length.
for(UINT j = 0; j < count; ++j)
{
    // Convert the property type from a WORD to a string.
    PropertyTypeFromWORD(
        pPropBuffer[j].type, str.PropertyType, MAX_PROP_TYPE_SIZE);

    printf("Property Item %d\n", j);
    printf(" id: 0x%04X\n", pPropBuffer[j].id);
    wprintf(L" type: %s\n", str.PropertyType);
    printf(" length: %d bytes\n\n", pPropBuffer[j].length);
}

free(pPropBuffer);
delete bitmap;
GdiplusShutdown(gdipplusToken);
return 0;
} // main

// Helper function
HRESULT PropertyTypeFromWORD(WORD index, WCHAR* string, UINT maxChars)
{
    HRESULT hr = E_FAIL;

    WCHAR* propertyTypes[] = {
        L"Nothing",           // 0
        L"PropertyTagTypeByte", // 1
        L"PropertyTagTypeASCII", // 2
        L"PropertyTagTypeShort", // 3
        L"PropertyTagTypeLong", // 4
        L"PropertyTagTypeRational", // 5
        L"Nothing",           // 6
        L"PropertyTagTypeUndefined", // 7
        L"Nothing",           // 8
        L"PropertyTagTypeSLONG", // 9
        L"PropertyTagTypeSRational"}; // 10

    hr = StringCchCopyW(string, maxChars, propertyTypes[index]);
    return hr;
}

```

The preceding console application produces the following output:

```

Property Item 0
  id: 0x320
  type: PropertyTagTypeASCII
  length: 16 bytes
Property Item 1
  id: 0x10f
  type: PropertyTagTypeASCII
  length: 17 bytes
Property Item 2
  id: 0x110
  type: PropertyTagTypeASCII
  length: 7 bytes
Property Item 3
  id: 0x9003
  type: PropertyTagTypeASCII
  length: 20 bytes
Property Item 4
  id: 0x829a
  type: PropertyTagTypeRational
  length: 8 bytes
Property Item 5
  id: 0x5090
  type: PropertyTagTypeShort
  length: 128 bytes
Property Item 6
  id: 0x5091
  type: PropertyTagTypeShort
  length: 128 bytes

```

The preceding output shows a hexadecimal ID number for each property item. You can look up those ID numbers in [Image Property Tag Constants](#) and find out that they represent the following property tags.

HEXADECIMAL VALUE	PROPERTY TAG
0x0320	PropertyTagImageTitle
0x010f	PropertyTagEquipMake
0x0110	PropertyTagEquipModel
0x9003	PropertyTagExifDTOriginal
0x829a	PropertyTagExifExposureTime
0x5090	PropertyTagLuminanceTable
0x5091	PropertyTagChrominanceTable

The second (index 1) property item in the list has **id** `PropertyTagEquipMake` and **type** `PropertyTagTypeASCII`. The following example, which is a continuation of the previous console application, displays the value of that property item:

```
printf("The equipment make is %s.\n", pPropBuffer[1].value);
```

The preceding line of code produces the following output:

```
The equipment make is Northwind Traders.
```

The fifth (index 4) property item in the list has **id** `PropertyTagExifExposureTime` and **type** `PropertyTagTypeRational`. That data type (`PropertyTagTypeRational`) is a pair of **LONGs**. The following example, which is a continuation of the previous console application, displays those two **LONG** values as a fraction. That fraction, 1/125, is the exposure time measured in seconds.

```
long* ptrLong = (long*)(pPropBuffer[4].value);
printf("The exposure time is %d/%d.\n", ptrLong[0], ptrLong[1]);
```

The preceding code produces the following output:

```
The exposure time is 1/125.
```

Writing Metadata to a File

To write an item of metadata to an [Image](#) object, initialize a [PropertyItem](#) object and then pass the address of that [PropertyItem](#) object to the [SetPropertyItem](#) method of the [Image](#) object.

The following console application writes one item (the image title) of metadata to an [Image](#) object and then saves the image in the disk file FakePhoto2.jpg. The main function relies on the helper function [GetEncoderClid](#), which is shown in the topic [Retrieving the Class Identifier for an Encoder](#).

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;
INT main()
{
    // Initialize <tla rid="tla_gdiplus"/>.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
    Status stat;
    CLSID clsid;
    char propertyValue[] = "Fake Photograph";
    Bitmap* bitmap = new Bitmap(L"FakePhoto.jpg");
    PropertyItem* propertyItem = new PropertyItem;
    // Get the CLSID of the JPEG encoder.
    GetEncoderClid(L"image/jpeg", &clsid);
    propertyItem->id = PropertyTagImageTitle;
    propertyItem->length = 16; // string length including NULL terminator
    propertyItem->type = PropertyTagTypeASCII;
    propertyItem->value = propertyValue;
    bitmap-> SetPropertyItem(propertyItem);
    stat = bitmap->Save(L"FakePhoto2.jpg", &clsid, NULL);
    if(stat == Ok)
        printf("FakePhoto2.jpg saved successfully.\n");

    delete propertyItem;
    delete bitmap;
    GdiplusShutdown(gdiplusToken);
    return 0;
}
```

Using Image Encoders and Decoders

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides the [Image](#) class and the [Bitmap](#) class for storing images in memory and manipulating images in memory. GDI+ writes images to disk files with the help of image encoders and loads images from disk files with the help of image decoders. An encoder translates the data in an [Image](#) or [Bitmap](#) object into a designated disk file format. A decoder translates the data in a disk file to the format required by the [Image](#) and [Bitmap](#) objects. GDI+ has built-in encoders and decoders that support the following file types:

- BMP
- GIF
- JPEG
- PNG
- TIFF

GDI+ also has built-in decoders that support the following file types:

- WMF
- EMF
- ICON

The following topics discuss encoders and decoders in more detail:

- [Listing Installed Encoders](#)
- [Listing Installed Decoders](#)
- [Retrieving the Class Identifier for an Encoder](#)
- [Determining the Parameters Supported by an Encoder](#)
- [Converting a BMP Image to a PNG Image](#)
- [Setting JPEG Compression Level](#)
- [Transforming a JPEG Image Without Loss of Information](#)
- [Creating and Saving a Multiple-Frame Image](#)
- [Copying Individual Frames from a Multiple-Frame Image](#)

Listing Installed Encoders

11/2/2020 • 2 minutes to read • [Edit Online](#)

GDI+ provides the [GetImageEncoders](#) function so that you can determine which image encoders are available on your computer. [GetImageEncoders](#) returns an array of [ImageCodecInfo](#) objects. Before you call [GetImageEncoders](#), you must allocate a buffer large enough to receive that array. You can call [GetImageEncodersSize](#) to determine the size of the required buffer.

The following console application lists the available image encoders:

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    UINT num;          // number of image encoders
    UINT size;         // size, in bytes, of the image encoder array

    ImageCodecInfo* pImageCodecInfo;

    // How many encoders are there?
    // How big (in bytes) is the array of all ImageCodecInfo objects?
    GetImageEncodersSize(&num, &size);

    // Create a buffer large enough to hold the array of ImageCodecInfo
    // objects that will be returned by GetImageEncoders.
    pImageCodecInfo = (ImageCodecInfo*)(malloc(size));

    // GetImageEncoders creates an array of ImageCodecInfo objects
    // and copies that array into a previously allocated buffer.
    // The third argument, imageCodecInfo, is a pointer to that buffer.
    GetImageEncoders(num, size, pImageCodecInfo);

    // Display the graphics file format (MimeType)
    // for each ImageCodecInfo object.
    for(UINT j = 0; j < num; ++j)
    {
        wprintf(L"%s\n", pImageCodecInfo[j].MimeType);
    }

    free(pImageCodecInfo);
    GdiplusShutdown(gdiplusToken);
    return 0;
}
```

When you run the preceding console application, the output will be similar to the following:

image/bmp

image/jpeg

image/gif

image/tiff

image/png

Listing Installed Decoders

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides the [GetImageDecoders](#) function so that you can determine which image decoders are available on your computer. [GetImageDecoders](#) returns an array of [ImageCodecInfo](#) objects. Before you call [GetImageDecoders](#), you must allocate a buffer large enough to receive that array. You can call [GetImageDecodersSize](#) to determine the size of the required buffer.

The following console application lists the available image decoders:

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    UINT num;          // number of image decoders
    UINT size;         // size, in bytes, of the image decoder array

    ImageCodecInfo* pImageCodecInfo;

    // How many decoders are there?
    // How big (in bytes) is the array of all ImageCodecInfo objects?
    GetImageDecodersSize(&num, &size);

    // Create a buffer large enough to hold the array of ImageCodecInfo
    // objects that will be returned by GetImageDecoders.
    pImageCodecInfo = (ImageCodecInfo*)(malloc(size));

    // GetImageDecoders creates an array of ImageCodecInfo objects
    // and copies that array into a previously allocated buffer.
    // The third argument, imageCodecInfo, is a pointer to that buffer.
    GetImageDecoders(num, size, pImageCodecInfo);

    // Display the graphics file format (MimeType)
    // for each ImageCodecInfo object.
    for(UINT j = 0; j < num; ++j)
    {
        wprintf(L"%s\n", pImageCodecInfo[j].MimeType);
    }

    free(pImageCodecInfo);
    GdiplusShutdown(gdiplusToken);
    return 0;
}
```

When you run the preceding console application, the output will be similar to the following:

image/bmp
image/jpeg
image/gif
image/x-emf
image/x-wmf
image/tiff
image/png
image/x-icon

Retrieving the Class Identifier for an Encoder

11/2/2020 • 2 minutes to read • [Edit Online](#)

The function `GetEncoderClid` in the following example receives the MIME type of an encoder and returns the class identifier (`CLSID`) of that encoder. The MIME types of the encoders built into Windows GDI+ are as follows:

- `image/bmp`
- `image/jpeg`
- `image/gif`
- `image/tiff`
- `image/png`

The function calls `GetImageEncoders` to get an array of `ImageCodecInfo` objects. If one of the `ImageCodecInfo` objects in that array represents the requested encoder, the function returns the index of the `ImageCodecInfo` object and copies the `CLSID` into the variable pointed to by `pClid`. If the function fails, it returns `-1`.

```
int GetEncoderClid(const WCHAR* format, CLSID* pClid)
{
    UINT num = 0;           // number of image encoders
    UINT size = 0;          // size of the image encoder array in bytes

    ImageCodecInfo* pImageCodecInfo = NULL;

    GetImageEncodersSize(&num, &size);
    if(size == 0)
        return -1; // Failure

    pImageCodecInfo = (ImageCodecInfo*)(malloc(size));
    if(pImageCodecInfo == NULL)
        return -1; // Failure

    GetImageEncoders(num, size, pImageCodecInfo);

    for(UINT j = 0; j < num; ++j)
    {
        if( wcscmp(pImageCodecInfo[j].MimeType, format) == 0 )
        {
            *pClid = pImageCodecInfo[j].Clid;
            free(pImageCodecInfo);
            return j; // Success
        }
    }

    free(pImageCodecInfo);
    return -1; // Failure
}
```

The following console application calls the `GetEncoderClid` function to determine the `CLSID` of the PNG encoder:

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

#include "GdiplusHelperFunctions.h"

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    CLSID encoderClsid;
    INT result;
    WCHAR strGuid[39];

    result = GetEncoderClsid(L"image/png", &encoderClsid);

    if(result < 0)
    {
        printf("The PNG encoder is not installed.\n");
    }
    else
    {
        StringFromGUID2(encoderClsid, strGuid, 39);
        printf("An ImageCodecInfo object representing the PNG encoder\n");
        printf("was found at position %d in the array.\n", result);
        wprintf(L"The CLSID of the PNG encoder is %s.\n", strGuid);
    }

    GdiplusShutdown(gdiplusToken);
    return 0;
}
```

When you run the preceding console application, you get an output similar to the following:

```
An ImageCodecInfo object representing the PNG encoder
was found at position 4 in the array.
The CLSID of the PNG encoder is {557CF406-1A04-11D3-9A73-0000F81EF32E}.
```

Determining the Parameters Supported by an Encoder

11/2/2020 • 3 minutes to read • [Edit Online](#)

The [Image](#) class provides the [Image::GetEncoderParameterList](#) method so that you can determine the parameters that are supported by a given image encoder. The [Image::GetEncoderParameterList](#) method returns an array of [EncoderParameter](#) objects. You must allocate a buffer to receive that array before you call [Image::GetEncoderParameterList](#). You can call [Image::GetEncoderParameterListSize](#) to determine the size of the required buffer.

The following console application obtains the parameter list for the JPEG encoder. The main function relies on the helper function `GetEncoderClid`, which is shown in [Retrieving the Class Identifier for an Encoder](#).

```

#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT GetEncoderClsid(const WCHAR* format, CLSID* pClsid); // helper function

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    // Create Bitmap (inherited from Image) object so that we can call
    // GetParameterListSize and GetParameterList.
    Bitmap* bitmap = new Bitmap(1, 1);

    // Get the JPEG encoder CLSID.
    CLSID encoderClsid;
    GetEncoderClsid(L"image/jpeg", &encoderClsid);

    // How big (in bytes) is the JPEG encoder's parameter list?
    UINT listSize = 0;
    listSize = bitmap->GetEncoderParameterListSize(&encoderClsid);
    printf("The parameter list requires %d bytes.\n", listSize);

    // Allocate a buffer large enough to hold the parameter list.
    EncoderParameters* pEncoderParameters = NULL;
    pEncoderParameters = (EncoderParameters*)malloc(listSize);

    // Get the parameter list for the JPEG encoder.
    bitmap->GetEncoderParameterList(
        &encoderClsid, listSize, pEncoderParameters);

    // pEncoderParameters points to an EncoderParameters object, which
    // has a Count member and an array of EncoderParameter objects.
    // How many EncoderParameter objects are in the array?
    printf("There are %d EncoderParameter objects in the array.\n",
        pEncoderParameters->Count);

    free(pEncoderParameters);
    delete(bitmap);
    GdiplusShutdown(gdiplusToken);
    return 0;
}

```

When you run the preceding console application, you get an output similar to the following:

```

The parameter list requires 172 bytes.
There are 4 EncoderParameter objects in the array.

```

Each of the **EncoderParameter** objects in the array has the following four public data members:

The following code is a continuation of the console application shown in the preceding example. The code looks at the second **EncoderParameter** object in the array returned by **Image::GetEncoderParameterList**. The code calls **StringFromGUID2**, which is a system function declared in Objbase.h, to convert the Guid member of the **EncoderParameter** object to a string.

```

// Look at the second (index 1) EncoderParameter object in the array.
printf("Parameter[1]\n");

WCHAR strGuid[39];
StringFromGUID2(pEncoderParameters->Parameter[1].Guid, strGuid, 39);
wprintf(L"    The GUID is %s.\n", strGuid);

printf("    The value type is %d.\n",
    pEncoderParameters->Parameter[1].Type);

printf("    The number of values is %d.\n",
    pEncoderParameters->Parameter[1].NumberOfValues);

```

The preceding code produces the following output:

```

Parameter[1]
The GUID is {1D5BE4B5-FA4A-452D-9CDD-5DB35105E7EB}.
The value type is 6.
The number of values is 1.

```

You can look up the GUID in Gdiplusimaging.h and find out that the category of this [EncoderParameter](#) object is EncoderQuality. You can use this category (EncoderQuality) of parameter to set the compression level of a JPEG image.

In Gdiplusenums.h, the [EncoderParameterValueType](#) enumeration indicates that data type 6 is **ValueLongRange**. A long range is a pair of ULONG values.

The number of values is one, so we know that the **Value** member of the [EncoderParameter](#) object is a pointer to an array that has one element. That one element is a pair of ULONG values.

The following code is a continuation of the console application that is shown in the preceding two examples. The code defines a data type called **PLONGRANGE** (pointer to a long range). A variable of type **PLONGRANGE** is used to extract the minimum and maximum values that can be passed as a quality setting to the JPEG encoder.

```

typedef struct
{
    long min;
    long max;
}* PLONGRANGE;

PLONGRANGE pLongRange =
    (PLONGRANGE)(pEncoderParameters->Parameter[1].Value);

printf("    The minimum possible quality value is %d.\n",
    pLongRange->min);

printf("    The maximum possible quality value is %d.\n",
    pLongRange->max);

```

The preceding code produces the following output:

```

The minimum possible quality value is 0.
The maximum possible quality value is 100.

```

In the preceding example, the value returned in the [EncoderParameter](#) object is a pair of ULONG values that indicate the minimum and maximum possible values for the quality parameter. In some cases, the values returned in an [EncoderParameter](#) object are members of the [EncoderValue](#) enumeration. The following topics discuss the [EncoderValue](#) enumeration and methods for listing possible parameter values in more detail:

- [Using the EncoderValue Enumeration](#)
- [Listing Parameters and Values for All Encoders](#)

Using the EncoderValue Enumeration

11/2/2020 • 3 minutes to read • [Edit Online](#)

A given encoder supports certain parameter categories, and for each of those categories, that encoder allows certain values. For example, the JPEG encoder supports the EncoderValueQuality parameter category, and the allowable parameter values are the integers 0 through 100. Some of the allowable parameter values are the same across several encoders. These standard values are defined in the [EncoderValue](#) enumeration in Gdiplusenums.h:

```
enum EncoderValue
{
    EncoderValueColorTypeCMYK,           // 0
    EncoderValueColorTypeYCKC,          // 1
    EncoderValueCompressionLZW,         // 2
    EncoderValueCompressionCCITT3,      // 3
    EncoderValueCompressionCCITT4,      // 4
    EncoderValueCompressionRle,         // 5
    EncoderValueCompressionNone,        // 6
    EncoderValueScanMethodInterlaced,   // 7
    EncoderValueScanMethodNonInterlaced, // 8
    EncoderValueVersionGif87,           // 9
    EncoderValueVersionGif89,           // 10
    EncoderValueRenderProgressive,      // 11
    EncoderValueRenderNonProgressive,   // 12
    EncoderValueTransformRotate90,       // 13
    EncoderValueTransformRotate180,      // 14
    EncoderValueTransformRotate270,      // 15
    EncoderValueTransformFlipHorizontal, // 16
    EncoderValueTransformFlipVertical,   // 17
    EncoderValueMultiFrame,             // 18
    EncoderValueLastFrame,              // 19
    EncoderValueFlush,                 // 20
    EncoderValueFrameDimensionTime,     // 21
    EncoderValueFrameDimensionResolution, // 22
    EncoderValueFrameDimensionPage,     // 23
};
```

One of the parameter categories supported by the JPEG encoder is the EncoderTransformation category. By examining the [EncoderValue](#) enumeration, you might speculate (and you would be correct) that the EncoderTransformation category allows the following five values:

```
EncoderValueTransformRotate90,        // 13
EncoderValueTransformRotate180,        // 14
EncoderValueTransformRotate270,        // 15
EncoderValueTransformFlipHorizontal,   // 16
EncoderValueTransformFlipVertical,     // 17
```

The following console application verifies that the JPEG encoder supports the EncoderTransformation parameter category and that there are five allowable values for such a parameter. The main function relies on the helper function GetEncoderClsid, which is shown in [Retrieving the Class Identifier for an Encoder](#).

```

#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;
INT GetEncoderClsid(const WCHAR* format, CLSID* pClsid);
INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
    // Create a Bitmap (inherited from Image) object so that we can call
    // GetParameterListSize and GetParameterList.
    Bitmap* bitmap = new Bitmap(1, 1);
    // Get the JPEG encoder CLSID.
    CLSID encoderClsid;
    GetEncoderClsid(L"image/jpeg", &encoderClsid);
    // How big (in bytes) is the JPEG encoder's parameter list?
    UINT listSize = 0;
    listSize = bitmap->GetEncoderParameterListSize(&encoderClsid);
    printf("The parameter list requires %d bytes.\n", listSize);
    // Allocate a buffer large enough to hold the parameter list.
    EncoderParameters* pEncoderParameters = NULL;
    pEncoderParameters = (EncoderParameters*)malloc(listSize);
    // Get the parameter list for the JPEG encoder.
    bitmap->GetEncoderParameterList(
        &encoderClsid, listSize, pEncoderParameters);
    // pEncoderParameters points to an EncoderParameters object, which
    // has a Count member and an array of EncoderParameter objects.
    // How many EncoderParameter objects are in the array?
    printf("There are %d EncoderParameter objects in the array.\n",
        pEncoderParameters->Count);
    // Look at the first (index 0) EncoderParameter object in the array.
    printf("Parameter[0]\n");
    WCHAR strGuid[39];
    StringFromGUID2(pEncoderParameters->Parameter[0].Guid, strGuid, 39);
    wprintf(L"    The guid is %s.\n", strGuid);
    printf("    The data type is %d.\n",
        pEncoderParameters->Parameter[0].Type);
    printf("    The number of values is %d.\n",
        pEncoderParameters->Parameter[0].NumberOfValues);
    free(pEncoderParameters);
    delete bitmap;
    GdiplusShutdown(gdiplusToken);
    return 0;
}

```

When you run the preceding console application, you get an output similar to the following:

```

The parameter list requires 172 bytes.
There are 4 EncoderParameter objects in the array.
Parameter[0]
    The GUID is {8D0EB2D1-A58E-4EA8-AA14-108074B7B6F9}.
    The value type is 4.
    The number of values is 5.

```

You can look up the GUID in Gdiplusimaging.h and find out that the category of this [EncoderParameter](#) object is EncoderTransformation. In Gdiplusenums.h, the [EncoderParameterValueType](#) enumeration indicates that data type 4 is ValueLong (32-bit unsigned integer). The number of values is five, so we know that the **Value** member of the **EncoderParameter** object is a pointer to an array of five **ULONG** values.

The following code is a continuation of the console application that is shown in the preceding example. The code lists the allowable values for an EncoderTransformation parameter:

```
ULONG* pUlong = (ULONG*)(pEncoderParameters->Parameter[0].Value);
ULONG numVals = pEncoderParameters->Parameter[0].NumberOfValues;
printf("%s", " The allowable values are");
for(ULONG j = 0; j < numVals; ++j)
{
    printf(" %d", pUlong[j]);
}
```

The preceding code produces the following output:

```
The allowable values are 13 14 15 16 17
```

The allowable values (13, 14, 15, 16, and 17) correspond to the following members of the [EncoderValue](#) enumeration:

```
EncoderValueTransformRotate90,      // 13
EncoderValueTransformRotate180,     // 14
EncoderValueTransformRotate270,     // 15
EncoderValueTransformFlipHorizontal, // 16
EncoderValueTransformFlipVertical,   // 17
```

Listing Parameters and Values for All Encoders

11/2/2020 • 5 minutes to read • [Edit Online](#)

The following console application lists all the parameters supported by the various encoders installed on the computer. The main function calls [GetImageEncoders](#) to discover which encoders are available. For each available encoder, the main function calls the helper function [ShowAllEncoderParameters](#).

The [ShowAllEncoderParameters](#) function calls the [Image::GetEncoderParameterList](#) method to discover which parameters are supported by a given encoder. For each supported parameter, the function lists the category, data type, and number of values. The [ShowAllEncoderParameters](#) function relies on two helper functions: [EncoderParameterCategoryFromGUID](#) and [ValueTypeFromULONG](#).

```
#include <windows.h>
#include <gdiplus.h>
#include <strsafe.h>
using namespace Gdiplus;

// Helper functions
void ShowAllEncoderParameters(ImageCodecInfo* );
HRESULT EncoderParameterCategoryFromGUID(GUID guid, WCHAR* category, UINT maxChars);
HRESULT ValueTypeFromULONG(ULONG index, WCHAR* strValueType, UINT maxChars);

INT main()
{
    // Initialize GDI+
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    UINT num;           // Number of image encoders
    UINT size;          // Size of the image encoder array in bytes

    ImageCodecInfo* pImageCodecInfo;

    // How many encoders are there?
    // How big (in bytes) is the array of all ImageCodecInfo objects?
    GetImageEncodersSize(&num, &size);

    // Create a buffer large enough to hold the array of ImageCodecInfo
    // objects that will be returned by GetImageEncoders.
    pImageCodecInfo = (ImageCodecInfo*)(malloc(size));

    // GetImageEncoders creates an array of ImageCodecInfo objects
    // and copies that array into a previously allocated buffer.
    // The third argument, imageCodecInfos, is a pointer to that buffer.
    GetImageEncoders(num, size, pImageCodecInfo);

    // For each ImageCodecInfo object in the array, show all parameters.
    for(UINT j = 0; j < num; ++j)
    {
        ShowAllEncoderParameters(&(pImageCodecInfo[j]));
    }

    GdiplusShutdown(gdiplusToken);
    return 0;
}

///////////////
// Helper functions
```

```

VOID ShowAllEncoderParameters(ImageCodecInfo* pImageCodecInfo)
{
    CONST MAX_CATEGORY_LENGTH = 50;
    CONST MAX_VALUE_TYPE_LENGTH = 50;
    WCHAR strParameterCategory[MAX_CATEGORY_LENGTH] = L"";;
    WCHAR strValueType[MAX_VALUE_TYPE_LENGTH] = L"";

    wprintf(L"\n\n%s\n", pImageCodecInfo->MimeType);

    // Create a Bitmap (inherited from Image) object so that we can call
    // GetParameterListSize and GetParameterList.
    Bitmap bitmap(1, 1);

    // How big (in bytes) is the encoder's parameter list?
    UINT listSize = 0;
    listSize = bitmap.GetEncoderParameterListSize(&pImageCodecInfo->Clid);
    printf(" The parameter list requires %d bytes.\n", listSize);

    if(listSize == 0)
        return;

    // Allocate a buffer large enough to hold the parameter list.
    EncoderParameters* pEncoderParameters = NULL;
    pEncoderParameters = (EncoderParameters*)malloc(listSize);

    if(pEncoderParameters == NULL)
        return;

    // Get the parameter list for the encoder.
    bitmap.GetEncoderParameterList(
        &pImageCodecInfo->Clid, listSize, pEncoderParameters);

    // pEncoderParameters points to an EncoderParameters object, which
    // has a Count member and an array of EncoderParameter objects.
    // How many EncoderParameter objects are in the array?
    printf(" There are %d EncoderParameter objects in the array.\n",
        pEncoderParameters->Count);

    // For each EncoderParameter object in the array, list the
    // parameter category, data type, and number of values.
    for(UINT k = 0; k < pEncoderParameters->Count; ++k)
    {
        EncoderParameterCategoryFromGUID(
            pEncoderParameters->Parameter[k].Guid, strParameterCategory, MAX_CATEGORY_LENGTH);

        ValueTypeFromULONG(
            pEncoderParameters->Parameter[k].Type, strValueType, MAX_VALUE_TYPE_LENGTH);

        printf(" Parameter[%d]\n", k);
        wprintf(L" The category is %s.\n", strParameterCategory);
        wprintf(L" The data type is %s.\n", strValueType);

        printf(" The number of values is %d.\n",
            pEncoderParameters->Parameter[k].NumberOfValues);
    } // for

    free(pEncoderParameters);
} // ShowAllEncoderParameters

HRESULT EncoderParameterCategoryFromGUID(GUID guid, WCHAR* category, UINT maxChars)
{
    HRESULT hr = E_FAIL;

    if(guid == EncoderCompression)
        hr = StringCchCopyW(category, maxChars, L"Compression");
    else if(guid == EncoderColorDepth)
        hr = StringCchCopyW(category, maxChars, L"ColorDepth");
}

```

```

else if(guid == EncoderScanMethod)
    hr = StringCchCopyW(category, maxChars, L"ScanMethod");
else if(guid == EncoderVersion)
    hr = StringCchCopyW(category, maxChars, L"Version");
else if(guid == EncoderRequestMethod)
    hr = StringCchCopyW(category, maxChars, L"RequestMethod");
else if(guid == EncoderQuality)
    hr = StringCchCopyW(category, maxChars, L"Quality");
else if(guid == EncoderTransformation)
    hr = StringCchCopyW(category, maxChars, L"Transformation");
else if(guid == EncoderLuminanceTable)
    hr = StringCchCopyW(category, maxChars, L"LuminanceTable");
else if(guid == EncoderChrominanceTable)
    hr = StringCchCopyW(category, maxChars, L"ChrominanceTable");
else if(guid == EncoderSaveFlag)
    hr = StringCchCopyW(category, maxChars, L"SaveFlag");
else
    hr = StringCchCopyW(category, maxChars, L"Unknown category");

return hr;
} // EncoderParameterCategoryFromGUID

```

```

HRESULT ValueTypeFromULONG(ULONG index, WCHAR* strValueType, UINT maxChars)
{
    HRESULT hr = E_FAIL;

    WCHAR* valueTypes[] = {
        L"Nothing",           // 0
        L"ValueTypeByte",     // 1
        L"ValueTypeASCII",    // 2
        L"ValueTypeShort",    // 3
        L"ValueTypeLong",     // 4
        L"ValueTypeRational", // 5
        L"ValueTypeLongRange",// 6
        L"ValueTypeUndefined",// 7
        L"ValueTypeRationalRange"}; // 8

    hr = StringCchCopyW(strValueType, maxChars, valueTypes[index]);
    return hr;
} // ValueTypeFromULONG

```

When you run the preceding console application, you get an output similar to the following:

```

image/bmp
The parameter list requires 0 bytes.

image/jpeg
The parameter list requires 172 bytes.
There are 4 EncoderParameter objects in the array.
Parameter[0]
The category is Transformation.
The data type is Long.
The number of values is 5.
Parameter[1]
The category is Quality.
The data type is LongRange.
The number of values is 1.
Parameter[2]
The category is LuminanceTable.
The data type is Short.
The number of values is 0.
Parameter[3]
The category is ChrominanceTable.
The data type is Short.
The number of values is 0.

image/gif
The parameter list requires 0 bytes.

image/tiff
The parameter list requires 160 bytes.
There are 3 EncoderParameter objects in the array.
Parameter[0]
The category is Compression.
The data type is Long.
The number of values is 5.
Parameter[1]
The category is ColorDepth.
The data type is Long.
The number of values is 5.
Parameter[2]
The category is SaveFlag.
The data type is Long.
The number of values is 1.

image/png
The parameter list requires 0 bytes.

```

You can draw the following conclusions by examining the preceding program output:

- The JPEG encoder supports the Transformation, Quality, LuminanceTable, and ChrominanceTable parameter categories.
- The TIFF encoder supports the Compression, ColorDepth, and SaveFlag parameter categories.

You can also see the number of acceptable values for each parameter category. For example, you can see that the ColorDepth parameter category (TIFF codec) has five values of type **ULONG**. The following code lists those five values. Assume that **pEncoderParameters** is a pointer to an **EncoderParameters** object that represents the TIFF encoder.

```

ULONG* pUlong = (ULONG*)(pEncoderParameters->Parameter[1].Value);
ULONG numVals = pEncoderParameters->Parameter[1].NumberOfValues;
printf("\nThe allowable values for ColorDepth are\n");

for(ULONG k = 0; k < numVals; ++k)
{
    printf(" %u\n", pUlong[k]);
}

```

The preceding code produces the following output:

```

The allowable values for ColorDepth are
1
4
8
24
32

```

NOTE

In some cases, the values in an **EncoderParameter** object are the numeric values of elements of the **EncoderValue** enumeration. However, the numbers in the preceding list do not relate to the **EncoderValue** enumeration. The numbers mean 1 bit per pixel, 2 bits per pixel, and so on.

If you write code similar to the preceding example to investigate the allowable values for the other parameter categories, you will obtain a result similar to the following.

JPEG ENCODER PARAMETER	ALLOWABLE VALUES
Transformation	EncoderValueTransformRotate90 EncoderValueTransformRotate180 EncoderValueTransformRotate270 EncoderValueTransformFlipHorizontal EncoderValueTransformFlipVertical
Quality	0 through 100

TIFF ENCODER PARAMETER	ALLOWABLE VALUES
Compression	EncoderValueCompressionLZW EncoderValueCompressionCCITT3 EncoderValueCompressionCCITT4 EncoderValueCompressionRle EncoderValueCompressionNone
ColorDepth	1, 4, 8, 24, 32
SaveFlag	EncoderValueMultiFrame

NOTE

If the width and height of a JPEG image are multiples of 16, you can apply any of the transformations allowed by the EncoderTransformation parameter category (for example, 90-degree rotation) without loss of information.

Converting a BMP Image to a PNG Image

11/2/2020 • 2 minutes to read • [Edit Online](#)

To save an image to a disk file, call the [Save](#) method of the [Image](#) class. The following console application loads a BMP image from a disk file, converts the image to the PNG format, and saves the converted image to a new disk file. The main function relies on the helper function [GetEncoderClid](#), which is shown in [Retrieving the Class Identifier for an Encoder](#).

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT GetEncoderClid(const WCHAR* format, CLSID* pClid); // helper function

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    CLSID encoderClid;
    Status stat;
    Image* image = new Image(L"Bird.bmp");

    // Get the CLSID of the PNG encoder.
    GetEncoderClid(L"image/png", &encoderClid);

    stat = image->Save(L"Bird.png", &encoderClid, NULL);

    if(stat == Ok)
        printf("Bird.png was saved successfully\n");
    else
        printf("Failure: stat = %d\n", stat);

    delete image;
    GdiplusShutdown(gdiplusToken);
    return 0;
}
```

Setting JPEG Compression Level

11/2/2020 • 2 minutes to read • [Edit Online](#)

To specify the compression level when you save a JPEG image, initialize an [EncoderParameters](#) object and pass the address of that object to the [Save](#) method of the [Image](#) class. Initialize the [EncoderParameters](#) object so that it has an array consisting of one [EncoderParameter](#) object. Initialize that one [EncoderParameter](#) object so that its [Value](#) member points to a [ULONG](#) value from 0 through 100. Set the [Guid](#) member of the [EncoderParameter](#) object to [EncoderQuality](#).

The following console application saves three JPEG images, each with a different quality level. A quality level of 0 corresponds to the greatest compression, and a quality level of 100 corresponds to the least compression.

The main function relies on the helper function [GetEncoderClid](#), which is shown in [Retrieving the Class Identifier for an Encoder](#):

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT GetEncoderClid(const WCHAR* format, CLSID* pClid); // helper function

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    CLSID encoderClid;
    EncoderParameters encoderParameters;
    ULONG quality;
    Status stat;

    // Get an image from the disk.
    Image* image = new Image(L"Shapes.bmp");

    // Get the CLSID of the JPEG encoder.
    GetEncoderClid(L"image/jpeg", &encoderClid);

    // Before we call Image::Save, we must initialize an
    // EncoderParameters object. The EncoderParameters object
    // has an array of EncoderParameter objects. In this
    // case, there is only one EncoderParameter object in the array.
    // The one EncoderParameter object has an array of values.
    // In this case, there is only one value (of type ULONG)
    // in the array. We will let this value vary from 0 to 100.

    encoderParameters.Count = 1;
    encoderParameters.Parameter[0].Guid = EncoderQuality;
    encoderParameters.Parameter[0].Type = EncoderParameterValueTypeLong;
    encoderParameters.Parameter[0].NumberOfValues = 1;

    // Save the image as a JPEG with quality level 0.
    quality = 0;
    encoderParameters.Parameter[0].Value = &quality;
    stat = image->Save(L"Shapes001.jpg", &encoderClid, &encoderParameters);

    if(stat == Ok)
        wprintf(L"%s saved successfully.\n", L"Shapes001.jpg");
}
```

```
else
    wprintf(L"%d Attempt to save %s failed.\n", stat, L"Shapes001.jpg");

// Save the image as a JPEG with quality level 50.
quality = 50;
encoderParameters.Parameter[0].Value = &quality;
stat = image->Save(L"Shapes050.jpg", &encoderClsid, &encoderParameters);

if(stat == Ok)
    wprintf(L"%s saved successfully.\n", L"Shapes050.jpg");
else
    wprintf(L"%d Attempt to save %s failed.\n", stat, L"Shapes050.jpg");

    // Save the image as a JPEG with quality level 100.
quality = 100;
encoderParameters.Parameter[0].Value = &quality;
stat = image->Save(L"Shapes100.jpg", &encoderClsid, &encoderParameters);

if(stat == Ok)
    wprintf(L"%s saved successfully.\n", L"Shapes100.jpg");
else
    wprintf(L"%d Attempt to save %s failed.\n", stat, L"Shapes100.jpg");

delete image;
GdiplusShutdown(gdiplusToken);
return 0;
}
```

Lossless transform of a JPEG image

11/2/2020 • 3 minutes to read • [Edit Online](#)

When you compress a JPEG image, some of the information in the image is lost. If you open a JPEG file, alter the image, and save it to another JPEG file, the quality will decrease. If you repeat that process many times, you will see a substantial degradation in the image quality.

Because JPEG is one of the most popular image formats on the Web, and because people often like to modify JPEG images, GDI+ provides the following transformations that can be performed on JPEG images without loss of information:

- Rotate 90 degrees
- Rotate 180 degrees
- Rotate 270 degrees
- Flip horizontally
- Flip vertically

You can apply one of the transformations shown in the preceding list when you call the [Save](#) method of an [Image](#) object. If the following conditions are met, then the transformation will proceed without loss of information:

- The file used to construct the [Image](#) object is a JPEG file.
- The width and height of the image are both multiples of 16.

If the width and height of the image are not both multiples of 16, GDI+ will do its best to preserve the image quality when you apply one of the rotation or flipping transformations shown in the preceding list.

To transform a JPEG image, initialize an [EncoderParameters](#) object and pass the address of that object to the [Save](#) method of the [Image](#) class. Initialize the [EncoderParameters](#) object so that it has an array that consists of one [EncoderParameter](#) object. Initialize that one [EncoderParameter](#) object so that its [Value](#) member points to a [ULONG](#) variable that holds one of the following elements of the [EncoderValue](#) enumeration:

- [EncoderValueTransformRotate90](#),
- [EncoderValueTransformRotate180](#),
- [EncoderValueTransformRotate270](#),
- [EncoderValueTransformFlipHorizontal](#),
- [EncoderValueTransformFlipVertical](#)

Set the [Guid](#) member of the [EncoderParameter](#) object to [EncoderTransformation](#).

The following console application creates an [Image](#) object from a JPEG file and then saves the image to a new file. During the save process, the image is rotated 90 degrees. If the width and height of the image are both multiples of 16, the process of rotating and saving the image causes no loss of information.

The main function relies on the helper function [GetEncoderClid](#), which is shown in [Retrieving the Class Identifier for an Encoder](#).

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT GetEncoderClid(const WCHAR* format, CLSID* pClid); // helper function
```

```

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    CLSID encoderClSID;
    EncoderParameters encoderParameters;
    ULONG transformation;
    UINT width;
    UINT height;
    Status stat;

    // Get a JPEG image from the disk.
    Image* image = new Image(L"Shapes.jpg");

    // Determine whether the width and height of the image
    // are multiples of 16.
    width = image->GetWidth();
    height = image->GetHeight();

    printf("The width of the image is %u", width);
    if(width / 16.0 - width / 16 == 0)
        printf(", which is a multiple of 16.\n");
    else
        printf(", which is not a multiple of 16.\n");

    printf("The height of the image is %u", height);
    if(height / 16.0 - height / 16 == 0)
        printf(", which is a multiple of 16.\n");
    else
        printf(", which is not a multiple of 16.\n");

    // Get the CLSID of the JPEG encoder.
    GetEncoderClsid(L"image/jpeg", &encoderClSID);

    // Before we call Image::Save, we must initialize an
    // EncoderParameters object. The EncoderParameters object
    // has an array of EncoderParameter objects. In this
    // case, there is only one EncoderParameter object in the array.
    // The one EncoderParameter object has an array of values.
    // In this case, there is only one value (of type ULONG)
    // in the array. We will set that value to EncoderValueTransformRotate90.

    encoderParameters.Count = 1;
    encoderParameters.Parameter[0].Guid = EncoderTransformation;
    encoderParameters.Parameter[0].Type = EncoderParameterValueTypeLong;
    encoderParameters.Parameter[0].NumberOfValues = 1;

    // Rotate and save the image.
    transformation = EncoderValueTransformRotate90;
    encoderParameters.Parameter[0].Value = &transformation;
    stat = image->Save(L"ShapesR90.jpg", &encoderClSID, &encoderParameters);

    if(stat == Ok)
        wprintf(L"%s saved successfully.\n", L"ShapesR90.jpg");
    else
        wprintf(L"%d Attempt to save %s failed.\n", stat, L"ShapesR90.jpg");

    delete image;
    GdiplusShutdown(gdiplusToken);
    return 0;
}

```


Creating and Saving a Multiple-Frame Image

11/2/2020 • 3 minutes to read • [Edit Online](#)

With certain file formats, you can save multiple images (frames) to a single file. For example, you can save several pages to a single TIFF file. To save the first page, call the [Save](#) method of the [Image](#) class. To save subsequent pages, call the [SaveAdd](#) method of the [Image](#) class.

NOTE

You cannot use [SaveAdd](#) to add frames to an animated gif file.

The following console application creates a TIFF file with four pages. The images that become the pages of the TIFF file come from four disk files: Shapes.bmp, Cereal.gif, Iron.jpg, and House.png. The code first constructs four [Image](#) objects: **multi**, **page2**, **page3**, and **page4**. At first, **multi** contains only the image from Shapes.bmp, but eventually it contains all four images. As the individual pages are added to the **multi** [Image](#) object, they are also added to the disk file Multiframe.tif.

Note that the code calls [Save](#) (not [SaveAdd](#)) to save the first page. The first argument passed to the Save method is the name of the disk file that will eventually contain several frames. The second argument passed to the Save method specifies the encoder that will be used to convert the data in the **multi** [Image](#) object to the format (in this case TIFF) required by the disk file. That same encoder is used automatically by all subsequent calls to the SaveAdd method of the **multi** [Image](#) object.

The third argument passed to the [Save](#) method is the address of an [EncoderParameters](#) object. The [EncoderParameters](#) object has an array that contains a single [EncoderParameter](#) object. The **Guid** member of that [EncoderParameter](#) object is set to EncoderSaveFlag. The **Value** member of the [EncoderParameter](#) object points to a **ULONG** that contains the value EncoderValueMultiFrame.

The code saves the second, third, and fourth pages by calling the [SaveAdd](#) method of the **multi** [Image](#) object. The first argument passed to the SaveAdd method is the address of an [Image](#) object. The image in that [Image](#) object is added to the **multi** [Image](#) object and is also added to the Multiframe.tif disk file. The second argument passed to the SaveAdd method is the address of the same [EncoderParameters](#) object that was used by the [Save](#) method. The difference is that the **ULONG** pointed to by the **Value** member now contains the value EncoderValueFrameDimensionPage.

The main function relies on the helper function GetEncoderClid, which is shown in [Retrieving the Class Identifier for an Encoder](#).

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT GetEncoderClid(const WCHAR* format, CLSID* pClid); // helper function

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
```

```

EncoderParameters encoderParameters;
ULONG parameterValue;
Status stat;

// An EncoderParameters object has an array of
// EncoderParameter objects. In this case, there is only
// one EncoderParameter object in the array.
encoderParameters.Count = 1;

// Initialize the one EncoderParameter object.
encoderParameters.Parameter[0].Guid = EncoderSaveFlag;
encoderParameters.Parameter[0].Type = EncoderParameterValueTypeLong;
encoderParameters.Parameter[0].NumberOfValues = 1;
encoderParameters.Parameter[0].Value = &parameterValue;

// Get the CLSID of the TIFF encoder.
CLSID encoderClsid;
GetEncoderClsid(L"image/tiff", &encoderClsid);

// Create four image objects.
Image* multi = new Image(L"Shapes.bmp");
Image* page2 = new Image(L"Cereal.gif");
Image* page3 = new Image(L"Iron.jpg");
Image* page4 = new Image(L"House.png");

// Save the first page (frame).
parameterValue = EncoderValueMultiFrame;
stat = multi->Save(L"MultiFrame.tif", &encoderClsid, &encoderParameters);
if(stat == Ok)
    printf("Page 1 saved successfully.\n");

// Save the second page (frame).
parameterValue = EncoderValueFrameDimensionPage;
stat = multi->SaveAdd(page2, &encoderParameters);
if(stat == Ok)
    printf("Page 2 saved successfully.\n");

// Save the third page (frame).
parameterValue = EncoderValueFrameDimensionPage;
stat = multi->SaveAdd(page3, &encoderParameters);
if(stat == Ok)
    printf("Page 3 saved successfully.\n");

// Save the fourth page (frame).
parameterValue = EncoderValueFrameDimensionPage;
stat = multi->SaveAdd(page4, &encoderParameters);
if(stat == Ok)
    printf("Page 4 saved successfully.\n");

// Close the multiframe file.
parameterValue = EncoderValueFlush;
stat = multi->SaveAdd(&encoderParameters);
if(stat == Ok)
    printf("File closed successfully.\n");

delete multi;
delete page2;
delete page3;
delete page4;
GdiplusShutdown(gdiplusToken);
return 0;
}

```

Copy individual frames from a multiple-frame image

2/22/2020 • 2 minutes to read • [Edit Online](#)

The following example retrieves the individual frames from a multiple-frame TIFF file. When the TIFF file was created, the individual frames were added to the Page dimension (see [Creating and Saving a Multiple-Frame Image](#)). The code displays each of the four pages and saves each page to a separate PNG disk file.

The code constructs an **Image** object from the multiple-frame TIFF file. To retrieve the individual frames (pages), the code calls the **Image::SelectActiveFrame** method of that **Image** object. The first argument passed to the **Image::SelectActiveFrame** method is the address of a GUID that specifies the dimension in which the frames were previously added to the multiple-frame TIFF file. The GUID FrameDimensionPage is defined in Gdipplusimaging.h. Other GUIDs defined in that header file are FrameDimensionTime and FrameDimensionResolution. The second argument passed to the **Image::SelectActiveFrame** method is the zero-based index of the desired page.

The code relies on the helper function GetEncoderClsid, which is shown in [Retrieving the Class Identifier for an Encoder](#).

```
GUID    pageGuid = FrameDimensionPage;
CLSID   encoderClSID;
Image   multi(L"Multiframe.tif");

// Get the CLSID of the PNG encoder.
GetEncoderClSID(L"image/png", &encoderClSID);

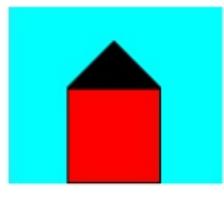
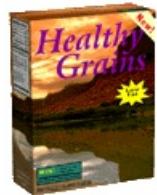
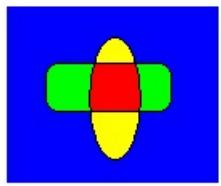
// Display and save the first page (index 0).
multi.SelectActiveFrame(&pageGuid, 0);
graphics.DrawImage(&multi, 10, 10);
multi.Save(L"Page0.png", &encoderClSID, NULL);

// Display and save the second page.
multi.SelectActiveFrame(&pageGuid, 1);
graphics.DrawImage(&multi, 200, 10);
multi.Save(L"Page1.png", &encoderClSID, NULL);

// Display and save the third page.
multi.SelectActiveFrame(&pageGuid, 2);
graphics.DrawImage(&multi, 10, 150);
multi.Save(L"Page2.png", &encoderClSID, NULL);

// Display and save the fourth page.
multi.SelectActiveFrame(&pageGuid, 3);
graphics.DrawImage(&multi, 200, 150);
multi.Save(L"Page3.png", &encoderClSID, NULL);
```

The following illustration shows the individual pages as displayed by the preceding code.



Alpha Blending Lines and Fills

2/22/2020 • 2 minutes to read • [Edit Online](#)

In Windows GDI+, a color is a 32-bit value with 8 bits each for alpha, red, green, and blue. The alpha value indicates the transparency of the color — the extent to which the color is blended with the background color. Alpha values range from 0 through 255, where 0 represents a fully transparent color, and 255 represents a fully opaque color.

Alpha blending is a pixel-by-pixel blending of source and background color data. Each of the three components (red, green, blue) of a given source color is blended with the corresponding component of the background color according to the following formula:

$$\text{displayColor} = \text{sourceColor} \times \text{alpha} / 255 + \text{backgroundColor} \times (255 - \text{alpha}) / 255$$

For example, suppose the red component of the source color is 150 and the red component of the background color is 100. If the alpha value is 200, the red component of the resultant color is calculated as follows:

$$150 \times 200 / 255 + 100 \times (255 - 200) / 255 = 139$$

The following topics cover alpha blending in more detail:

- [Drawing Opaque and Semitransparent Lines](#)
- [Drawing with Opaque and Semitransparent Brushes](#)
- [Using Compositing Mode to Control Alpha Blending](#)
- [Using a Color Matrix to Set Alpha Values in Images](#)
- [Setting the Alpha Values of Individual Pixels](#)

Drawing Opaque and Semitransparent Lines

11/2/2020 • 2 minutes to read • [Edit Online](#)

When you draw a line, you must pass the address of a **Pen** object to the **DrawLine** method of the **Graphics** class. One of the parameters of the **Pen** constructor is a **Color** object. To draw an opaque line, set the alpha component of the color to 255. To draw a semitransparent line, set the alpha component to any value from 1 through 254.

When you draw a semitransparent line over a background, the color of the line is blended with the colors of the background. The alpha component specifies how the line and background colors are mixed; alpha values near 0 place more weight on the background colors, and alpha values near 255 place more weight on the line color.

The following example draws an image and then draws three lines that use the image as a background. The first line uses an alpha component of 255, so it is opaque. The second and third lines use an alpha component of 128, so they are semitransparent; you can see the background image through the lines. The call to **Graphics::SetCompositingQuality** causes the blending for the third line to be done in conjunction with gamma correction.

```
Image image(L"Texture1.jpg");
graphics.DrawImage(&image, 10, 5, image.GetWidth(), image.GetHeight());
Pen opaquePen(Color(255, 0, 0, 255), 15);
Pen semiTransPen(Color(128, 0, 0, 255), 15);
graphics.DrawLine(&opaquePen, 0, 20, 100, 20);
graphics.DrawLine(&semiTransPen, 0, 40, 100, 40);
graphics.SetCompositingQuality(CompositingQualityGammaCorrected);
graphics.DrawLine(&semiTransPen, 0, 60, 100, 60);
```

The following illustration shows the output of the preceding code.



Drawing with Opaque and Semitransparent Brushes

2/22/2020 • 2 minutes to read • [Edit Online](#)

When you fill a shape, you must pass the address of a **Brush** object to one of the fill methods of the **Graphics** class. The one parameter of the **SolidBrush** constructor is a **Color** object. To fill an opaque shape, set the alpha component of the color to 255. To fill a semitransparent shape, set the alpha component to any value from 1 through 254.

When you fill a semitransparent shape, the color of the shape is blended with the colors of the background. The alpha component specifies how the shape and background colors are mixed; alpha values near 0 place more weight on the background colors, and alpha values near 255 place more weight on the shape color.

The following example draws an image and then fills three ellipses that overlap the image. The first ellipse uses an alpha component of 255, so it is opaque. The second and third ellipses use an alpha component of 128, so they are semitransparent; you can see the background image through the ellipses. The call to **Graphics::SetCompositingQuality** causes the blending for the third ellipse to be done in conjunction with gamma correction.

```
Image image(L"Texture1.jpg");
graphics.DrawImage(&image, 50, 50, image.GetWidth(), image.GetHeight());
SolidBrush opaqueBrush(Color(255, 0, 0, 255));
SolidBrush semiTransBrush(Color(128, 0, 0, 255));
graphics.FillEllipse(&opaqueBrush, 35, 45, 45, 30);
graphics.FillEllipse(&semiTransBrush, 86, 45, 45, 30);
graphics.SetCompositingQuality(CompositingQualityGammaCorrected);
graphics.FillEllipse(&semiTransBrush, 40, 90, 86, 30);
```

The following illustration shows the output of the preceding code.



Using Compositing Mode to Control Alpha Blending

2/26/2020 • 2 minutes to read • [Edit Online](#)

There might be times when you want to create an off-screen bitmap that has the following characteristics:

- Colors have alpha values that are less than 255.
- Colors are not alpha blended with each other as you create the bitmap.
- When you display the finished bitmap, colors in the bitmap are alpha blended with the background colors on the display device.

To create such a bitmap, construct a blank **Bitmap** object, and then construct a **Graphics** object based on that bitmap. Set the compositing mode of the **Graphics** object to **CompositingModeSourceCopy**.

The following example creates a **Graphics** object based on a **Bitmap** object. The code uses the **Graphics** object along with two semitransparent brushes (alpha = 160) to paint on the bitmap. The code fills a red ellipse and a green ellipse using the semitransparent brushes. The green ellipse overlaps the red ellipse, but the green is not blended with the red because the compositing mode of the **Graphics** object is set to **CompositingModeSourceCopy**.

Next the code prepares to draw on the screen by calling **BeginPaint** and creating a **Graphics** object based on a device context. The code draws the bitmap on the screen twice: once on a white background and once on a multicolored background. The pixels in the bitmap that are part of the two ellipses have an alpha component of 160, so the ellipses are blended with the background colors on the screen.

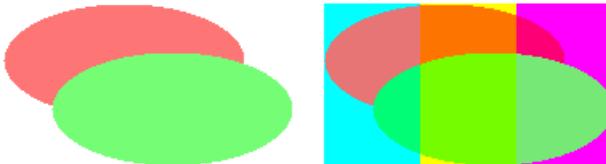
```

// Create a blank bitmap.
Bitmap bitmap(180, 100);
// Create a Graphics object that can be used to draw on the bitmap.
Graphics bitmapGraphics(&bitmap);
// Create a red brush and a green brush, each with an alpha value of 160.
SolidBrush redBrush(Color(210, 255, 0, 0));
SolidBrush greenBrush(Color(210, 0, 255, 0));
// Set the compositing mode so that when overlapping ellipses are drawn,
// the colors of the ellipses are not blended.
bitmapGraphics.SetCompositingMode(CompositingModeSourceCopy);
// Fill an ellipse using a red brush that has an alpha value of 160.
bitmapGraphics.FillEllipse(&redBrush, 0, 0, 150, 70);
// Fill a second ellipse using green brush that has an alpha value of 160.
// The green ellipse overlaps the red ellipse, but the green is not
// blended with the red.
bitmapGraphics.FillEllipse(&greenBrush, 30, 30, 150, 70);
// Prepare to draw on the screen.
hdc = BeginPaint(hWnd, &ps);
Graphics* pGraphics = new Graphics(hdc);
pGraphics->SetCompositingQuality(CompositingQualityGammaCorrected);
// Draw a multicolored background.
SolidBrush brush(Color((ARGB)Color::Aqua));
pGraphics->FillRectangle(&brush, 200, 0, 60, 100);
brushSetColor(Color((ARGB)Color::Yellow));
pGraphics->FillRectangle(&brush, 260, 0, 60, 100);
brushSetColor(Color((ARGB)Color::Fuchsia));
pGraphics->FillRectangle(&brush, 320, 0, 60, 100);

// Display the bitmap on a white background.
pGraphics->DrawImage(&bitmap, 0, 0);
// Display the bitmap on a multicolored background.
pGraphics->DrawImage(&bitmap, 200, 0);
delete pGraphics;
EndPaint(hWnd, &ps);

```

The following illustration shows the output of the preceding code. Note that the ellipses are blended with the background, but they are not blended with each other.



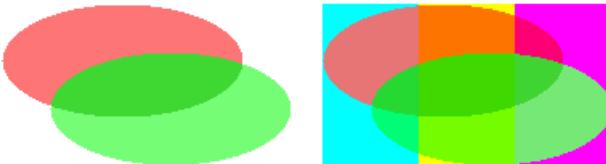
The preceding code example has the following statement:

```
bitmapGraphics.SetCompositingMode(CompositingModeSourceCopy);
```

If you want the ellipses to be blended with each other as well as with the background, change that statement to the following:

```
bitmapGraphics.SetCompositingMode(CompositingModeSourceOver);
```

The following illustration shows the output of the revised code.



Using a Color Matrix to Set Alpha Values in Images

11/2/2020 • 2 minutes to read • [Edit Online](#)

The **Bitmap** class (which inherits from the **Image** class) and the **ImageAttributes** class provide functionality for getting and setting pixel values. You can use the **ImageAttributes** class to modify the alpha values for an entire image, or you can call the **Bitmap::SetPixel** method of the **Bitmap** class to modify individual pixel values. For more information on setting individual pixel values, see [Setting the Alpha Values of Individual Pixels](#).

The following example draws a wide black line and then displays an opaque image that covers part of that line.

```
Bitmap bitmap(L"Texture1.jpg");
Pen pen(Color(255, 0, 0, 0), 25);
// First draw a wide black line.
graphics.DrawLine(&pen, Point(10, 35), Point(200, 35));
// Now draw an image that covers part of the black line.
graphics.DrawImage(&bitmap,
    Rect(30, 0, bitmap.GetWidth(), bitmap.GetHeight()));
```

The following illustration shows the resulting image, which is drawn at (30, 0). Note that the wide black line doesn't show through the image.



The **ImageAttributes** class has many properties that you can use to modify images during rendering. In the following example, an **ImageAttributes** object is used to set all the alpha values to 80 percent of what they were. This is done by initializing a color matrix and setting the alpha scaling value in the matrix to 0.8. The address of the color matrix is passed to the **ImageAttributes::SetColorMatrix** method of the **ImageAttributes** object, and the address of the **ImageAttributes** object is passed to the **DrawImage** method of a **Graphics** object.

```

// Create a Bitmap object and load it with the texture image.
Bitmap bitmap(L"Texture1.jpg");
Pen pen(Color(255, 0, 0, 0), 25);
// Initialize the color matrix.
// Notice the value 0.8 in row 4, column 4.
ColorMatrix colorMatrix = {1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
                           0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
                           0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
                           0.0f, 0.0f, 0.0f, 0.8f, 0.0f,
                           0.0f, 0.0f, 0.0f, 1.0f};

// Create an ImageAttributes object and set its color matrix.
ImageAttributes imageAtt;
imageAttSetColorMatrix(&colorMatrix, ColorMatrixFlagsDefault,
    ColorAdjustTypeBitmap);
// First draw a wide black line.
graphics.DrawLine(&pen, Point(10, 35), Point(200, 35));
// Now draw the semitransparent bitmap image.
INT iWidth = bitmap.GetWidth();
INT iHeight = bitmap.GetHeight();
graphics.DrawImage(
    &bitmap,
    Rect(30, 0, iWidth, iHeight), // Destination rectangle
    0,                          // Source rectangle X
    0,                          // Source rectangle Y
    iWidth,                     // Source rectangle width
    iHeight,                    // Source rectangle height
    UnitPixel,
    &imageAtt);

```

During rendering, the alpha values in the bitmap are converted to 80 percent of what they were. This results in an image that is blended with the background. As the following illustration shows, the bitmap image looks transparent; you can see the solid black line through it.



Where the image is over the white portion of the background, the image has been blended with the color white. Where the image crosses the black line, the image is blended with the color black.

Setting the Alpha Values of Individual Pixels

2/22/2020 • 2 minutes to read • [Edit Online](#)

The topic [Using a Color Matrix to Set Alpha Values in Images](#) shows a nondestructive method for changing the alpha values of an image. The example in that topic renders an image semitransparently, but the pixel data in the bitmap is not changed. The alpha values are altered only during rendering.

The following example shows how to change the alpha values of individual pixels. The code in the example actually changes the alpha information in a [Bitmap](#) object. The approach is much slower than using a color matrix and an [ImageAttributes](#) object but gives you control over the individual pixels in the bitmap.

```
INT iWidth = bitmap.GetWidth();
INT iHeight = bitmap.GetHeight();
Color color, colorTemp;
for(INT iRow = 0; iRow < iHeight; iRow++)
{
    for(INT iColumn = 0; iColumn < iWidth; iColumn++)
    {
        bitmap.GetPixel(iColumn, iRow, &color);
        colorTemp.SetValue(color.MakeARGB(
            (BYTE)(255 * iColumn / iWidth),
            color.GetRed(),
            color.GetGreen(),
            color.GetBlue()));
        bitmap.SetPixel(iColumn, iRow, colorTemp);
    }
}
// First draw a wide black line.
Pen pen(Color(255, 0, 0, 0), 25);
graphics.DrawLine(&pen, 10, 35, 200, 35);
// Now draw the modified bitmap.
graphics.DrawImage(&bitmap, 30, 0, iWidth, iHeight);
```

The following illustration shows the resulting image.



The preceding code example uses nested loops to change the alpha value of each pixel in the bitmap. For each pixel, [Bitmap::GetPixel](#) gets the existing color, [Color::SetValue](#) creates a temporary color that contains the new alpha value, and then [Bitmap::SetPixel](#) sets the new color. The alpha value is set based on the column of the bitmap. In the first column, alpha is set to 0. In the last column, alpha is set to 255. So the resulting image goes from fully transparent (on the left edge) to fully opaque (on the right edge).

[Bitmap::GetPixel](#) and [Bitmap::SetPixel](#) give you control of the individual pixel values. However, using [Bitmap::GetPixel](#) and [Bitmap::SetPixel](#) is not nearly as fast as using the [ImageAttributes](#) class and the [ColorMatrix](#) structure.

Using Text and Fonts

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides several classes that form the foundation for drawing text. The [Graphics](#) class has several [DrawString](#) methods that allow you to specify various features of text, such as location, bounding rectangle, font, and format. Other classes that contribute to text rendering include [FontFamily](#), [Font](#), [StringFormat](#), [InstalledFontCollection](#), and [PrivateFontCollection](#).

The following topics cover text and fonts in more detail:

- [Constructing Font Families and Fonts](#)
- [Drawing Text](#)
- [Formatting Text](#)
- [Enumerating Installed Fonts](#)
- [Creating a Private Font Collection](#)
- [Obtaining Font Metrics](#)
- [Antialiasing with Text](#)

Constructing Font Families and Fonts

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ groups fonts with the same typeface but different styles into font families. For example, the Arial font family contains the following fonts:

- Arial Regular
- Arial Bold
- Arial Italic
- Arial Bold Italic

GDI+ uses four styles to form families: regular, bold, italic, and bold italic. Adjectives such as *narrow* and *rounded* are not considered styles; rather they are part of the family name. For example, Arial Narrow is a font family whose members are the following:

- Arial Narrow Regular
- Arial Narrow Bold
- Arial Narrow Italic
- Arial Narrow Bold Italic

Before you can draw text with GDI+, you need to construct a **FontFamily** object and a **Font** object. The **FontFamily** objects specifies the typeface (for example, Arial), and the **Font** object specifies the size, style, and units.

The following example constructs a regular style Arial font with a size of 16 pixels:

```
FontFamily fontFamily(L"Arial");
Font font(&fontFamily, 16, FontStyleRegular, UnitPixel);
```

In the preceding code, the first argument passed to the **Font** constructor is the address of the **FontFamily** object. The second argument specifies the size of the font measured in units identified by the fourth argument. The third argument identifies the style.

UnitPixel is a member of the **Unit** enumeration, and **FontStyleRegular** is a member of the **FontStyle** enumeration. Both enumerations are declared in **Gdiplusenums.h**.

Drawing Text (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

You can use the [DrawString](#) method of the [Graphics](#) class to draw text at a specified location or within a specified rectangle.

- [Drawing Text at a Specified Location](#)
- [Drawing Text in a Rectangle](#)

Drawing Text at a Specified Location

To draw text at a specified location, you need [Graphics](#), [FontFamily](#), [Font](#), [PointF](#), and [Brush](#) objects.

The following example draws the string "Hello" at location (30, 10). The font family is Times New Roman. The font, which is an individual member of the font family, is Times New Roman, size 24 pixels, regular style. Assume that `graphics` is an existing [Graphics](#) object.

```
FontFamily fontFamily(L"Times New Roman");
Font      font(&fontFamily, 24, FontStyleRegular, UnitPixel);
PointF    pointF(30.0f, 10.0f);
SolidBrush solidBrush(Color(255, 0, 0, 255));

graphics.DrawString(L"Hello", -1, &font, pointF, &solidBrush);
```

The following illustration shows the output of the preceding code.



In the preceding example, the [FontFamily](#) constructor receives a string that identifies the font family. The address of the [FontFamily](#) object is passed as the first argument to the [Font](#) constructor. The second argument passed to the [Font](#) constructor specifies the size of the font measured in units given by the fourth argument. The third argument specifies the style (regular, bold, italic, and so on) of the font.

The [DrawString](#) method receives five arguments. The first argument is the string to be drawn, and the second argument is the length (in characters, not bytes) of that string. If the string is null-terminated, you can pass `-1` for the length. The third argument is the address of the [Font](#) object that was constructed previously. The fourth argument is a [PointF](#) object that contains the coordinates of the upper-left corner of the string. The fifth argument is the address of a [SolidBrush](#) object that will be used to fill the characters of the string.

Drawing Text in a Rectangle

One of the [DrawString](#) methods of the [Graphics](#) class has a *RectF* parameter. By calling that [DrawString](#) method, you can draw text that wraps in a specified rectangle. To draw text in a rectangle, you need [Graphics](#), [FontFamily](#), [Font](#), [RectF](#), and [Brush](#) objects.

The following example creates a rectangle with upper-left corner (30, 10), width 100, and height 122. Then the code draws a string inside that rectangle. The string is restricted to the rectangle and wraps in such a way that individual words are not broken.

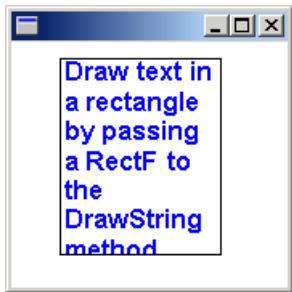
```
WCHAR string[] =
    L"Draw text in a rectangle by passing a RectF to the DrawString method.';

FontFamily fontFamily(L"Arial");
Font font(&fontFamily, 12, FontStyleBold, UnitPoint);
RectF rectF(30.0f, 10.0f, 100.0f, 122.0f);
SolidBrush solidBrush(Color(255, 0, 0, 255));

graphics.DrawString(string, -1, &font, rectF, NULL, &solidBrush);

Pen pen(Color(255, 0, 0, 0));
graphics.DrawRectangle(&pen, rectF);
```

The following illustration shows the text drawn in the rectangle.



In the preceding example, the fourth argument passed to the [DrawString](#) method is a [RectF](#) object that specifies the bounding rectangle for the text. The fifth parameter is of type [StringFormat](#)—the argument is **NULL** because no special string formatting is required.

Formatting Text (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

To apply special formatting to text, initialize a [StringFormat](#) object and pass the address of that object to the [DrawString](#) method of the [Graphics](#) class.

To draw formatted text in a rectangle, you need [Graphics](#), [FontFamily](#), [Font](#), [RectF](#), [StringFormat](#), and [Brush](#) objects.

- [Aligning Text](#)
- [Setting Tab Stops](#)
- [Drawing Vertical Text](#)

Aligning Text

The following example draws text in a rectangle. Each line of text is centered (side to side), and the entire block of text is centered (top to bottom) in the rectangle.

```
WCHAR string[] =
    L"Use StringFormat and RectF objects to center text in a rectangle./";

FontFamily    fontFamily(L"Arial");
Font         font(&fontFamily, 12, FontStyleBold, UnitPoint);
RectF        rectF(30.0f, 10.0f, 120.0f, 140.0f);
StringFormat stringFormat;
SolidBrush    solidBrush(Color(255, 0, 0, 255));

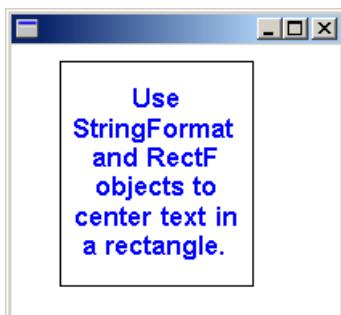
// Center-justify each line of text.
stringFormat.SetAlignment(StringAlignmentCenter);

// Center the block of text (top to bottom) in the rectangle.
stringFormat.SetLineAlignment(StringAlignmentCenter);

graphics.DrawString(string, -1, &font, rectF, &stringFormat, &solidBrush);

Pen pen(Color(255, 0, 0, 0));
graphics.DrawRectangle(&pen, rectF);
```

The following illustration shows the rectangle and the centered text.



The preceding code calls two methods of the [StringFormat](#) object: [StringFormat::SetAlignment](#) and [StringFormat::SetLineAlignment](#). The call to [StringFormat::SetAlignment](#) specifies that each line of text is centered in the rectangle given by the third argument passed to the [DrawString](#) method. The call to [StringFormat::SetLineAlignment](#) specifies that the block of text is centered (top to bottom) in the rectangle.

The value **StringAlignmentCenter** is an element of the **StringAlignment** enumeration, which is declared in Gdiplusenums.h.

Setting Tab Stops

You can set tab stops for text by calling the **StringFormat::SetTabStops** method of a **StringFormat** object and then passing the address of that **StringFormat** object to the **DrawString** method of the **Graphics** class.

The following example sets tab stops at 150, 250, and 350. Then the code displays a tabbed list of names and test scores.

```
WCHAR string[150] =
L"Name\tTest 1\tTest 2\tTest 3\n";

StringCchCatW(string, 150, L"Joe\t95\t88\t91\n");
StringCchCatW(string, 150, L"Mary\t98\t84\t90\n");
StringCchCatW(string, 150, L"Sam\t42\t76\t98\n");
StringCchCatW(string, 150, L"Jane\t65\t73\t92\n");

FontFamily fontFamily(L"Courier New");
Font font(&fontFamily, 12, FontStyleRegular, UnitPoint);
RectF rectF(10.0f, 10.0f, 450.0f, 100.0f);
StringFormat stringFormat;
SolidBrush solidBrush(Color(255, 0, 0, 255));
REAL tabs[] = {150.0f, 100.0f, 100.0f};

stringFormat.SetTabStops(0.0f, 3, tabs);

graphics.DrawString(string, -1, &font, rectF, &stringFormat, &solidBrush);

Pen pen(Color(255, 0, 0, 0));
graphics.DrawRectangle(&pen, rectF);
```

The following illustration shows the tabbed text.

Name	Test 1	Test 2	Test 3
Joe	95	88	91
Mary	98	84	90
Sam	42	76	98
Jane	65	73	92

The preceding code passes three arguments to the **StringFormat::SetTabStops** method. The third argument is the address of an array containing the tab offsets. The second argument indicates that there are three offsets in that array. The first argument passed to **StringFormat::SetTabStops** is 0, which indicates that the first offset in the array is measured from position 0, the left edge of the bounding rectangle.

Drawing Vertical Text

You can use a **StringFormat** object to specify that text be drawn vertically rather than horizontally.

The following example passes the value **StringFormatFlagsDirectionVertical** to the **StringFormat::SetFormatFlags** method of a **StringFormat** object. The address of that **StringFormat** object is passed to the **DrawString** method of the **Graphics** class. The value **StringFormatFlagsDirectionVertical** is an element of the **StringFormatFlags** enumeration, which is declared in Gdiplusenums.h.

```
WCHAR string[] = L"Vertical text";

FontFamily fontFamily(L"Lucida Console");
Font font(&fontFamily, 14, FontStyleRegular, UnitPoint);
PointF pointF(40.0f, 10.0f);
StringFormat stringFormat;
SolidBrush solidBrush(Color(255, 0, 0, 255));

stringFormat.SetFormatFlags(StringFormatFlagsDirectionVertical);

graphics.DrawString(string, -1, &font, pointF, &stringFormat, &solidBrush);
```

The following illustration shows the vertical text.



Enumerating Installed Fonts

11/2/2020 • 2 minutes to read • [Edit Online](#)

The [InstalledFontCollection](#) class inherits from the [FontCollection](#) abstract base class. You can use an [InstalledFontCollection](#) object to enumerate the fonts installed on the computer. The [FontCollection::GetFamilies](#) method of an [InstalledFontCollection](#) object returns an array of [FontFamily](#) objects. Before you call [FontCollection::GetFamilies](#), you must allocate a buffer large enough to hold that array. To determine the size of the required buffer, call the [FontCollection::GetFamilyCount](#) method and multiply the return value by `sizeof(FontFamily)`.

The following example lists the names of all the font families installed on the computer. The code retrieves the font family names by calling the [FontFamily::GetFamilyName](#) method of each [FontFamily](#) object in the array returned by [FontCollection::GetFamilies](#). As the family names are retrieved, they are concatenated to form a comma-separated list. Then the [DrawString](#) method of the [Graphics](#) class draws the comma-separated list in a rectangle.

```

FontFamily fontFamily(L"Arial");
Font font(&fontFamily, 8, FontStyleRegular, UnitPoint);
RectF rectF(10.0f, 10.0f, 500.0f, 500.0f);
SolidBrush solidBrush(Color(255, 0, 0, 0));

INT count = 0;
INT found = 0;
WCHAR familyName[LF_FACESIZE]; // enough space for one family name
WCHAR* familyList = NULL;
FontFamily* pFontFamily = NULL;

InstalledFontCollection installedFontCollection;

// How many font families are installed?
count = installedFontCollection.GetFamilyCount();

// Allocate a buffer to hold the array of FontFamily
// objects returned by GetFamilies.
pFontFamily = new FontFamily[count];

// Get the array of FontFamily objects.
installedFontCollection.GetFamilies(count, pFontFamily, &found);

// The loop below creates a large string that is a comma-separated
// list of all font family names.
// Allocate a buffer large enough to hold that string.
familyList = new WCHAR[count*(sizeof(familyName)+ 3)];
StringCchCopy(familyList, 1, L"");

for(INT j = 0; j < count; ++j)
{
    pFontFamily[j].GetFamilyName(familyName);
    StringCchCatW(familyList, count*(sizeof(familyName)+ 3), familyName);
    StringCchCatW(familyList, count*(sizeof(familyName)+ 3), L", ");
}

// Draw the large string (list of all families) in a rectangle.
graphics.DrawString(
    familyList, -1, &font, rectF, NULL, &solidBrush);

delete [] pFontFamily;
delete [] familyList;

```

The following illustration shows a possible output of the preceding code. If you run the code, the output might be different, depending on the fonts installed on your computer.



Creating a Private Font Collection

11/2/2020 • 4 minutes to read • [Edit Online](#)

The [PrivateFontCollection](#) class inherits from the [FontCollection](#) abstract base class. You can use a [PrivateFontCollection](#) object to maintain a set of fonts specifically for your application.

A private font collection can include installed system fonts as well as fonts that have not been installed on the computer. To add a font file to a private font collection, call the [PrivateFontCollection::AddFontFile](#) method of a [PrivateFontCollection](#) object.

NOTE

When you use the GDI+ API, you must never allow your application to download arbitrary fonts from untrusted sources. The operating system requires elevated privileges to assure that all installed fonts are trusted.

The [FontCollection::GetFamilies](#) method of a [PrivateFontCollection](#) object returns an array of [FontFamily](#) objects. Before you call [FontCollection::GetFamilies](#), you must allocate a buffer large enough to hold that array. To determine the size of the required buffer, call the [FontCollection::GetFamilyCount](#) method and multiply the return value by `sizeof(FontFamily)`.

The number of font families in a private font collection is not necessarily the same as the number of font files that have been added to the collection. For example, suppose you add the files ArialBd.ttf, Times.ttf, and TimesBd.ttf to a collection. There will be three files but only two families in the collection because Times.ttf and TimesBd.ttf belong to the same family.

The following example adds the following three font files to a [PrivateFontCollection](#) object:

- C:\WINNT\Fonts\Arial.ttf (Arial, regular)
- C:\WINNT\Fonts\CourBI.ttf (Courier New, bold italic)
- C:\WINNT\Fonts\TimesBd.ttf (Times New Roman, bold)

The code calls the [FontCollection::GetFamilyCount](#) method of the [PrivateFontCollection](#) object to determine the number of families in the private collection, and then calls [FontCollection::GetFamilies](#) to retrieve an array of [FontFamily](#) objects.

For each [FontFamily](#) object in the collection, the code calls the [FontFamily::IsStyleAvailable](#) method to determine whether various styles (regular, bold, italic, bold italic, underline, and strikeout) are available. The arguments passed to the [FontFamily::IsStyleAvailable](#) method are members of the [FontStyle](#) enumeration, which is declared in Gdiplusenums.h.

If a particular family/style combination is available, a [Font](#) object is constructed using that family and style. The first argument passed to the [Font](#) constructor is the font family name (not a [FontFamily](#) object as is the case for other variations of the [Font](#) constructor), and the final argument is the address of the [PrivateFontCollection](#) object. After the [Font](#) object is constructed, its address is passed to the [DrawString](#) method of the [Graphics](#) class to display the family name along with the name of the style.

```
#define MAX_STYLE_SIZE 20
#define MAX_FACEANDSTYLE_SIZE (LF_FACESIZE + MAX_STYLE_SIZE + 2)

PointF      pointF(10.0f, 0.0f);
SolidBrush  solidBrush(Color(255, 0, 0, 0));
```

```

INT count = 0;
INT found = 0;
WCHAR familyName[LF_FACESIZE];
WCHAR familyNameAndStyle[MAX_FACEANDSTYLE_SIZE];
FontFamily* pFontFamily;
PrivateFontCollection privateFontCollection;

// Add three font files to the private collection.
privateFontCollection.AddFontFile(L"c:\\Winnt\\Fonts\\Arial.ttf");
privateFontCollection.AddFontFile(L"c:\\Winnt\\Fonts\\CourBI.ttf");
privateFontCollection.AddFontFile(L"c:\\Winnt\\Fonts\\TimesBd.ttf");

// How many font families are in the private collection?
count = privateFontCollection.GetFamilyCount();

// Allocate a buffer to hold the array of FontFamily
// objects returned by GetFamilies.
pFontFamily = new FontFamily[count];

// Get the array of FontFamily objects.
privateFontCollection.GetFamilies(count, pFontFamily, &found);

// Display the name of each font family in the private collection
// along with the available styles for that font family.
for(INT j = 0; j < count; ++j)
{
    // Get the font family name.
    pFontFamily[j].GetFamilyName(familyName);

    // Is the regular style available?
    if(pFontFamily[j].IsStyleAvailable(FontStyleRegular))
    {
        StringCchCopyW(familyNameAndStyle, LF_FACESIZE, familyName);
        StringCchCatW(familyNameAndStyle, MAX_FACEANDSTYLE_SIZE, L" Regular");

        Font* pFont = new Font(
            familyName, 16, FontStyleRegular, UnitPixel, &privateFontCollection);

        graphics.DrawString(familyNameAndStyle, -1, pFont, pointF, &solidBrush);

        pointF.Y += pFont->GetHeight(0.0f);
        delete(pFont);
    }

    // Is the bold style available?
    if(pFontFamily[j].IsStyleAvailable(FontStyleBold))
    {
        StringCchCopyW(familyNameAndStyle, LF_FACESIZE, familyName);
        StringCchCatW(familyNameAndStyle, MAX_FACEANDSTYLE_SIZE, L" Bold");

        Font* pFont = new Font(
            familyName, 16, FontStyleBold, UnitPixel, &privateFontCollection);

        graphics.DrawString(familyNameAndStyle, -1, pFont, pointF, &solidBrush);

        pointF.Y += pFont->GetHeight(0.0f);
        delete(pFont);
    }

    // Is the italic style available?
    if(pFontFamily[j].IsStyleAvailable(FontStyleItalic))
    {
        StringCchCopyW(familyNameAndStyle, LF_FACESIZE, familyName);
        StringCchCatW(familyNameAndStyle, MAX_FACEANDSTYLE_SIZE, L" Italic");

        Font* pFont = new Font(
            familyName, 16, FontStyleItalic, UnitPixel, &privateFontCollection);

        graphics.DrawString(familyNameAndStyle, -1, pFont, pointF, &solidBrush);
    }
}

```

```

graphics.DrawString(familyNameAndStyle, -1, pFont, pointF, &solidBrush);

pointF.Y += pFont->GetHeight(0.0f);
delete(pFont);
}

// Is the bold italic style available?
if(pFontFamily[j].IsStyleAvailable(FontStyleBoldItalic))
{
    StringCchCopyW(familyNameAndStyle, LF_FACESIZE, familyName);
    StringCchCatW(familyNameAndStyle, MAX_FACEANDSTYLE_SIZE, L" BoldItalic");

    Font* pFont = new Font(familyName, 16,
        FontStyleBoldItalic, UnitPixel, &privateFontCollection);

    graphics.DrawString(familyNameAndStyle, -1, pFont, pointF, &solidBrush);

    pointF.Y += pFont->GetHeight(0.0f);
    delete(pFont);
}

// Is the underline style available?
if(pFontFamily[j].IsStyleAvailable(FontStyleUnderline))
{
    StringCchCopyW(familyNameAndStyle, LF_FACESIZE, familyName);
    StringCchCatW(familyNameAndStyle, MAX_FACEANDSTYLE_SIZE, L" Underline");

    Font* pFont = new Font(familyName, 16,
        FontStyleUnderline, UnitPixel, &privateFontCollection);

    graphics.DrawString(familyNameAndStyle, -1, pFont, pointF, &solidBrush);

    pointF.Y += pFont->GetHeight(0.0f);
    delete(pFont);
}

// Is the strikeout style available?
if(pFontFamily[j].IsStyleAvailable(FontStyleStrikeout))
{
    StringCchCopyW(familyNameAndStyle, LF_FACESIZE, familyName);
    StringCchCatW(familyNameAndStyle, MAX_FACEANDSTYLE_SIZE, L" Strikeout");

    Font* pFont = new Font(familyName, 16,
        FontStyleStrikeout, UnitPixel, &privateFontCollection);

    graphics.DrawString(familyNameAndStyle, -1, pFont, pointF, &solidBrush);

    pointF.Y += pFont->GetHeight(0.0f);
    delete(pFont);
}

// Separate the families with white space.
pointF.Y += 10.0f;

} // for

delete pFontFamily;

```

The following illustration shows the output of the preceding code.



Arial.ttf (which was added to the private font collection in the preceding code example) is the font file for the Arial regular style. Note, however, that the program output shows several available styles other than regular for the Arial font family. That is because Windows GDI+ can simulate the bold, italic, and bold italic styles from the regular style. GDI+ can also produce underlines and strikeouts from the regular style.

Similarly, GDI+ can simulate the bold italic style from either the bold style or the italic style. The program output shows that the bold italic style is available for the Times family even though TimesBd.ttf (Times New Roman, bold) is the only Times file in the collection.

This table specifies the non-system fonts that GDI+ supports.

	GDI	GDI+ ON WINDOWS 7	GDI+ ON WINDOWS 8	DIRECTWRITE
.FON	yes	no	no	no
.FNT	yes	no	no	no
.TTF	yes	yes	yes	yes
.OTF with TrueType	yes	yes	yes	yes
.OTF with Adobe CFF	yes	no	yes	yes
Adobe Type 1	yes	no	no	no

Obtaining Font Metrics

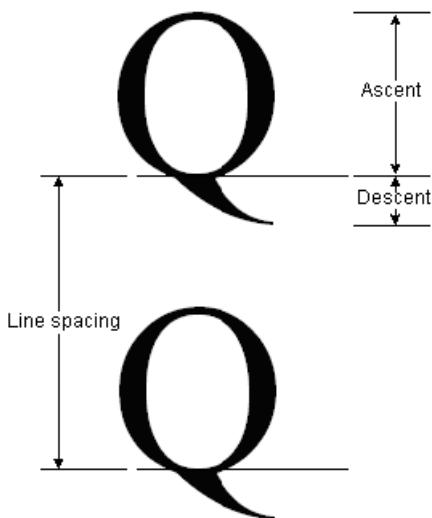
11/2/2020 • 2 minutes to read • [Edit Online](#)

The [FontFamily](#) class provides the following methods that retrieve various metrics for a particular family/style combination:

- [FontFamily::GetEmHeight\(FontStyle\)](#)
- [FontFamily::GetCellAscent\(FontStyle\)](#)
- [FontFamily::GetCellDescent\(FontStyle\)](#)
- [FontFamily::GetLineSpacing\(FontStyle\)](#)

The numbers returned by these methods are in font design units, so they are independent of the size and units of a particular [Font](#) object.

The following illustration shows ascent, descent, and line spacing.



The following example displays the metrics for the regular style of the Arial font family. The code also creates a [Font](#) object (based on the Arial family) with size 16 pixels and displays the metrics (in pixels) for that particular [Font](#) object.

```
#define INFO_STRING_SIZE 100 // one line of output including null terminator
WCHAR infoString[INFO_STRING_SIZE] = L"";

UINT ascent; // font family ascent in design units
REAL ascentPixel; // ascent converted to pixels
UINT descent; // font family descent in design units
REAL descentPixel; // descent converted to pixels
UINT lineSpacing; // font family line spacing in design units
REAL lineSpacingPixel; // line spacing converted to pixels

FontFamily fontFamily(L"Arial");
Font font(&fontFamily, 16, FontStyleRegular, UnitPixel);
PointF pointF(10.0f, 10.0f);
SolidBrush solidBrush(Color(255, 0, 0, 0));

// Display the font size in pixels.
StringCchPrintf(
    infoString,
    INFO_STRING_SIZE,
    L"font.GetSize() returns %f.", font.GetSize());
```

```
graphics.DrawString(
    infoString, -1, &font, pointF, &solidBrush);

// Move down one line.
pointF.Y += font.GetHeight(0.0);

// Display the font family em height in design units.
StringCchPrintf(
    infoString,
    INFO_STRING_SIZE,
    L"fontFamily.GetEmHeight() returns %d.",
    fontFamily.GetEmHeight(FontStyleRegular));

graphics.DrawString(infoString, -1, &font, pointF, &solidBrush);

// Move down two lines.
pointF.Y += 2.0f * font.GetHeight(0.0f);

// Display the ascent in design units and pixels.
ascent = fontFamily.GetCellAscent(FontStyleRegular);

// 14.484375 = 16.0 * 1854 / 2048
ascentPixel =
    font.GetSize() * ascent / fontFamily.GetEmHeight(FontStyleRegular);

StringCchPrintf(
    infoString,
    INFO_STRING_SIZE,
    L"The ascent is %d design units, %f pixels.",
    ascent,
    ascentPixel);

graphics.DrawString(infoString, -1, &font, pointF, &solidBrush);

// Move down one line.
pointF.Y += font.GetHeight(0.0f);

// Display the descent in design units and pixels.
descent = fontFamily.GetCellDescent(FontStyleRegular);

// 3.390625 = 16.0 * 434 / 2048
descentPixel =
    font.GetSize() * descent / fontFamily.GetEmHeight(FontStyleRegular);

StringCchPrintf(
    infoString,
    INFO_STRING_SIZE,
    L"The descent is %d design units, %f pixels.",
    descent,
    descentPixel);

graphics.DrawString(infoString, -1, &font, pointF, &solidBrush);

// Move down one line.
pointF.Y += font.GetHeight(0.0f);

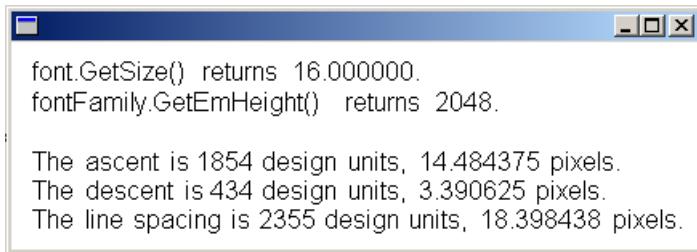
// Display the line spacing in design units and pixels.
lineSpacing = fontFamily.GetLineSpacing(FontStyleRegular);

// 18.398438 = 16.0 * 2355 / 2048
lineSpacingPixel =
    font.GetSize() * lineSpacing / fontFamily.GetEmHeight(FontStyleRegular);

StringCchPrintf(
    infoString,
    INFO_STRING_SIZE,
    L"The line spacing is %d design units, %f pixels.",
    lineSpacing,
    lineSpacingPixel);
```

```
graphics.DrawString(infoString, -1, &font, pointF, &solidBrush);
```

The following illustration shows the output of the preceding code.



Note the first two lines of output in the preceding illustration. The **Font** object returns a size of 16, and the **FontFamily** object returns an em height of 2,048. These two numbers (16 and 2,048) are the key to converting between font design units and the units (in this case pixels) of the **Font** object.

For example, you can convert the ascent from design units to pixels as follows:

$$\frac{1854 \text{ design units}}{1} \times \frac{16 \text{ pixels}}{2048 \text{ design units}} = 14.484375 \text{ pixels}$$

The preceding code positions text vertically by setting the *y* data member of a **PointF** object. The *y*-coordinate is increased by `font.GetHeight(0.0f)` for each new line of text. The **Font::GetHeight** method of a **Font** object returns the line spacing (in pixels) for that particular **Font** object. In this example, the number returned by **Font::GetHeight** is 18.398438. Note that this is the same as the number obtained by converting the line spacing metric to pixels.

Antialiasing with Text

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides various quality levels for drawing text. Typically, higher quality rendering takes more processing time than lower quality rendering.

The quality level is a property of the **Graphics** class. To set the quality level, call the **Graphics::SetTextRenderingHint** method of a **Graphics** object. The **Graphics::SetTextRenderingHint** method receives one of the elements of the **TextRenderingHint** enumeration, which is declared in **Gdiplusenums.h**.

GDI+ provides traditional antialiasing and a new kind of antialiasing based on Microsoft ClearType display technology only available on Windows XP and Windows Server 2003 and later versions of Windows. ClearType smoothing improves readability on color LCD monitors that have a digital interface, such as the monitors in laptops and high-quality flat desktop displays. Readability on CRT screens is also somewhat improved.

ClearType is dependent on the orientation and ordering of the LCD stripes. Currently, ClearType is implemented only for vertical stripes that are ordered RGB. This might be a concern if you are using a tablet PC, where the display can be oriented in any direction, or if you are using a screen that can be turned from landscape to portrait.

The following example draws text with two different quality settings:

```
FontFamily fontFamily(L"Times New Roman");
Font font(&fontFamily, 32, FontStyleRegular, UnitPixel);
SolidBrush solidBrush(Color(255, 0, 0, 255));
WCHAR string1[] = L"SingleBitPerPixel";
WCHAR string2[] = L"AntiAlias";

graphics.SetTextRenderingHint(TextRenderingHintSingleBitPerPixel);
graphics.DrawString(string1, -1, &font, PointF(10.0f, 10.0f), &solidBrush);

graphics.SetTextRenderingHint(TextRenderingHintAntiAlias);
graphics.DrawString(string2, -1, &font, PointF(10.0f, 60.0f), &solidBrush);
```

The following illustration shows the output of the preceding code.

SingleBitPerPixel

AntiAlias

Constructing and Drawing Curves

2/22/2020 • 2 minutes to read • [Edit Online](#)

GDI+ supports several types of curves: ellipses, arcs, cardinal splines, and Bézier splines. An ellipse is defined by its bounding rectangle; an arc is a portion of an ellipse defined by a starting angle and a sweep angle. A cardinal spline is defined by an array of points and a tension parameter — the curve passes smoothly through each point in the array, and the tension parameter influences the way the curve bends. A Bézier spline is defined by two end points and two control points — the curve does not pass through the control points, but the control points influence the direction and bend as the curve goes from one end point to the other.

The following topics cover cardinal splines and Bézier splines in more detail:

- [Drawing Cardinal Splines](#)
- [Drawing Bézier Splines](#)

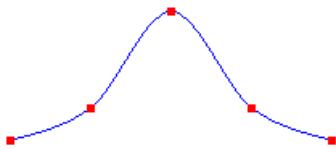
Drawing Cardinal Splines

11/2/2020 • 2 minutes to read • [Edit Online](#)

A cardinal spline is a curve that passes smoothly through a given set of points. To draw a cardinal spline, create a **Graphics** object and pass the address of an array of points to the **Graphics::DrawCurve** method. The following example draws a bell-shaped cardinal spline that passes through five designated points:

```
Point points[] = {Point(0, 100),
                  Point(50, 80),
                  Point(100, 20),
                  Point(150, 80),
                  Point(200, 100)};  
  
Pen pen(Color(255, 0, 0, 255));
graphics.DrawCurve(&pen, points, 5);
```

The following illustration shows the curve and five points.

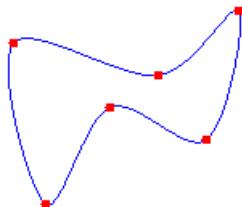


You can use the **Graphics::DrawClosedCurve** method of the **Graphics** class to draw a closed cardinal spline. In a closed cardinal spline, the curve continues through the last point in the array and connects with the first point in the array.

The following example draws a closed cardinal spline that passes through six designated points.

```
Point points[] = {Point(60, 60),
                  Point(150, 80),
                  Point(200, 40),
                  Point(180, 120),
                  Point(120, 100),
                  Point(80, 160)};  
  
Pen pen(Color(255, 0, 0, 255));
graphics.DrawClosedCurve(&pen, points, 6);
```

The following illustration shows the closed spline along with the six points:

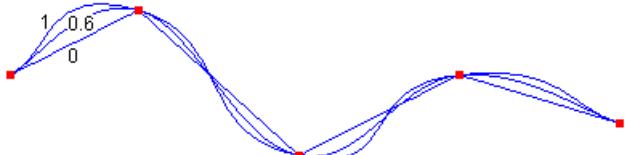


You can change the way a cardinal spline bends by passing a tension argument to the **Graphics::DrawCurve** method. The following example draws three cardinal splines that pass through the same set of points:

```
Point points[] = {Point(20, 50),
                  Point(100, 10),
                  Point(200, 100),
                  Point(300, 50),
                  Point(400, 80)};

Pen pen(Color(255, 0, 0, 255));
graphics.DrawCurve(&pen, points, 5, 0.0f); // tension 0.0
graphics.DrawCurve(&pen, points, 5, 0.6f); // tension 0.6
graphics.DrawCurve(&pen, points, 5, 1.0f); // tension 1.0
```

The following illustration shows the three splines along with their tension values. Note that when the tension is 0, the points are connected by straight lines.



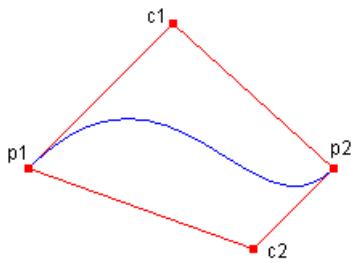
Drawing Bezier Splines

11/2/2020 • 2 minutes to read • [Edit Online](#)

A Bézier spline is defined by four points: a start point, two control points, and an end point. The following example draws a Bézier spline with start point (10, 100) and end point (200, 100). The control points are (100, 10) and (150, 150):

```
Point p1(10, 100); // start point
Point c1(100, 10); // first control point
Point c2(150, 150); // second control point
Point p2(200, 100); // end point
Pen pen(Color(255, 0, 0, 255));
graphics.DrawBezier(&pen, p1, c1, c2, p2);
```

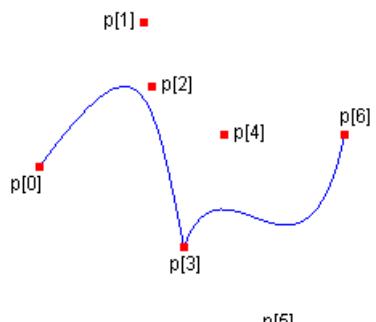
The following illustration shows the resulting Bézier spline along with its start point, control points, and end point. The illustration also shows the spline's convex hull, which is a polygon formed by connecting the four points with straight lines.



You can use the `DrawBeziers` method of the `Graphics` class to draw a sequence of connected Bézier splines. The following example draws a curve that consists of two connected Bézier splines. The end point of the first Bézier spline is the start point of the second Bézier spline.

```
Point p[] = {
    Point(10, 100), // start point of first spline
    Point(75, 10), // first control point of first spline
    Point(80, 50), // second control point of first spline
    Point(100, 150), // end point of first spline and
                      // start point of second spline
    Point(125, 80), // first control point of second spline
    Point(175, 200), // second control point of second spline
    Point(200, 80)}; // end point of second spline
Pen pen(Color(255, 0, 0, 255));
graphics.DrawBeziers(&pen, p, 7);
```

The following illustration shows the connected splines along with the seven points.



Filling Shapes with a Gradient Brush

2/22/2020 • 2 minutes to read • [Edit Online](#)

You can use a gradient brush to fill a shape with a gradually changing color. For example, you can use a horizontal gradient to fill a shape with color that changes gradually as you move from the left edge of the shape to the right edge. Imagine a rectangle with a left edge that is black (represented by red, green, and blue components 0, 0, 0) and a right edge that is red (represented by 255, 0, 0). If the rectangle is 256 pixels wide, the red component of a given pixel will be one greater than the red component of the pixel to its left. The leftmost pixel in a row has color components (0, 0, 0), the second pixel has (1, 0, 0), the third pixel has (2, 0, 0), and so on, until you get to the rightmost pixel, which has color components (255, 0, 0). These interpolated color values make up the color gradient.

A linear gradient changes color as you move horizontally, vertically, or parallel to a specified slanted line. A path gradient changes color as you move about the interior and boundary of a path. You can customize path gradients to achieve a wide variety of effects.

GDI+ provides the [LinearGradientBrush](#) and [PathGradientBrush](#) classes, both of which inherit from the [Brush](#) class.

The following topics cover linear and path gradients in more detail:

- [Creating a Linear Gradient](#)
- [Creating a Path Gradient](#)
- [Applying Gamma Correction to a Gradient](#)

Creating a Linear Gradient

2/22/2020 • 3 minutes to read • [Edit Online](#)

GDI+ provides horizontal, vertical, and diagonal linear gradients. By default, the color in a linear gradient changes uniformly. However, you can customize a linear gradient so that the color changes in a non-uniform fashion.

- [Horizontal Linear Gradients](#)
- [Customizing Linear Gradients](#)
- [Diagonal Linear Gradients](#)

Horizontal Linear Gradients

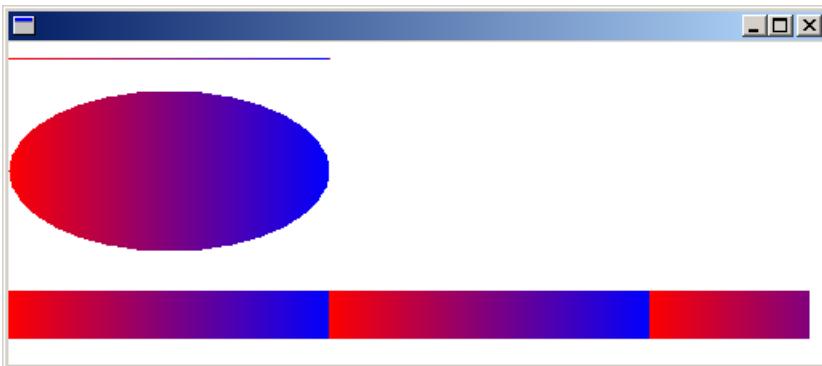
The following example uses a horizontal linear gradient brush to fill a line, an ellipse, and a rectangle:

```
LinearGradientBrush linGrBrush(  
    Point(0, 10),  
    Point(200, 10),  
    Color(255, 255, 0, 0), // opaque red  
    Color(255, 0, 0, 255)); // opaque blue  
  
Pen pen(&linGrBrush);  
  
graphics.DrawLine(&pen, 0, 10, 200, 10);  
graphics.FillEllipse(&linGrBrush, 0, 30, 200, 100);  
graphics.FillRectangle(&linGrBrush, 0, 155, 500, 30);
```

The [LinearGradientBrush](#) constructor receives four arguments: two points and two colors. The first point (0, 10) is associated with the first color (red), and the second point (200, 10) is associated with the second color (blue). As you would expect, the line drawn from (0, 10) to (200, 10) changes gradually from red to blue.

The 10s in the points (50, 10) and (200, 10) are not important. What's important is that the two points have the same second coordinate — the line connecting them is horizontal. The ellipse and the rectangle also change gradually from red to blue as the horizontal coordinate goes from 0 to 200.

The following illustration shows the line, the ellipse, and the rectangle. Note that the color gradient repeats itself as the horizontal coordinate increases beyond 200.



Customizing Linear Gradients

In the preceding example, the color components change linearly as you move from a horizontal coordinate of 0 to a horizontal coordinate of 200. For example, a point whose first coordinate is halfway between 0 and 200 will have a blue component that is halfway between 0 and 255.

GDI+ allows you to adjust the way a color varies from one edge of a gradient to the other. Suppose you want to create a gradient brush that changes from black to red according to the following table.

HORIZONTAL COORDINATE	RGB COMPONENTS
0	(0, 0, 0)
40	(128, 0, 0)
200	(255, 0, 0)

Note that the red component is at half intensity when the horizontal coordinate is only 20 percent of the way from 0 to 200.

The following example calls the [LinearGradientBrush::SetBlend](#) method of a [LinearGradientBrush](#) object to associate three relative intensities with three relative positions. As in the preceding table, a relative intensity of 0.5 is associated with a relative position of 0.2. The code fills an ellipse and a rectangle with the gradient brush.

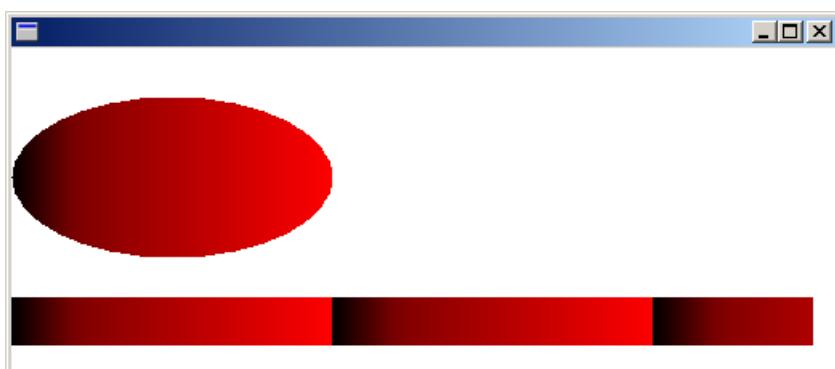
```
LinearGradientBrush linGrBrush(
    Point(0, 10),
    Point(200, 10),
    Color(255, 0, 0, 0), // opaque black
    Color(255, 255, 0, 0)); // opaque red

REAL relativeIntensities[] = {0.0f, 0.5f, 1.0f};
REAL relativePositions[] = {0.0f, 0.2f, 1.0f};

linGrBrush.SetBlend(relativeIntensities, relativePositions, 3);

graphics.FillEllipse(&linGrBrush, 0, 30, 200, 100);
graphics.FillRectangle(&linGrBrush, 0, 155, 500, 30);
```

The following illustration shows the resulting ellipse and rectangle.



Diagonal Linear Gradients

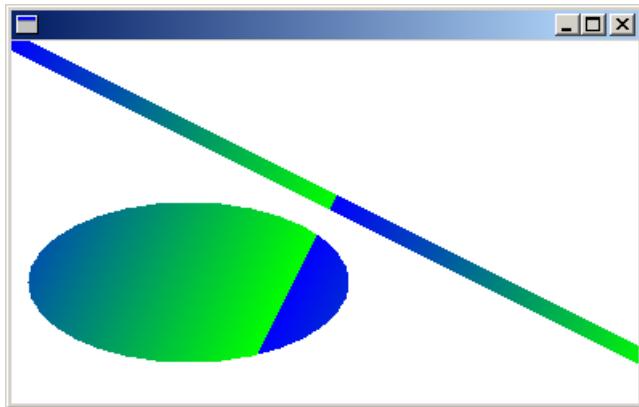
The gradients in the preceding examples have been horizontal; that is, the color changes gradually as you move along any horizontal line. You can also define vertical gradients and diagonal gradients. The following code passes the points (0, 0) and (200, 100) to a [LinearGradientBrush](#) constructor. The color blue is associated with (0, 0), and the color green is associated with (200, 100). A line (with pen width 10) and an ellipse are filled with the linear gradient brush.

```
LinearGradientBrush linGrBrush(
    Point(0, 0),
    Point(200, 100),
    Color(255, 0, 0, 255), // opaque blue
    Color(255, 0, 255, 0)); // opaque green

Pen pen(&linGrBrush, 10);

graphics.DrawLine(&pen, 0, 0, 600, 300);
graphics.FillEllipse(&linGrBrush, 10, 100, 200, 100);
```

The following illustration shows the line and the ellipse. Note that the color in the ellipse changes gradually as you move along any line that is parallel to the line passing through (0, 0) and (200, 100).



Creating a Path Gradient

11/2/2020 • 9 minutes to read • [Edit Online](#)

The **PathGradientBrush** class allows you to customize the way you fill a shape with gradually changing colors. A **PathGradientBrush** object has a boundary path and a center point. You can specify one color for the center point and another color for the boundary. You can also specify separate colors for each of several points along the boundary.

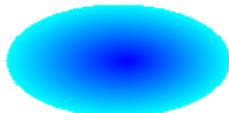
NOTE

In GDI+, a path is a sequence of lines and curves maintained by a **GraphicsPath** object. For more information about GDI+ paths, see [Paths](#) and [Constructing and Drawing Paths](#).

The following example fills an ellipse with a path gradient brush. The center color is set to blue and the boundary color is set to aqua.

```
// Create a path that consists of a single ellipse.  
GraphicsPath path;  
path.AddEllipse(0, 0, 140, 70);  
  
// Use the path to construct a brush.  
PathGradientBrush pthGrBrush(&path);  
  
// Set the color at the center of the path to blue.  
pthGrBrush.SetCenterColor(Color(255, 0, 0, 255));  
  
// Set the color along the entire boundary of the path to aqua.  
Color colors[] = {Color(255, 0, 255, 255)};  
int count = 1;  
pthGrBrush.SetSurroundColors(colors, &count);  
  
graphics.FillEllipse(&pthGrBrush, 0, 0, 140, 70);
```

The following illustration shows the filled ellipse.



By default, a path gradient brush does not extend outside the boundary of the path. If you use the path gradient brush to fill a shape that extends beyond the boundary of the path, the area of the screen outside the path will not be filled. The following illustration shows what happens if you change the **Graphics::FillEllipse** call in the preceding code to `graphics.FillRectangle(&pthGrBrush, 0, 10, 200, 40)`.



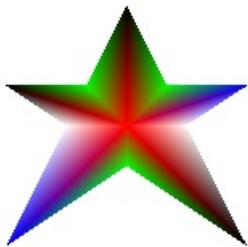
Specifying Points on the Boundary

The following example constructs a path gradient brush from a star-shaped path. The code calls the **PathGradientBrush::SetCenterColor** method to set the color at the centroid of the star to red. Then the code

calls the [PathGradientBrush::SetSurroundColors](#) method to specify various colors (stored in the [colors](#) array) at the individual points in the [points](#) array. The final code statement fills the star-shaped path with the path gradient brush.

```
// Put the points of a polygon in an array.  
Point points[] = {Point(75, 0), Point(100, 50),  
    Point(150, 50), Point(112, 75),  
    Point(150, 150), Point(75, 100),  
    Point(0, 150), Point(37, 75),  
    Point(0, 50), Point(50, 50)};  
  
// Use the array of points to construct a path.  
GraphicsPath path;  
path.AddLines(points, 10);  
  
// Use the path to construct a path gradient brush.  
PathGradientBrush pthGrBrush(&path);  
  
// Set the color at the center of the path to red.  
pthGrBrush.SetCenterColor(Color(255, 255, 0, 0));  
  
// Set the colors of the points in the array.  
Color colors[] = {Color(255, 0, 0, 0), Color(255, 0, 255, 0),  
    Color(255, 0, 0, 255), Color(255, 255, 255, 255),  
    Color(255, 0, 0, 0), Color(255, 0, 255, 0),  
    Color(255, 0, 0, 255), Color(255, 255, 255, 255),  
    Color(255, 0, 0, 0), Color(255, 0, 255, 0)};  
  
int count = 10;  
pthGrBrush.SetSurroundColors(colors, &count);  
  
// Fill the path with the path gradient brush.  
graphics.FillPath(&pthGrBrush, &path);
```

The following illustration shows the filled star.



The following example constructs a path gradient brush based on an array of points. A color is assigned to each of the five points in the array. If you were to connect the five points by straight lines, you would get a five-sided polygon. A color is also assigned to the center (centroid) of that polygon — in this example, the center (80, 75) is set to white. The final code statement in the example fills a rectangle with the path gradient brush.

The color used to fill the rectangle is white at (80, 75) and changes gradually as you move away from (80, 75) toward the points in the array. For example, as you move from (80, 75) to (0, 0), the color changes gradually from white to red, and as you move from (80, 75) to (160, 0), the color changes gradually from white to green.

```

// Construct a path gradient brush based on an array of points.
PointF ptsF[] = {PointF(0.0f, 0.0f),
                 PointF(160.0f, 0.0f),
                 PointF(160.0f, 200.0f),
                 PointF(80.0f, 150.0f),
                 PointF(0.0f, 200.0f)};

PathGradientBrush pBrush(ptsF, 5);

// An array of five points was used to construct the path gradient
// brush. Set the color of each point in that array.
Color colors[] = {Color(255, 255, 0, 0), // (0, 0) red
                  Color(255, 0, 255, 0), // (160, 0) green
                  Color(255, 0, 255, 0), // (160, 200) green
                  Color(255, 0, 0, 255), // (80, 150) blue
                  Color(255, 255, 0, 0)}; // (0, 200) red

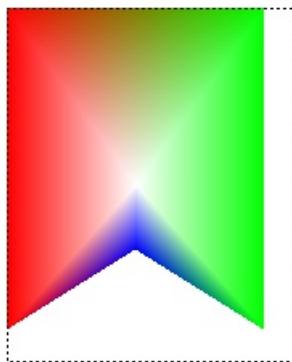
int count = 5;
pBrush.SetSurroundColors(colors, &count);

// Set the center color to white.
pBrush.SetCenterColor(Color(255, 255, 255, 255));

// Use the path gradient brush to fill a rectangle.
graphics.FillRectangle(&pBrush, Rect(0, 0, 180, 220));

```

Note that there is no [GraphicsPath](#) object in the preceding code. The particular [PathGradientBrush](#) constructor in the example receives a pointer to an array of points but does not require a [GraphicsPath](#) object. Also, note that the path gradient brush is used to fill a rectangle, not a path. The rectangle is larger than the path used to define the brush, so some of the rectangle is not painted by the brush. The following illustration shows the rectangle (dotted line) and the portion of the rectangle painted by the path gradient brush.



Customizing a Path Gradient

One way to customize a path gradient brush is to set its focus scales. The focus scales specify an inner path that lies inside the main path. The center color is displayed everywhere inside that inner path rather than only at the center point. To set the focus scales of a path gradient brush, call the [PathGradientBrush::SetFocusScales](#) method.

The following example creates a path gradient brush based on an elliptical path. The code sets the boundary color to blue, sets the center color to aqua, and then uses the path gradient brush to fill the elliptical path.

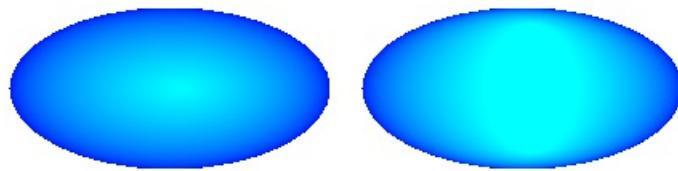
Next the code sets the focus scales of the path gradient brush. The x focus scale is set to 0.3, and the y focus scale is set to 0.8. The code calls the [Graphics::TranslateTransform](#) method of a [Graphics](#) object so that the subsequent call to [Graphics::FillPath](#) fills an ellipse that sits to the right of the first ellipse.

To see the effect of the focus scales, imagine a small ellipse that shares its center with the main ellipse. The small (inner) ellipse is the main ellipse scaled (about its center) horizontally by a factor of 0.3 and vertically by a factor of 0.8. As you move from the boundary of the outer ellipse to the boundary of the inner ellipse, the color changes

gradually from blue to aqua. As you move from the boundary of the inner ellipse to the shared center, the color remains aqua.

```
// Create a path that consists of a single ellipse.  
GraphicsPath path;  
path.AddEllipse(0, 0, 200, 100);  
  
// Create a path gradient brush based on the elliptical path.  
PathGradientBrush pthGrBrush(&path);  
pthGrBrush.SetGammaCorrection(TRUE);  
  
// Set the color along the entire boundary to blue.  
Color color(Color(255, 0, 0, 255));  
INT num = 1;  
pthGrBrush.SetSurroundColors(&color, &num);  
  
// Set the center color to aqua.  
pthGrBrush.SetCenterColor(Color(255, 0, 255, 255));  
  
// Use the path gradient brush to fill the ellipse.  
graphics.FillPath(&pthGrBrush, &path);  
  
// Set the focus scales for the path gradient brush.  
pthGrBrush.SetFocusScales(0.3f, 0.8f);  
  
// Use the path gradient brush to fill the ellipse again.  
// Show this filled ellipse to the right of the first filled ellipse.  
graphics.TranslateTransform(220.0f, 0.0f);  
graphics.FillPath(&pthGrBrush, &path);
```

The following illustration shows the output of the preceding code. The ellipse on the left is aqua only at the center point. The ellipse on the right is aqua everywhere inside the inner path.



Another way to customize a path gradient brush is to specify an array of preset colors and an array of interpolation positions.

The following example creates a path gradient brush based on a triangle. The code calls the [PathGradientBrush::SetInterpolationColors](#) method of the path gradient brush to specify an array of interpolation colors (dark green, aqua, blue) and an array of interpolation positions (0, 0.25, 1). As you move from the boundary of the triangle to the center point, the color changes gradually from dark green to aqua and then from aqua to blue. The change from dark green to aqua happens in 25 percent of the distance from dark green to blue.

```

// Vertices of the triangle
Point points[] = {Point(100, 0),
                  Point(200, 200),
                  Point(0, 200)};

// No GraphicsPath object is created. The PathGradient
// brush is constructed directly from the array of points.
PathGradientBrush pthGrBrush(points, 3);

Color presetColors[] = {
    Color(255, 0, 128, 0), // Dark green
    Color(255, 0, 255, 255), // Aqua
    Color(255, 0, 0, 255)}; // Blue

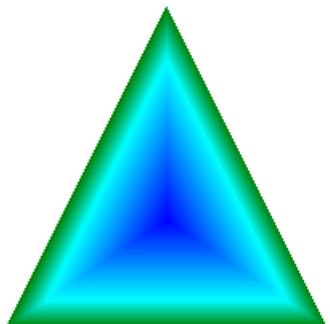
REAL interpPositions[] = {
    0.0f, // Dark green is at the boundary of the triangle.
    0.25f, // Aqua is 25 percent of the way from the boundary
            // to the center point.
    1.0f}; // Blue is at the center point.

pthGrBrush.SetInterpolationColors(presetColors, interpPositions, 3);

// Fill a rectangle that is larger than the triangle
// specified in the Point array. The portion of the
// rectangle outside the triangle will not be painted.
graphics.FillRectangle(&pthGrBrush, 0, 0, 200, 200);

```

The following illustration shows the output of the preceding code.



Setting the Center Point

By default, the center point of a path gradient brush is at the centroid of the path used to construct the brush. You can change the location of the center point by calling the [PathGradientBrush::SetCenterPoint](#) method of the [PathGradientBrush](#) class.

The following example creates a path gradient brush based on an ellipse. The center of the ellipse is at (70, 35), but the center point of the path gradient brush is set to (120, 40).

```

// Create a path that consists of a single ellipse.
GraphicsPath path;
path.AddEllipse(0, 0, 140, 70);

// Use the path to construct a brush.
PathGradientBrush pthGrBrush(&path);

// Set the center point to a location that is not the centroid of the path.
pthGrBrush.SetCenterPoint(Point(120, 40));

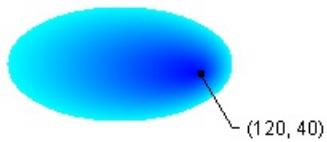
// Set the color at the center point to blue.
pthGrBrush.SetCenterColor(Color(255, 0, 0, 255));

// Set the color along the entire boundary of the path to aqua.
Color colors[] = {Color(255, 0, 255, 255)};
int count = 1;
pthGrBrush.SetSurroundColors(colors, &count);

graphics.FillEllipse(&pthGrBrush, 0, 0, 140, 70);

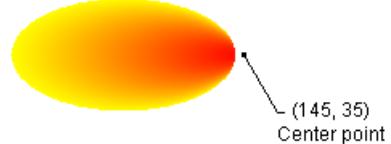
```

The following illustration shows the filled ellipse and the center point of the path gradient brush.



You can set the center point of a path gradient brush to a location outside the path that was used to construct the brush. In the preceding code, if you replace the call to [PathGradientBrush::SetCenterPoint](#) with

`pthGrBrush.SetCenterPoint(Point(145, 35))`, you will get the following result.



In the preceding illustration, the points at the far right of the ellipse are not pure blue (although they are very close). The colors in the gradient are positioned as if the fill had been allowed to reach the point (145, 35), the color would have reached pure blue (0, 0, 255). But the fill never reaches (145, 35) because a path gradient brush paints only inside its path.

Applying Gamma Correction to a Gradient

2/22/2020 • 2 minutes to read • [Edit Online](#)

You can enable gamma correction for a gradient brush by passing **TRUE** to the [PathGradientBrush::SetGammaCorrection](#) method of that brush. You can disable gamma correction by passing **FALSE** to the [PathGradientBrush::SetGammaCorrection](#) method. Gamma correction is disabled by default.

The following example creates a linear gradient brush and uses that brush to fill two rectangles. The first rectangle is filled without gamma correction and the second rectangle is filled with gamma correction.

```
LinearGradientBrush linGrBrush(  
    Point(0, 10),  
    Point(200, 10),  
    Color(255, 255, 0, 0), // Opaque red  
    Color(255, 0, 0, 255)); // Opaque blue  
  
graphics.FillRectangle(&linGrBrush, 0, 0, 200, 50);  
linGrBrush.SetGammaCorrection(TRUE);  
graphics.FillRectangle(&linGrBrush, 0, 60, 200, 50);
```

The following illustration shows the two filled rectangles. The top rectangle, which does not have gamma correction, appears dark in the middle. The bottom rectangle, which has gamma correction, appears to have more uniform intensity.



The following example creates a path gradient brush based on a star-shaped path. The code uses the path gradient brush with gamma correction disabled (the default) to fill the path. Then the code passes **TRUE** to the [PathGradientBrush::SetGammaCorrection](#) method to enable gamma correction for the path gradient brush. The call to [Graphics::TranslateTransform](#) sets the world transformation of a [Graphics](#) object so that the subsequent call to [Graphics::FillPath](#) fills a star that sits to the right of the first star.

```

// Put the points of a polygon in an array.
Point points[] = {Point(75, 0), Point(100, 50),
                  Point(150, 50), Point(112, 75),
                  Point(150, 150), Point(75, 100),
                  Point(0, 150), Point(37, 75),
                  Point(0, 50), Point(50, 50)};

// Use the array of points to construct a path.
GraphicsPath path;
path.AddLines(points, 10);

// Use the path to construct a path gradient brush.
PathGradientBrush pthGrBrush(&path);

// Set the color at the center of the path to red.
pthGrBrush.SetCenterColor(Color(255, 255, 0, 0));

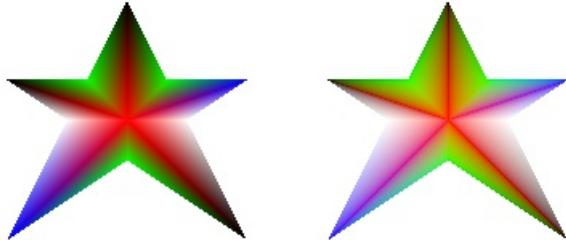
// Set the colors of the points in the array.
Color colors[] = {Color(255, 0, 0, 0), Color(255, 0, 255, 0),
                  Color(255, 0, 0, 255), Color(255, 255, 255, 255),
                  Color(255, 0, 0, 0), Color(255, 0, 255, 0),
                  Color(255, 0, 0, 255), Color(255, 255, 255, 255),
                  Color(255, 0, 0, 0), Color(255, 0, 255, 0)};

int count = 10;
pthGrBrush.SetSurroundColors(colors, &count);

// Fill the path with the path gradient brush.
graphics.FillPath(&pthGrBrush, &path);
pthGrBrush.SetGammaCorrection(TRUE);
graphics.TranslateTransform(200.0f, 0.0f);
graphics.FillPath(&pthGrBrush, &path);

```

The following illustration shows the output of the preceding code. The star on the right has gamma correction. Note that the star on the left, which does not have gamma correction, has areas that appear dark.



Constructing and Drawing Paths

2/22/2020 • 2 minutes to read • [Edit Online](#)

A path is a sequence of graphics primitives (lines, rectangles, curves, text, and the like) that can be manipulated and drawn as a single unit. A path can be divided into *figures* that are either open or closed. A figure can contain several primitives.

You can draw a path by calling the [Graphics::DrawPath](#) method of the [Graphics](#) class, and you can fill a path by calling the [Graphics::FillPath](#) method of the [Graphics](#) class.

The following topics cover paths in more detail:

- [Creating Figures from Lines, Curves, and Shapes](#)
- [Filling Open Figures](#)

Creating Figures from Lines, Curves, and Shapes

11/2/2020 • 2 minutes to read • [Edit Online](#)

To create a path, construct a **GraphicsPath** object, and then call methods, such as **AddLine** and **AddCurve**, to add primitives to the path.

The following example creates a path that has a single arc. The arc has a sweep angle of -180 degrees, which is counterclockwise in the default coordinate system.

```
Pen pen(Color(255, 255, 0, 0));
GraphicsPath path;
path.AddArc(175, 50, 50, 50, 0, -180);
graphics.DrawPath(&pen, &path);
```

The following example creates a path that has two figures. The first figure is an arc followed by a line. The second figure is a line followed by a curve, followed by a line. The first figure is left open, and the second figure is closed.

```
Point points[] = {Point(40, 60), Point(50, 70), Point(30, 90)};

Pen pen(Color(255, 255, 0, 0), 2);
GraphicsPath path;

// The first figure is started automatically, so there is
// no need to call StartFigure here.
path.AddArc(175, 50, 50, 50, 0.0f, -180.0f);
path.AddLine(100, 0, 250, 20);

path.StartFigure();
path.AddLine(50, 20, 5, 90);
path.AddCurve(points, 3);
path.AddLine(50, 150, 150, 180);
path.CloseFigure();

graphics.DrawPath(&pen, &path);
```

In addition to adding lines and curves to paths, you can add closed shapes: rectangles, ellipses, pies, and polygons. The following example creates a path that has two lines, a rectangle, and an ellipse. The code uses a pen to draw the path and a brush to fill the path.

```
GraphicsPath path;
Pen         pen(Color(255, 255, 0, 0), 2);
SolidBrush   brush(Color(255, 0, 0, 200));

path.AddLine(10, 10, 100, 40);
path.AddLine(100, 60, 30, 60);
path.AddRectangle(Rect(50, 35, 20, 40));
path.AddEllipse(10, 75, 40, 30);

graphics.DrawPath(&pen, &path);
graphics.FillPath(&brush, &path);
```

The path in the preceding example has three figures. The first figure consists of the two lines, the second figure consists of the rectangle, and the third figure consists of the ellipse. Even when there are no calls to **GraphicsPath::CloseFigure** or **GraphicsPath::StartFigure**, intrinsically closed shapes, such as rectangles and ellipses, are considered separate figures.

Filling Open Figures

2/22/2020 • 2 minutes to read • [Edit Online](#)

You can fill a path by passing the address of a **GraphicsPath** object to the **Graphics::FillPath** method. The **Graphics::FillPath** method fills the path according to the fill mode (alternate or winding) currently set for the path. If the path has any open figures, the path is filled as if those figures were closed. GDI+ closes a figure by drawing a straight line from its ending point to its starting point.

The following example creates a path that has one open figure (an arc) and one closed figure (an ellipse). The **Graphics::FillPath** method fills the path according to the default fill mode, which is **FillModeAlternate**.

```
GraphicsPath path;

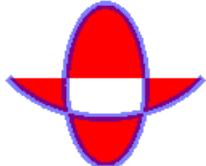
// Add an open figure.
path.AddArc(0, 0, 150, 120, 30, 120);

// Add an intrinsically closed figure.
path.AddEllipse(50, 50, 50, 100);

Pen pen(Color(128, 0, 0, 255), 5);
SolidBrush brush(Color(255, 255, 0, 0));

// The fill mode is FillModeAlternate by default.
graphics.FillPath(&brush, &path);
graphics.DrawPath(&pen, &path);
```

The following illustration shows the output of the preceding code. Note that path is filled (according to **FillModeAlternate**) as if the open figure were closed by a straight line from its ending point to its starting point.



Using Graphics Containers

11/2/2020 • 2 minutes to read • [Edit Online](#)

A **Graphics** object provides methods such as [DrawLine](#), [DrawImage](#), and [DrawString](#) for displaying vector images, raster images, and text. A **Graphics** object also has several properties that influence the quality and orientation of the items that are drawn. For example, the smoothing mode property determines whether antialiasing is applied to lines and curves, and the world transformation property influences the position and rotation of the items that are drawn.

A **Graphics** object is often associated with a particular display device. When you use a **Graphics** object to draw in a window, the **Graphics** object is also associated with that particular window.

A **Graphics** object can be thought of as a container because it holds a set of properties that influence drawing, and it is linked to device-specific information. You can create a secondary container within an existing **Graphics** object by calling the [BeginContainer](#) method of that **Graphics** object.

The following topics cover **Graphics** objects and nested containers in more detail:

- [The State of a Graphics Object](#)
- [Nested Graphics Containers](#)

The State of a Graphics Object

11/2/2020 • 3 minutes to read • [Edit Online](#)

The **Graphics** class is at the heart of Windows GDI+. To draw anything, you create a **Graphics** object, set its properties, and call its methods ([DrawLine](#), [DrawImage](#), [DrawString](#), and the like).

The following example constructs a **Graphics** object and a **Pen** object and then calls the [Graphics::DrawRectangle](#) method of the **Graphics** object:

```
HDC           hdc;
PAINTSTRUCT  ps;

hdc = BeginPaint(hWnd, &ps);
{
    Graphics graphics(hdc);
    Pen pen(Color(255, 0, 0, 255)); // opaque blue
    graphics.DrawRectangle(&pen, 10, 10, 200, 100);
}
EndPaint(hWnd, &ps);
```

In the preceding code, the [BeginPaint](#) method returns a handle to a device context, and that handle is passed to the **Graphics** constructor. A device context is a structure (maintained by Windows) that holds information about the particular display device being used.

Graphics State

A **Graphics** object does more than provide drawing methods, such as [DrawLine](#) and [DrawRectangle](#). A **Graphics** object also maintains graphics state, which can be divided into the following categories:

- A link to a device context
- Quality settings
- Transformations
- A clipping region

Device Context

As an application programmer, you don't have to think about the interaction between a **Graphics** object and its device context. This interaction is handled by GDI+ behind the scenes.

Quality Settings

A **Graphics** object has several properties that influence the quality of the items that are drawn on the screen. You can view and manipulate these properties by calling get and set methods. For example, you can call the [Graphics::SetTextRenderingHint](#) method to specify the type of antialiasing (if any) applied to text. Other set methods that influence quality are [Graphics::SetSmoothingMode](#), [Graphics::SetCompositingMode](#), [Graphics::SetCompositingQuality](#), and [Graphics::SetInterpolationMode](#).

The following example draws two ellipses, one with the smoothing mode set to [SmoothingModeAntiAlias](#) and one with the smoothing mode set to [SmoothingModeHighSpeed](#):

```

Graphics graphics(hdc);
Pen pen(Color(255, 0, 255, 0)); // opaque green

graphics.SetSmoothingMode(SmoothingModeAntiAlias);
graphics.DrawEllipse(&pen, 0, 0, 200, 100);
graphics.SetSmoothingMode(SmoothingModeHighSpeed);
graphics.DrawEllipse(&pen, 0, 150, 200, 100);

```

Transformations

A [Graphics](#) object maintains two transformations (world and page) that are applied to all items drawn by that [Graphics](#) object. Any affine transformation can be stored in the world transformation. Affine transformations include scaling, rotating, reflecting, skewing, and translating. The page transformation can be used for scaling and for changing units (for example, pixels to inches). For more information on transformations, see [Coordinate Systems and Transformations](#).

The following example sets the world and page transformations of a [Graphics](#) object. The world transformation is set to a 30-degree rotation. The page transformation is set so that the coordinates passed to the second [Graphics::DrawEllipse](#) will be treated as millimeters instead of pixels. The code makes two identical calls to the [Graphics::DrawEllipse](#) method. The world transformation is applied to the first [Graphics::DrawEllipse](#) call, and both transformations (world and page) are applied to the second [Graphics::DrawEllipse](#) call.

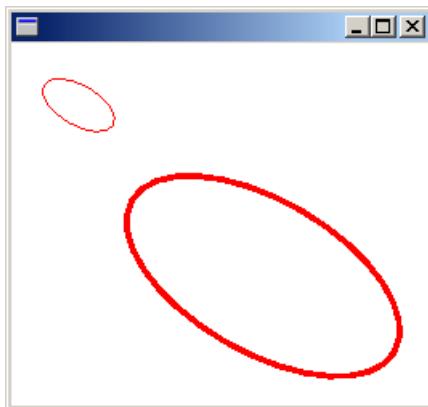
```

Graphics graphics(hdc);
Pen pen(Color(255, 255, 0, 0));

graphics.ResetTransform();
graphics.RotateTransform(30.0f);           // World transformation
graphics.DrawEllipse(&pen, 30, 0, 50, 25);
graphics.SetPageUnit(UnitMillimeter);    // Page transformation
graphics.DrawEllipse(&pen, 30, 0, 50, 25);

```

The following illustration shows the two ellipses. Note that the 30-degree rotation is about the origin of the coordinate system (upper-left corner of the client area), not about the centers of the ellipses. Also note that the pen width of 1 means 1 pixel for the first ellipse and 1 millimeter for the second ellipse.



Clipping Region

A [Graphics](#) object maintains a clipping region that applies to all items drawn by that [Graphics](#) object. You can set the clipping region by calling the [SetClip](#) method.

The following example creates a plus-shaped region by forming the union of two rectangles. That region is designated as the clipping region of a [Graphics](#) object. Then the code draws two lines that are restricted to the interior of the clipping region.

```
Graphics graphics(hdc);
Pen pen(Color(255, 255, 0, 0), 5); // opaque red, width 5
SolidBrush brush(Color(255, 180, 255)); // opaque aqua

// Create a plus-shaped region by forming the union of two rectangles.
Region region(Rect(50, 0, 50, 150));
region.Union(Rect(0, 50, 150, 50));
graphics.FillRegion(&brush, &region);

// Set the clipping region.
graphics.SetClip(&region);

// Draw two clipped lines.
graphics.DrawLine(&pen, 0, 30, 150, 160);
graphics.DrawLine(&pen, 40, 20, 190, 150);
```

The following illustration shows the clipped lines.



Nested Graphics Containers

11/2/2020 • 4 minutes to read • [Edit Online](#)

Windows GDI+ provides containers that you can use to temporarily replace or augment part of the state in a **Graphics** object. You create a container by calling the **Graphics::BeginContainer** method of a **Graphics** object. You can call **Graphics::BeginContainer** repeatedly to form nested containers.

Transformations in Nested Containers

The following example creates a **Graphics** object and a container within that **Graphics** object. The world transformation of the **Graphics** object is a translation 100 units in the x direction and 80 units in the y direction. The world transformation of the container is a 30-degree rotation. The code makes the call

```
DrawRectangle(&pen, -60, -30, 120, 60)
```

twice. The first call to **Graphics::DrawRectangle** is *inside the container*, that is, the call is in between the calls to **Graphics::BeginContainer** and **Graphics::EndContainer**. The second call to **Graphics::DrawRectangle** is after the call to **Graphics::EndContainer**.

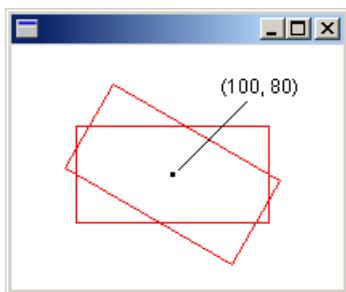
```
Graphics      graphics(hdc);
Pen          pen(Color(255, 255, 0, 0));
GraphicsContainer  graphicsContainer;

graphics.TranslateTransform(100.0f, 80.0f);

graphicsContainer = graphics.BeginContainer();
graphics.RotateTransform(30.0f);
graphics.DrawRectangle(&pen, -60, -30, 120, 60);
graphics.EndContainer(graphicsContainer);

graphics.DrawRectangle(&pen, -60, -30, 120, 60);
```

In the preceding code, the rectangle drawn from inside the container is transformed first by the world transformation of the container (rotation) and then by the world transformation of the **Graphics** object (translation). The rectangle drawn from outside the container is transformed only by the world transformation of the **Graphics** object (translation). The following illustration shows the two rectangles.



Clipping in Nested Containers

The following example illustrates how nested containers handle clipping regions. The code creates a **Graphics** object and a container within that **Graphics** object. The clipping region of the **Graphics** object is a rectangle, and

the clipping region of the container is an ellipse. The code makes two calls to the **Graphics::DrawLine** method. The first call to **Graphics::DrawLine** is inside the container, and the second call to **Graphics::DrawLine** is outside the container (after the call to **Graphics::EndContainer**). The first line is clipped by the intersection of the two clipping regions. The second line is clipped only by the rectangular clipping region of the **Graphics** object.

```
Graphics         graphics(hdc);
GraphicsContainer graphicsContainer;
Pen             redPen(Color(255, 255, 0, 0), 2);
Pen             bluePen(Color(255, 0, 0, 255), 2);
SolidBrush       aquaBrush(Color(255, 180, 255, 255));
SolidBrush       greenBrush(Color(255, 150, 250, 130));

graphics.SetClip(Rect(50, 65, 150, 120));
graphics.FillRectangle(&aquaBrush, 50, 65, 150, 120);

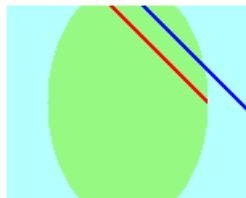
graphicsContainer = graphics.BeginContainer();
// Create a path that consists of a single ellipse.
GraphicsPath path;
path.AddEllipse(75, 50, 100, 150);

// Construct a region based on the path.
Region region(&path);
graphics.FillRegion(&greenBrush, &region);

graphics.SetClip(&region);
graphics.DrawLine(&redPen, 50, 0, 350, 300);
graphics.EndContainer(graphicsContainer);

graphics.DrawLine(&bluePen, 70, 0, 370, 300);
```

The following illustration shows the two clipped lines.



As the two preceding examples show, transformations and clipping regions are cumulative in nested containers. If you set the world transformations of the container and the **Graphics** object, both transformations will apply to items drawn from inside the container. The transformation of the container will be applied first, and the transformation of the **Graphics** object will be applied second. If you set the clipping regions of the container and the **Graphics** object, items drawn from inside the container will be clipped by the intersection of the two clipping regions.

Quality Settings in Nested Containers

Quality settings (**SmoothingMode**, **TextRenderingHint**, and the like) in nested containers are not cumulative; rather, the quality settings of the container temporarily replace the quality settings of a **Graphics** object. When you create a new container, the quality settings for that container are set to default values. For example, suppose you have a **Graphics** object with a smoothing mode of **SmoothingModeAntiAlias**. When you create a container, the smoothing mode inside the container is the default smoothing mode. You are free to set the smoothing mode of the container, and any items drawn from inside the container will be drawn according to the mode you set. Items drawn after the call to **Graphics::EndContainer** will be drawn according to the smoothing mode (**SmoothingModeAntiAlias**) that was in place before the call to **Graphics::BeginContainer**.

Several Layers of Nested Containers

You are not limited to one container in a [Graphics](#) object. You can create a sequence of containers, each nested in the preceding, and you can specify the world transformation, clipping region, and quality settings of each of those nested containers. If you call a drawing method from inside the innermost container, the transformations will be applied in order, starting with the innermost container and ending with the outermost container. Items drawn from inside the innermost container will be clipped by the intersection of all the clipping regions.

The following example creates a [Graphics](#) object and sets its text rendering hint to [TextRenderingHintAntiAlias](#). The code creates two containers, one nested within the other. The text rendering hint of the outer container is set to [TextRenderingHintSingleBitPerPixel](#), and the text rendering hint of the inner container is set to [TextRenderingHintAntiAlias](#). The code draws three strings: one from the inner container, one from the outer container, and one from the [Graphics](#) object itself.

```
Graphics graphics(hdc);
GraphicsContainer innerContainer;
GraphicsContainer outerContainer;
SolidBrush brush(Color(255, 0, 0, 255));
FontFamily fontFamily(L"Times New Roman");
Font font(&fontFamily, 36, FontStyleRegular, UnitPixel);

graphics.SetTextRenderingHint(TextRenderingHintAntiAlias);

outerContainer = graphics.BeginContainer();

graphics.SetTextRenderingHint(TextRenderingHintSingleBitPerPixel);

innerContainer = graphics.BeginContainer();
graphics.SetTextRenderingHint(TextRenderingHintAntiAlias);
graphics.DrawString(L"Inner Container", 15, &font,
    PointF(20, 10), &brush);
graphics.EndContainer(innerContainer);

graphics.DrawString(L"Outer Container", 15, &font, PointF(20, 50), &brush);

graphics.EndContainer(outerContainer);

graphics.DrawString(L"Graphics Object", 15, &font, PointF(20, 90), &brush);
```

The following illustration shows the three strings. The strings drawn from the inner container and the [Graphics](#) object are smoothed by antialiasing. The string drawn from the outer container is not smoothed by antialiasing because of the [TextRenderingHintSingleBitPerPixel](#) setting.

**Inner Container
Outer Container
Graphics Object**

Transformations

2/22/2020 • 2 minutes to read • [Edit Online](#)

Affine transformations include rotating, scaling, reflecting, shearing, and translating. In Windows GDI+, the [Matrix](#) class provides the foundation for performing affine transformations on vector drawings, images, and text.

The following topics cover transformations in more detail:

- [Using the World Transformation](#)
- [Why Transformation Order Is Significant](#)

Using the World Transformation

2/22/2020 • 2 minutes to read • [Edit Online](#)

The world transformation is a property of the **Graphics** class. The numbers that specify the world transformation are stored in a **Matrix** object, which represents a 3×3 matrix. The **Matrix** and **Graphics** classes have several methods for setting the numbers in the world transformation matrix. The examples in this section manipulate rectangles because rectangles are easy to draw and it is easy to see the effects of transformations on rectangles.

We start by creating a 50 by 50 rectangle and locating it at the origin (0, 0). The origin is at the upper-left corner of the client area.

```
Rect rect(0, 0, 50, 50);
Pen pen(Color(255, 255, 0, 0), 0);
graphics.DrawRectangle(&pen, rect);
```

The following code applies a scaling transformation that expands the rectangle by a factor of 1.75 in the x direction and shrinks the rectangle by a factor of 0.5 in the y direction:

```
Rect rect(0, 0, 50, 50);
Pen pen(Color(255, 255, 0, 0), 0);
graphics.ScaleTransform(1.75f, 0.5f);
graphics.DrawRectangle(&pen, rect);
```

The result is a rectangle that is longer in the x direction and shorter in the y direction than the original.

To rotate the rectangle instead of scaling it, use the following code instead of the preceding code:

```
Rect rect(0, 0, 50, 50);
Pen pen(Color(255, 255, 0, 0), 0);
graphics.RotateTransform(28.0f);
graphics.DrawRectangle(&pen, rect);
```

To translate the rectangle, use the following code:

```
Rect rect(0, 0, 50, 50);
Pen pen(Color(255, 255, 0, 0), 0);
graphics.TranslateTransform(150.0f, 150.0f);
graphics.DrawRectangle(&pen, rect);
```

Why Transformation Order Is Significant

2/22/2020 • 2 minutes to read • [Edit Online](#)

A single [Matrix](#) object can store a single transformation or a sequence of transformations. The latter is called a *composite transformation*. The matrix of a composite transformation is obtained by multiplying the matrices of the individual transformations.

In a composite transformation, the order of the individual transformations is important. For example, if you first rotate, then scale, then translate, you get a different result than if you first translate, then rotate, then scale. In Windows GDI+, composite transformations are built from left to right. If S, R, and T are scale, rotation, and translation matrices respectively, then the product SRT (in that order) is the matrix of the composite transformation that first scales, then rotates, then translates. The matrix produced by the product SRT is different from the matrix produced by the product TRS.

One reason order is significant is that transformations like rotation and scaling are done with respect to the origin of the coordinate system. Scaling an object that is centered at the origin produces a different result than scaling an object that has been moved away from the origin. Similarly, rotating an object that is centered at the origin produces a different result than rotating an object that has been moved away from the origin.

The following example combines scaling, rotation and translation (in that order) to form a composite transformation. The argument [MatrixOrderAppend](#) passed to the [Graphics::RotateTransform](#) method specifies that the rotation will follow the scaling. Likewise, the argument [MatrixOrderAppend](#) passed to the [Graphics::TranslateTransform](#) method specifies that the translation will follow the rotation.

```
Rect rect(0, 0, 50, 50);
Pen pen(Color(255, 255, 0, 0), 0);
graphics.ScaleTransform(1.75f, 0.5f);
graphics.RotateTransform(28.0f, MatrixOrderAppend);
graphics.TranslateTransform(150.0f, 150.0f, MatrixOrderAppend);
graphics.DrawRectangle(&pen, rect);
```

The following example makes the same method calls as the previous example, but the order of the calls is reversed. The resulting order of operations is first translate, then rotate, then scale, which produces a very different result than first scale, then rotate, then translate:

```
Rect rect(0, 0, 50, 50);
Pen pen(Color(255, 255, 0, 0), 0);
graphics.TranslateTransform(150.0f, 150.0f);
graphics.RotateTransform(28.0f, MatrixOrderAppend);
graphics.ScaleTransform(1.75f, 0.5f, MatrixOrderAppend);
graphics.DrawRectangle(&pen, rect);
```

One way to reverse the order of the individual transformations in a composite transformation is to reverse the order of a sequence of method calls. A second way to control the order of operations is to change the matrix order argument. The following example is the same as the previous example, except that [MatrixOrderAppend](#) has been changed to [MatrixOrderPrepend](#). The matrix multiplication is done in the order SRT, where S, R, and T are the matrices for scale, rotate, and translate, respectively. The order of the composite transformation is first scale, then rotate, then translate.

```
Rect rect(0, 0, 50, 50);
Pen pen(Color(255, 255, 0, 0), 0);
graphics.TranslateTransform(150.0f, 150.0f, MatrixOrderPrepend);
graphics.RotateTransform(28.0f, MatrixOrderPrepend);
graphics.ScaleTransform(1.75f, 0.5f, MatrixOrderPrepend);
graphics.DrawRectangle(&pen, rect);
```

The result of the preceding example is the same result that we achieved in the first example of this section. This is because we reversed both the order of the method calls and the order of the matrix multiplication.

Using Regions (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

The Windows GDI+ **Region** class allows you to define a custom shape. The shape can be made up of lines, polygons, and curves.

Two common uses for regions are hit testing and clipping. Hit testing is determining whether the mouse was clicked in a certain region of the screen. Clipping is restricting drawing to a certain region.

The following topics cover regions in more detail:

- [Hit Testing with a Region](#)
- [Clipping with a Region](#)

Hit Testing with a Region

2/22/2020 • 2 minutes to read • [Edit Online](#)

The purpose of hit testing is to determine whether the cursor is over a given object, such as an icon or a button. The following example creates a plus-shaped region by forming the union of two rectangular regions. Assume that the variable **point** holds the location of the most recent click. The code checks to see whether **point** is in the plus-shaped region. If **point** is in the region (a hit), the region is filled with an opaque red brush. Otherwise, the region is filled with a semitransparent red brush.

```
Point point(60, 10);
// Assume that the variable "point" contains the location
// of the most recent click.
// To simulate a hit, assign (60, 10) to point.
// To simulate a miss, assign (0, 0) to point.
SolidBrush solidBrush(Color());
Region region1(Rect(50, 0, 50, 150));
Region region2(Rect(0, 50, 150, 50));
// Create a plus-shaped region by forming the union of region1 and region2.
// The union replaces region1.
region1.Union(&region2);
if(region1.IsVisible(point, &graphics))
{
    // The point is in the region. Use an opaque brush.
    solidBrush.SetColor(Color(255, 255, 0, 0));
}
else
{
    // The point is not in the region. Use a semitransparent brush.
    solidBrush.SetColor(Color(64, 255, 0, 0));
}
graphics.FillRegion(&solidBrush, &region1);
```

Clipping with a Region

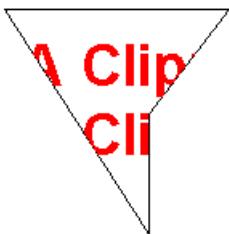
2/22/2020 • 2 minutes to read • [Edit Online](#)

One of the properties of the **Graphics** class is the clipping region. All drawing done by a given **Graphics** object is restricted to the clipping region of that **Graphics** object. You can set the clipping region by calling the **SetClip** method.

The following example constructs a path that consists of a single polygon. Then the code constructs a region based on that path. The address of the region is passed to the **SetClip** method of a **Graphics** object, and then two strings are drawn.

```
// Create a path that consists of a single polygon.  
Point polyPoints[] = {Point(10, 10), Point(150, 10),  
    Point(100, 75), Point(100, 150)};  
GraphicsPath path;  
path.AddPolygon(polyPoints, 4);  
// Construct a region based on the path.  
Region region(&path);  
// Draw the outline of the region.  
Pen pen(Color(255, 0, 0, 0));  
graphics.DrawPath(&pen, &path);  
// Set the clipping region of the Graphics object.  
graphics.SetClip(&region);  
// Draw some clipped strings.  
FontFamily fontFamily(L"Arial");  
Font font(&fontFamily, 36, FontStyleBold, UnitPixel);  
SolidBrush solidBrush(Color(255, 255, 0, 0));  
graphics.DrawString(L"A Clipping Region", 20, &font,  
    PointF(15, 25), &solidBrush);  
graphics.DrawString(L"A Clipping Region", 20, &font,  
    PointF(15, 68), &solidBrush);
```

The following illustration shows the clipped strings.



Recoloring

2/22/2020 • 2 minutes to read • [Edit Online](#)

Recoloring is the process of adjusting image colors. Some examples of recoloring are changing one color to another, adjusting a color's intensity relative to another color, adjusting the brightness or contrast of all colors, and converting colors to shades of gray.

The following topics cover recoloring in more detail:

- [Using a Color Matrix to Transform a Single Color](#)
- [Translating Colors](#)
- [Scaling Colors](#)
- [Rotating Colors](#)
- [Shearing Colors](#)
- [Using a Color Remap Table](#)

Using a Color Matrix to Transform a Single Color

11/2/2020 • 3 minutes to read • [Edit Online](#)

Windows GDI+ provides the [Image](#) and [Bitmap](#) classes for storing and manipulating images. **Image** and **Bitmap** objects store the color of each pixel as a 32-bit number: 8 bits each for red, green, blue, and alpha. Each of the four components is a number from 0 through 255, with 0 representing no intensity and 255 representing full intensity. The alpha component specifies the transparency of the color: 0 is fully transparent, and 255 is fully opaque.

A color vector is a 4-tuple of the form (red, green, blue, alpha). For example, the color vector (0, 255, 0, 255) represents an opaque color that has no red or blue, but has green at full intensity.

Another convention for representing colors uses the number 1 for maximum intensity and the number 0 for minimum intensity. Using that convention, the color described in the preceding paragraph would be represented by the vector (0, 1, 0, 1). GDI+ uses the convention of 1 as full intensity when it performs color transformations.

You can apply linear transformations (rotation, scaling, and the like) to color vectors by multiplying by a 4×4 matrix. However, you cannot use a 4×4 matrix to perform a translation (nonlinear). If you add a dummy fifth coordinate (for example, the number 1) to each of the color vectors, you can use a 5×5 matrix to apply any combination of linear transformations and translations. A transformation consisting of a linear transformation followed by a translation is called an affine transformation. A 5×5 matrix that represents an affine transformation is called a homogeneous matrix for a 4-space transformation. The element in the fifth row and fifth column of a 5×5 homogeneous matrix must be 1, and all of the other entries in the fifth column must be 0.

For example, suppose you want to start with the color (0.2, 0.0, 0.4, 1.0) and apply the following transformations:

1. Double the red component
2. Add 0.2 to the red, green, and blue components

The following matrix multiplication will perform the pair of transformations in the order listed.

$$\begin{bmatrix} 0.2 & 0.0 & 0.4 & 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0.2 & 0.2 & 0.2 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.6 & 0.2 & 0.6 & 1.0 & 1.0 \end{bmatrix}$$

The elements of a color matrix are indexed (zero-based) by row and then column. For example, the entry in the fifth row and third column of matrix M is denoted by M[4][2].

The 5×5 identity matrix (shown in the following illustration) has 1s on the diagonal and 0s everywhere else. If you multiply a color vector by the identity matrix, the color vector does not change. A convenient way to form the matrix of a color transformation is to start with the identity matrix and make a small change that produces the desired transformation.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Identity Matrix

For a more detailed discussion of matrices and transformations, see [Coordinate Systems and Transformations](#).

The following example takes an image that is all one color (0.2, 0.0, 0.4, 1.0) and applies the transformation described in the preceding paragraphs.

```

Image           image(L"InputColor.bmp");
ImageAttributes imageAttributes;
UINT           width = image.GetWidth();
UINT           height = image.GetHeight();

ColorMatrix colorMatrix = {
    2.0f, 0.0f, 0.0f, 0.0f, 0.0f,
    0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
    0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
    0.2f, 0.2f, 0.2f, 0.0f, 1.0f};

imageAttributesSetColorMatrix(
    &colorMatrix,
    ColorMatrixFlagsDefault,
    ColorAdjustTypeBitmap);

graphics.DrawImage(&image, 10, 10);

graphics.DrawImage(
    &image,
    Rect(120, 10, width, height), // destination rectangle
    0, 0,           // upper-left corner of source rectangle
    width,          // width of source rectangle
    height,         // height of source rectangle
    UnitPixel,
    &imageAttributes);

```

The following illustration shows the original image on the left and the transformed image on the right.



The code in the preceding example uses the following steps to perform the recoloring:

1. Initialize a **ColorMatrix** structure.
2. Create an **ImageAttributes** object and pass the address of the **ColorMatrix** structure to the **ImageAttributes::SetColorMatrix** method of the **ImageAttributes** object.
3. Pass the address of the **ImageAttributes** object to the **DrawImage Methods** method of a **Graphics** object.

Translating Colors

2/22/2020 • 2 minutes to read • [Edit Online](#)

A translation adds a value to one or more of the four color components. The color matrix entries that represent translations are given in the following table.

COMPONENT TO BE TRANSLATED	MATRIX ENTRY
Red	[4][0]
Green	[4][1]
Blue	[4][2]
Alpha	[4][3]

The following example constructs an [Image](#) object from the file ColorBars.bmp. Then the code adds 0.75 to the red component of each pixel in the image. The original image is drawn alongside the transformed image.

```
Image         image(L"ColorBars.bmp");
ImageAttributes imageAttributes;
UINT          width = image.GetWidth();
UINT          height = image.GetHeight();

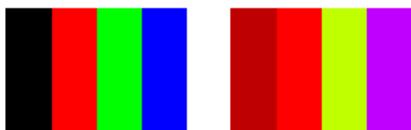
ColorMatrix colorMatrix = {
    1.0f,  0.0f, 0.0f, 0.0f, 0.0f,
    0.0f,  1.0f, 0.0f, 0.0f, 0.0f,
    0.0f,  0.0f, 1.0f, 0.0f, 0.0f,
    0.0f,  0.0f, 0.0f, 1.0f, 0.0f,
    0.75f, 0.0f, 0.0f, 0.0f, 1.0f};

imageAttributesSetColorMatrix(
    &colorMatrix,
    ColorMatrixFlagsDefault,
    ColorAdjustTypeBitmap);

graphics.DrawImage(&image, 10, 10, width, height);

graphics.DrawImage(
    &image,
    Rect(150, 10, width, height), // destination rectangle
    0, 0,           // upper-left corner of source rectangle
    width,          // width of source rectangle
    height,         // height of source rectangle
    UnitPixel,
    &imageAttributes);
```

The following illustration shows the original image on the left and the transformed image on the right.



The following table lists the color vectors for the four bars before and after the red translation. Note that because the maximum value for a color component is 1, the red component in the second row does not change. (Similarly, the minimum value for a color component is 0.)

ORIGINAL	TRANSLATED
Black (0, 0, 0, 1)	(0.75, 0, 0, 1)
Red (1, 0, 0, 1)	(1, 0, 0, 1)
Green (0, 1, 0, 1)	(0.75, 1, 0, 1)
Blue (0, 0, 1, 1)	(0.75, 0, 1, 1)

Scaling Colors

2/22/2020 • 2 minutes to read • [Edit Online](#)

A scaling transformation multiplies one or more of the four color components by a number. The color matrix entries that represent scaling are given in the following table.

COMPONENT TO BE SCALED	MATRIX ENTRY
Red	[0][0]
Green	[1][1]
Blue	[2][2]
Alpha	[3][3]

The following example constructs an **Image** object from the file ColorBars2.bmp. Then the code scales the blue component of each pixel in the image by a factor of 2. The original image is drawn alongside the transformed image.

```
Image         image(L"ColorBars2.bmp");
ImageAttributes imageAttributes;
UINT          width = image.GetWidth();
UINT          height = image.GetHeight();

ColorMatrix colorMatrix = {
    1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
    0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
    0.0f, 0.0f, 2.0f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 0.0f, 1.0f};

imageAttributesSetColorMatrix(
    &colorMatrix,
    ColorMatrixFlagsDefault,
    ColorAdjustTypeBitmap);

graphics.DrawImage(&image, 10, 10, width, height);

graphics.DrawImage(
    &image,
    Rect(150, 10, width, height), // destination rectangle
    0, 0,           // upper-left corner of source rectangle
    width,          // width of source rectangle
    height,          // height of source rectangle
    UnitPixel,
    &imageAttributes);
```

The following illustration shows the original image on the left and the scaled image on the right.



The following table shows the color vectors for the four bars before and after the blue scaling. Note that the blue component in the fourth color bar went from 0.8 to 0.6. That is because GDI+ retains only the fractional part of the result. For example, $(2)(0.8) = 1.6$, and the fractional part of 1.6 is 0.6. Retaining only the fractional part ensures that the result is always in the interval $[0, 1]$.

ORIGINAL	SCALED
(0.4, 0.4, 0.4, 1)	(0.4, 0.4, 0.8, 1)
(0.4, 0.2, 0.2, 1)	(0.4, 0.2, 0.4, 1)
(0.2, 0.4, 0.2, 1)	(0.2, 0.4, 0.4, 1)
(0.4, 0.4, 0.8, 1)	(0.4, 0.4, 0.6, 1)

The following example constructs an [Image](#) object from the file ColorBars2.bmp. Then the code scales the red, green, and blue components of each pixel in the image. The red components are scaled down 25 percent, the green components are scaled down 35 percent, and the blue components are scaled down 50 percent.

```

Image           image(L"ColorBars.bmp");
ImageAttributes imageAttributes;
UINT            width = image.GetWidth();
UINT            height = image.GetHeight();

ColorMatrix colorMatrix = {
    0.75f, 0.0f, 0.0f, 0.0f, 0.0f,
    0.0f, 0.65f, 0.0f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.5f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 0.0f, 1.0f};

imageAttributesSetColorMatrix(
    &colorMatrix,
    ColorMatrixFlagsDefault,
    ColorAdjustTypeBitmap);

graphics.DrawImage(&image, 10, 10, width, height);

graphics.DrawImage(
    &image,
    Rect(150, 10, width, height), // destination rectangle
    0, 0, // upper-left corner of source rectangle
    width, // width of source rectangle
    height, // height of source rectangle
    UnitPixel,
    &imageAttributes);

```

The following illustration shows the original image on the left and the scaled image on the right.



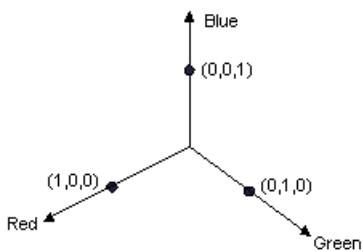
The following table shows the color vectors for the four bars before and after the red, green and blue scaling.

ORIGINAL	SCALED
(0.6, 0.6, 0.6, 1)	(0.45, 0.39, 0.3, 1)
(0, 1, 1, 1)	(0, 0.65, 0.5, 1)
(1, 1, 0, 1)	(0.75, 0.65, 0, 1)
(1, 0, 1, 1)	(0.75, 0, 0.5, 1)

Rotating Colors

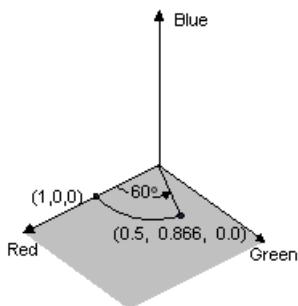
2/22/2020 • 2 minutes to read • [Edit Online](#)

Rotation in a four-dimensional color space is difficult to visualize. We can make it easier to visualize rotation by agreeing to keep one of the color components fixed. Suppose we agree to keep the alpha component fixed at 1 (fully opaque). Then we can visualize a three-dimensional color space with red, green, and blue axes as shown in the following illustration.



A color can be thought of as a point in 3-D space. For example, the point $(1, 0, 0)$ in space represents the color red, and the point $(0, 1, 0)$ in space represents the color green.

The following illustration shows what it means to rotate the color $(1, 0, 0)$ through an angle of 60 degrees in the Red-Green plane. Rotation in a plane parallel to the Red-Green plane can be thought of as rotation about the blue axis.



The following illustration shows how to initialize a color matrix to perform rotations about each of the three coordinate axes (red, green, blue).

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 & 0 \\ 0 & -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

©Indicates that the axis comes out of the page toward the reader

The following example takes an image that is all one color $(1, 0, 0.6)$ and applies a 60-degree rotation about the blue axis. The angle of the rotation is swept out in a plane that is parallel to the Red-Green plane.

```

Image           image(L"RotationInput.bmp");
ImageAttributes imageAttributes;
UINT            width = image.GetWidth();
UINT            height = image.GetHeight();
REAL             degrees = 60;
REAL             pi = acos(-1); // the angle whose cosine is -1.
REAL             r = degrees * pi / 180; // degrees to radians

ColorMatrix colorMatrix = {
    cos(r), sin(r), 0.0f, 0.0f, 0.0f,
    -sin(r), cos(r), 0.0f, 0.0f, 0.0f,
    0.0f, 0.0f, 1.0f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 0.0f, 1.0f};

imageAttributesSetColorMatrix(
    &colorMatrix,
    ColorMatrixFlagsDefault,
    ColorAdjustTypeBitmap);

graphics.DrawImage(&image, 10, 10, width, height);

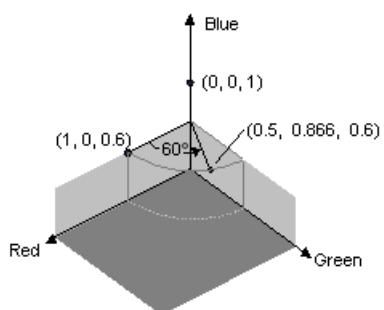
graphics.DrawImage(
    &image,
    Rect(130, 10, width, height), // destination rectangle
    0, 0, // upper-left corner of source rectangle
    width, // width of source rectangle
    height, // height of source rectangle
    UnitPixel,
    &imageAttributes);

```

The following illustration shows the original image on the left and the color-rotated image on the right.



The color rotation performed in the preceding code example can be visualized as follows.



The rotation takes place in the plane $\text{Blue}=0.6$, which is parallel to the Red-Green plane.

Shearing Colors

2/22/2020 • 2 minutes to read • [Edit Online](#)

Shearing increases or decreases a color component by an amount proportional to another color component. For example, consider the transformation where the red component is increased by one half the value of the blue component. Under such a transformation, the color (0.2, 0.5, 1) would become (0.7, 0.5, 1). The new red component is $0.2 + (1/2)(1) = 0.7$.

The following example constructs an **Image** object from the file ColorBars4.bmp. Then the code applies the shearing transformation described in the preceding paragraph to each pixel in the image.

```
Image           image(L"ColorBars4.bmp");
ImageAttributes imageAttributes;
UINT           width = image.GetWidth();
UINT           height = image.GetHeight();

ColorMatrix colorMatrix = {
    1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
    0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
    0.5f, 0.0f, 1.0f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 0.0f, 1.0f};

imageAttributesSetColorMatrix(
    &colorMatrix,
    ColorMatrixFlagsDefault,
    ColorAdjustTypeBitmap);

graphics.DrawImage(&image, 10, 10, width, height);

graphics.DrawImage(
    &image,
    Rect(150, 10, width, height), // destination rectangle
    0, 0,           // upper-left corner of source rectangle
    width,          // width of source rectangle
    height,         // height of source rectangle
    UnitPixel,
    &imageAttributes);
```

The following illustration shows the original image on the left and the sheared image on the right.



The following table shows the color vectors for the four bars before and after the shearing transformation.

ORIGINAL	SHEARED
(0, 0, 1, 1)	(0.5, 0, 1, 1)
(0.5, 1, 0.5, 1)	(0.75, 1, 0.5, 1)
(1, 1, 0, 1)	(1, 1, 0, 1)

ORIGINAL

SHEARED

(0.4, 0.4, 0.4, 1)

(0.6, 0.4, 0.4, 1)

Using a Color Remap Table

11/2/2020 • 2 minutes to read • [Edit Online](#)

Remapping is the process of converting the colors in an image according to a color remap table. The color remap table is an array of **ColorMap** structures. Each **ColorMap** structure in the array has an **oldColor** member and a **newColor** member.

When GDI+ draws an image, each pixel of the image is compared to the array of old colors. If a pixel's color matches an old color, its color is changed to the corresponding new color. The colors are changed only for rendering — the color values of the image itself (stored in an **Image** or **Bitmap** object) are not changed.

To draw a remapped image, initialize an array of **ColorMap** structures. Pass the address of that array to the **ImageAttributes::SetRemapTable** method of an **ImageAttributes** object, and then pass the address of the **ImageAttributes** object to the **DrawImage Methods** method of a **Graphics** object.

The following example creates an **Image** object from the file **RemapInput.bmp**. The code creates a color remap table that consists of a single **ColorMap** structure. The **oldColor** member of the **ColorMap** structure is red, and the **newColor** member is blue. The image is drawn once without remapping and once with remapping. The remapping process changes all the red pixels to blue.

```
Image           image(L"RemapInput.bmp");
ImageAttributes imageAttributes;
UINT            width = image.GetWidth();
UINT            height = image.GetHeight();
ColorMap        colorMap[1];

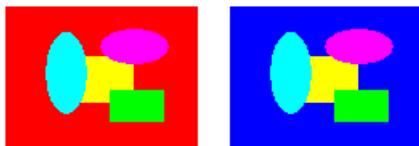
colorMap[0].oldColor = Color(255, 255, 0, 0); // opaque red
colorMap[0].newColor = Color(255, 0, 0, 255); // opaque blue

imageAttributes.SetRemapTable(1, colorMap, ColorAdjustTypeBitmap);

graphics.DrawImage(&image, 10, 10, width, height);

graphics.DrawImage(
    &image,
    Rect(150, 10, width, height), // destination rectangle
    0, 0,           // upper-left corner of source rectangle
    width,          // width of source rectangle
    height,         // height of source rectangle
    UnitPixel,
    &imageAttributes);
```

The following illustration shows the original image on the left and the remapped image on the right.



Printing (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

With a few minor adjustments to your code, you can send Windows GDI+ output to a printer rather than to a screen. To draw on a printer, obtain a device context handle for the printer and pass that handle to a **Graphics** constructor. Place your GDI+ drawing commands in between calls to **StartDoc** and **EndDoc**.

The following topics cover sending GDI+ output to printers in more detail:

- [Sending GDI+ Output to a Printer](#)
- [Displaying a Print Dialog Box](#)
- [Optimizing Printing by Providing a Printer Handle](#)

Sending GDI+ Output to a Printer

11/2/2020 • 2 minutes to read • [Edit Online](#)

Using Windows GDI+ to draw on a printer is similar to using GDI+ to draw on a computer screen. To draw on a printer, obtain a device context handle for the printer and then pass that handle to a [Graphics](#) constructor.

The following console application draws a line, a rectangle, and an ellipse on a printer named MyPrinter:

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    // Get a device context for the printer.
    HDC hdcPrint = CreateDC(NULL, TEXT("\\\\printserver\\\\print1"), NULL, NULL);

    DOCINFO docInfo;
    ZeroMemory(&docInfo, sizeof(docInfo));
    docInfo.cbSize = sizeof(docInfo);
    docInfo.lpszDocName = "GdiplusPrint";

    StartDoc(hdcPrint, &docInfo);
    StartPage(hdcPrint);
    Graphics* graphics = new Graphics(hdcPrint);
    Pen* pen = new Pen(Color(255, 0, 0, 0));
    graphics->DrawLine(pen, 50, 50, 350, 550);
    graphics->DrawRectangle(pen, 50, 50, 300, 500);
    graphics->DrawEllipse(pen, 50, 50, 300, 500);
    delete pen;
    delete graphics;
    EndPage(hdcPrint);
    EndDoc(hdcPrint);

    DeleteDC(hdcPrint);
    GdiplusShutdown(gdiplusToken);
    return 0;
}
```

In the preceding code, the three GDI+ drawing commands are in between calls to the [StartDoc](#) and [EndDoc](#) functions, each of which receives the printer device context handle. All graphics commands between StartDoc and EndDoc are routed to a temporary metafile. After the call to EndDoc, the printer driver converts the data in the metafile into the format required by the specific printer being used.

NOTE

If spooling is not enabled for the printer being used, the graphics output is not routed to a metafile. Instead, individual graphics commands are processed by the printer driver and then sent to the printer.

Generally you won't want to hard-code the name of a printer as was done in the preceding console application. One alternative to hard-coding the name is to call [GetDefaultPrinter](#) to obtain the name of the default printer. Before you call GetDefaultPrinter, you must allocate a buffer large enough to hold the printer name. You can determine the size of the required buffer by calling GetDefaultPrinter, passing **NULL** as the first argument.

NOTE

The [GetDefaultPrinter](#) function is supported only on Windows 2000 and later.

The following console application gets the name of the default printer and then draws a rectangle and an ellipse on that printer. The [Graphics::DrawRectangle](#) call is in between calls to [StartPage](#) and [EndPage](#), so the rectangle is on a page by itself. Similarly, the ellipse is on a page by itself.

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    DWORD size;
    HDC hdcPrint;

    DOCINFO docInfo;
    ZeroMemory(&docInfo, sizeof(docInfo));
    docInfo.cbSize = sizeof(docInfo);
    docInfo.lpszDocName = "GdiplusPrint";

    // Get the size of the default printer name.
    GetDefaultPrinter(NULL, &size);

    // Allocate a buffer large enough to hold the printer name.
    TCHAR* buffer = new TCHAR[size];

    // Get the printer name.
    if(!GetDefaultPrinter(buffer, &size))
    {
        printf("Failure");
    }
    else
    {
        // Get a device context for the printer.
        hdcPrint = CreateDC(NULL, buffer, NULL, NULL);

        StartDoc(hdcPrint, &docInfo);
        Graphics* graphics;
        Pen* pen = new Pen(Color(255, 0, 0, 0));

        StartPage(hdcPrint);
        graphics = new Graphics(hdcPrint);
        graphics->DrawRectangle(pen, 50, 50, 200, 300);
        delete graphics;
        EndPage(hdcPrint);

        StartPage(hdcPrint);
        graphics = new Graphics(hdcPrint);
        graphics->DrawEllipse(pen, 50, 50, 200, 300);
        delete graphics;
        EndPage(hdcPrint);

        delete pen;
        EndDoc(hdcPrint);

        DeleteDC(hdcPrint);
    }

    delete buffer;

    GdiplusShutdown(gdiplusToken);
    return 0;
}
```


Displaying a Print Dialog Box

11/2/2020 • 2 minutes to read • [Edit Online](#)

One way to get a device context handle for a printer is to display a print dialog box and allow the user to choose a printer. The [PrintDlg](#) function (which displays the dialog box) has one parameter that is the address of a [PRINTDLG](#) structure. The PRINTDLG structure has several members, but you can leave most of them set to their default values. The two members you need to set are **IStructSize** and **Flags**. Set **IStructSize** to the size of a PRINTDLG variable, and set **Flags** to PD_RETURNDC. Setting **Flags** to PC_RETURNDC specifies that you want the PrintDlg function to fill the **hDC** field with a device context handle for the printer chosen by the user.

```

#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    DOCINFO docInfo;
    ZeroMemory(&docInfo, sizeof(docInfo));
    docInfo.cbSize = sizeof(docInfo);
    docInfo.lpszDocName = "GdiplusPrint";

    // Create a PRINTDLG structure, and initialize the appropriate fields.
    PRINTDLG printDlg;
    ZeroMemory(&printDlg, sizeof(printDlg));
    printDlg.lStructSize = sizeof(printDlg);
    printDlg.Flags = PD_RETURNDC;

    // Display a print dialog box.
    if(!PrintDlg(&printDlg))
    {
        printf("Failure\n");
    }
    else
    {
        // Now that PrintDlg has returned, a device context handle
        // for the chosen printer is in printDlg->hDC.

        StartDoc(printDlg.hDC, &docInfo);
        StartPage(printDlg.hDC);
        Graphics* graphics = new Graphics(printDlg.hDC);
        Pen* pen = new Pen(Color(255, 0, 0, 0));
        graphics->DrawRectangle(pen, 200, 500, 200, 150);
        graphics->DrawEllipse(pen, 200, 500, 200, 150);
        graphics->DrawLine(pen, 200, 500, 400, 650);
        delete pen;
        delete graphics;
        EndPage(printDlg.hDC);
        EndDoc(printDlg.hDC);
    }
    if(printDlg.hDevMode)
        GlobalFree(printDlg.hDevMode);
    if(printDlg.hDevNames)
        GlobalFree(printDlg.hDevNames);
    if(printDlg.hDC)
        DeleteDC(printDlg.hDC);

    GdiplusShutdown(gdiplusToken);
    return 0;
}

```

Optimizing Printing by Providing a Printer Handle

11/2/2020 • 2 minutes to read • [Edit Online](#)

One of the constructors for the [Graphics](#) class receives a device context handle and a printer handle. When you send Windows GDI+ commands to certain PostScript printers, the performance will be better if you create your [Graphics](#) object with that particular constructor.

The following console application calls [GetDefaultPrinter](#) to get the name of the default printer. The code passes the printer name to [CreateDC](#) to obtain a device context handle for the printer. The code also passes the printer name to [OpenPrinter](#) to obtain a printer handle. Both the device context handle and the printer handle are passed to the [Graphics](#) constructor. Then two figures are drawn on the printer.

NOTE

The [GetDefaultPrinter](#) function is supported only on Windows 2000 and later.

```

#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;

INT main()
{
    // Initialize GDI+.
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    DWORD    size;
    HDC      hdcPrint;
    HANDLE   printerHandle;

    DOCINFO docInfo;
    ZeroMemory(&docInfo, sizeof(docInfo));
    docInfo.cbSize = sizeof(docInfo);
    docInfo.lpszDocName = "GdiplusPrint";

    // Get the length of the printer name.
    GetDefaultPrinter(NULL, &size);
    TCHAR* buffer = new TCHAR[size];

    // Get the printer name.
    if(!GetDefaultPrinter(buffer, &size))
    {
        printf("Failure");
    }
    else
    {
        // Get a device context for the printer.
        hdcPrint = CreateDC(NULL, buffer, NULL, NULL);

        // Get a printer handle.
        OpenPrinter(buffer, &printerHandle, NULL);

        StartDoc(hdcPrint, &docInfo);
        StartPage(hdcPrint);
        Graphics* graphics = new Graphics(hdcPrint, printerHandle);
        Pen* pen = new Pen(Color(255, 0, 0, 0));
        graphics->DrawRectangle(pen, 200, 500, 200, 150);
        graphics->DrawEllipse(pen, 200, 500, 200, 150);
        delete(pen);
        delete(graphics);
        EndPage(hdcPrint);
        EndDoc(hdcPrint);

        ClosePrinter(printerHandle);
        DeleteDC(hdcPrint);
    }

    delete buffer;

    GdiplusShutdown(gdiplusToken);
    return 0;
}

```

GDI+ Reference

2/22/2020 • 2 minutes to read • [Edit Online](#)

The following topics provide reference information about the Windows GDI+ API with the C++ programming language:

- [Classes](#)
- [Functions](#)
- [Constants](#)
- [Enumerations](#)
- [Structures](#)

Classes (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides the following classes:

- [AdjustableArrowCap](#)
- [Bitmap](#)
- [BitmapData](#)
- [Blur](#)
- [BrightnessContrast](#)
- [Brush](#)
- [CachedBitmap](#)
- [CharacterRange](#)
- [Color](#)
- [ColorBalance](#)
- [ColorCurve](#)
- [ColorLUT](#)
- [ColorMatrixEffect](#)
- [CustomLineCap](#)
- [Effect](#)
- [EncoderParameter](#)
- [EncoderParameters](#)
- [Font](#)
- [FontCollection](#)
- [FontFamily](#)
- [GdiplusBase](#)
- [Graphics](#)
- [GraphicsPath](#)
- [GraphicsPathIterator](#)
- [HatchBrush](#)
- [HueSaturationLightness](#)
- [Image](#)
- [ImageAttributes](#)
- [ImageCodeclnfo](#)
- [ImageItemData](#)
- [InstalledFontCollection](#)
- [Levels](#)
- [LinearGradientBrush](#)
- [Matrix](#)
- [Metafile](#)
- [MetafileHeader](#)
- [PathData](#)
- [PathGradientBrush](#)
- [Pen](#)

- [Point](#)
- [PointF](#)
- [PrivateFontCollection](#)
- [PropertyItem](#)
- [Rect](#)
- [RectF](#)
- [RedEyeCorrection](#)
- [Region](#)
- [Sharpen](#)
- [Size](#)
- [SizeF](#)
- [SolidBrush](#)
- [StringFormat](#)
- [TextureBrush](#)
- [Tint](#)

AdjustableArrowCap Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [AdjustableArrowCap](#) class. For a complete class listing, see [AdjustableArrowCap](#).

- [CustomLineCap::Clone](#)
- [CustomLineCap::GetBaseCap](#)
- [CustomLineCap::GetBaseInset](#)
- [AdjustableArrowCap::GetHeight](#)
- [CustomLineCap::GetLastStatus](#)
- [AdjustableArrowCap::GetMiddleInset](#)
- [CustomLineCap::GetStrokeCaps](#)
- [CustomLineCap::GetStrokeJoin](#)
- [AdjustableArrowCap::GetWidth](#)
- [CustomLineCap::GetWidthScale](#)
- [AdjustableArrowCap::IsFilled](#)
- [CustomLineCap::SetBaseCap](#)
- [CustomLineCap::SetBaseInset](#)
- [AdjustableArrowCap::SetFillState](#)
- [AdjustableArrowCap::SetHeight](#)
- [AdjustableArrowCap::SetMiddleInset](#)
- [CustomLineCap::SetStrokeCap](#)
- [CustomLineCap::SetStrokeCaps](#)
- [CustomLineCap::SetStrokeJoin](#)
- [AdjustableArrowCap::setWidth](#)
- [CustomLineCap::setWidthScale](#)

Bitmap Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Bitmap](#) class. For a complete class listing, see [Bitmap Class](#).

- [ApplyEffect\(Bitmap**,INT,Effect*,RECT*,RECT*,Bitmap**\)](#)
- [ApplyEffect\(Effect*,RECT*\)](#)
- [Clone\(Rect&,PixelFormat\)](#)
- [Clone\(RectF&,PixelFormat\)](#)
- [Clone\(REAL,REAL,REAL,REAL,PixelFormat\)](#)
- [Clone\(INT,INT,INT,INT,PixelFormat\)](#)
- [Clone](#)
- [ConvertFormat](#)
- [FindFirstItem](#)
- [FindNextItem](#)
- [FromBITMAPINFO](#)
- [FromDirectDrawSurface7](#)
- [FromFile](#)
- [FromFile](#)
- [FromHBITMAP](#)
- [FromHICON](#)
- [FromResource](#)
- [FromStream](#)
- [FromStream](#)
- [GetAllPropertyItems](#)
- [GetBounds](#)
- [GetEncoderParameterList](#)
- [GetEncoderParameterListSize](#)
- [GetFlags](#)
- [GetFrameCount](#)
- [GetFrameDimensionsCount](#)
- [GetFrameDimensionsList](#)
- [GetHBITMAP](#)
- [GetHeight](#)
- [GetHICON](#)
- [GetHistogram](#)
- [GetHistogramSize](#)
- [GetHorizontalResolution](#)
- [GetImageData](#)
- [GetLastStatus](#)
- [GetPalette](#)
- [GetPaletteSize](#)
- [GetPhysicalDimension](#)
- [GetPixel](#)

- [GetPixelFormat](#)
- [GetPropertyCount](#)
- [GetPropertyIdList](#)
- [GetPropertyItem](#)
- [GetPropertyItemSize](#)
- [GetPropertySize](#)
- [GetRawFormat](#)
- [GetThumbnailImage](#)
- [GetType](#)
- [GetVerticalResolution](#)
- [GetWidth](#)
- [InitializePalette](#)
- [LockBits](#)
- [RemovePropertyItem](#)
- [RotateFlip](#)
- [Save\(IStream*,CLSID*,EncoderParameters*\)](#)
- [Save\(WCHAR*,CLSID*,EncoderParameters*\)](#)
- [SaveAdd\(EncoderParameters*\)](#)
- [SaveAdd\(Image*,EncoderParameters*\)](#)
- [SelectActiveFrame](#)
- [SetAbort](#)
- [SetPalette](#)
- [SetPixel](#)
- [SetPropertyItem](#)
- [SetResolution](#)
- [UnlockBits](#)

Bitmap.Clone methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Clone methods of the [Bitmap](#) class. For a complete list of methods for the [Bitmap](#) class, see [Bitmap Methods](#).

Overload list

METHOD	DESCRIPTION
Image::Clone	The Image::Clone method creates a new Image object and initializes it with the contents of this Image object.
Clone(Rect&,PixelFormat)	The Bitmap::Clone method creates a new Bitmap object by copying a portion of this bitmap.
Clone(RectF&,PixelFormat)	The Bitmap::Clone method creates a new Bitmap object by copying a portion of this bitmap.
Clone(INT,INT,INT,INT,PixelFormat)	The Bitmap::Clone method creates a new Bitmap object by copying a portion of this bitmap.
Clone(REAL,REAL,REAL,REAL,PixelFormat)	The Bitmap::Clone method creates a new Bitmap object by copying a portion of this bitmap.

Bitmap.ApplyEffect methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the ApplyEffect methods of the [Bitmap](#) class. For a complete list of methods for the [Bitmap](#) class, see [Bitmap Methods](#).

Overload list

METHOD	DESCRIPTION
ApplyEffect(Effect*,RECT*)	The Bitmap::ApplyEffect method alters this Bitmap object by applying a specified effect.
ApplyEffect(Bitmap**,INT,Effect*,RECT*,RECT*,Bitmap**)	The Bitmap::ApplyEffect method creates a new Bitmap object by applying a specified effect to an existing Bitmap object.

Blur Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **Blur** class. For a complete class listing, see **Blur**.

- [GetParameters](#)
- [SetParameters](#)

BrightnessContrast Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **BrightnessContrast** class. For a complete class listing, see **BrightnessContrast**.

- [GetParameters](#)
- [SetParameters](#)

Brush Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **Brush** class. For a complete class listing, see **Brush Class**.

- [Clone](#)
- [GetLastStatus](#)
- [GetType](#)

CharacterRange.CharacterRange constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [CharacterRange](#) class. For a complete class listing, see [CharacterRange Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
CharacterRange()	Creates a CharacterRange object with the data members set to zero.
CharacterRange(INT,INT)	Creates a CharacterRange object and initializes the data members to the values specified.

Requirements

Header	Gdiplustypes.h
--------	----------------

Color.Color constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [Color](#) class. For a complete class listing, see [Color Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Color(ARGB)	Creates a Color::Color object by using an ARGB value.
Color(BYTE,BYTE,BYTE)	Creates a Color::Color object by using specified values for the red, green, and blue components. This constructor sets the alpha component to 255 (opaque).
Color(BYTE,BYTE,BYTE,BYTE)	Creates a Color::Color object by using specified values for the alpha, red, green, and blue components.
Color()	Creates a Color::Color object and initializes it to opaque black. This is the default constructor.

Color Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Color](#) class. For a complete class listing, see [Color Class](#).

- [GetA](#)
- [GetAlpha](#)
- [GetB](#)
- [GetBlue](#)
- [GetG](#)
- [GetGreen](#)
- [GetR](#)
- [GetRed](#)
- [GetValue](#)
- [MakeARGB](#)
- [SetFromCOLORREF](#)
- [SetValue](#)
- [ToCOLORREF](#)

ColorBalance Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **ColorBalance** class. For a complete class listing, see **ColorBalance**.

- [GetParameters](#)
- [SetParameters](#)

ColorCurve Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [ColorCurve](#) class. For a complete class listing, see [ColorCurve](#).

- [GetParameters](#)
- [SetParameters](#)

ColorLUT Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **ColorLUT** class. For a complete class listing, see **ColorLUT**.

- [GetParameters](#)
- [SetParameters](#)

ColorMatrixEffect Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **ColorMatrixEffect** class. For a complete class listing, see **ColorMatrixEffect**.

- [GetParameters](#)
- [SetParameters](#)

CustomLineCap Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [CustomLineCap](#) class. For a complete class listing, see [CustomLineCap Class](#).

- [Clone](#)
- [GetBaseCap](#)
- [GetBaseInset](#)
- [GetLastStatus](#)
- [GetStrokeCaps](#)
- [GetStrokeJoin](#)
- [GetWidthScale](#)
- [SetBaseCap](#)
- [SetBaseInset](#)
- [SetStrokeCap](#)
- [SetStrokeCaps](#)
- [SetStrokeJoin](#)
- [SetWidthScale](#)

Effect Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Effect](#) class. For a complete class listing, see [Effect](#).

- [GetAuxData](#)
- [GetAuxDataSize](#)
- [GetParameterSize](#)
- [UseAuxData](#)

Font.Font constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [Font](#) class. For a complete class listing, see [Font Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Font(HDC,HFONT)	Creates a Font::Font object indirectly from a GDI logical font by using a handle to a GDI LOGFONT structure.
Font(HDC,LOGFONTA*)	Creates a Font::Font object directly from a GDI logical font. The GDI logical font is a LOGFONTA structure, which is the one-byte character version of a logical font. This constructor is provided for compatibility with GDI.
Font(HDC,LOGFONTW*)	Creates a Font::Font object directly from a GDI logical font. The GDI logical font is a LOGFONTW structure, which is the wide character version of a logical font. This constructor is provided for compatibility with GDI.
Font(FontFamily*,REAL,INT,Unit)	Creates a Font::Font object based on a FontFamily object, a size, a font style, and a unit of measurement.
Font(WCHAR*,REAL,INT,Unit,FontCollection*)	Creates a Font::Font object based on a font family, a size, a font style, a unit of measurement, and a FontCollection object.
Font(HDC)	Creates a Font::Font object based on the GDI font object that is currently selected into a specified device context. This constructor is provided for compatibility with GDI.

Font Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Font](#) class. For a complete class listing, see [Font Class](#).

- [Clone](#)
- [GetFamily](#)
- [GetHeight\(Graphics*\)](#)
- [GetHeight\(REAL\)](#)
- [GetLastStatus](#)
- [GetLogFontA](#)
- [GetLogFontW](#)
- [GetSize](#)
- [GetStyle](#)
- [GetUnit](#)
- [IsAvailable](#)

Font.GetHeight methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the GetHeight methods of the [Font](#) class. For a complete list of methods for the [Font](#) class, see [Font Methods](#).

Overload list

METHOD	DESCRIPTION
GetHeight(REAL)	The Font::GetHeight method gets the line spacing, in pixels, of this font. The line spacing is the vertical distance between the base lines of two consecutive lines of text. Thus, the line spacing includes the blank space between lines along with the height of the character itself.
GetHeight(Graphics*)	The Font::GetHeight method gets the line spacing of this font in the current unit of a specified Graphics object. The line spacing is the vertical distance between the base lines of two consecutive lines of text. Thus, the line spacing includes the blank space between lines along with the height of the character itself.

FontCollection Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [FontCollection](#) class. For a complete class listing, see [FontCollection Class](#).

- [GetFamilies](#)
- [GetFamilyCount](#)
- [GetLastStatus](#)

FontFamily.FontFamily constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [FontFamily](#) class. For a complete class listing, see [FontFamilyClass](#).

Overload list

CONSTRUCTOR	DESCRIPTION
FontFamily()	Creates an empty FontFamily::FontFamily object.
FontFamily(WCHAR*,FontCollection*)	Creates a FontFamily::FontFamily object based on a specified font family.

FontFamily Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [FontFamily](#) class. For a complete class listing, see [FontFamilyClass](#).

- [Clone](#)
- [GenericMonospace](#)
- [GenericSansSerif](#)
- [GenericSerif](#)
- [GetCellAscent](#)
- [GetCellDescent](#)
- [GetEmHeight](#)
- [GetFamilyName](#)
- [GetLastStatus](#)
- [GetLineSpacing](#)
- [IsAvailable](#)
- [IsStyleAvailable](#)

GdiplusBase Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **GdiplusBase** class. For a complete class listing, see **GdiplusBase Class**.

- [operator delete](#)
- [operator delete\[\]](#)
- [operator new](#)
- [operator new\[\]](#)

Graphics.Graphics constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the **Graphics** class. For a complete class listing, see [Graphics Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Graphics(HDC)	Creates a Graphics::Graphics object that is associated with a specified device context.
Graphics(Image*)	Creates a Graphics::Graphics object that is associated with an Image object.
Graphics(HWND,BOOL)	Creates a Graphics::Graphics object that is associated with a specified window.
Graphics(HDC,HANDLE)	Creates a Graphics::Graphics object that is associated with a specified device context and a specified device.

Graphics Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **Graphics** class. For a complete class listing, see [Graphics Class](#).

- [AddMetafileComment](#)
- [BeginContainer\(\)](#)
- [BeginContainer\(RectF&,RectF&,Unit\)](#)
- [BeginContainer\(Rect&,Rect&,Unit\)](#)
- [Clear](#)
- [DrawArc\(Pen*,Rect&,REAL,REAL\)](#)
- [DrawArc\(Pen*,RectF&,REAL,REAL\)](#)
- [DrawArc\(Pen*,REAL,REAL,REAL,REAL,REAL\)](#)
- [DrawArc\(Pen*,INT,INT,INT,INT,REAL,REAL\)](#)
- [DrawBezier\(Pen*,POINT&,POINT&,POINT&,POINT&\)](#)
- [DrawBezier\(Pen*,POINTERF&,POINTERF&,POINTERF&,POINTERF&\)](#)
- [DrawBezier\(Pen*,REAL,REAL,REAL,REAL,REAL,REAL,REAL\)](#)
- [DrawBezier\(Pen*,INT,INT,INT,INT,INT,INT,INT\)](#)
- [DrawBeziers\(Pen*,Point*,INT\)](#)
- [DrawBeziers\(Pen*,PointF*,INT\)](#)
- [DrawCachedBitmap](#)
- [DrawClosedCurve\(Pen*,Point*,INT\)](#)
- [DrawClosedCurve\(Pen*,PointF*,INT\)](#)
- [DrawClosedCurve\(Pen*,PointF*,INT,REAL\)](#)
- [DrawClosedCurve\(Pen*,Point*,INT,REAL\)](#)
- [DrawCurve\(Pen*,Point*,INT\)](#)
- [DrawCurve\(Pen*,PointF*,INT\)](#)
- [DrawCurve\(Pen*,PointF*,INT,REAL\)](#)
- [DrawCurve\(Pen*,Point*,INT,INT,INT,REAL\)](#)
- [DrawCurve\(Pen*,PointF*,INT,INT,INT,REAL\)](#)
- [DrawCurve\(Pen*,Point*,INT,REAL\)](#)
- [DrawDriverString](#)
- [DrawEllipse\(Pen*,Rect&\)](#)
- [DrawEllipse\(Pen*,REAL,REAL,REAL,REAL\)](#)
- [DrawEllipse\(Pen*,RectF&\)](#)
- [DrawEllipse\(Pen*,INT,INT,INT,INT\)](#)
- [DrawImage\(Image*,Point*,INT\)](#)
- [DrawImage\(Image*,INT,INT\)](#)
- [DrawImage\(Image*,Point&\)](#)
- [DrawImage\(Image*,REAL,REAL\)](#)
- [DrawImage\(Image*,PointF&\)](#)
- [DrawImage\(Image*,PointF*,INT\)](#)
- [DrawImage\(Image*,REAL,REAL,REAL,REAL,REAL,REAL,Unit\)](#)
- [DrawImage\(Image*,RectF&\)](#)

- `DrawImage(Image*,INT,INT,INT,INT)`
- `DrawImage(Image*,PointF*,INT,REAL,REAL,REAL,REAL,Unit,ImageAttributes*,DrawImageAbort,VOID*)`
- `DrawImage(Image*,Rect&,INT,INT,INT,INT,Unit,ImageAttributes*,DrawImageAbort,VOID*)`
- `DrawImage(Image*,Point*,INT,INT,INT,INT,Unit,ImageAttributes*,DrawImageAbort,VOID*)`
- `DrawImage(Image*,REAL,REAL,REAL,REAL)`
- `DrawImage(Image*,Rect&)`
- `DrawImage(Image*,INT,INT,INT,INT,INT,Unit)`
- `DrawImage(Image*,RectF&,REAL,REAL,REAL,REAL,Unit,ImageAttributes*,DrawImageAbort,VOID*)`
- `DrawImage(Image*,RectF&,RectF&,Unit,ImageAttributes*)`
- `DrawImage(Image*,RectF*,Matrix*,Effect*,ImageAttributes*,Unit*)`
- `DrawLine(Pen*,Point&,Point&)`
- `DrawLine(Pen*,PointF&,PointF&)`
- `DrawLine(Pen*,REAL,REAL,REAL,REAL)`
- `DrawLine(Pen*,INT,INT,INT,INT)`
- `DrawLines(Pen*,Point*,INT)`
- `DrawLines(Pen*,PointF*,INT)`
- `DrawPath`
- `DrawPie(Pen*,Rect&,REAL,REAL)`
- `DrawPie(Pen*,INT,INT,INT,INT,REAL,REAL)`
- `DrawPie(Pen*,REAL,REAL,REAL,REAL,REAL,REAL)`
- `DrawPie(Pen*,RectF&,REAL,REAL)`
- `DrawPolygon(Pen*,Point*,INT*)`
- `DrawPolygon(Pen*,PointF*,INT*)`
- `DrawRectangle(Pen*,Rect&)`
- `DrawRectangle(Pen*,INT,INT,INT,INT)`
- `DrawRectangle(Pen*,REAL,REAL,REAL,REAL)`
- `DrawRectangle(Pen*,RectF&)`
- `DrawRectangles(Pen*,Rect*,INT)`
- `DrawRectangles(Pen*,RectF*,INT)`
- `DrawString(WCHAR*,INT,Font*,RectF&,StringFormat*,Brush*)`
- `DrawString(WCHAR*,INT,Font*,PointF&,Brush*)`
- `DrawString(WCHAR*,INT,Font*,PointF&,StringFormat*,Brush*)`
- `EndContainer`
- `EnumerateMetafile(Metafile*,Metafile&,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,Point*,INT,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,Point&,Rect&,Unit,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,Point*,INT,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,Rect&,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,RectF&,RectF&,Unit,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,RectF&,RectF&,Unit,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,Point&,Rect&,Unit,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,Point*,INT,Rect&,Unit,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,Rect&,Rect&,Unit,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,Point*,INT,RectF&,Unit,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `EnumerateMetafile(Metafile*,PointF&,EnumerateMetafileProc,VOID*,ImageAttributes*)`
- `ExcludeClip(Rect&)`

- [ExcludeClip\(RectF&\)](#)
- [ExcludeClip\(Region*\)](#)
- [FillClosedCurve\(Brush*,Point*,INT\)](#)
- [FillClosedCurve\(Brush*,Point*,INT,FillMode,REAL\)](#)
- [FillClosedCurve\(Brush*,PointF*,INT\)](#)
- [FillClosedCurve\(Brush*,PointF*,INT,FillMode,REAL\)](#)
- [FillEllipse\(Brush*,Rect&\)](#)
- [FillEllipse\(Brush*,REAL,REAL,REAL,REAL\)](#)
- [FillEllipse\(Brush*,RectF&\)](#)
- [FillEllipse\(Brush*,INT,INT,INT,INT\)](#)
- [FillPath](#)
- [FillPie\(Brush*,Rect&,REAL,REAL\)](#)
- [FillPie\(Brush*,INT,INT,INT,INT,REAL,REAL\)](#)
- [FillPie\(Brush*,REAL,REAL,REAL,REAL,REAL,REAL\)](#)
- [FillPie\(Brush*,RectF&,REAL,REAL\)](#)
- [FillPolygon\(Brush*,Point*,INT\)](#)
- [FillPolygon\(Brush*,PointF*,INT\)](#)
- [FillPolygon\(Brush*,Point*,INT,FillMode\)](#)
- [FillPolygon\(Brush*,PointF*,INT,FillMode\)](#)
- [FillRectangle\(Brush*,Rect&\)](#)
- [FillRectangle\(Brush*,RectF&\)](#)
- [FillRectangle\(Brush*,REAL,REAL,REAL,REAL\)](#)
- [FillRectangle\(Brush*,INT,INT,INT,INT\)](#)
- [FillRectangles\(Brush*,Rect*,INT\)](#)
- [FillRectangles\(Brush*,RectF*,INT\)](#)
- [FillRegion](#)
- [Flush](#)
- [FromHDC\(HDC\)](#)
- [FromHDC\(HDD,HANDLE\)](#)
- [FromHWND](#)
- [FromImage](#)
- [GetClip](#)
- [GetClipBounds\(Rect*\)](#)
- [GetClipBounds\(RectF*\)](#)
- [GetCompositingMode](#)
- [GetCompositingQuality](#)
- [GetDpiX](#)
- [GetDpiY](#)
- [GetHalftonePalette](#)
- [GetHDC](#)
- [GetInterpolationMode](#)
- [GetLastStatus](#)
- [GetNearestColor](#)
- [GetPageScale](#)
- [GetPageUnit](#)
- [GetPixelOffsetMode](#)

- [GetRenderingOrigin](#)
- [GetSmoothingMode](#)
- [GetTextContrast](#)
- [GetTextRenderingHint](#)
- [GetTransform](#)
- [GetVisibleClipBounds\(Rect*\)](#)
- [GetVisibleClipBounds\(RectF*\)](#)
- [IntersectClip\(Rect&\)](#)
- [IntersectClip\(Region*\)](#)
- [IntersectClip\(RectF&\)](#)
- [IsClipEmpty](#)
- [IsVisible\(Point&\)](#)
- [IsVisible\(Rect&\)](#)
- [IsVisible\(REAL,REAL\)](#)
- [IsVisible\(RectF&\)](#)
- [IsVisible\(INT,INT,INT,INT\)](#)
- [IsVisible\(INT,INT\)](#)
- [IsVisible\(PointF&\)](#)
- [IsVisible\(REAL,REAL,REAL,REAL\)](#)
- [IsVisibleClipEmpty](#)
- [MeasureCharacterRanges](#)
- [MeasureDriverString](#)
- [MeasureString\(WCHAR*,INT,Font*,RectF&,RectF*\)](#)
- [MeasureString\(WCHAR*,INT,Font*,PointF&,StringFormat*,RectF*\)](#)
- [MeasureString\(WCHAR*,INT,Font*,RectF&,StringFormat*,RectF*,INT*,INT*\)](#)
- [MeasureString\(WCHAR*,INT,Font*,SizeF&,StringFormat*,SizeF*,INT*,INT*\)](#)
- [MeasureString\(WCHAR*,INT,Font*,PointF&,RectF*\)](#)
- [MultiplyTransform](#)
- [ReleaseHDC](#)
- [ResetClip](#)
- [ResetTransform](#)
- [Restore](#)
- [RotateTransform](#)
- [Save](#)
- [ScaleTransform](#)
- [SetAbort](#)
- [SetClip\(Graphics*,CombineMode\)](#)
- [SetClip\(GraphicsPath*,CombineMode\)](#)
- [SetClip\(Region*,CombineMode\)](#)
- [SetClip\(Rect&,CombineMode\)](#)
- [SetClip\(HRGN,CombineMode\)](#)
- [SetClip\(RectF&,CombineMode\)](#)
- [SetCompositingMode](#)
- [SetCompositingQuality](#)
- [SetInterpolationMode](#)
- [SetPageScale](#)

- [SetPageUnit](#)
- [SetPixelOffsetMode](#)
- [SetRenderingOrigin](#)
- [SetSmoothingMode](#)
- [SetTextContrast](#)
- [SetTextRenderingHint](#)
- [SetTransform](#)
- [TransformPoints](#)
- [TranslateClip\(INT,INT\)](#)
- [TranslateClip\(REAL,REAL\)](#)
- [TranslateTransform](#)

Graphics.BeginContainer methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the BeginContainer methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
BeginContainer()	The Graphics::BeginContainer method begins a new graphics container.
BeginContainer(Rect&,Rect&,Unit)	The Graphics::BeginContainer method begins a new graphics container.
BeginContainer(RectF&,RectF&,Unit)	The Graphics::BeginContainer method begins a new graphics container.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawArc methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawArc methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawArc(Pen*,Rect&,REAL,REAL)	The Graphics::DrawArc method draws an arc. The arc is part of an ellipse.
DrawArc(Pen*,RectF&,REAL,REAL)	The Graphics::DrawArc method draws an arc. The arc is part of an ellipse.
DrawArc(Pen*,INT,INT,INT,INT,REAL,REAL)	The Graphics::DrawArc method draws an arc. The arc is part of an ellipse.
DrawArc(Pen*,REAL,REAL,REAL,REAL,REAL,REAL)	The Graphics::DrawArc method draws an arc. The arc is part of an ellipse.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawBezier methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawBezier methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawBezier(Pen*,POINT&,POINT&,POINT&,POINT&)	The Graphics::DrawBezier method draws a Bézier spline.
DrawBezier(Pen*,INT,INT,INT,INT,INT,INT,INT,INT)	The Graphics::DrawBezier method draws a Bézier spline.
DrawBezier(Pen*,POINTF&,POINTF&,POINTF&,POINTF&)	The Graphics::DrawBezier method draws a Bézier spline.
DrawBezier(Pen*,REAL,REAL,REAL,REAL,REAL,REAL,REAL,REAL)	The Graphics::DrawBezier method draws a Bézier spline.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawBeziers methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawBeziers methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawBeziers(Pen*,Point*,INT)	The Graphics::DrawBeziers method draws a sequence of connected Bézier splines.
DrawBeziers(Pen*,PointF*,INT)	The Graphics::DrawBeziers method draws a sequence of connected Bézier splines.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawClosedCurve methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawClosedCurve methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawClosedCurve(Pen*,Point*,INT)	The Graphics::DrawClosedCurve method draws a closed cardinal spline.
DrawClosedCurve(Pen*,PointF*,INT)	The Graphics::DrawClosedCurve method draws a closed cardinal spline.
DrawClosedCurve(Pen*,Point*,INT,REAL)	The Graphics::DrawClosedCurve method draws a closed cardinal spline.
DrawClosedCurve(Pen*,PointF*,INT,REAL)	The Graphics::DrawClosedCurve method draws a closed cardinal spline.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawCurve methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawCurve methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawCurve(Pen*,Point*,INT)	The Graphics::DrawCurve method draws a cardinal spline.
DrawCurve(Pen*,PointF*,INT)	The Graphics::DrawCurve method draws a cardinal spline.
DrawCurve(Pen*,Point*,INT,REAL)	The Graphics::DrawCurve method draws a cardinal spline.
DrawCurve(Pen*,PointF*,INT,REAL)	The Graphics::DrawCurve method draws a cardinal spline.
DrawCurve(Pen*,Point*,INT,INT,INT,REAL)	The Graphics::DrawCurve method draws a cardinal spline.
DrawCurve(Pen*,PointF*,INT,INT,INT,REAL)	The Graphics::DrawCurve method draws a cardinal spline.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawEllipse methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawEllipse methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawEllipse(Pen*,Rect&)	The Graphics::DrawEllipse method draws an ellipse.
DrawEllipse(Pen*,RectF&)	The Graphics::DrawEllipse method draws an ellipse.
DrawEllipse(Pen*,INT,INT,INT,INT)	The Graphics::DrawEllipse method draws an ellipse.
DrawEllipse(Pen*,REAL,REAL,REAL,REAL)	The Graphics::DrawEllipse method draws an ellipse.

Graphics.DrawImage methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawImage methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawImage(Image*,Rect&)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,Point&)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,RectF&)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,INT,INT)	The Graphics::DrawImage method draws an image at a specified location.
DrawImage(Image*,PointF&)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,REAL,REAL)	The Graphics::DrawImage method draws an image at a specified location.
DrawImage(Image*,Point*,INT)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,PointF*,INT)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,INT,INT,INT,INT)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,REAL,REAL,REAL,REAL)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,INT,INT,INT,INT,INT,INT,Unit)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,REAL,REAL,REAL,REAL,REAL,REAL,Unit)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,RectF&,RectF&,Unit,ImageAttributes*)	The Graphics::DrawImage method draws a specified portion of an image at a specified location.
DrawImage(Image*,RectF*,Matrix*,Effect*,ImageAttributes*,Unit*)	The method draws a portion of an image after applying a specified effect.
DrawImage(Image*,Rect&,INT,INT,INT,INT,Unit,ImageAttributes*,DrawImageAbort,VOID*)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,Point*,INT,INT,INT,INT,INT,Unit,ImageAttributes*,DrawImageAbort,VOID*)	The Graphics::DrawImage method draws an image.
DrawImage(Image*,RectF&,REAL,REAL,REAL,REAL,Unit,ImageAttributes*,DrawImageAbort,VOID*)	The Graphics::DrawImage method draws an image.

METHOD	DESCRIPTION
<code>DrawImage(Image*,PointF*,INT,REAL,REAL,REAL,REAL, Unit,ImageAttributes*,DrawImageAbort,VOID*)</code>	The Graphics::DrawImage method draws an image.

Graphics.DrawLine methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawLine methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawLine(Pen*,Point&,Point&)	The Graphics::DrawLine method draws a line that connects two points.
DrawLine(Pen*,INT,INT,INT,INT)	The Graphics::DrawLine method draws a line that connects two points.
DrawLine(Pen*,PointF&,PointF&)	The Graphics::DrawLine method draws a line that connects two points.
DrawLine(Pen*,REAL,REAL,REAL,REAL)	The Graphics::DrawLine method draws a line that connects two points.

Graphics.DrawLines methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawLines methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawLines(Pen*,Point*,INT)	The Graphics::DrawLines method draws a sequence of connected lines.
DrawLines(Pen*,PointF*,INT)	The Graphics::DrawLines method draws a sequence of connected lines.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawPie methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawPie methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawPie(Pen*,Rect&,REAL,REAL)	The Graphics::DrawPie method draws a pie.
DrawPie(Pen*,RectF&,REAL,REAL)	The Graphics::DrawPie method draws a pie.
DrawPie(Pen*,INT,INT,INT,INT,REAL,REAL)	The Graphics::DrawPie method draws a pie.
DrawPie(Pen*,REAL,REAL,REAL,REAL,REAL,REAL)	The Graphics::DrawPie method draws a pie.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawPolygon methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawPolygon methods of the **Graphics** class. For a complete list of methods for the **Graphics** class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawPolygon(Pen*,Point*,INT*)	The Graphics::DrawPolygon method draws a polygon.
DrawPolygon(Pen*,PointF*,INT*)	The Graphics::DrawPolygon method draws a polygon.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawRectangle methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawRectangle methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawRectangle(Pen*,Rect&)	The Graphics::DrawRectangle method draws a rectangle.
DrawRectangle(Pen*,RectF&)	The Graphics::DrawRectangle method draws a rectangle.
DrawRectangle(Pen*,INT,INT,INT,INT)	The Graphics::DrawRectangle method draws a rectangle.
DrawRectangle(Pen*,REAL,REAL,REAL,REAL)	The Graphics::DrawRectangle method draws a rectangle.

Graphics.DrawRectangles methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the DrawRectangles methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
DrawRectangles(Pen*,Rect*,INT)	The Graphics::DrawRectangles method draws a sequence of rectangles.
DrawRectangles(Pen*,RectF*,INT)	The Graphics::DrawRectangles method draws a sequence of rectangles.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.DrawString methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the `DrawString` methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
<code>DrawString(WCHAR*,INT,Font*,PointF&,Brush*)</code>	The Graphics::DrawString method draws a string based on a font and an origin for the string.
<code>DrawString(WCHAR*,INT,Font*,RectF&,StringFormat*,Brush*)</code>	The Graphics::DrawString method draws a string based on a font, a layout rectangle, and a format.
<code>DrawString(WCHAR*,INT,Font*,PointF&,StringFormat*,Brush*)</code>	The Graphics::DrawString method draws a string based on a font, a string origin, and a format.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.EnumerateMetafile methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the `EnumerateMetafile` methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
<code>EnumerateMetafile(Metafile*, Rect&, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.
<code>EnumerateMetafile(Metafile*, RectF&, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.
<code>EnumerateMetafile(Metafile*, PointF&, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.
<code>EnumerateMetafile(Metafile*, Metafile&, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.
<code>EnumerateMetafile(Metafile*, Point*, INT, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.
<code>EnumerateMetafile(Metafile*, Rect&, Rect&, Unit, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.
<code>EnumerateMetafile(Metafile*, Point&, Rect&, Unit, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.
<code>EnumerateMetafile(Metafile*, RectF&, RectF&, Unit, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.
<code>EnumerateMetafile(Metafile*, Point*, INT, Rect&, Unit, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.
<code>EnumerateMetafile(Metafile*, Point*, INT, RectF&, Unit, EnumerateMetafileProc, VOID*, ImageAttributes*)</code>	The Graphics::EnumerateMetafileMetafile::PlayRecord in the callback function.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.ExcludeClip methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the ExcludeClip methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
ExcludeClip(Rect&)	The Graphics::ExcludeClip method updates the clipping region to the portion of itself that does not intersect the specified rectangle.
ExcludeClip(RectF&)	The Graphics::ExcludeClip method updates the clipping region to the portion of itself that does not intersect the specified rectangle.
ExcludeClip(Region*)	The Graphics::ExcludeClip method updates the clipping region with the portion of itself that does not overlap the specified region.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.FillClosedCurve methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the FillClosedCurve methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
FillClosedCurve(Brush*,Point*,INT)	The Graphics::FillClosedCurve method creates a closed cardinal spline from an array of points and uses a brush to fill the interior of the spline.
FillClosedCurve(Brush*,PointF*,INT)	The Graphics::FillClosedCurve method creates a closed cardinal spline from an array of points and uses a brush to fill the interior of the spline.
FillClosedCurve(Brush*,Point*,INT,FillMode,REAL)	The Graphics::FillClosedCurve method creates a closed cardinal spline from an array of points and uses a brush to fill, according to a specified mode, the interior of the spline.
FillClosedCurve(Brush*,PointF*,INT,FillMode,REAL)	The Graphics::FillClosedCurve method creates a closed cardinal spline from an array of points and uses a brush to fill, according to a specified mode, the interior of the spline.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.FillEllipse methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the FillEllipse methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
FillEllipse(Brush*,Rect&)	The Graphics::FillEllipse method uses a brush to fill the interior of an ellipse that is specified by a rectangle.
FillEllipse(Brush*,RectF&)	The Graphics::FillEllipse method uses a brush to fill the interior of an ellipse that is specified by a rectangle.
FillEllipse(Brush*,INT,INT,INT,INT)	The Graphics::FillEllipse method uses a brush to fill the interior of an ellipse that is specified by coordinates and dimensions.
FillEllipse(Brush*,REAL,REAL,REAL,REAL)	The Graphics::FillEllipse method uses a brush to fill the interior of an ellipse that is specified by coordinates and dimensions.

Graphics.FillPie methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the FillPie methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
FillPie(Brush*,Rect&,REAL,REAL)	The Graphics::FillPie method uses a brush to fill the interior of a pie.
FillPie(Brush*,RectF&,REAL,REAL)	The Graphics::FillPie method uses a brush to fill the interior of a pie.
FillPie(Brush*,INT,INT,INT,INT,REAL,REAL)	The Graphics::FillPie method uses a brush to fill the interior of a pie.
FillPie(Brush*,REAL,REAL,REAL,REAL,REAL,REAL)	The Graphics::FillPie method uses a brush to fill the interior of a pie.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.FillPolygon methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the FillPolygon methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
FillPolygon(Brush*,Point*,INT)	The Graphics::FillPolygon method uses a brush to fill the interior of a polygon.
FillPolygon(Brush*,PointF*,INT)	The Graphics::FillPolygon method uses a brush to fill the interior of a polygon.
FillPolygon(Brush*,Point*,INT,FillMode)	The Graphics::FillPolygon method uses a brush to fill the interior of a polygon.
FillPolygon(Brush*,PointF*,INT,FillMode)	The Graphics::FillPolygon method uses a brush to fill the interior of a polygon.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.FillRectangle methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the FillRectangle methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
FillRectangle(Brush*,Rect&)	The Graphics::FillRectangle method uses a brush to fill the interior of a rectangle.
FillRectangle(Brush*,RectF&)	The Graphics::FillRectangle method uses a brush to fill the interior of a rectangle.
FillRectangle(Brush*,INT,INT,INT,INT)	The Graphics::FillRectangle method uses a brush to fill the interior of a rectangle.
FillRectangle(Brush*,REAL,REAL,REAL,REAL)	The Graphics::FillRectangle method uses a brush to fill the interior of a rectangle.

Graphics.FillRectangles methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the FillRectangles methods of the **Graphics** class. For a complete list of methods for the **Graphics** class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
FillRectangles(Brush*,Rect*,INT)	The Graphics::FillRectangles method uses a brush to fill the interior of a sequence of rectangles.
FillRectangles(Brush*,RectF*,INT)	The Graphics::FillRectangles method uses a brush to fill the interior of a sequence of rectangles.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.FromHDC methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the FromHDC methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
FromHDC(HDC)	The Graphics::FromHDCGraphics object that is associated with a specified device context.
FromHDC(HD,DHANDLE)	The Graphics::FromHDCGraphics object that is associated with a specified device context and a specified device.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.GetClipBounds methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the GetClipBounds methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
GetClipBounds(Rect*)	The Graphics::GetClipBounds method gets a rectangle that encloses the clipping region of this Graphics object.
GetClipBounds(RectF*)	The Graphics::GetClipBoundsGraphics object.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.GetVisibleClipBounds methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the GetVisibleClipBounds methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
GetVisibleClipBounds(Rect*)	The <code>Graphics::GetVisibleClipBounds</code> method returns the clipping region of the window.
GetVisibleClipBounds(RectF*)	The <code>Graphics::GetVisibleClipBounds</code> method returns the clipping region of the window.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Graphics.IntersectClip methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the `IntersectClip` methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
IntersectClip(Rect&)	The <code>Graphics::IntersectClipGraphics</code> object.
IntersectClip(RectF&)	The <code>Graphics::IntersectClipGraphics</code> object.
IntersectClip(Region*)	The <code>Graphics::IntersectClipGraphics</code> object.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

Region::IsVisible methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the IsVisible methods of the [Region](#) class. For a complete list of methods for the [Region](#) class, see [Region Methods](#).

Overload list

METHOD	DESCRIPTION
IsVisible(Rect&,Graphics*)	The Region::IsVisible method determines whether a rectangle intersects this region.
IsVisible(Point&,Graphics*)	The Region::IsVisible method determines whether a point is inside this region.
IsVisible(RectF&,Graphics*)	The Region::IsVisible method determines whether a rectangle intersects this region.
IsVisible(INT,INT,Graphics*)	The Region::IsVisible method determines whether a point is inside this region.
IsVisible(PointF&,Graphics*)	The Region::IsVisible method determines whether a point is inside this region.
IsVisible(REAL,REAL,Graphics*)	The Region::IsVisible method determines whether a point is inside this region.
IsVisible(INT,INT,INT,INT,Graphics*)	The Region::IsVisible method determines whether a rectangle intersects this region.
IsVisible(REAL,REAL,REAL,REAL,Graphics*)	The Region::IsVisible method determines whether a rectangle intersects this region.

Graphics.MeasureString methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the `MeasureString` methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
MeasureString(WCHAR*,INT,Font*,RectF&,RectF*)	The Graphics::MeasureString method measures the extent of the string in the specified font and layout rectangle.
MeasureString(WCHAR*,INT,Font*,PointF&,RectF*)	The Graphics::MeasureString method measures the extent of the string in the specified font and layout rectangle.
MeasureString(WCHAR*,INT,Font*,PointF&,StringFormat at*,RectF*)	The Graphics::MeasureString method measures the extent of the string in the specified font, format, and layout rectangle.
MeasureString(WCHAR*,INT,Font*,RectF&,StringFormat t*,RectF*,INT*,INT*)	The Graphics::MeasureString method measures the extent of the string in the specified font, format, and layout rectangle.
MeasureString(WCHAR*,INT,Font*,SizeF&,StringFormat *,SizeF*,INT*,INT*)	The Graphics::MeasureString method measures the extent of the string in the specified font, format, and layout rectangle.

Graphics.SetClip methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the SetClip methods of the [Graphics](#) class. For a complete list of methods for the [Graphics](#) class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
SetClip(HRGN,CombineMode)	The Graphics::SetClip method updates the clipping region of this Graphics object to a region that is the combination of itself and a GDI region.
SetClip(Rect&,CombineMode)	The Graphics::SetClip method updates the clipping region of this Graphics object to a region that is the combination of itself and a rectangle.
SetClip(RectF&,CombineMode)	The Graphics::SetClip method updates the clipping region of this Graphics object to a region that is the combination of itself and a rectangle.
SetClip(Region*,CombineMode)	The Graphics::SetClip method updates the clipping region of this Graphics object to a region that is the combination of itself and the region specified by a Region object.
SetClip(Graphics*,CombineMode)	The Graphics::SetClip method updates the clipping region of this Graphics object to a region that is the combination of itself and the clipping region of another Graphics object.
SetClip(GraphicsPath*,CombineMode)	The Graphics::SetClip method updates the clipping region of this Graphics object to a region that is the combination of itself and the region specified by a graphics path. If a figure in the path is not closed, this method treats the nonclosed figure as if it were closed by a straight line that connects the figure's starting and ending points.

Graphics.TranslateClip methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the TranslateClip methods of the **Graphics** class. For a complete list of methods for the **Graphics** class, see [Graphics](#).

Overload list

METHOD	DESCRIPTION
TranslateClip(INT,INT)	The Graphics::TranslateClip method translates the clipping region of this Graphics object.
TranslateClip(REAL,REAL)	The Graphics::TranslateClip method translates the clipping region of this Graphics object.

Requirements

Header	Gdiplusgraphics.h
--------	-------------------

GraphicsPath.GraphicsPath constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [GraphicsPath](#) class. For a complete class listing, see [GraphicsPath Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
GraphicsPath(FillMode)	Creates a GraphicsPath::GraphicsPath object and initializes the fill mode. This is the default constructor.
GraphicsPath(Point*,BYTE*,INT,FillMode)	Creates a GraphicsPath::GraphicsPath object based on an array of points, an array of types, and a fill mode.
GraphicsPath(PointF*,BYTE*,INT,FillMode)	Creates a GraphicsPath::GraphicsPath object based on an array of points, an array of types, and a fill mode.

GraphicsPath Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [GraphicsPath](#) class. For a complete class listing, see [GraphicsPath Class](#).

- [AddArc\(Rect&,REAL,REAL\)](#)
- [AddArc\(RectF&,REAL,REAL\)](#)
- [AddArc\(INT,INT,INT,INT,REAL,REAL\)](#)
- [AddArc\(REAL,REAL,REAL,REAL,REAL,REAL\)](#)
- [AddBezier\(Point&,Point&,Point&,Point&\)](#)
- [AddBezier\(REAL,REAL,REAL,REAL,REAL,REAL,REAL,REAL\)](#)
- [AddBezier\(PointF&,PointF&,PointF&,PointF&\)](#)
- [AddBezier\(INT,INT,INT,INT,INT,INT,INT,INT\)](#)
- [AddBeziers\(Point*,INT\)](#)
- [AddBeziers\(PointF*,INT\)](#)
- [AddClosedCurve\(Point*,INT\)](#)
- [AddClosedCurve\(Point*,INT,REAL\)](#)
- [AddClosedCurve\(PointF*,INT,REAL\)](#)
- [AddClosedCurve\(PointF*,INT\)](#)
- [AddCurve\(Point*,INT\)](#)
- [AddCurve\(PointF*,INT,REAL\)](#)
- [AddCurve\(PointF*,INT\)](#)
- [AddCurve\(Point*,INT,INT,INT,REAL\)](#)
- [AddCurve\(Point*,INT,REAL\)](#)
- [AddCurve\(PointF*,INT,INT,INT,REAL\)](#)
- [AddEllipse\(Rect&\)](#)
- [AddEllipse\(RectF&\)](#)
- [AddEllipse\(INT,INT,INT,INT\)](#)
- [AddEllipse\(REAL,REAL,REAL,REAL\)](#)
- [AddLine\(Point&,Point&\)](#)
- [AddLine\(PointF&,PointF&\)](#)
- [AddLine\(REAL,REAL,REAL,REAL\)](#)
- [AddLine\(INT,INT,INT,INT\)](#)
- [AddLines\(Point*,INT\)](#)
- [AddLines\(PointF*,INT\)](#)
- [AddPath](#)
- [AddPie\(Rect&,REAL,REAL\)](#)
- [AddPie\(INT,INT,INT,INT,REAL,REAL\)](#)
- [AddPie\(REAL,REAL,REAL,REAL,REAL,REAL\)](#)
- [AddPie\(RectF&,REAL,REAL\)](#)
- [AddPolygon\(Point*,INT\)](#)
- [AddPolygon\(PointF*,INT\)](#)
- [AddRectangle\(Rect&\)](#)
- [AddRectangle\(RectF&\)](#)

- [AddRectangles\(Rect*,INT\)](#)
- [AddRectangles\(RectF*,INT\)](#)
- [AddString\(WCHAR*,INT,FontFamily*,INT,REAL,Rect&,StringFormat*\)](#)
- [AddString\(WCHAR*,INT,FontFamily*,INT,REAL,PointF&,StringFormat*\)](#)
- [AddString\(WCHAR*,INT,FontFamily*,INT,REAL,Point&,StringFormat*\)](#)
- [AddString\(WCHAR*,INT,FontFamily*,INT,REAL,RectF&,StringFormat*\)](#)
- [ClearMarkers](#)
- [Clone](#)
- [CloseAllFigures](#)
- [CloseFigure](#)
- [Flatten](#)
- [GetBounds\(Rect*,Matrix*,Pen*\)](#)
- [GetBounds\(RectF*,Matrix*,Pen*\)](#)
- [GetFillMode](#)
- [GetLastPoint](#)
- [GetLastStatus](#)
- [GetPathData](#)
- [GetPathPoints\(Point*,INT\)](#)
- [GetPathPoints\(PointF*,INT\)](#)
- [GetPathTypes](#)
- [GetPointCount](#)
- [IsOutlineVisible\(Point&,Pen*,Graphics*\)](#)
- [IsOutlineVisible\(REAL,REAL,Pen*,Graphics*\)](#)
- [IsOutlineVisible\(INT,INT,Pen*,Graphics*\)](#)
- [IsOutlineVisible\(PointF&,Pen*,Graphics*\)](#)
- [IsVisible\(REAL,REAL,Graphics*\)](#)
- [IsVisible\(PointF&,Graphics*\)](#)
- [IsVisible\(INT,INT,Graphics*\)](#)
- [IsVisible\(Point&,Graphics*\)](#)
- [Outline](#)
- [Reset](#)
- [Reverse](#)
- [SetFillMode](#)
- [SetMarker](#)
- [StartFigure](#)
- [Transform](#)
- [Warp](#)
- [Widen](#)

GraphicsPath.AddArc methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddArc methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddArc(Rect&,REAL,REAL)	The GraphicsPath::AddArc method adds an elliptical arc to the current figure of this path.
AddArc(RectF&,REAL,REAL)	The GraphicsPath::AddArc method adds an elliptical arc to the current figure of this path.
AddArc(INT,INT,INT,INT,REAL,REAL)	The GraphicsPath::AddArc method adds an elliptical arc to the current figure of this path.
AddArc(REAL,REAL,REAL,REAL,REAL,REAL)	The GraphicsPath::AddArc method adds an elliptical arc to the current figure of this path.

GraphicsPath.AddBezier methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddBezier methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddBezier(Point&, Point&, Point&, Point&)	The GraphicsPath::AddBezier method adds a Bézier spline to the current figure of this path.
AddBezier(INT, INT, INT, INT, INT, INT, INT, INT)	The GraphicsPath::AddBezier method adds a Bézier spline to the current figure of this path.
AddBezier(PointF&, PointF&, PointF&, PointF&)	The GraphicsPath::AddBezier method adds a Bézier spline to the current figure of this path.
AddBezier(REAL, REAL, REAL, REAL, REAL, REAL, REAL, REAL)	The GraphicsPath::AddBezier method adds a Bézier spline to the current figure of this path.

GraphicsPath.AddBeziers methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddBeziers methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddBeziers(Point*,INT)	The GraphicsPath::AddBeziers method adds a sequence of connected Bézier splines to the current figure of this path.
AddBeziers(PointF*,INT)	The GraphicsPath::AddBeziers method adds a sequence of connected Bézier splines to the current figure of this path.

GraphicsPath.AddClosedCurve methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddClosedCurve methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddClosedCurve(Point*,INT)	The GraphicsPath::AddClosedCurve method adds a closed cardinal spline to this path.
AddClosedCurve(PointF*,INT)	The GraphicsPath::AddClosedCurve method adds a closed cardinal spline to this path.
AddClosedCurve(Point*,INT,REAL)	The GraphicsPath::AddClosedCurve method adds a closed cardinal spline to this path.
AddClosedCurve(PointF*,INT,REAL)	The GraphicsPath::AddClosedCurve method adds a closed cardinal spline to this path.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPath.AddCurve methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddCurve methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddCurve(Point*,INT)	The GraphicsPath::AddCurve method adds a cardinal spline to the current figure of this path.
AddCurve(PointF*,INT)	The GraphicsPath::AddCurve method adds a cardinal spline to the current figure of this path.
AddCurve(Point*,INT,REAL)	The GraphicsPath::AddCurve method adds a cardinal spline to the current figure of this path.
AddCurve(PointF*,INT,REAL)	The GraphicsPath::AddCurve method adds a cardinal spline to the current figure of this path.
AddCurve(Point*,INT,INT,INT,REAL)	The GraphicsPath::AddCurve method adds a cardinal spline to the current figure of this path.
AddCurve(PointF*,INT,INT,INT,REAL)	The GraphicsPath::AddCurve method adds a cardinal spline to the current figure of this path.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPath.AddEllipse methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddEllipse methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddEllipse(Rect&)	The GraphicsPath::AddEllipse method adds an ellipse to this path.
AddEllipse(RectF&)	The GraphicsPath::AddEllipse method adds an ellipse to this path.
AddEllipse(INT,INT,INT,INT)	The GraphicsPath::AddEllipse method adds an ellipse to this path.
AddEllipse(REAL,REAL,REAL,REAL)	The GraphicsPath::AddEllipse method adds an ellipse to this path.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPath.AddLine methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddLine methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddLine(Point&,Point&)	The GraphicsPath::AddLine method adds a line to the current figure of this path.
AddLine(INT,INT,INT,INT)	The GraphicsPath::AddLine method adds a line to the current figure of this path.
AddLine(PointF&,PointF&)	The GraphicsPath::AddLine method adds a line to the current figure of this path.
AddLine(REAL,REAL,REAL,REAL)	The GraphicsPath::AddLine method adds a line to the current figure of this path.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPath.AddLines methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddLines methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddLines(Point*,INT)	The GraphicsPath::AddLines method adds a sequence of connected lines to the current figure of this path.
AddLines(PointF*,INT)	The GraphicsPath::AddLines method adds a sequence of connected lines to the current figure of this path.

GraphicsPath.AddPie methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddPie methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddPie(Rect&,REAL,REAL)	The GraphicsPath::AddPie method adds a pie to this path. An arc is a portion of an ellipse, and a pie is a portion of the area enclosed by an ellipse. A pie is bounded by an arc and two lines (edges) that go from the center of the ellipse to the endpoints of the arc.
AddPie(RectF&,REAL,REAL)	The GraphicsPath::AddPie method adds a pie to this path. An arc is a portion of an ellipse, and a pie is a portion of the area enclosed by an ellipse. A pie is bounded by an arc and two lines (edges) that go from the center of the ellipse to the endpoints of the arc.
AddPie(INT,INT,INT,INT,REAL,REAL)	The GraphicsPath::AddPie method adds a pie to this path. An arc is a portion of an ellipse, and a pie is a portion of the area enclosed by an ellipse. A pie is bounded by an arc and two lines (edges) that go from the center of the ellipse to the endpoints of the arc.
AddPie(REAL,REAL,REAL,REAL,REAL,REAL)	The GraphicsPath::AddPie method adds a pie to this path. An arc is a portion of an ellipse, and a pie is a portion of the area enclosed by an ellipse. A pie is bounded by an arc and two lines (edges) that go from the center of the ellipse to the endpoints of the arc.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPath.AddPolygon methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddPolygon methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddPolygon(Point*,INT)	The GraphicsPath::AddPolygon method adds a polygon to this path.
AddPolygon(PointF*,INT)	The GraphicsPath::AddPolygon method adds a polygon to this path.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPath.AddRectangle methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddRectangle methods of the **GraphicsPath** class. For a complete list of methods for the **GraphicsPath** class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddRectangle(Rect&)	The GraphicsPath::AddRectangle method adds a rectangle to this path.
AddRectangle(RectF&)	The GraphicsPath::AddRectangle method adds a rectangle to this path.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPath.AddRectangles methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddRectangles methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddRectangles(Rect*,INT)	The GraphicsPath::AddRectangles method adds a sequence of rectangles to this path.
AddRectangles(RectF*,INT)	The GraphicsPath::AddRectangles method adds a sequence of rectangles to this path.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPath.AddString methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the AddString methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
AddString(WCHAR*,INT,FontFamily*,INT,REAL,Rect&,StringFormat*)	The GraphicsPath::AddString method adds the outline of a string to this path.
AddString(WCHAR*,INT,FontFamily*,INT,REAL,Point&,StringFormat*)	The GraphicsPath::AddString method adds the outlines of a string to this path.
AddString(WCHAR*,INT,FontFamily*,INT,REAL,RectF&,StringFormat*)	The GraphicsPath::AddString method adds the outline of a string to this path.
AddString(WCHAR*,INT,FontFamily*,INT,REAL,PointF&,StringFormat*)	The GraphicsPath::AddString method adds the outline of a string to this path.

Region.GetBounds methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the GetBounds methods of the [Region](#) class. For a complete list of methods for the [Region](#) class, see [Region Methods](#).

Overload list

METHOD	DESCRIPTION
GetBounds(Rect*,Graphics*)	The Region::GetBounds method gets a rectangle that encloses this region.
GetBounds(RectF*,Graphics*)	The Region::GetBounds method gets a rectangle that encloses this region.

GraphicsPath::GetPathPoints methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the GetPathPoints methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
GetPathPoints(Point*,INT)	The GraphicsPath::GetPathPoints method gets this path's array of points. The array contains the endpoints and control points of the lines and Bézier splines that are used to draw the path.
GetPathPoints(PointF*,INT)	The GraphicsPath::GetPathPoints method gets this path's array of points. The array contains the endpoints and control points of the lines and Bézier splines that are used to draw the path.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPath::IsOutlineVisible methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the `IsOutlineVisible` methods of the [GraphicsPath](#) class. For a complete list of methods for the [GraphicsPath](#) class, see [GraphicsPath](#).

Overload list

METHOD	DESCRIPTION
IsOutlineVisible(Point&,Pen*,Graphics*)	The GraphicsPath::IsOutlineVisible method determines whether a specified point touches the outline of this path when the path is drawn by a specified Graphics object and a specified pen.
IsOutlineVisible(INT,INT,Pen*,Graphics*)	The GraphicsPath::IsOutlineVisible method determines whether a specified point touches the outline of this path when the path is drawn by a specified Graphics object and a specified pen.
IsOutlineVisible(PointF&,Pen*,Graphics*)	The GraphicsPath::IsOutlineVisibleGraphics object and a specified pen.
IsOutlineVisible(REAL,REAL,Pen*,Graphics*)	The GraphicsPath::IsOutlineVisible method determines whether a specified point touches the outline of this path when the path is drawn by a specified Graphics object and a specified pen.

GraphicsPathIterator Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **GraphicsPathIterator** class. For a complete class listing, see [GraphicsPathIterator Class](#).

- [CopyData](#)
- [Enumerate](#)
- [GetCount](#)
- [GetLastStatus](#)
- [GetSubpathCount](#)
- [HasCurve](#)
- [NextMarker\(GraphicsPath*\)](#)
- [NextMarker\(INT*,INT*\)](#)
- [NextPathType](#)
- [NextSubpath\(GraphicsPath*,BOOL*\)](#)
- [NextSubpath\(INT*,INT*,BOOL*\)](#)
- [Rewind](#)

GraphicsPathIterator.NextMarker methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the NextMarker methods of the [GraphicsPathIterator](#) class. For a complete list of methods for the [GraphicsPathIterator](#) class, see [GraphicsPathIterator Methods](#).

Overload list

METHOD	DESCRIPTION
NextMarker(INT*,INT*)	The GraphicsPathIterator::NextMarker method gets the starting index and the ending index of the next marker-delimited section in this iterator's associated path.
NextMarker(GraphicsPath*)	The GraphicsPathIterator::NextMarker method gets the next marker-delimited section of this iterator's associated path.

Requirements

Header	Gdipluspath.h
--------	---------------

GraphicsPathIterator.NextSubpath methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the NextSubpath methods of the **GraphicsPathIterator** class. For a complete list of methods for the **GraphicsPathIterator** class, see [GraphicsPathIterator Methods](#).

Overload list

METHOD	DESCRIPTION
NextSubpath(INT*,INT*,BOOL*)	The GraphicsPathIterator::NextSubpath method gets the starting index and the ending index of the next subpath (figure) in this iterator's associated path.
NextSubpath(GraphicsPath*,BOOL*)	The GraphicsPathIterator::NextSubpath method gets the next figure (subpath) from this iterator's associated path.

Requirements

Header	Gdipluspath.h
--------	---------------

HatchBrush Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **HatchBrush** class. For a complete class listing, see [HatchBrush Class](#).

- [Clone](#)
- [GetBackgroundColor](#)
- [GetForegroundColor](#)
- [GetHatchStyle](#)
- [GetLastStatus](#)
- [GetType](#)

HueSaturationLightness Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [HueSaturationLightness](#) class. For a complete class listing, see [HueSaturationLightness](#).

- [GetParameters](#)
- [SetParameters](#)

Image.Image constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [Image](#) class. For a complete class listing, see [Image Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Image(WCHAR*,BOOL)	Creates an Image::Image object based on a file.
Image(IStream*,BOOL)	Creates an Image::Image object based on a stream.

Image Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Image](#) class. For a complete class listing, see [Image Class](#).

- [Clone](#)
- [FindFirstItem](#)
- [FindNextItem](#)
- [FromFile](#)
- [FromStream](#)
- [GetAllPropertyItems](#)
- [GetBounds](#)
- [GetEncoderParameterList](#)
- [GetEncoderParameterListSize](#)
- [GetFlags](#)
- [GetFrameCount](#)
- [GetFrameDimensionsCount](#)
- [GetFrameDimensionsList](#)
- [GetHeight](#)
- [GetHorizontalResolution](#)
- [GetImageData](#)
- [GetLastStatus](#)
- [GetPalette](#)
- [GetPaletteSize](#)
- [GetPhysicalDimension](#)
- [GetPixelFormat](#)
- [GetPropertyCount](#)
- [GetPropertyIdList](#)
- [GetPropertyItem](#)
- [GetPropertyItemSize](#)
- [GetPropertySize](#)
- [GetRawFormat](#)
- [GetThumbnailImage](#)
- [GetType](#)
- [GetVerticalResolution](#)
- [GetWidth](#)
- [RemovePropertyItem](#)
- [RotateFlip](#)
- [Save\(IStream*,CLSID*,EncoderParameters*\)](#)
- [Save\(WCHAR*,CLSID*,EncoderParameters*\)](#)
- [SaveAdd\(EncoderParameters*\)](#)
- [SaveAdd\(Image*,EncoderParameters*\)](#)
- [SelectActiveFrame](#)
- [SetAbort](#)

- [SetPalette](#)
- [SetPropertyItem](#)

Image.Save methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Save methods of the [Image](#) class. For a complete list of methods for the [Image](#) class, see [Image Methods](#).

Overload list

METHOD	DESCRIPTION
Save(WCHAR*,CLSID*,EncoderParameters*)	The Image::Save method saves this image to a file.
Save(IStream*,CLSID*,EncoderParameters*)	The Image::Save method saves this image to a stream.

Image.SaveAdd methods

12/18/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the SaveAdd methods of the [Image](#) class. For a complete list of methods for the [Image](#) class, see [Image Methods](#).

Overload list

METHOD	DESCRIPTION
SaveAdd(EncoderParameters*)	The Image::SaveAdd method adds a frame to a file or stream specified in a previous call to the Save method. Use this method to save selected frames from a multiple-frame image to another multiple-frame image.
SaveAdd(Image*,EncoderParameters*)	The Image::SaveAdd method adds a frame to a file or stream specified in a previous call to the Save method.

Requirements

REQUIREMENT	VALUE
Minimum supported client	Windows XP; Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Product	GDI+ 1.0
Header	Gdiplusheaders.h (include Gdiplus.h)
Library	Gdiplus.lib

ImageAttributes Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [ImageAttributes](#) class. For a complete class listing, see [ImageAttributes Class](#).

- [ClearBrushRemapTable](#)
- [ClearColorKey](#)
- [ClearColorMatrices](#)
- [ClearColorMatrix](#)
- [ClearGamma](#)
- [ClearNoOp](#)
- [ClearOutputChannel](#)
- [ClearOutputChannelColorProfile](#)
- [ClearRemapTable](#)
- [ClearThreshold](#)
- [Clone](#)
- [GetAdjustedPalette](#)
- [GetLastStatus](#)
- [Reset](#)
- [SetBrushRemapTable](#)
- [SetColorKey](#)
- [SetColorMatrices](#)
- [SetColorMatrix](#)
- [SetGamma](#)
- [SetNoOp](#)
- [SetOutputChannel](#)
- [SetOutputChannelColorProfile](#)
- [SetRemapTable](#)
- [SetThreshold](#)
- [SetTolerance](#)
- [SetWrapMode](#)

InstalledFontCollection Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [InstalledFontCollection](#) class. For a complete class listing, see [InstalledFontCollection Class](#).

- [FontCollection::GetFamilies](#)
- [FontCollection::GetFamilyCount](#)
- [FontCollection::GetLastStatus](#)

Levels Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Levels](#) class. For a complete class listing, see [Levels](#).

- [GetParameters](#)
- [SetParameters](#)

LinearGradientBrush.LinearGradientBrush constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [LinearGradientBrush](#) class. For a complete class listing, see [LinearGradientBrush Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
LinearGradientBrush(Point&, Point&, Color&, Color&)	Creates a LinearGradientBrush::LinearGradientBrush object from a set of boundary points and boundary colors.
LinearGradientBrush(PointF&, PointF&, Color&, Color&)	Creates a LinearGradientBrush::LinearGradientBrush object from a set of boundary points and boundary colors.
LinearGradientBrush(Rect&, Color&, Color&, REAL, BOOL)	Creates a LinearGradientBrush::LinearGradientBrush object from a rectangle and angle of direction.
LinearGradientBrush(Rect&, Color&, Color&, LinearGradientMode)	Creates a LinearGradientBrush::LinearGradientBrush object based on a rectangle and mode of direction.
LinearGradientBrush(RectF&, Color&, Color&, LinearGradientMode)	Creates a LinearGradientBrush::LinearGradientBrush object based on a rectangle and mode of direction.

LinearGradientBrush Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [LinearGradientBrush](#) class. For a complete class listing, see [LinearGradientBrush Class](#).

- [Clone](#)
- [GetBlend](#)
- [GetBlendCount](#)
- [GetGammaCorrection](#)
- [GetInterpolationColorCount](#)
- [GetInterpolationColors](#)
- [GetLastStatus](#)
- [GetLinearColors](#)
- [GetRectangle\(Rect*\)](#)
- [GetRectangle\(RectF*\)](#)
- [GetTransform](#)
- [GetType](#)
- [GetWrapMode](#)
- [MultiplyTransform](#)
- [ResetTransform](#)
- [RotateTransform](#)
- [ScaleTransform](#)
- [SetBlend](#)
- [SetBlendBellShape](#)
- [SetBlendTriangularShape](#)
- [SetGammaCorrection](#)
- [SetInterpolationColors](#)
- [SetLinearColors](#)
- [SetTransform](#)
- [SetWrapMode](#)
- [TranslateTransform](#)

PathGradientBrush::GetRectangle methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the GetRectangle methods of the **PathGradientBrush** class. For a complete list of methods for the **PathGradientBrush** class, see [PathGradientBrush Methods](#).

Overload list

METHOD	DESCRIPTION
GetRectangle(Rect*)	The PathGradientBrush::GetRectangle method gets the smallest rectangle that encloses the boundary path of this path gradient brush.
GetRectangle(RectF*)	The PathGradientBrush::GetRectangle method gets the smallest rectangle that encloses the boundary path of this path gradient brush.

Matrix.Matrix constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [Matrix](#) class. For a complete class listing, see [Matrix Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Matrix(Rect&, Point*)	Creates a Matrix::Matrix object based on a rectangle and a point.
Matrix(RectF&, PointF*)	Creates a Matrix::Matrix object based on a rectangle and a point.
Matrix(REAL,REAL,REAL,REAL,REAL,REAL)	Creates and initializes a Matrix::Matrix object based on six numbers that define an affine transformation.
Matrix()	Creates and initializes a Matrix::Matrix object that represents the identity matrix.

Matrix Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **Matrix** class. For a complete class listing, see [Matrix Class](#).

- [Clone](#)
- [Equals](#)
- [GetElements](#)
- [GetLastStatus](#)
- [Invert](#)
- [IsIdentity](#)
- [IsInvertible](#)
- [Multiply](#)
- [OffsetX](#)
- [OffsetY](#)
- [Reset](#)
- [Rotate](#)
- [RotateAt](#)
- [Scale](#)
- [SetElements](#)
- [Shear](#)
- [TransformPoints\(Point*,INT\)](#)
- [TransformPoints\(PointF*,INT\)](#)
- [TransformVectors\(Point*,INT\)](#)
- [TransformVectors\(PointF*,INT\)](#)
- [Translate](#)

Matrix.TransformPoints methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the TransformPoints methods of the [Matrix](#) class. For a complete list of methods for the [Matrix](#) class, see [Matrix Methods](#).

Overload list

METHOD	DESCRIPTION
TransformPoints(Point*,INT)	The Matrix::TransformPoints method multiplies each point in an array by this matrix. Each point is treated as a row matrix. The multiplication is performed with the row matrix on the left and this matrix on the right.
TransformPoints(PointF*,INT)	The Matrix::TransformPoints method multiplies each point in an array by this matrix. Each point is treated as a row matrix. The multiplication is performed with the row matrix on the left and this matrix on the right.

Matrix.TransformVectors methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the TransformVectors methods of the [Matrix](#) class. For a complete list of methods for the [Matrix](#) class, see [Matrix Methods](#).

Overload list

METHOD	DESCRIPTION
TransformVectors(Point*,INT)	The Matrix::TransformVectors method multiplies each vector in an array by this matrix. The translation elements of this matrix (third row) are ignored. Each vector is treated as a row matrix. The multiplication is performed with the row matrix on the left and this matrix on the right.
TransformVectors(PointF*,INT)	The Matrix::TransformVectors method multiplies each vector in an array by this matrix. The translation elements of this matrix (third row) are ignored. Each vector is treated as a row matrix. The multiplication is performed with the row matrix on the left and this matrix on the right.

Requirements

Header	Gdiplusmatrix.h
--------	-----------------

Metafile.Metafile constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [Metafile](#) class. For a complete class listing, see [Metafile Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Metafile(WCHAR*)	Creates a Metafile::Metafile object for playback.
Metafile(IStream*)	Creates a Metafile::Metafile object from an IStream interface for playback.
Metafile(HENHMETAFILE,BOOL)	Creates a GDI+ Metafile::Metafile object for playback based on a GDI Enhanced Metafile (EMF) file.
Metafile(HDC,EmfType,WCHAR*)	Creates a Metafile::Metafile object for recording.
Metafile(WCHAR*,HDC,EmfType,WCHAR*)	Creates a Metafile::Metafile object for recording.
Metafile(IStream*,HDC,EmfType,WCHAR*)	Creates a Metafile::Metafile object for recording to an IStream interface.
Metafile(HMETAFILE,WmfPlaceableFileHeader*,BOOL)	Creates a GDI+ Metafile::Metafile object for recording. The format will be placeable metafile.
Metafile(HDC,Rect&,MetafileFrameUnit,EmfType,WCHAR*)	Creates a Metafile::Metafile object for recording.
Metafile(HDC,RectF&,MetaFileFrameUnit,EmfType,WCHAR*)	Creates a Metafile::Metafile object for recording.
Metafile(WCHAR*,HDC,Rect&,MetaFileFrameUnit,EmfType,WCHAR*)	Creates a Metafile::Metafile object for recording.
Metafile(WCHAR*,HDC,RectF&,MetafileFrameUnit,EmfType,WCHAR*)	Creates a Metafile::Metafile object for recording.
Metafile(IStream*,HDC,Rect&,MetafileFrameUnit,EmfType,WCHAR*)	Creates a Metafile::Metafile object for recording to an IStream interface.
Metafile(IStream*,HDC,RectF&,MetafileFrameUnit,EmfType,WCHAR*)	Creates a Metafile::Metafile object for recording to an IStream interface.

Metafile Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Metafile](#) class. For a complete class listing, see [Metafile Class](#).

- [Clone](#)
- [ConvertToEmfPlus](#)
- [ConvertToEmfPlus](#)
- [ConvertToEmfPlus](#)
- [EmfToWmfBits](#)
- [FindFirstItem](#)
- [FindNextItem](#)
- [FromFile](#)
- [FromStream](#)
- [GetAllPropertyItems](#)
- [GetBounds](#)
- [GetDownLevelRasterizationLimit](#)
- [GetEncoderParameterList](#)
- [GetEncoderParameterListSize](#)
- [GetFlags](#)
- [GetFrameCount](#)
- [GetFrameDimensionsCount](#)
- [GetFrameDimensionsList](#)
- [GetHeight](#)
- [GetHENHMETAFILE](#)
- [GetHorizontalResolution](#)
- [GetItemData](#)
- [GetLastStatus](#)
- [GetMetafileHeader\(WCHAR*,MetafileHeader*\)](#)
- [GetMetafileHeader\(HENHMETAFILE*,MetafileHeader*\)](#)
- [GetMetafileHeader\(HMETAFILE,WmfPlaceableFileHeader*,MetafileHeader*\)](#)
- [GetMetafileHeader\(MetafileHeader*\)](#)
- [GetMetafileHeader\(IStream*,MetafileHeader*\)](#)
- [GetPalette](#)
- [GetPaletteSize](#)
- [GetPhysicalDimension](#)
- [GetPixelFormat](#)
- [GetPropertyCount](#)
- [GetPropertyIdList](#)
- [GetPropertyItem](#)
- [GetPropertyItemSize](#)
- [GetPropertySize](#)
- [GetRawFormat](#)
- [GetThumbnailImage](#)

- [GetType](#)
- [GetVerticalResolution](#)
- [GetWidth](#)
- [PlayRecord](#)
- [RemovePropertyItem](#)
- [RotateFlip](#)
- [Save\(IStream*,CLSID*,EncoderParameters*\)](#)
- [Save\(WCHAR*,CLSID*,EncoderParameters*\)](#)
- [SaveAdd\(EncoderParameters*\)](#)
- [SaveAdd\(Image*,EncoderParameters*\)](#)
- [SelectActiveFrame](#)
- [SetAbort](#)
- [SetDownLevelRasterizationLimit](#)
- [SetPalette](#)
- [SetPropertyItem](#)

Metafile.ConvertToEmfPlus methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the ConvertToEmfPlus methods of the [Metafile](#) class. For a complete list of methods for the [Metafile](#) class, see [Metafile Methods](#).

Overload list

METHOD	DESCRIPTION
ConvertToEmfPlus(Graphics*,BOOL*,EmfType,WCHAR*)	The Metafile::ConvertToEmfPlus method converts this Metafile object to the EMF+ format.
ConvertToEmfPlus(Graphics*,WCHAR*,BOOL*,EmfType,WCHAR*)	The Metafile::ConvertToEmfPlus method converts this Metafile object to the EMF+ format.
ConvertToEmfPlus(Graphics*,IStream*,BOOL*,EmfType,WCHAR*)	The Metafile::ConvertToEmfPlus method converts this Metafile object to the EMF+ format.

Metafile.GetMetafileHeader methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the GetMetafileHeader methods of the [Metafile](#) class. For a complete list of methods for the [Metafile](#) class, see [Metafile Methods](#).

Overload list

METHOD	DESCRIPTION
GetMetafileHeader(MetafileHeader*)	The Metafile::GetMetafileHeader method gets the header.
GetMetafileHeader(WCHAR*,MetafileHeader*)	The Metafile::GetMetafileHeader method gets the header.
GetMetafileHeader(IStream*,MetafileHeader*)	The Metafile::GetMetafileHeader method gets the header.
GetMetafileHeader(HENHMETAFILE*,MetafileHeader*)	The Metafile::GetMetafileHeader method gets the header.
GetMetafileHeader(HMETAFILE,WmfPlaceableFileHeader*,MetafileHeader*)	The Metafile::GetMetafileHeader method gets the metafile header of this metafile.

MetafileHeader Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [MetafileHeader](#) class. For a complete class listing, see [MetafileHeader Class](#).

- [GetBounds](#)
- [GetDpiX](#)
- [GetDpiY](#)
- [GetEmfHeader](#)
- [GetEmfPlusFlags](#)
- [GetMetafileSize](#)
- [GetType](#)
- [GetVersion](#)
- [GetWmfHeader](#)
- [IsDisplay](#)
- [IsEmf](#)
- [IsEmfOrEmfPlus](#)
- [IsEmfPlus](#)
- [IsEmfPlusDual](#)
- [IsEmfPlusOnly](#)
- [IsWmf](#)
- [IsWmfPlaceable](#)

PathGradientBrush.PathGradientBrush constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the **PathGradientBrush** class. For a complete class listing, see [PathGradientBrushXX Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
PathGradientBrush(GraphicsPath*)	Creates a PathGradientBrush::PathGradientBrush object based on a GraphicsPath object.
PathGradientBrush(Point*,INT,WrapMode)	Creates a PathGradientBrush::PathGradientBrush object based on an array of points. Initializes the wrap mode of the path gradient brush.
PathGradientBrush(PointF*,INT,WrapMode)	Creates a PathGradientBrush object based on an array of points. Initializes the wrap mode of the path gradient brush.

PathGradientBrush Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [PathGradientBrush](#) class. For a complete class listing, see [PathGradientBrushXX Class](#).

- [Clone](#)
- [GetBlend](#)
- [GetBlendCount](#)
- [GetCenterColor](#)
- [GetCenterPoint\(Point*\)](#)
- [GetCenterPoint\(PointF*\)](#)
- [GetFocusScales](#)
- [GetGammaCorrection](#)
- [GetGraphicsPath](#)
- [GetInterpolationColorCount](#)
- [GetInterpolationColors](#)
- [GetLastStatus](#)
- [GetPointCount](#)
- [GetRectangle\(RectF*\)](#)
- [GetRectangle\(Rect*\)](#)
- [GetSurroundColorCount](#)
- [GetSurroundColors](#)
- [GetTransform](#)
- [GetType](#)
- [GetWrapMode](#)
- [MultiplyTransform](#)
- [ResetTransform](#)
- [RotateTransform](#)
- [ScaleTransform](#)
- [SetBlend](#)
- [SetBlendBellShape](#)
- [SetBlendTriangularShape](#)
- [SetCenterColor](#)
- [SetCenterPoint\(Point&\)](#)
- [SetCenterPoint\(PointF&\)](#)
- [SetFocusScales](#)
- [SetGammaCorrection](#)
- [SetGraphicsPath](#)
- [SetInterpolationColors](#)
- [SetSurroundColors](#)
- [SetTransform](#)
- [SetWrapMode](#)
- [TranslateTransform](#)

PathGradientBrush::GetCenterPoint methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the GetCenterPoint methods of the [PathGradientBrush](#) class. For a complete list of methods for the [PathGradientBrush](#) class, see [PathGradientBrush Methods](#).

Overload list

METHOD	DESCRIPTION
GetCenterPoint(Point*)	The PathGradientBrush::GetCenterPoint method gets the center point of this path gradient brush.
GetCenterPoint(PointF*)	The PathGradientBrush::GetCenterPoint method gets the center point of this path gradient brush.

PathGradientBrush::SetCenterPoint methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the SetCenterPoint methods of the [PathGradientBrush](#) class. For a complete list of methods for the [PathGradientBrush](#) class, see [PathGradientBrush Methods](#).

Overload list

METHOD	DESCRIPTION
SetCenterPoint(Point&)	The PathGradientBrush::SetCenterPoint method sets the center point of this path gradient brush. By default, the center point is at the centroid of the brush's boundary path, but you can set the center point to any location inside or outside the path.
SetCenterPoint(PointF&)	The PathGradientBrush::SetCenterPoint method sets the center point of this path gradient brush. By default, the center point is at the centroid of the brush's boundary path, but you can set the center point to any location inside or outside the path.

Requirements

Header	Gdipluspath.h
--------	---------------

Pen.Pen constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [Pen](#) class. For a complete class listing, see [Pen Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Pen(Brush*,REAL)	Creates a Pen object that uses the attributes of a brush and a real number to set the width of this Pen object.
Pen(Color&,REAL)	Creates a Pen object that uses a specified color and width.

Pen Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Pen](#) class. For a complete class listing, see [Pen Class](#).

- [Clone](#)
- [GetAlignment](#)
- [GetBrush](#)
- [GetColor](#)
- [GetCompoundArray](#)
- [GetCompoundArrayCount](#)
- [GetCustomEndCap](#)
- [GetCustomStartCap](#)
- [GetDashCap](#)
- [GetDashOffset](#)
- [GetDashPattern](#)
- [GetDashPatternCount](#)
- [GetDashStyle](#)
- [GetEndCap](#)
- [GetLastStatus](#)
- [GetLineJoin](#)
- [GetMiterLimit](#)
- [GetPenType](#)
- [GetStartCap](#)
- [GetTransform](#)
- [GetWidth](#)
- [MultiplyTransform](#)
- [ResetTransform](#)
- [RotateTransform](#)
- [ScaleTransform](#)
- [SetAlignment](#)
- [SetBrush](#)
- [SetColor](#)
- [SetCompoundArray](#)
- [SetCustomEndCap](#)
- [SetCustomStartCap](#)
- [SetDashCap](#)
- [SetDashOffset](#)
- [SetDashPattern](#)
- [SetDashStyle](#)
- [SetEndCap](#)
- [SetLineCap](#)
- [SetLineJoin](#)
- [SetMiterLimit](#)

- [SetStartCap](#)
- [SetTransform](#)
- [SetWidth](#)

Point.Point constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [Point](#) class. For a complete class listing, see [Point Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Point(Size&)	Creates a Point object using a Size object to initialize the X and Y data members.
Point(Point&)	Creates a new Point object and copies the data members from another Point object.
Point(INT,INT)	Creates a Point object using two integers to initialize the X and Y data members.
Point()	Creates a Point object and initializes the X and Y data members to zero. This is the default constructor.

Point Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Point](#) class. For a complete class listing, see [Point Class](#).

- [Equals](#)
- [operator-\(Point&\)](#)
- [operator+\(Point&\)](#)

PointF.PointF constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [PointF](#) class. For a complete class listing, see [PointF Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
PointF(SizeF&)	Creates a PointF object using a SizeF object to specify the X and Y data members.
PointF(PointF&)	Creates a new PointF object and copies the data from another PointF object.
PointF(REAL,REAL)	Creates a PointF object using two real numbers to specify the X and Y data members.
PointF()	Creates a PointF object and initializes the X and Y data members to zero. This is the default constructor.

PointF Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [PointF](#) class. For a complete class listing, see [PointF Class](#).

- [Equals](#)
- [operator-\(PointF&\)](#)
- [operator+\(PointF&\)](#)

PrivateFontCollection Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [PrivateFontCollection](#) class. For a complete class listing, see [PrivateFontCollection Class](#).

- [AddFontFile](#)
- [AddMemoryFont](#)
- [GetFamilies](#)
- [GetFamilyCount](#)
- [GetLastStatus](#)

Rect.Rect constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the **Rect** class. For a complete class listing, see [Rect Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Rect(Point&,Size&)	Creates a Rect object by using a Point object to initialize the X and Y data members and a Size object to initialize the Width and Height data members.
Rect(INT,INT,INT,INT)	Creates a Rect object by using four integers to initialize the X , Y , Width , and Height data members.
Rect()	Creates a Rect object whose x-coordinate, y-coordinate, width, and height are all zero. This is the default constructor.

Rect Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Rect](#) class. For a complete class listing, see [Rect Class](#).

- [Clone](#)
- [Contains\(Point&\)](#)
- [Contains\(Rect&\)](#)
- [Contains\(INT,INT\)](#)
- [Equals](#)
- [GetBottom](#)
- [GetBounds](#)
- [GetLeft](#)
- [GetLocation](#)
- [GetRight](#)
- [GetSize](#)
- [GetTop](#)
- [Inflate\(INT,INT\)](#)
- [Inflate\(Point&\)](#)
- [Intersect\(Rect&,Rect&,Rect&\)](#)
- [Intersect\(Rect&\)](#)
- [IntersectsWith](#)
- [IsEmptyArea](#)
- [Offset\(INT,INT\)](#)
- [Offset\(Point&\)](#)
- [Union](#)

RectF.Contains methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Contains methods of the [RectF](#) class. For a complete list of methods for the [RectF](#) class, see [RectF Methods](#).

Overload list

METHOD	DESCRIPTION
Contains(RectF&)	The RectF::Contains method determines whether another rectangle is inside this rectangle.
Contains(PointF&)	The RectF::Contains method determines whether a point is inside this rectangle.
Contains(REAL,REAL)	The RectF::Contains method determines whether the point (x, y) is inside this rectangle.

RectF Inflate methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Inflate methods of the [RectF](#) class. For a complete list of methods for the [RectF](#) class, see [RectF Methods](#).

Overload list

METHOD	DESCRIPTION
Inflate(PointF&)	The <code>RectF::Inflate</code> <i>point.X</i> <i>point.Y</i> on the top and bottom edges.
Inflate(REAL,REAL)	The <code>RectF::Inflate</code> <i>dxdy</i> on the top and bottom edges.

Requirements

Header	Gdiplustypes.h
--------	----------------

Region.Intersect methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Intersect methods of the [Region](#) class. For a complete list of methods for the [Region](#) class, see [Region Methods](#).

Overload list

METHOD	DESCRIPTION
Intersect(Rect&)	The Region::Intersect method updates this region to the portion of itself that intersects the specified rectangle's interior.
Intersect(RectF&)	The Region::Intersect method updates this region to the portion of itself that intersects the specified rectangle's interior.
Intersect(Region*)	The Region::Intersect method updates this region to the portion of itself that intersects another region.
Intersect(GraphicsPath*)	The Region::Intersect method updates this region to the portion of itself that intersects the specified path's interior.

RectF.Offset methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Offset methods of the [RectF](#) class. For a complete list of methods for the [RectF](#) class, see [Rect Methods](#).

Overload list

METHOD	DESCRIPTION
Offset(PointF&)	The RectF::Offset <i>pointX</i> <i>pointY</i> .
Offset(REAL,REAL)	The RectF::Offset <i>dxdy</i> vertically.

RectF.RectF constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [RectF](#) class. For a complete class listing, see [RectF Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
RectF(PointF&,SizeF&)	Creates a RectF object by using a PointF object to initialize the X and Y data members and uses a SizeF object to initialize the Width and Height data members of this rectangle.
RectF(REAL,REAL,REAL,REAL)	Creates a RectF object by using four integers to initialize the X , Y , Width , and Height data members.
RectF()	Creates a RectF object and initializes the X , Y , Width , and Height data members to zero. This is the default constructor.

RectF Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [RectF](#) class. For a complete class listing, see [RectF Class](#).

- [Clone](#)
- [Contains\(RectF&\)](#)
- [Contains\(PointF&\)](#)
- [Contains\(REAL,REAL\)](#)
- [Equals](#)
- [GetBottom](#)
- [GetBounds](#)
- [GetLeft](#)
- [GetLocation](#)
- [GetRight](#)
- [GetSize](#)
- [GetTop](#)
- [Inflate\(PointF&\)](#)
- [Inflate\(REAL,REAL\)](#)
- [Intersect\(RectF&\)](#)
- [Intersect\(RectF&,RectF&,RectF&\)](#)
- [IntersectsWith](#)
- [IsEmptyArea](#)
- [Offset\(PointF&\)](#)
- [Offset\(REAL,REAL\)](#)
- [Union](#)

RedEyeCorrection Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **RedEyeCorrection** class. For a complete class listing, see **RedEyeCorrection**.

- [GetParameters](#)
- [SetParameters](#)

Region.Region constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [Region](#) class. For a complete class listing, see [Region Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Region(HRGN)	Creates a region that is identical to the region that is specified by a handle to a GDI region.
Region(Rect&)	Creates a region that is defined by a rectangle.
Region(RectF&)	Creates a region that is defined by a rectangle.
Region(BYTE*,INT)	Creates a region that is defined by data obtained from another region.
Region(GraphicsPath*)	Creates a region that is defined by a path (a GraphicsPath object) and has a fill mode that is contained in the GraphicsPath object.
Region()	Creates a region that is infinite. This is the default constructor.

Region Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Region](#) class. For a complete class listing, see [Region Class](#).

- [Clone](#)
- [Complement\(GraphicsPath*\)](#)
- [Complement\(Region*\)](#)
- [Complement\(Rect&\)](#)
- [Complement\(RectF&\)](#)
- [Equals](#)
- [Exclude\(GraphicsPath*\)](#)
- [Exclude\(RectF&\)](#)
- [Exclude\(Rect&\)](#)
- [Exclude\(Region*\)](#)
- [FromHRGN](#)
- [GetBounds\(Rect*,Graphics*\)](#)
- [GetBounds\(RectF*,Graphics*\)](#)
- [GetData](#)
- [GetDataSize](#)
- [GetHRGN](#)
- [GetLastStatus](#)
- [GetRegionScans\(Matrix*,Rect*,INT*\)](#)
- [GetRegionScans\(Matrix*,RectF*,INT*\)](#)
- [GetRegionScansCount](#)
- [Intersect\(Rect&\)](#)
- [Intersect\(GraphicsPath*\)](#)
- [Intersect\(RectF&\)](#)
- [Intersect\(Region*\)](#)
- [IsEmpty](#)
- [IsInfinite](#)
- [IsVisible\(PointF&,Graphics*\)](#)
- [IsVisible\(RectF&,Graphics*\)](#)
- [IsVisible\(Rect&,Graphics*\)](#)
- [IsVisible\(INT,INT,Graphics*\)](#)
- [IsVisible\(REAL,REAL,Graphics*\)](#)
- [IsVisible\(INT,INT,INT,INT,Graphics*\)](#)
- [IsVisible\(Point&,Graphics*\)](#)
- [IsVisible\(REAL,REAL,REAL,REAL,Graphics*\)](#)
- [MakeEmpty](#)
- [MakeInfinite](#)
- [Transform](#)
- [Translate\(REAL,REAL\)](#)
- [Translate\(INT,INT\)](#)

- [Union\(Rect&\)](#)
- [Union\(Region*\)](#)
- [Union\(RectF&\)](#)
- [Union\(GraphicsPath*\)](#)
- [Xor\(GraphicsPath*\)](#)
- [Xor\(RectF&\)](#)
- [Xor\(Rect&\)](#)
- [Xor\(Region*\)](#)

Region.Complement methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Complement methods of the [Region](#) class. For a complete list of methods for the [Region](#) class, see [Region Methods](#).

Overload list

METHOD	DESCRIPTION
Complement(Rect&)	The Region::Complement method updates this region to the portion of the specified rectangle's interior that does not intersect this region.
Complement(RectF&)	The Region::Complement method updates this region to the portion of the specified rectangle's interior that does not intersect this region.
Complement(Region*)	The Region::Complement method updates this region to the portion of another region that does not intersect this region.
Complement(GraphicsPath*)	The Region::Complement method updates this region to the portion of the specified path's interior that does not intersect this region.

Region.Exclude methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Exclude methods of the [Region](#) class. For a complete list of methods for the [Region](#) class, see [Region Methods](#).

Overload list

METHOD	DESCRIPTION
Exclude(Rect&)	The Region::Exclude method updates this region to the portion of itself that does not intersect the specified rectangle's interior.
Exclude(RectF&)	The Region::Exclude method updates this region to the portion of itself that does not intersect the specified rectangle's interior.
Exclude(Region*)	The Region::Exclude method updates this region to the portion of itself that does not intersect another region.
Exclude(GraphicsPath*)	The Region::Exclude method updates this region to the portion of itself that does not intersect the specified path's interior.

Region.GetRegionScans methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the GetRegionScans methods of the [Region](#) class. For a complete list of methods for the [Region](#) class, see [Region Methods](#).

Overload list

METHOD	DESCRIPTION
GetRegionScans(Matrix*,Rect*,INT*)	The Region::GetRegionScans method gets an array of rectangles that approximate this region. The region is transformed by a specified matrix before the rectangles are calculated.
GetRegionScans(Matrix*,RectF*,INT*)	The Region::GetRegionScans method gets an array of rectangles that approximate this region. The region is transformed by a specified matrix before the rectangles are calculated.

Region.Translate methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Translate methods of the [Region](#) class. For a complete list of methods for the [Region](#) class, see [Region Methods](#).

Overload list

METHOD	DESCRIPTION
Translate(INT,INT)	The Region::Translate method offsets this region by specified amounts in the horizontal and vertical directions.
Translate(REAL,REAL)	The Region::Translate method offsets this region by specified amounts in the horizontal and vertical directions.

Region.Union methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Union methods of the [Region](#) class. For a complete list of methods for the [Region](#) class, see [Region Methods](#).

Overload list

METHOD	DESCRIPTION
Union(Rect&)	The Region::Union method updates this region to all portions (intersecting and nonintersecting) of itself and all portions of the specified rectangle's interior.
Union(RectF&)	The Region::Union method updates this region to all portions (intersecting and nonintersecting) of itself and all portions of the specified rectangle's interior.
Union(Region*)	The Region::Union method updates this region to all portions (intersecting and nonintersecting) of itself and all portions of another region.
Union(GraphicsPath*)	The Region::Union method updates this region to all portions (intersecting and nonintersecting) of itself and all portions of the specified path's interior.

Region.Xor methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the Xor methods of the [Region](#) class. For a complete list of methods for the [Region](#) class, see [Region Methods](#).

Overload list

METHOD	DESCRIPTION
Xor(Rect&)	The Region::Xor method updates this region to the nonintersecting portions of itself and the specified rectangle's interior.
Xor(RectF&)	The Region::Xor method updates this region to the nonintersecting portions of itself and the specified rectangle's interior.
Xor(Region*)	The Region::Xor method updates this region to the nonintersecting portions of itself and another region.
Xor(GraphicsPath*)	The Region::Xor method updates this region to the nonintersecting portions of itself and the specified path's interior.

Sharpen Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **Sharpen** class. For a complete class listing, see **Sharpen**.

- [GetParameters](#)
- [SetParameters](#)

Size.Size constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [Size](#) class. For a complete class listing, see [Size Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
Size(Size&)	Creates a Size object and initializes its members by copying the members of another Size object.
Size(INT,INT)	Creates a Size object and initializes its Width and Height data members.
Size()	Creates a new Size object and initializes the members to zero. This is the default constructor.

Size Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Size](#) class. For a complete class listing, see [Size Class](#).

- [Empty](#)
- [Equals](#)
- [operator-\(Size&\)](#)
- [operator+\(Size&\)](#)

SizeF.SizeF constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [SizeF](#) class. For a complete class listing, see [SizeF Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
SizeF(SizeF&)	Creates a SizeF object and initializes its members by copying the members of another SizeF object.
SizeF(REAL,REAL)	Creates a SizeF object and initializes its Width and Height data members.
SizeF()	Creates a SizeF object and initializes the members to zero. This is the default constructor.

SizeF Methods

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [SizeF](#) class. For a complete class listing, see [SizeF Class](#).

- [Empty](#)
- [Equals](#)
- [operator-\(SizeF&\)](#)
- [operator+\(SizeF&\)](#)

SolidBrush Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the **SolidBrush** class. For a complete class listing, see **SolidBrush Class**.

- [Clone](#)
- [GetColor](#)
- [GetLastStatus](#)
- [GetType](#)
- [SetColor](#)

StringFormat.StringFormat constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [StringFormat](#) class. For a complete class listing, see [StringFormat Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
StringFormat(INT,LANGID)	Creates a StringFormat object based on string format flags and a language.
StringFormat(StringFormat*)	Creates a StringFormat object from another StringFormat object.

Requirements

Header	Gdiplusstringformat.h
--------	-----------------------

StringFormat Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [StringFormat](#) class. For a complete class listing, see [StringFormat Class](#).

- [Clone](#)
- [GenericDefault](#)
- [GenericTypographic](#)
- [GetAlignment](#)
- [GetDigitSubstitutionLanguage](#)
- [GetDigitSubstitutionMethod](#)
- [GetFormatFlags](#)
- [GetHotkeyPrefix](#)
- [GetLastStatus](#)
- [GetLineAlignment](#)
- [GetMeasurableCharacterRangeCount](#)
- [GetTabStopCount](#)
- [GetTabStops](#)
- [GetTrimming](#)
- [SetAlignment](#)
- [SetDigitSubstitution](#)
- [SetFormatFlags](#)
- [SetHotkeyPrefix](#)
- [SetLineAlignment](#)
- [SetMeasurableCharacterRanges](#)
- [SetTabStops](#)
- [SetTrimming](#)

TextureBrush.TextureBrush constructors

11/2/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the constructors of the [TextureBrush](#) class. For a complete class listing, see [TextureBrush Class](#).

Overload list

CONSTRUCTOR	DESCRIPTION
TextureBrush(Image*,WrapMode)	Creates a TextureBrush object based on an image and a wrap mode. The size of the brush defaults to the size of the image, so the entire image is used by the brush.
TextureBrush(Image*,WrapMode,Rect&)	Creates a TextureBrush object based on an image, a wrap mode, and a defining rectangle.
TextureBrush(Image*,wrapMode,RectF&)	Creates a TextureBrush object based on an image, a wrap mode, and a defining rectangle.
TextureBrush(Image*,Rect&,ImageAttributes*)	Creates a TextureBrush object based on an image, a defining rectangle, and a set of image properties.
TextureBrush(Image*,RectF&,ImageAttributes*)	Creates a TextureBrush object based on an image, a defining rectangle, and a set of image properties.
TextureBrush(Image*,WrapMode,INT,INT,INT,INT)	Creates a TextureBrush object based on an image, a wrap mode, and a defining set of coordinates.
TextureBrush(Image*,WrapMode,REAL,REAL,REAL,REAL,REAL)	Creates a TextureBrush object based on an image, a wrap mode, and a defining set of coordinates.

TextureBrush Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [TextureBrush](#) class. For a complete class listing, see [TextureBrush Class](#).

- [Clone](#)
- [GetImage](#)
- [GetLastStatus](#)
- [GetTransform](#)
- [GetType](#)
- [GetWrapMode](#)
- [MultiplyTransform](#)
- [ResetTransform](#)
- [RotateTransform](#)
- [ScaleTransform](#)
- [SetTransform](#)
- [SetWrapMode](#)
- [TranslateTransform](#)

Tint Methods

2/22/2020 • 2 minutes to read • [Edit Online](#)

This topic lists the methods of the [Tint](#) class. For a complete class listing, see [Tint](#).

- [GetParameters](#)
- [SetParameters](#)

GDI+ functions

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ provides the following functions:

- [GdiplusShutdown](#)
- [GdiplusStartup](#)
- [GetImageDecoders](#)
- [GetImageDecodersSize](#)
- [GetImageEncoders](#)
- [GetImageEncodersSize](#)
- [GetPixelFormatSize](#)
- [IsAlphaPixelFormat](#)
- [IsCanonicalPixelFormat](#)
- [IsExtendedPixelFormat](#)
- [IsIndexedPixelFormat](#)
- [ObjectTypeIsValid](#)

Constants (GDI+)

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ defines constants in the following categories:

- [Image File Format Constants](#)
- [Image Frame Dimension Constants](#)
- [Image Encoder Constants](#)
- [Image Pixel Format Constants](#)
- [Image Property Tag Type Constants](#)
- [Image Property Tag Constants](#)

Image Effect Constants

2/22/2020 • 2 minutes to read • [Edit Online](#)

The [Effect](#) class has several descendants that can be used to apply a color effect or adjustment to a bitmap. Each of the descendants is identified by a GUID. The following constants, defined in Gdipluseffects.h, represent GUIDs that identify the various effects.

- [BlurEffectGuid](#)
- [BrightnessContrastEffectGuid](#)
- [ColorBalanceEffectGuid](#)
- [ColorCurveEffectGuid](#)
- [ColorLUTEffectGuid](#)
- [ColorMatrixEffectGuid](#)
- [HueSaturationLightnessEffectGuid](#)
- [LevelsEffectGuid](#)
- [RedEyeCorrectionEffectGuid](#)
- [SharpenEffectGuid](#)
- [TintEffectGuid](#)

Image Encoder Constants

11/2/2020 • 2 minutes to read • [Edit Online](#)

The [Image::Save Methods](#) and [Image::SaveAdd Methods](#) methods of the [Image](#) class receive an [EncoderParameters](#) object that contains an array of [EncoderParameter](#) objects. Each [EncoderParameter](#) object has a GUID data member that specifies the parameter category. The following constants, defined in [Gdiplusimaging.h](#), represent GUIDs that specify the various parameter categories.

- EncoderChrominanceTable
- EncoderColorDepth
- EncoderColorSpace
- EncoderCompression
- EncoderLuminanceTable
- EncoderQuality
- EncoderRenderMethod
- EncoderSaveFlag
- EncoderScanMethod
- EncoderTransformation
- EncoderVersion
- EncoderImageItems
- EncoderSaveAsCMYK

Image File Format Constants

2/22/2020 • 2 minutes to read • [Edit Online](#)

The [Image::GetRawFormat](#) method returns a GUID that indicates the file format of an image. The following constants, defined in Gdipplusimaging.h, represent the GUIDs that identify those file formats.

CONSTANT	DESCRIPTION
<code>ImageFormatBMP</code>	Indicates the Windows BMP format.
<code>ImageFormatEMF</code>	Indicates the EMF format.
<code>ImageFormatEXIF</code>	Indicates the Exif (Exchangeable Image File) format.
<code>ImageFormatGIF</code>	Indicates the GIF format.
<code>ImageFormatHEIF</code>	Indicates the HEIF (High Efficiency Image Format) format.
<code>ImageFormatIcon</code>	Indicates the Icon format.
<code>ImageFormatJPEG</code>	Indicates the JPEG format.
<code>ImageFormatMemoryBMP</code>	Indicates that the image was constructed from a memory bitmap.
<code>ImageFormatPNG</code>	Indicates the PNG format.
<code>ImageFormatTIFF</code>	Indicates the TIFF format.
<code>ImageFormatUndefined</code>	Indicates that GDI+ is unable to determine the format.
<code>ImageFormatWEBP</code>	Indicates the WebP format.
<code>ImageFormatWMF</code>	Indicates the WMF format.

Requirements

Header	Gdiplusimaging.h
--------	------------------

Image Frame Dimension Constants

2/22/2020 • 2 minutes to read • [Edit Online](#)

The Graphics Interchange Format (GIF) and Tagged Image File Format (TIFF) image file formats enable you to store multiple frames in a single image file. Multiple frames in a GIF file are used for animation, so the frames are said to be in the time dimension. Multiple frames in a TIFF file are typically used as separate pages, so the frames are said to be in the page dimension.

The following constants, defined in Gdiplusimaging.h, represent GUIDs that identify the page and time dimensions.

- `FrameDimensionPage`
- `FrameDimensionTime`

Image Pixel Format Constants

2/22/2020 • 2 minutes to read • [Edit Online](#)

The following constants, defined in Gdippluspixelformats.h, specify various pixel formats used in bitmaps.

CONSTANT	DESCRIPTION
PixelFormat1bppIndexed	Specifies that the format is 1 bit per pixel, indexed.
PixelFormat4bppIndexed	Specifies that the format is 4 bits per pixel, indexed.
PixelFormat8bppIndexed	Specifies that the format is 8 bits per pixel, indexed.
PixelFormat16bppARGB1555	Specifies that the format is 16 bits per pixel; 1 bit is used for the alpha component, and 5 bits each are used for the red, green, and blue components.
PixelFormat16bppGrayScale	Specifies that the format is 16 bits per pixel, grayscale.
PixelFormat16bppRGB555	Specifies that the format is 16 bits per pixel; 5 bits each are used for the red, green, and blue components. The remaining bit is not used.
PixelFormat16bppRGB565	Specifies that the format is 16 bits per pixel; 5 bits are used for the red component, 6 bits are used for the green component, and 5 bits are used for the blue component.
PixelFormat24bppRGB	Specifies that the format is 24 bits per pixel; 8 bits each are used for the red, green, and blue components.
PixelFormat32bppARGB	Specifies that the format is 32 bits per pixel; 8 bits each are used for the alpha, red, green, and blue components.
PixelFormat32bppPARGB	Specifies that the format is 32 bits per pixel; 8 bits each are used for the alpha, red, green, and blue components. The red, green, and blue components are premultiplied according to the alpha component.
PixelFormat32bppRGB	Specifies that the format is 32 bits per pixel; 8 bits each are used for the red, green, and blue components. The remaining 8 bits are not used.
PixelFormat48bppRGB	Specifies that the format is 48 bits per pixel; 16 bits each are used for the red, green, and blue components.

CONSTANT	DESCRIPTION
PixelFormat64bppARGB	Specifies that the format is 64 bits per pixel; 16 bits each are used for the alpha, red, green, and blue components.
PixelFormat64bppPARGB	Specifies that the format is 64 bits per pixel; 16 bits each are used for the alpha, red, green, and blue components. The red, green, and blue components are premultiplied according to the alpha component.

Remarks

PixelFormat48bppRGB, **PixelFormat64bppARGB**, and **PixelFormat64bppPARGB** use 16 bits per color component (channel). Windows GDI+ version 1.0 can read 16-bits-per-channel images, but such images are converted to an 8-bits-per-channel format for processing, displaying, and saving.

Requirements

Header Gdippluspixelformats.h	
--------------------------------------	--

Image Property Tag Type Constants

11/2/2020 • 2 minutes to read • [Edit Online](#)

You can store and retrieve image metadata with the help of a [PropertyItem](#) object. The **type** data member of a [PropertyItem](#) object specifies the data type of the values stored in the **value** data member of that same [PropertyItem](#) object.

The following constants, defined in Gdiplusimaging.h, can be assigned to the **type** data member of a [PropertyItem](#) object.

CONSTANT	DESCRIPTION
<code>PixelFormat4bppIndexed</code>	Specifies that the format is 4 bits per pixel, indexed.
<code>PropertyTagTypeASCII</code>	Specifies that the value data member is a null-terminated ASCII string. If you set the type data member of a PropertyItem object to <code>PropertyTagTypeASCII</code> , you should set the length data member to the length of the string including the NULL terminator. For example, the string <code>HELLO</code> would have a length of 6.
<code>PropertyTagTypeByte</code>	Specifies that the value data member is an array of bytes.
<code>PropertyTagTypeLong</code>	Specifies that the value data member is an array of unsigned long (32-bit) integers.
<code>PropertyTagTypeRational</code>	Specifies that the value data member is an array of pairs of unsigned long integers. Each pair represents a fraction; the first integer is the numerator and the second integer is the denominator.
<code>PropertyTagTypeShort</code>	Specifies that the value data member is an array of unsigned short (16-bit) integers.
<code>PropertyTagTypeSLONG</code>	Specifies that the value data member is an array of signed long (32-bit) integers.
<code>PropertyTagTypeSRational</code>	Specifies that the value data member is an array of pairs of signed long integers. Each pair represents a fraction; the first integer is the numerator and the second integer is the denominator.
<code>PropertyTagTypeUndefined</code>	Specifies that the value data member is an array of bytes that can hold values of any data type.

Requirements

Header

Gdipusimaging.h

See also

[Image Property Tag Constants](#)

[Image File Format Specifications](#)

[Reading and Writing Metadata](#)

Image Property Tag Constants

2/22/2020 • 2 minutes to read • [Edit Online](#)

Several image file formats enable you to store metadata along with an image. Metadata is information about an image, for example, title, width, camera model, and artist. Windows GDI+ provides a uniform way of storing and retrieving metadata from image files in the Tagged Image File Format (TIFF), JPEG, Portable Network Graphics (PNG), and Exchangeable Image File (EXIF) formats.

In GDI+, a piece of metadata is called a *property item*, and an individual property item is identified by a numerical constant called a *property tag*. You can store and retrieve metadata by calling the [Image::SetPropertyItem](#) and [Image::GetPropertyItem](#) methods of the [Image](#) class, and you don't have to be concerned with the details of how a particular file format stores that metadata.

The following topics list and describe the property items supported by GDI+. The descriptions are brief and general so that they apply to a variety of file formats. For a detailed description of how a property item is used in a particular file format, see the specification for that file format. For links to several file format specifications, see [Image File Format Specifications](#). For more information about metadata, see [Reading and Writing Metadata](#) and [Image Property Tag Type Constants](#).

- [Property Tags in Numerical Order](#)
- [Property Tags in Alphabetical Order](#)
- [Property Item Descriptions](#)

Property Tags in Numerical Order

2/22/2020 • 2 minutes to read • [Edit Online](#)

The following table lists the Windows GDI+ image property tags in numerical order.

PROPERTY TAG	VALUE
PropertyTagGpsVer	0x0000
PropertyTagGpsLatitudeRef	0x0001
PropertyTagGpsLatitude	0x0002
PropertyTagGpsLongitudeRef	0x0003
PropertyTagGpsLongitude	0x0004
PropertyTagGpsAltitudeRef	0x0005
PropertyTagGpsAltitude	0x0006
PropertyTagGpsGpsTime	0x0007
PropertyTagGpsGpsSatellites	0x0008
PropertyTagGpsGpsStatus	0x0009
PropertyTagGpsGpsMeasureMode	0x000A
PropertyTagGpsGpsDop	0x000B
PropertyTagGpsSpeedRef	0x000C
PropertyTagGpsSpeed	0x000D
PropertyTagGpsTrackRef	0x000E
PropertyTagGpsTrack	0x000F
PropertyTagGpsImgDirRef	0x0010
PropertyTagGpsImgDir	0x0011
PropertyTagGpsMapDatum	0x0012
PropertyTagGpsDestLatRef	0x0013
PropertyTagGpsDestLat	0x0014

PROPERTY TAG	VALUE
PropertyTagGpsDestLongRef	0x0015
PropertyTagGpsDestLong	0x0016
PropertyTagGpsDestBearRef	0x0017
PropertyTagGpsDestBear	0x0018
PropertyTagGpsDestDistRef	0x0019
PropertyTagGpsDestDist	0x001A
PropertyTagNewSubfileType	0x00FE
PropertyTagSubfileType	0x00FF
PropertyTagImageWidth	0x0100
PropertyTagImageHeight	0x0101
PropertyTagBitsPerSample	0x0102
PropertyTagCompression	0x0103
PropertyTagPhotometricInterp	0x0106
PropertyTagThreshHolding	0x0107
PropertyTagCellWidth	0x0108
PropertyTagCellHeight	0x0109
PropertyTagFillOrder	0x010A
PropertyTagDocumentName	0x010D
PropertyTagImageDescription	0x010E
PropertyTagEquipMake	0x010F
PropertyTagEquipModel	0x0110
PropertyTagStripOffsets	0x0111
PropertyTagOrientation	0x0112
PropertyTagSamplesPerPixel	0x0115
PropertyTagRowsPerStrip	0x0116

PROPERTY TAG	VALUE
PropertyTagStripBytesCount	0x0117
PropertyTagMinSampleValue	0x0118
PropertyTagMaxSampleValue	0x0119
PropertyTagXResolution	0x011A
PropertyTagYResolution	0x011B
PropertyTagPlanarConfig	0x011C
PropertyTagPageName	0x011D
PropertyTagXPosition	0x011E
PropertyTagYPosition	0x011F
PropertyTagFreeOffset	0x0120
PropertyTagFreeByteCounts	0x0121
PropertyTagGrayResponseUnit	0x0122
PropertyTagGrayResponseCurve	0x0123
PropertyTagT4Option	0x0124
PropertyTagT6Option	0x0125
PropertyTagResolutionUnit	0x0128
PropertyTagPageNumber	0x0129
PropertyTagTransferFunction	0x012D
PropertyTagSoftwareUsed	0x0131
PropertyTagDateTime	0x0132
PropertyTagArtist	0x013B
PropertyTagHostComputer	0x013C
PropertyTagPredictor	0x013D
PropertyTagWhitePoint	0x013E
PropertyTagPrimaryChromaticities	0x013F

PROPERTY TAG	VALUE
PropertyTagColorMap	0x0140
PropertyTagHalftoneHints	0x0141
PropertyTagTileWidth	0x0142
PropertyTagTileLength	0x0143
PropertyTagTileOffset	0x0144
PropertyTagTileByteCounts	0x0145
PropertyTagInkSet	0x014C
PropertyTagInkNames	0x014D
PropertyTagNumberOflinks	0x014E
PropertyTagDotRange	0x0150
PropertyTagTargetPrinter	0x0151
PropertyTagExtraSamples	0x0152
PropertyTagSampleFormat	0x0153
PropertyTagSMinSampleValue	0x0154
PropertyTagSMaxSampleValue	0x0155
PropertyTagTransferRange	0x0156
PropertyTagJPEGProc	0x0200
PropertyTagJPEGInterFormat	0x0201
PropertyTagJPEGInterLength	0x0202
PropertyTagJPEGRestartInterval	0x0203
PropertyTagJPEGLosslessPredictors	0x0205
PropertyTagJPEGPointTransforms	0x0206
PropertyTagJPEGQTables	0x0207
PropertyTagJPEGDCTables	0x0208
PropertyTagJPEGACTables	0x0209

PROPERTY TAG	VALUE
PropertyTagYCbCrCoefficients	0x0211
PropertyTagYCbCrSubsampling	0x0212
PropertyTagYCbCrPositioning	0x0213
PropertyTagREFBlackWhite	0x0214
PropertyTagGamma	0x0301
PropertyTagICCPProfileDescriptor	0x0302
PropertyTagSRGBRenderingIntent	0x0303
PropertyTagImageTitle	0x0320
PropertyTagResolutionXUnit	0x5001
PropertyTagResolutionYUnit	0x5002
PropertyTagResolutionXLengthUnit	0x5003
PropertyTagResolutionYLengthUnit	0x5004
PropertyTagPrintFlags	0x5005
PropertyTagPrintFlagsVersion	0x5006
PropertyTagPrintFlagsCrop	0x5007
PropertyTagPrintFlagsBleedWidth	0x5008
PropertyTagPrintFlagsBleedWidthScale	0x5009
PropertyTagHalftoneLPI	0x500A
PropertyTagHalftoneLPIUnit	0x500B
PropertyTagHalftoneDegree	0x500C
PropertyTagHalftoneShape	0x500D
PropertyTagHalftoneMisc	0x500E
PropertyTagHalftoneScreen	0x500F
PropertyTagJPEGQuality	0x5010
PropertyTagGridSize	0x5011

PROPERTY TAG	VALUE
PropertyTagThumbnailFormat	0x5012
PropertyTagThumbnailWidth	0x5013
PropertyTagThumbnailHeight	0x5014
PropertyTagThumbnailColorDepth	0x5015
PropertyTagThumbnailPlanes	0x5016
PropertyTagThumbnailRawBytes	0x5017
PropertyTagThumbnailSize	0x5018
PropertyTagThumbnailCompressedSize	0x5019
PropertyTagColorTransferFunction	0x501A
PropertyTagThumbnailData	0x501B
PropertyTagThumbnailImageWidth	0x5020
PropertyTagThumbnailImageHeight	0x5021
PropertyTagThumbnailBitsPerSample	0x5022
PropertyTagThumbnailCompression	0x5023
PropertyTagThumbnailPhotometricInterp	0x5024
PropertyTagThumbnailImageDescription	0x5025
PropertyTagThumbnailEquipMake	0x5026
PropertyTagThumbnailEquipModel	0x5027
PropertyTagThumbnailStripOffsets	0x5028
PropertyTagThumbnailOrientation	0x5029
PropertyTagThumbnailSamplesPerPixel	0x502A
PropertyTagThumbnailRowsPerStrip	0x502B
PropertyTagThumbnailStripBytesCount	0x502C
PropertyTagThumbnailResolutionX	0x502D
PropertyTagThumbnailResolutionY	0x502E

PROPERTY TAG	VALUE
PropertyTagThumbnailPlanarConfig	0x502F
PropertyTagThumbnailResolutionUnit	0x5030
PropertyTagThumbnailTransferFunction	0x5031
PropertyTagThumbnailSoftwareUsed	0x5032
PropertyTagThumbnailDateTime	0x5033
PropertyTagThumbnailArtist	0x5034
PropertyTagThumbnailWhitePoint	0x5035
PropertyTagThumbnailPrimaryChromaticities	0x5036
PropertyTagThumbnailYCbCrCoefficients	0x5037
PropertyTagThumbnailYCbCrSubsampling	0x5038
PropertyTagThumbnailYCbCrPositioning	0x5039
PropertyTagThumbnailRefBlackWhite	0x503A
PropertyTagThumbnailCopyRight	0x503B
PropertyTagLuminanceTable	0x5090
PropertyTagChrominanceTable	0x5091
PropertyTagFrameDelay	0x5100
PropertyTagLoopCount	0x5101
PropertyTagGlobalPalette	0x5102
PropertyTagIndexBackground	0x5103
PropertyTagIndexTransparent	0x5104
PropertyTagPixelUnit	0x5110
PropertyTagPixelPerUnitX	0x5111
PropertyTagPixelPerUnitY	0x5112
PropertyTagPaletteHistogram	0x5113
PropertyTagCopyright	0x8298

PROPERTY TAG	VALUE
PropertyTagExifExposureTime	0x829A
PropertyTagExifFNumber	0x829D
PropertyTagExifIFD	0x8769
PropertyTagICCPProfile	0x8773
PropertyTagExifExposureProg	0x8822
PropertyTagExifSpectralSense	0x8824
PropertyTagGpsIFD	0x8825
PropertyTagExifISOSpeed	0x8827
PropertyTagExifOECF	0x8828
PropertyTagExifVer	0x9000
PropertyTagExifDTOrig	0x9003
PropertyTagExifDTDigitized	0x9004
PropertyTagExifCompConfig	0x9101
PropertyTagExifCompBPP	0x9102
PropertyTagExifShutterSpeed	0x9201
PropertyTagExifAperture	0x9202
PropertyTagExifBrightness	0x9203
PropertyTagExifExposureBias	0x9204
PropertyTagExifMaxAperture	0x9205
PropertyTagExifSubjectDist	0x9206
PropertyTagExifMeteringMode	0x9207
PropertyTagExifLightSource	0x9208
PropertyTagExifFlash	0x9209
PropertyTagExifFocalLength	0x920A
PropertyTagExifMakerNote	0x927C

PROPERTY TAG	VALUE
PropertyTagExifUserComment	0x9286
PropertyTagExifDTSubsec	0x9290
PropertyTagExifDТОrigSS	0x9291
PropertyTagExifDTDigSS	0x9292
PropertyTagExifFPXVer	0xA000
PropertyTagExifColorSpace	0xA001
PropertyTagExifPixXDim	0xA002
PropertyTagExifPixYDim	0xA003
PropertyTagExifRelatedWav	0xA004
PropertyTagExifInterop	0xA005
PropertyTagExifFlashEnergy	0xA20B
PropertyTagExifSpatialFR	0xA20C
PropertyTagExifFocalXRes	0xA20E
PropertyTagExifFocalYRes	0xA20F
PropertyTagExifFocalResUnit	0xA210
PropertyTagExifSubjectLoc	0xA214
PropertyTagExifExposureIndex	0xA215
PropertyTagExifSensingMethod	0xA217
PropertyTagExifFileSource	0xA300
PropertyTagExifSceneType	0xA301
PropertyTagExifCfaPattern	0xA302

Related topics

[Image Property Tag Constants](#)

[Image Property Tag Type Constants](#)

[Property Item Descriptions](#)

Property Tags in Alphabetical Order

Reading and Writing Metadata

Property Tags in Alphabetical Order

2/22/2020 • 2 minutes to read • [Edit Online](#)

The following table lists the Windows GDI+ image property tags in alphabetical order.

PROPERTY TAG	VALUE
PropertyTagArtist	0x013B
PropertyTagBitsPerSample	0x0102
PropertyTagCellHeight	0x0109
PropertyTagCellWidth	0x0108
PropertyTagChrominanceTable	0x5091
PropertyTagColorMap	0x0140
PropertyTagColorTransferFunction	0x501A
PropertyTagCompression	0x0103
PropertyTagCopyright	0x8298
PropertyTagDateTime	0x0132
PropertyTagDocumentName	0x010D
PropertyTagDotRange	0x0150
PropertyTagEquipMake	0x010F
PropertyTagEquipModel	0x0110
PropertyTagExifAperture	0x9202
PropertyTagExifBrightness	0x9203
PropertyTagExifCfaPattern	0xA302
PropertyTagExifColorSpace	0xA001
PropertyTagExifCompBPP	0x9102
PropertyTagExifCompConfig	0x9101
PropertyTagExifDTDigitized	0x9004

PROPERTY TAG	VALUE
PropertyTagExifDTDigSS	0x9292
PropertyTagExifDТОrig	0x9003
PropertyTagExifDТОrigSS	0x9291
PropertyTagExifDTSubsec	0x9290
PropertyTagExifExposureBias	0x9204
PropertyTagExifExposureIndex	0xA215
PropertyTagExifExposureProg	0x8822
PropertyTagExifExposureTime	0x829A
PropertyTagExifFileSource	0xA300
PropertyTagExifFlash	0x9209
PropertyTagExifFlashEnergy	0xA20B
PropertyTagExifFNumber	0x829D
PropertyTagExifFocalLength	0x920A
PropertyTagExifFocalResUnit	0xA210
PropertyTagExifFocalXRes	0xA20E
PropertyTagExifFocalYRes	0xA20F
PropertyTagExifFPXVer	0xA000
PropertyTagExifIFD	0x8769
PropertyTagExifInterop	0xA005
PropertyTagExifISOSpeed	0x8827
PropertyTagExifLightSource	0x9208
PropertyTagExifMakerNote	0x927C
PropertyTagExifMaxAperture	0x9205
PropertyTagExifMeteringMode	0x9207
PropertyTagExifOECF	0x8828

PROPERTY TAG	VALUE
PropertyTagExifPixXDim	0xA002
PropertyTagExifPixYDim	0xA003
PropertyTagExifRelatedWav	0xA004
PropertyTagExifSceneType	0xA301
PropertyTagExifSensingMethod	0xA217
PropertyTagExifShutterSpeed	0x9201
PropertyTagExifSpatialFR	0xA20C
PropertyTagExifSpectralSense	0x8824
PropertyTagExifSubjectDist	0x9206
PropertyTagExifSubjectLoc	0xA214
PropertyTagExifUserComment	0x9286
PropertyTagExifVer	0x9000
PropertyTagExtraSamples	0x0152
PropertyTagFillOrder	0x010A
PropertyTagFrameDelay	0x5100
PropertyTagFreeByteCounts	0x0121
PropertyTagFreeOffset	0x0120
PropertyTagGamma	0x0301
PropertyTagGlobalPalette	0x5102
PropertyTagGpsAltitude	0x0006
PropertyTagGpsAltitudeRef	0x0005
PropertyTagGpsDestBear	0x0018
PropertyTagGpsDestBearRef	0x0017
PropertyTagGpsDestDist	0x001A
PropertyTagGpsDestDistRef	0x0019

PROPERTY TAG	VALUE
PropertyTagGpsDestLat	0x0014
PropertyTagGpsDestLatRef	0x0013
PropertyTagGpsDestLong	0x0016
PropertyTagGpsDestLongRef	0x0015
PropertyTagGpsGpsDop	0x000B
PropertyTagGpsGpsMeasureMode	0x000A
PropertyTagGpsGpsSatellites	0x0008
PropertyTagGpsGpsStatus	0x0009
PropertyTagGpsGpsTime	0x0007
PropertyTagGpsIFD	0x8825
PropertyTagGpsImgDir	0x0011
PropertyTagGpsImgDirRef	0x0010
PropertyTagGpsLatitude	0x0002
PropertyTagGpsLatitudeRef	0x0001
PropertyTagGpsLongitude	0x0004
PropertyTagGpsLongitudeRef	0x0003
PropertyTagGpsMapDatum	0x0012
PropertyTagGpsSpeed	0x000D
PropertyTagGpsSpeedRef	0x000C
PropertyTagGpsTrack	0x000F
PropertyTagGpsTrackRef	0x000E
PropertyTagGpsVer	0x0000
PropertyTagGrayResponseCurve	0x0123
PropertyTagGrayResponseUnit	0x0122
PropertyTagGridSize	0x5011

PROPERTY TAG	VALUE
PropertyTagHalftoneDegree	0x500C
PropertyTagHalftoneHints	0x0141
PropertyTagHalftoneLPI	0x500A
PropertyTagHalftoneLPIUnit	0x500B
PropertyTagHalftoneMisc	0x500E
PropertyTagHalftoneScreen	0x500F
PropertyTagHalftoneShape	0x500D
PropertyTagHostComputer	0x013C
PropertyTagICCPProfile	0x8773
PropertyTagICCPProfileDescriptor	0x0302
PropertyTagImageDescription	0x010E
PropertyTagImageHeight	0x0101
PropertyTagImageTitle	0x0320
PropertyTagImageWidth	0x0100
PropertyTagIndexBackground	0x5103
PropertyTagIndexTransparent	0x5104
PropertyTagInkNames	0x014D
PropertyTagInkSet	0x014C
PropertyTagJPEGACTables	0x0209
PropertyTagJPEGDCTables	0x0208
PropertyTagJPEGInterFormat	0x0201
PropertyTagJPEGInterLength	0x0202
PropertyTagJPEGLosslessPredictors	0x0205
PropertyTagJPEGPointTransforms	0x0206
PropertyTagJPEGProc	0x0200

PROPERTY TAG	VALUE
PropertyTagJPEGTables	0x0207
PropertyTagJPEGQuality	0x5010
PropertyTagJPEGRestartInterval	0x0203
PropertyTagLoopCount	0x5101
PropertyTagLuminanceTable	0x5090
PropertyTagMaxSampleValue	0x0119
PropertyTagMinSampleValue	0x0118
PropertyTagNewSubfileType	0x00FE
PropertyTagNumberOflinks	0x014E
PropertyTagOrientation	0x0112
PropertyTagPageName	0x011D
PropertyTagPageNumber	0x0129
PropertyTagPaletteHistogram	0x5113
PropertyTagPhotometricInterp	0x0106
PropertyTagPixelPerUnitX	0x5111
PropertyTagPixelPerUnitY	0x5112
PropertyTagPixelUnit	0x5110
PropertyTagPlanarConfig	0x011C
PropertyTagPredictor	0x013D
PropertyTagPrimaryChromaticities	0x013F
PropertyTagPrintFlags	0x5005
PropertyTagPrintFlagsBleedWidth	0x5008
PropertyTagPrintFlagsBleedWidthScale	0x5009
PropertyTagPrintFlagsCrop	0x5007
PropertyTagPrintFlagsVersion	0x5006

PROPERTY TAG	VALUE
PropertyTagREFBlackWhite	0x0214
PropertyTagResolutionUnit	0x0128
PropertyTagResolutionXLengthUnit	0x5003
PropertyTagResolutionXUnit	0x5001
PropertyTagResolutionYLengthUnit	0x5004
PropertyTagResolutionYUnit	0x5002
PropertyTagRowsPerStrip	0x0116
PropertyTagSampleFormat	0x0153
PropertyTagSamplesPerPixel	0x0115
PropertyTagSMaxSampleValue	0x0155
PropertyTagSMinSampleValue	0x0154
PropertyTagSoftwareUsed	0x0131
PropertyTagSRGBRenderingIntent	0x0303
PropertyTagStripBytesCount	0x0117
PropertyTagStripOffsets	0x0111
PropertyTagSubfileType	0x00FF
PropertyTagT4Option	0x0124
PropertyTagT6Option	0x0125
PropertyTagTargetPrinter	0x0151
PropertyTagThreshHolding	0x0107
PropertyTagThumbnailArtist	0x5034
PropertyTagThumbnailBitsPerSample	0x5022
PropertyTagThumbnailColorDepth	0x5015
PropertyTagThumbnailCompressedSize	0x5019
PropertyTagThumbnailCompression	0x5023

PROPERTY TAG	VALUE
PropertyTagThumbnailCopyRight	0x503B
PropertyTagThumbnailData	0x501B
PropertyTagThumbnailDateTime	0x5033
PropertyTagThumbnailEquipMake	0x5026
PropertyTagThumbnailEquipModel	0x5027
PropertyTagThumbnailFormat	0x5012
PropertyTagThumbnailHeight	0x5014
PropertyTagThumbnailImageDescription	0x5025
PropertyTagThumbnailImageHeight	0x5021
PropertyTagThumbnailImageWidth	0x5020
PropertyTagThumbnailOrientation	0x5029
PropertyTagThumbnailPhotometricInterp	0x5024
PropertyTagThumbnailPlanarConfig	0x502F
PropertyTagThumbnailPlanes	0x5016
PropertyTagThumbnailPrimaryChromaticities	0x5036
PropertyTagThumbnailRawBytes	0x5017
PropertyTagThumbnailRefBlackWhite	0x503A
PropertyTagThumbnailResolutionUnit	0x5030
PropertyTagThumbnailResolutionX	0x502D
PropertyTagThumbnailResolutionY	0x502E
PropertyTagThumbnailRowsPerStrip	0x502B
PropertyTagThumbnailSamplesPerPixel	0x502A
PropertyTagThumbnailSize	0x5018
PropertyTagThumbnailSoftwareUsed	0x5032
PropertyTagThumbnailStripBytesCount	0x502C

PROPERTY TAG	VALUE
PropertyTagThumbnailStripOffsets	0x5028
PropertyTagThumbnailTransferFunction	0x5031
PropertyTagThumbnailWhitePoint	0x5035
PropertyTagThumbnailWidth	0x5013
PropertyTagThumbnailYCbCrCoefficients	0x5037
PropertyTagThumbnailYCbCrPositioning	0x5039
PropertyTagThumbnailYCbCrSubsampling	0x5038
PropertyTagTileByteCounts	0x0145
PropertyTagTileLength	0x0143
PropertyTagTileOffset	0x0144
PropertyTagTileWidth	0x0142
PropertyTagTransferFunction	0x012D
PropertyTagTransferRange	0x0156
PropertyTagWhitePoint	0x013E
PropertyTagXPosition	0x011E
PropertyTagXResolution	0x011A
PropertyTagYCbCrCoefficients	0x0211
PropertyTagYCbCrPositioning	0x0213
PropertyTagYCbCrSubsampling	0x0212
PropertyTagYPosition	0x011F
PropertyTagYResolution	0x011B

Related topics

[Image Property Tag Constants](#)

[Image Property Tag Type Constants](#)

[Property Item Descriptions](#)

Property Tags in Numerical Order

Reading and Writing Metadata

Property Item Descriptions

2/22/2020 • 33 minutes to read • [Edit Online](#)

The following list gives descriptions of the property items supported by Windows GDI+. The descriptions are brief and general so that they apply to a variety of image file formats. For a detailed description of how a property item is used by a particular file format, see the specification for that file format. For links to several file specifications and other documents that describe metadata in detail, see [Image File Format Specifications](#).

The Exchangeable Image File (EXIF) format is a Japan Electronic Industry Development Association (JEIDA) standard, revised June 1998 as version 2.1. Portions of the EXIF specification are used with permission of JEIDA.

PropertyTagGpsVer

Version of the Global Positioning Systems (GPS) IFD, given as 2.0.0.0. This tag is mandatory when the PropertyTagGpsIFD tag is present. When the version is 2.0.0.0, the tag value is 0x02000000.

Tag	0x0000
Type	PropertyTagTypeByte
Count	4

PropertyTagGpsLatitudeRef

Null-terminated character string that specifies whether the latitude is north or south. **N** specifies north latitude, and **S** specifies south latitude.

Tag	0x0001
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsLatitude

Latitude. Latitude is expressed as three rational values giving the degrees, minutes, and seconds respectively. When degrees, minutes, and seconds are expressed, the format is dd/1, mm/1, ss/1. When degrees and minutes are used and, for example, fractions of minutes are given up to two decimal places, the format is dd/1, mmmm/100, 0/1.

Tag	0x0002
Type	PropertyTagTypeRational
Count	3

PropertyTagGpsLongitudeRef

Null-terminated character string that specifies whether the longitude is east or west longitude. **E** specifies east longitude, and **W** specifies west longitude.

Tag	0x0003
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsLongitude

Longitude. Longitude is expressed as three rational values giving the degrees, minutes, and seconds respectively. When degrees, minutes and seconds are expressed, the format is ddd/1, mm/1, ss/1. When degrees and minutes are used and, for example, fractions of minutes are given up to two decimal places, the format is ddd/1, mmmm/100, 0/1.

Tag	0x0004
Type	PropertyTagTypeRational
Count	3

PropertyTagGpsAltitudeRef

Reference altitude, in meters.

Tag	0x0005
Type	PropertyTagTypeByte
Count	1

PropertyTagGpsAltitude

Altitude, in meters, based on the reference altitude specified by PropertyTagGpsAltitudeRef.

Tag	0x0006
Type	PropertyTagTypeRational
Count	1

PropertyTagGpsGpsTime

Time as Coordinated Universal Time (UTC). The value is expressed as three rational numbers that give the hour, minute, and second.

Tag	0x0007
Type	PropertyTagTypeRational
Count	3

PropertyTagGpsGpsSatellites

Null-terminated character string that specifies the GPS satellites used for measurements. This tag can be used to specify the ID number, angle of elevation, azimuth, SNR, and other information about each satellite. The format is not specified. If the GPS receiver is incapable of taking measurements, the value of the tag must be set to NULL.

Tag	0x0008
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagGpsGpsStatus

Null-terminated character string that specifies the status of the GPS receiver when the image is recorded. A means measurement is in progress, and V means the measurement is Interoperability.

Tag	0x0009
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsGpsMeasureMode

Null-terminated character string that specifies the GPS measurement mode. [2](#) specifies 2-D measurement, and [3](#) specifies 3-D measurement.

Tag	0x000A
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsGpsDop

GPS DOP (data degree of precision). An HDOP value is written during 2-D measurement, and a PDOP value is written during 3-D measurement.

Tag	0x000B
Type	PropertyTagTypeRational
Count	1

PropertyTagGpsSpeedRef

Null-terminated character string that specifies the unit used to express the GPS receiver speed of movement. [K](#), [M](#), and [N](#) represent kilometers per hour, miles per hour, and knots respectively.

Tag	0x000C
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsSpeed

Speed of the GPS receiver movement.

Tag	0x000D
Type	PropertyTagTypeRational
Count	1

PropertyTagGpsTrackRef

Null-terminated character string that specifies the reference for giving the direction of GPS receiver movement. [T](#) specifies true direction, and [M](#) specifies magnetic direction.

Tag	0x000E
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsTrack

Direction of GPS receiver movement. The range of values is from 0.00 to 359.99.

Tag	0x000F
Type	PropertyTagTypeRational
Count	1

PropertyTagGpsImgDirRef

Null-terminated character string that specifies the reference for the direction of the image when it is captured. specifies true direction, and specifies magnetic direction.

Tag	0x0010
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsImgDir

Direction of the image when it was captured. The range of values is from 0.00 to 359.99.

Tag	0x0011
Type	PropertyTagTypeRational
Count	1

PropertyTagGpsMapDatum

Null-terminated character string that specifies geodetic survey data used by the GPS receiver. If the survey data is restricted to Japan, the value of this tag is TOKYO or WGS-84.

Tag	0x0012
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagGpsDestLatRef

Null-terminated character string that specifies whether the latitude of the destination point is north or south latitude. specifies north latitude, and specifies south latitude.

Tag	0x0013
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsDestLat

Latitude of the destination point. The latitude is expressed as three rational values giving the degrees, minutes, and seconds respectively. When degrees, minutes, and seconds are expressed, the format is dd/1, mm/1, ss/1. When degrees and minutes are used and, for example, fractions of minutes are given up to two decimal places, the

format is dd/1, mmmm/100, 0/1.

Tag	0x0014
Type	PropertyTagTypeRational
Count	3

PropertyTagGpsDestLongRef

Null-terminated character string that specifies whether the longitude of the destination point is east or west longitude. **E** specifies east longitude, and **W** specifies west longitude.

Tag	0x0015
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsDestLong

Longitude of the destination point. The longitude is expressed as three rational values giving the degrees, minutes, and seconds respectively. When degrees, minutes, and seconds are expressed, the format is ddd/1, mm/1, ss/1. When degrees and minutes are used and, for example, fractions of minutes are given up to two decimal places, the format is ddd/1, mmmm/100, 0/1.

Tag	0x0016
Type	PropertyTagTypeRational
Count	3

PropertyTagGpsDestBearRef

Null-terminated character string that specifies the reference used for giving the bearing to the destination point. **T** specifies true direction, and **M** specifies magnetic direction.

Tag	0x0017
Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsDestBear

Bearing to the destination point. The range of values is from 0.00 to 359.99.

Tag	0x0018
Type	PropertyTagTypeRational
Count	1

PropertyTagGpsDestDistRef

Null-terminated character string that specifies the unit used to express the distance to the destination point. K, M, and N represent kilometers, miles, and knots respectively.

Tag	0x0019
-----	--------

Type	PropertyTagTypeASCII
Count	2 (one character plus the NULL terminator)

PropertyTagGpsDestDist

Distance to the destination point.

Tag	0x001A
Type	PropertyTagTypeRational
Count	1

PropertyTagNewSubfileType

Type of data in a subfile.

Tag	0x00FE
Type	PropertyTagTypeLong
Count	1

PropertyTagSubfileType

Type of data in a subfile.

Tag	0x00FF
Type	PropertyTagTypeShort
Count	1

PropertyTagImageWidth

Number of pixels per row.

Tag	0x0100
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagImageHeight

Number of pixel rows.

Tag	0x0101
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagBitsPerSample

Number of bits per color component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0102
Type	PropertyTagTypeShort

Count	Number of samples (components) per pixel
-------	--

PropertyTagCompression

Compression scheme used for the image data.

Tag	0x0103
Type	PropertyTagTypeShort
Count	1

PropertyTagPhotometricInterp

How pixel data will be interpreted.

Tag	0x0106
Type	PropertyTagTypeShort
Count	1

PropertyTagThreshHolding

Technique used to convert from gray pixels to black and white pixels.

Tag	0x0107
Type	PropertyTagTypeShort
Count	1

PropertyTagCellWidth

Width of the dithering or halftoning matrix.

Tag	0x0108
Type	PropertyTagTypeShort
Count	1

PropertyTagCellHeight

Height of the dithering or halftoning matrix.

Tag	0x0109
Type	PropertyTagTypeShort
Count	1

PropertyTagFillOrder

Logical order of bits in a byte.

Tag	0x010A
Type	PropertyTagTypeShort
Count	1

PropertyTagDocumentName

Null-terminated character string that specifies the name of the document from which the image was scanned.

Tag	0x010D
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagImageDescription

Null-terminated character string that specifies the title of the image.

Tag	0x010E
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagEquipMake

Null-terminated character string that specifies the manufacturer of the equipment used to record the image.

Tag	0x010F
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagEquipModel

Null-terminated character string that specifies the model name or model number of the equipment used to record the image.

Tag	0x0110
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagStripOffsets

For each strip, the byte offset of that strip. See also [PropertyTagRowsPerStrip](#) and [PropertyTagStripBytesCount](#).

Tag	0x0111
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	Number of strips

PropertyTagOrientation

Image orientation viewed in terms of rows and columns.

Tag

0x0112

Type

PropertyTagTypeShort

Count

1 - The 0th row is at the top of the visual image, and the 0th column is the visual left side. 2 - The 0th row is at the visual top of the image, and the 0th column is the visual right side. 3 - The 0th row is at the visual bottom of the image, and the 0th column is the visual right side. 4 - The 0th row is at the visual bottom of the image, and the 0th column is the visual left side. 5 - The 0th row is the visual left side of the image, and the 0th column is the visual top. 6 - The 0th row is the visual right side of the image, and the 0th column is the visual top. 7 - The 0th row is the visual right side of the image, and the 0th column is the visual bottom. 8 - The 0th row is the visual left side of the image, and the 0th column is the visual bottom.

PropertyTagSamplesPerPixel

Number of color components per pixel.

Tag	0x0115
Type	PropertyTagTypeShort
Count	1

PropertyTagRowsPerStrip

Number of rows per strip. See also [PropertyTagStripBytesCount](#) and [PropertyTagStripOffsets](#).

Tag	0x0116
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagStripBytesCount

For each strip, the total number of bytes in that strip.

Tag	0x0117
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	Number of strips

PropertyTagMinSampleValue

For each color component, the minimum value assigned to that component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0118
Type	PropertyTagTypeShort
Count	Number of samples (components) per pixel

PropertyTagMaxSampleValue

For each color component, the maximum value assigned to that component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0119
Type	PropertyTagTypeShort
Count	Number of samples (components) per pixel

PropertyTagXResolution

Number of pixels per unit in the image width (x) direction. The unit is specified by [PropertyTagResolutionUnit](#).

Tag	0x011A
Type	PropertyTagTypeRational
Count	1

PropertyTagYResolution

Number of pixels per unit in the image height (y) direction. The unit is specified by [PropertyTagResolutionUnit](#).

Tag	0x011B
Type	PropertyTagTypeRational
Count	1

PropertyTagPlanarConfig

Whether pixel components are recorded in chunky or planar format.

Tag	0x011C
Type	PropertyTagTypeShort
Count	1

PropertyTagPageName

Null-terminated character string that specifies the name of the page from which the image was scanned.

Tag	0x011D
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagXPosition

Offset from the left side of the page to the left side of the image. The unit of measure is specified by [PropertyTagResolutionUnit](#).

Tag	0x011E
Type	PropertyTagTypeRational
Count	1

PropertyTagYPosition

Offset from the top of the page to the top of the image. The unit of measure is specified by [PropertyTagResolutionUnit](#).

Tag	0x011F
Type	PropertyTagTypeRational
Count	1

PropertyTagFreeOffset

For each string of contiguous unused bytes, the byte offset of that string.

Tag	0x0120
Type	PropertyTagTypeLong

PropertyTagFreeByteCounts

For each string of contiguous unused bytes, the number of bytes in that string.

Tag	0x0121
Type	PropertyTagTypeLong
Count	Number of strings of contiguous unused bytes.

PropertyTagGrayResponseUnit

Precision of the number specified by PropertyTagGrayResponseCurve. 1 specifies tenths, 2 specifies hundredths, 3 specifies thousandths, and so on.

Tag	0x0122
Type	PropertyTagTypeShort
Count	1

PropertyTagGrayResponseCurve

For each possible pixel value in a grayscale image, the optical density of that pixel value.

Tag	0x0123
Type	PropertyTagTypeShort
Count	Number of possible pixel values

PropertyTagT4Option

Set of flags that relate to T4 encoding.

Tag	0x0124
Type	PropertyTagTypeLong
Count	1

PropertyTagT6Option

Set of flags that relate to T6 encoding.

Tag	0x0125
Type	PropertyTagTypeLong
Count	1

PropertyTagResolutionUnit

Unit of measure for the horizontal resolution and the vertical resolution.

Tag

0x0128

Type

PropertyTagTypeShort

Count

1

2 - inch 3 - centimeter

PropertyTagPageNumber

Page number of the page from which the image was scanned.

Tag	0x0129
Type	PropertyTagTypeShort
Count	1

PropertyTagTransferFunction

Tables that specify transfer functions for the image.

Tag	0x012D
Type	PropertyTagTypeShort
Count	Total number of 16-bit words required for the tables

PropertyTagSoftwareUsed

Null-terminated character string that specifies the name and version of the software or firmware of the device used to generate the image.

Tag	0x0131
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagDateTime

Date and time the image was created.

Tag	0x0132
Type	PropertyTagTypeASCII
Count	20

PropertyTagArtist

Null-terminated character string that specifies the name of the person who created the image.

Tag	0x013B
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagHostComputer

Null-terminated character string that specifies the computer and/or operating system used to create the image.

Tag	0x013C
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagPredictor

Type of prediction scheme that was applied to the image data before the encoding scheme was applied.

Tag	0x013D
Type	PropertyTagTypeShort
Count	1

PropertyTagWhitePoint

Chromaticity of the white point of the image.

Tag	0x013E
Type	PropertyTagTypeRational
Count	2

PropertyTagPrimaryChromaticities

For each of the three primary colors in the image, the chromaticity of that color.

Tag	0x013F
Type	PropertyTagTypeRational
Count	6

PropertyTagColorMap

Color palette (lookup table) for a palette-indexed image.

Tag	0x0140
Type	PropertyTagTypeShort
Count	Number of 16-bit words required for the palette

PropertyTagHalftoneHints

Information used by the halftone function

Tag	0x0141
Type	PropertyTagTypeShort
Count	2

PropertyTagTileWidth

Number of pixel columns in each tile.

Tag	0x0142
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagTileLength

Number of pixel rows in each tile.

Tag	0x0143
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagTileOffset

For each tile, the byte offset of that tile.

Tag	0x0144
Type	PropertyTagTypeLong
Count	Number of tiles

PropertyTagTileByteCounts

For each tile, the number of bytes in that tile.

Tag	0x0145
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	Number of tiles

PropertyTagInkSet

Set of inks used in a separated image.

Tag	0x014C
Type	PropertyTagTypeShort
Count	1

PropertyTagInkNames

Sequence of concatenated, null-terminated, character strings that specify the names of the inks used in a separated image.

Tag	0x014D
Type	PropertyTagTypeASCII

PropertyTagNumberOfInks

Number of inks.

Tag	0x014E
Type	PropertyTagTypeShort
Count	1

PropertyTagDotRange

Color component values that correspond to a 0 percent dot and a 100 percent dot.

Tag	0x0150
Type	PropertyTagTypeByte or PropertyTagTypeShort
Count	2 or $2 \times \text{PropertyTagSamplesPerPixel}$

PropertyTagTargetPrinter

Null-terminated character string that describes the intended printing environment.

Tag	0x0151
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagExtraSamples

Number of extra color components. For example, one extra component might hold an alpha value.

Tag	0x0152
Type	PropertyTagTypeShort
Count	1

PropertyTagSampleFormat

For each color component, the numerical format (unsigned, signed, floating point) of that component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0153
Type	PropertyTagTypeShort
Count	Number of samples (components) per pixel

PropertyTagSMinSampleValue

For each color component, the minimum value of that component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0154
Type	The type that best matches the pixel component data
Count	Number of samples (components) per pixel

PropertyTagSMaxSampleValue

For each color component, the maximum value of that component.
See also [PropertyTagSamplesPerPixel](#).

Tag	0x0155
Type	The type that best matches the pixel component data
Count	Number of samples (components) per pixel

PropertyTagTransferRange

Table of values that extends the range of the transfer function.

Tag	0x0156
Type	PropertyTagTypeShort
Count	6

PropertyTagJPEGProc

JPEG compression process.

Tag	0x0200
Type	PropertyTagTypeShort
Count	1

PropertyTagJPEGInterFormat

Offset to the start of a JPEG bitstream.

Tag	0x0201
Type	PropertyTagTypeLong
Count	1

PropertyTagJPEGInterLength

Length, in bytes, of the JPEG bitstream.

Tag	0x0202
Type	PropertyTagTypeLong
Count	1

PropertyTagJPEGRestartInterval

Length of the restart interval.

Tag	0x0203
Type	PropertyTagTypeShort
Count	1

PropertyTagJPEGLosslessPredictors

For each color component, a lossless predictor-selection value for that component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0205
Type	PropertyTagTypeShort
Count	Number of samples (components) per pixel

PropertyTagJPEGPointTransforms

For each color component, a point transformation value for that component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0206
Type	PropertyTagTypeShort
Count	Number of samples (components) per pixel

PropertyTagJPEGQTables

For each color component, the offset to the quantization table for that component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0207
Type	PropertyTagTypeLong
Count	Number of samples (components) per pixel

PropertyTagJPEGDCTables

For each color component, the offset to the DC Huffman table (or lossless Huffman table) for that component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0208
Type	PropertyTagTypeLong
Count	Number of samples (components) per pixel

PropertyTagJPEGACTables

For each color component, the offset to the AC Huffman table for that component. See also [PropertyTagSamplesPerPixel](#).

Tag	0x0209
Type	PropertyTagTypeLong
Count	Number of samples (components) per pixel

PropertyTagYCbCrCoefficients

Coefficients for transformation from RGB to YCbCr image data.

Tag	0x0211
Type	PropertyTagTypeRational
Count	3

PropertyTagYCbCrSubsampling

Sampling ratio of chrominance components in relation to the luminance component.

Tag	0x0212
Type	PropertyTagTypeShort
Count	2

PropertyTagYCbCrPositioning

Position of chrominance components in relation to the luminance component.

Tag	0x0213
Type	PropertyTagTypeShort
Count	1

PropertyTagREFBlackWhite

Reference black point value and reference white point value.

Tag	0x0214
Type	PropertyTagTypeRational
Count	6

PropertyTagGamma

Gamma value attached to the image. The gamma value is stored as a rational number (pair of `long`) with a numerator of 100000. For example, a gamma value of 2.2 is stored as the pair (100000, 45455).

Tag	0x0301
Type	PropertyTagTypeRational
Count	1

PropertyTagICCPProfileDescriptor

Null-terminated character string that identifies an ICC profile.

Tag	0x0302
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagSRGBRenderingIntent

How the image should be displayed as defined by the International Color Consortium (ICC). If a GDI+ `Image` object is constructed with the `useEmbeddedColorManagement` parameter set to `TRUE`, then GDI+ renders the image according to the specified rendering intent. The intent can be set to perceptual, relative colorimetric, saturation, or absolute colorimetric.

- Perceptual intent, which is suitable for photographs, gives good adaptation to the display device gamut at the expense of colorimetric accuracy.
- Relative colorimetric intent is suitable for images (for example, logos) that require color appearance matching that is relative to the display device white point.

- Saturation intent, which is suitable for charts and graphs, preserves saturation at the expense of hue and lightness.
- Absolute colorimetric intent is suitable for proofs (previews of images destined for a different display device) that require preservation of absolute colorimetry.

Tag

0x0303

Type

BYTE

Count

1

0 - perceptual 1 - relative colorimetric 2 - saturation 3 - absolute colorimetric

PropertyTagImageTitle

Null-terminated character string that specifies the title of the image.

Tag	0x0320
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagResolutionXUnit

Units in which to display horizontal resolution.

Tag

0x5001

Type

PropertyTagTypeShort

Count

1

1 - pixels per inch 2 - pixels per centimeter

PropertyTagResolutionYUnit

Units in which to display vertical resolution.

Tag

0x5002

Type

PropertyTagTypeShort

Count

1

1 - pixels per inch 2 - pixels per centimeter

PropertyTagResolutionXLengthUnit

Units in which to display the image width.

Tag

0x5003

Type

PropertyTagTypeShort

Count

1

1 - inches 2 - centimeters 3 - points 4 - picas 5 - columns

PropertyTagResolutionYLengthUnit

Units in which to display the image height.

Tag

0x5004

Type

PropertyTagTypeShort

Count

1

1 - inches 2 - centimeters 3 - points 4 - picas 5 - columns

PropertyTagPrintFlags

Sequence of one-byte Boolean values that specify printing options.

Tag	0x5005
Type	PropertyTagTypeASCII
Count	Number of flags

PropertyTagPrintFlagsVersion

Print flags version.

Tag	0x5006
Type	PropertyTagTypeShort
Count	1

PropertyTagPrintFlagsCrop

Print flags center crop marks.

Tag	0x5007
Type	PropertyTagTypeByte
Count	1

PropertyTagPrintFlagsBleedWidth

Print flags bleed width.

Tag	0x5008
Type	PropertyTagTypeLong
Count	1

PropertyTagPrintFlagsBleedWidthScale

Print flags bleed width scale.

Tag	0x5009
Type	PropertyTagTypeShort
Count	1

PropertyTagHalftoneLPI

Ink's screen frequency, in lines per inch.

Tag	0x500A
Type	PropertyTagTypeRational
Count	1

PropertyTagHalftoneLPIUnit

Units for the screen frequency.

Tag

0x500B

Type

PropertyTagTypeShort

Count

1

1 - lines per inch 2 - lines per centimeter

PropertyTagHalftoneDegree

Angle for screen.

Tag	0x500C
Type	PropertyTagTypeRational
Count	1

PropertyTagHalftoneShape

Shape of the halftone dots.

Tag

0x500D

Type

PropertyTagTypeShort

Count

1

0 - round 1 - ellipse 2 - line 3 - square 4 - cross 6 - diamond

PropertyTagHalftoneMisc

Miscellaneous halftone information.

Tag	0x500E
Type	PropertyTagTypeLong
Count	1

PropertyTagHalftoneScreen

Boolean value that specifies whether to use the printer's default screens.

Tag

0x500F

Type

PropertyTagTypeByte

Count

1

1 - use printer's default screens 2 - other

PropertyTagJPEGQuality

Private tag used by the Adobe Photoshop format. Not for public use.

Tag	0x5010
Type	PropertyTagTypeShort
Count	Any

PropertyTagGridSize

Block of information about grids and guides.

Tag	0x5011
Type	PropertyTagTypeUndefined
Count	Any

PropertyTagThumbnailFormat

Format of the thumbnail image.

Tag	0x5012
Type	PropertyTagTypeLong
Count	1
	0 - raw RGB 1 - JPEG

PropertyTagThumbnailWidth

Width, in pixels, of the thumbnail image.

Tag	0x5013
Type	PropertyTagTypeLong
Count	1

PropertyTagThumbnailHeight

Height, in pixels, of the thumbnail image.

Tag	0x5014
Type	PropertyTagTypeLong
Count	1

PropertyTagThumbnailColorDepth

bits per pixel (BPP) for the thumbnail image.

Tag	0x5015
Type	PropertyTagTypeShort
Count	1

PropertyTagThumbnailPlanes

Number of color planes for the thumbnail image.

Tag	0x5016
Type	PropertyTagTypeShort
Count	1

PropertyTagThumbnailRawBytes

Byte offset between rows of pixel data.

Tag	0x5017
Type	PropertyTagTypeLong
Count	1

PropertyTagThumbnailSize

Total size, in bytes, of the thumbnail image.

Tag	0x5018
Type	PropertyTagTypeLong
Count	1

PropertyTagThumbnailCompressedSize

Compressed size, in bytes, of the thumbnail image.

Tag	0x5019
Type	PropertyTagTypeLong
Count	1

PropertyTagColorTransferFunction

Table of values that specify color transfer functions.

Tag	0x501A
Type	PropertyTagTypeUndefined
Count	Any

PropertyTagThumbnailData

Raw thumbnail bits in JPEG or RGB format. Depends on PropertyTagThumbnailFormat.

Tag	0x501B
Type	PropertyTagTypeByte
Count	Variable

PropertyTagThumbnailImageWidth

Number of pixels per row in the thumbnail image.

Tag	0x5020
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagThumbnailImageHeight

Number of pixel rows in the thumbnail image.

Tag	0x5021
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagThumbnailBitsPerSample

Number of bits per color component in the thumbnail image. See also [PropertyTagThumbnailSamplesPerPixel](#).

Tag	0x5022
Type	PropertyTagTypeShort
Count	Number of samples (components) per pixel in the thumbnail image

PropertyTagThumbnailCompression

Compression scheme used for thumbnail image data.

Tag	0x5023
Type	PropertyTagTypeShort
Count	1

PropertyTagThumbnailPhotometricInterp

How thumbnail pixel data will be interpreted.

Tag	0x5024
Type	PropertyTagTypeShort
Count	1

PropertyTagThumbnailImageDescription

Null-terminated character string that specifies the title of the image.

Tag	0x5025
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagThumbnailEquipMake

Null-terminated character string that specifies the manufacturer of the equipment used to record the thumbnail image.

Tag	0x5026
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagThumbnailEquipModel

Null-terminated character string that specifies the model name or model number of the equipment used to record the thumbnail image.

Tag	0x5027
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagThumbnailStripOffsets

For each strip in the thumbnail image, the byte offset of that strip. See also [PropertyTagThumbnailRowsPerStrip](#) and [PropertyTagThumbnailStripBytesCount](#).

Tag	0x5028
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	Number of strips

PropertyTagThumbnailOrientation

Thumbnail image orientation in terms of rows and columns. See also [PropertyTagOrientation](#).

Tag	0x5029
Type	PropertyTagTypeShort
Count	1

PropertyTagThumbnailSamplesPerPixel

Number of color components per pixel in the thumbnail image.

Tag	0x502A
Type	PropertyTagTypeShort
Count	1

PropertyTagThumbnailRowsPerStrip

Number of rows per strip in the thumbnail image. See also [PropertyTagThumbnailStripBytesCount](#) and [PropertyTagThumbnailStripOffsets](#).

Tag	0x502B
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagThumbnailStripBytesCount

For each thumbnail image strip, the total number of bytes in that strip.

Tag	0x502C
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	Number of strips in the thumbnail image

PropertyTagThumbnailResolutionX

Thumbnail resolution in the width direction. The resolution unit is given in [PropertyTagThumbnailResolutionUnit](#).

Tag	0x502D
-----	--------

PropertyTagThumbnailResolutionY

Thumbnail resolution in the height direction. The resolution unit is given in [PropertyTagThumbnailResolutionUnit](#).

Tag	0x502E
-----	--------

PropertyTagThumbnailPlanarConfig

Whether pixel components in the thumbnail image are recorded in chunky or planar format. See also [PropertyTagPlanarConfig](#).

Tag	0x502F
Type	PropertyTagTypeShort
Count	1

PropertyTagThumbnailResolutionUnit

Unit of measure for the horizontal resolution and the vertical resolution of the thumbnail image. See also [PropertyTagResolutionUnit](#).

Tag	0x5030
Type	PropertyTagTypeShort
Count	1

PropertyTagThumbnailTransferFunction

Tables that specify transfer functions for the thumbnail image. See also [PropertyTagTransferFunction](#).

Tag	0x5031
Type	PropertyTagTypeShort
Count	Total number of 16-bit words required for the tables

PropertyTagThumbnailSoftwareUsed

Null-terminated character string that specifies the name and version of the software or firmware of the device used to generate the thumbnail image.

Tag	0x5032
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagThumbnailDateTime

Date and time the thumbnail image was created. See also [PropertyTagDateTime](#).

Tag	0x5033
Type	PropertyTagTypeASCII
Count	20

PropertyTagThumbnailArtist

Null-terminated character string that specifies the name of the person who created the thumbnail image.

Tag	0x5034
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagThumbnailWhitePoint

Chromaticity of the white point of the thumbnail image. See also [PropertyTagWhitePoint](#).

Tag	0x5035
Type	PropertyTagTypeRational
Count	2

PropertyTagThumbnailPrimaryChromaticities

For each of the three primary colors in the thumbnail image, the chromaticity of that color. See also [PropertyTagPrimaryChromaticities](#).

Tag	0x5036
Type	PropertyTagTypeRational
Count	6

PropertyTagThumbnailYCbCrCoefficients

Coefficients for transformation from RGB to YCbCr data for the thumbnail image. See also [PropertyTagYCbCrCoefficients](#).

Tag	0x5037
Type	PropertyTagTypeRational
Count	3

PropertyTagThumbnailYCbCrSubsampling

Sampling ratio of chrominance components in relation to the

luminance component for the thumbnail image. See also [PropertyTagYCbCrSubsampling](#).

Tag	0x5038
Type	PropertyTagTypeShort
Count	2

PropertyTagThumbnailYCbCrPositioning

Position of chrominance components in relation to the luminance component for the thumbnail image. See also [PropertyTagYCbCrPositioning](#).

Tag	0x5039
Type	PropertyTagTypeShort
Count	1

PropertyTagThumbnailRefBlackWhite

Reference black point value and reference white point value for the thumbnail image. See also [PropertyTagREFBlackWhite](#).

Tag	0x503A
Type	PropertyTagTypeRational
Count	6

PropertyTagThumbnailCopyRight

Null-terminated character string that contains copyright information for the thumbnail image.

Tag	0x503B
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagLuminanceTable

Luminance table. The luminance table and the chrominance table are used to control JPEG quality. A valid luminance or chrominance table has 64 entries of type PropertyTagTypeShort. If an image has either a luminance table or a chrominance table, then it must have both tables.

Tag	0x5090
Type	PropertyTagTypeShort
Count	64

PropertyTagChrominanceTable

Chrominance table. The luminance table and the chrominance table are used to control JPEG quality. A valid luminance or chrominance table has 64 entries of type PropertyTagTypeShort. If an image has either a luminance table or a chrominance table, then it must have both tables.

Tag	0x5091
Type	PropertyTagTypeShort

Count	64
-------	----

PropertyTagFrameDelay

Time delay, in hundredths of a second, between two frames in an animated GIF image.

Tag	0x5100
Type	PropertyTagTypeLong
Count	Number of frames in the image

PropertyTagLoopCount

For an animated GIF image, the number of times to display the animation. A value of 0 specifies that the animation should be displayed infinitely.

Tag	0x5101
Type	PropertyTagTypeShort
Count	1

PropertyTagGlobalPalette

Color palette for an indexed bitmap in a GIF image.

Tag	0x5102
Type	PropertyTagTypeByte
Count	3 x number of palette entries

PropertyTagIndexBackground

Index of the background color in the palette of a GIF image.

Tag	0x5103
Type	PropertyTagTypeByte
Count	1

PropertyTagIndexTransparent

Index of the transparent color in the palette of a GIF image.

Tag	0x5104
Type	PropertyTagTypeByte
Count	1

PropertyTagPixelUnit

Unit for PropertyTagPixelPerUnitX and PropertyTagPixelPerUnitY.

Tag

0x5110

Type

PropertyTagTypeByte

Count

1

0 - unknown

PropertyTagPixelPerUnitX

Pixels per unit in the x direction.

Tag	0x5111
Type	PropertyTagTypeLong
Count	1

PropertyTagPixelPerUnitY

Pixels per unit in the y direction.

Tag	0x5112
Type	PropertyTagTypeLong
Count	1

PropertyTagPaletteHistogram

Palette histogram.

Tag	0x5113
Type	PropertyTagTypeByte
Count	Length of the histogram

PropertyTagCopyright

Null-terminated character string that contains copyright information.

Tag	0x8298
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagExifExposureTime

Exposure time, measured in seconds.

Tag	0x829A
Type	PropertyTagTypeRational
Count	1

PropertyTagExifFNumber

F number.

Tag	0x829D
Type	RATIONAL
Count	1

PropertyTagExifIFD

Private tag used by GDI+. Not for public use. GDI+ uses this tag to locate Exif-specific information.

Tag	0x8769
Type	PropertyTagTypeLong
Count	1

PropertyTagICCPProfile

ICC profile embedded in the image.

Tag	0x8773
Type	PropertyTagTypeByte
Count	Length of the profile

PropertyTagExifExposureProg

Class of the program used by the camera to set exposure when the picture is taken.

Tag	0x8822
Type	SHORT
Count	1
Default	0
	0 - not defined 1 - manual 2 - normal program 3 - aperture priority 4 - shutter priority 5 - creative program (biased toward depth of field) 6 - action program (biased toward fast shutter speed) 7 - portrait mode (for close-up photos with the background out of focus) 8 - landscape mode (for landscape photos with the background in focus) 9 to 255 - reserved

PropertyTagExifSpectralSense

Null-terminated character string that specifies the spectral sensitivity of each channel of the camera used. The string is compatible with the standard developed by the ASTM Technical Committee.

Tag	0x8824
Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagGpsIFD

Offset to a block of GPS property items. Property items whose tags have the prefix PropertyTagGps are stored in the GPS block. The GPS property items are defined in the EXIF specification. GDI+ uses this tag to locate GPS information, but GDI+ does not expose this tag for public use.

Tag	0x8825
Type	PropertyTagTypeLong
Count	1

PropertyTagExifISO Speed

ISO speed and ISO latitude of the camera or input device as specified in ISO 12232.

Tag	0x8827
Type	PropertyTagTypeShort
Count	Any

PropertyTagExifOECF

Optoelectronic conversion function (OECF) specified in ISO 14524. The OECF is the relationship between the camera optical input and the image values.

Tag	0x8828
Type	PropertyTagTypeUndefined
Count	Any

PropertyTagExifVer

Version of the EXIF standard supported. Nonexistence of this field is taken to mean nonconformance to the standard. Conformance to the standard is indicated by recording 0210 as a 4-byte ASCII string. Because the type is PropertyTagTypeUndefined, there is no NULL terminator.

Tag	0x9000
Type	PropertyTagTypeUndefined
Count	4
Default	0210

PropertyTagExifDT Orig

Date and time when the original image data was generated. For a DSC, the date and time when the picture was taken. The format is YYYY:MM:DD HH:MM:SS with time shown in 24-hour format and the date and time separated by one blank character (0x2000). The character string length is 20 bytes including the NULL terminator. When the field is empty, it is treated as unknown.

Tag	0x9003
Type	PropertyTagTypeASCII
Count	20

PropertyTagExifDT Digitized

Date and time when the image was stored as digital data. If, for example, an image was captured by DSC and at the same time the file was recorded, then DateTimeOriginal and DateTimeDigitized will have the same contents.

The format is YYYY:MM:DD HH:MM:SS with time shown in 24-hour format and the date and time separated by one blank character (0x2000). The character string length is 20 bytes including the NULL terminator. When the field is empty, it is treated as unknown.

Tag	0x9004
Type	PropertyTagTypeASCII
Count	20

PropertyTagExifCompConfig

Information specific to compressed data. The channels of each component are arranged in order from the first component to the fourth. For uncompressed data, the data arrangement is given in the PropertyTagPhotometricInterp tag.

However, because PropertyTagPhotometricInterp can only express the order of Y, Cb, and Cr, this tag is provided for cases when compressed data uses components other than Y, Cb, and Cr and to support other sequences.

Tag

0x9101

Type

PropertyTagTypeUndefined

Count

4

Default

4 5 6 0 (if RGB uncompressed) 1 2 3 0 (other cases)

0 - does not exist 1 - Y 2 - Cb 3 - Cr 4 - R 5 - G 6 - B Other - reserved

PropertyTagExifCompBPP

Information specific to compressed data. The compression mode used for a compressed image is indicated in unit BPP.

Tag	0x9102
Type	PropertyTagTypeRational
Count	1

PropertyTagExifShutterSpeed

Shutter speed. The unit is the Additive System of Photographic Exposure (APEX) value.

Tag	0x9201
Type	PropertyTagTypeSRational
Count	1

PropertyTagExifAperture

Lens aperture. The unit is the APEX value.

Tag	0x9202
Type	PropertyTagTypeRational
Count	1

PropertyTagExifBrightness

Brightness value. The unit is the APEX value. Ordinarily it is given in the range of -99.99 to 99.99.

Tag	0x9203
Type	PropertyTagTypeSRational
Count	1

PropertyTagExifExposureBias

Exposure bias. The unit is the APEX value. Ordinarily it is given in the range -99.99 to 99.99.

Tag	0x9204
Type	PropertyTagTypeRational
Count	1

PropertyTagExifMaxAperture

Smallest F number of the lens. The unit is the APEX value. Ordinarily it is given in the range of 00.00 to 99.99, but it is not limited to this range.

Tag	0x9205
Type	PropertyTagTypeRational
Count	1

PropertyTagExifSubjectDist

Distance to the subject, measured in meters.

Tag	0x9206
Type	PropertyTagTypeRational
Count	1

PropertyTagExifMeteringMode

Metering mode.

Tag	0x9207
Type	PropertyTagTypeShort
Count	1
Default	0
0	0 - unknown 1 - Average 2 - CenterWeightedAverage 3 - Spot 4 - MultiSpot 5 - Pattern 6 - Partial 7 to 254 - reserved 255 - other

PropertyTagExifLightSource

Type of light source.

Tag	0x9208
Type	PropertyTagTypeShort
Count	1
Default	0
0	0 - unknown 1 - Daylight 2 - Flourescent 3 - Tungsten 17 - Standard Light A 18 - Standard Light B 19 - Standard Light C 20 - D55 21 - D65 22 - D75 23 to 254 - reserved 255 - other

PropertyTagExifFlash

Flash status. This tag is recorded when an image is taken using a strobe light (flash). Bit 0 indicates the flash firing status, and bits 1 and 2 indicate the flash return status.

Tag

0x9209

Type

PropertyTagTypeShort

Count

1

Values for bit 0 that indicate whether the flash fired: 0b - flash did not fire 1b - flash fired

Values for bits 1 and 2 that indicate the status of returned light: 00b - no strobe return detection function 01b - reserved 10b - strobe return light not detected 11b - strobe return light detected

Resulting flash tag values: 0x0000 - flash did not fire 0x0001 - flash fired 0x0005 - strobe return light not detected

PropertyTagExifFocalLength

Actual focal length, in millimeters, of the lens. Conversion is not made to the focal length of a 35 millimeter film camera.

Tag	0x920A
Type	PropertyTagTypeRational
Count	1

PropertyTagExifMakerNote

Note tag. A tag used by manufacturers of EXIF writers to record information. The contents are up to the manufacturer.

Tag	0x927C
Type	PropertyTagTypeUndefined
Count	Any

PropertyTagExifUserComment

Comment tag. A tag used by EXIF users to write keywords or comments about the image besides those in PropertyTagImageDescription and without the character-code limitations of the PropertyTagImageDescription tag.

Tag	0x9286
Type	PropertyTagTypeUndefined
Count	Any

The character code used in the PropertyTagExifUserComment tag is identified based on an ID code in a fixed 8-byte area at the start of the tag data area. The unused portion of the area is padded with null characters (0). ID codes are assigned by means of registration. Because the type is not ASCII, it is not necessary to use a NULL terminator.

PropertyTagExifDTSubsec

Null-terminated character string that specifies a fraction of a second for the PropertyTagDateTime tag.

Tag	0x9290

Type	PropertyTagTypeASCII
Count	Length of the string including the NULL terminator

PropertyTagExifDTOrigSS

Null-terminated character string that specifies a fraction of a second for the PropertyTagExifDTOrig tag.

Tag	0x9291
Type	PropertyTagTypeASCII
N	Length of the string including the NULL terminator

PropertyTagExifDTDigSS

Null-terminated character string that specifies a fraction of a second for the PropertyTagExifDTDigitized tag.

Tag	0x9292
Type	ASCII
N	Length of the string including the NULL terminator

PropertyTagExifFPXVer

FlashPix format version supported by an FPXR file. If the FPXR function supports FlashPix format version 1.0, this is indicated similarly to PropertyTagExiVer by recording 0100 as a 4-byte ASCII string. Because the type is PropertyTagTypeUndefined, there is no NULL terminator.

Tag	0xA000
Type	PropertyTagTypeUndefined
Count	4
Default	0100
0100 - FlashPix format version 1.0 Other - reserved	

PropertyTagExifColorSpace

Color space specifier. Normally sRGB (=1) is used to define the color space based on the PC monitor conditions and environment. If a color space other than sRGB is used, Uncalibrated (=0xFFFF) is set. Image data recorded as Uncalibrated can be treated as sRGB when it is converted to FlashPix.

Tag	0xA001
Type	PropertyTagTypeShort
Count	1
0x1 - sRGB 0xFFFF - uncalibrated Other - reserved	

PropertyTagExifPixXDim

Information specific to compressed data. When a compressed file is recorded, the valid width of the meaningful image must be recorded in this tag, whether or not there is padding data or a restart marker. This tag should not exist in an uncompressed file.

Tag	0xA002
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagExifPixYDim

Information specific to compressed data. When a compressed file is recorded, the valid height of the meaningful image must be recorded in this tag whether or not there is padding data or a restart marker. This tag should not exist in an uncompressed file. Because data padding is unnecessary in the vertical direction, the number of lines recorded in this valid image height tag will be the same as that recorded in the SOF.

Tag	0xA003
Type	PropertyTagTypeShort or PropertyTagTypeLong
Count	1

PropertyTagExifRelatedWav

The name of an audio file related to the image data. The only relational information recorded is the EXIF audio file name and extension (an ASCII string that consists of 8 characters plus a period (.), plus 3 characters). The path is not recorded. When you use this tag, audio files must be recorded in conformance with the EXIF audio format. Writers can also store audio data within APP2 as FlashPix extension stream data.

Tag	0xA004
Type	PropertyTagTypeASCII
Count	13

PropertyTagExifInterop

Offset to a block of property items that contain interoperability information.

Tag	0xA005
Type	PropertyTagTypeLong
Count	1

PropertyTagExifFlashEnergy

Strobe energy, in Beam Candle Power Seconds (BCPS), at the time the image was captured.

Tag	0xA20B
Type	PropertyTagTypeRational
Count	1

PropertyTagExifSpatialFR

Camera or input device spatial frequency table and SFR values in the

image width, image height, and diagonal direction, as specified in ISO 12233.

Tag	0xA20C
Type	PropertyTagTypeUndefined
Count	Any

PropertyTagExifFocalXRes

Number of pixels in the image width (x) direction per unit on the camera focal plane. The unit is specified in PropertyTagExifFocalResUnit.

Tag	0xA20E
Type	PropertyTagTypeRational
Count	1

PropertyTagExifFocalYRes

Number of pixels in the image height (y) direction per unit on the camera focal plane. The unit is specified in PropertyTagExifFocalResUnit.

Tag	0xA20F
Type	PropertyTagTypeRational
Count	1

PropertyTagExifFocalResUnit

Unit of measure for PropertyTagExifFocalXRes and PropertyTagExifFocalYRes.

Tag
0xA210
Type
PropertyTagTypeShort
Count
1
2 - inch 3 - centimeter

PropertyTagExifSubjectLoc

Location of the main subject in the scene. The value of this tag represents the pixel at the center of the main subject relative to the left edge. The first value indicates the column number, and the second value indicates the row number.

Tag	0xA214
Type	PropertyTagTypeShort
Count	2

PropertyTagExifExposureIndex

Exposure index selected on the camera or input device at the time the image was captured.

Tag	0xA215
-----	--------

Type	PropertyTagTypeRational
Count	1

PropertyTagExifSensingMethod

Image sensor type on the camera or input device.

Tag

0xA217

Type

PropertyTagTypeShort

Count

1

1 - not defined
2 - one-chip color area sensor
3 - two-chip color area sensor
4 - three-chip color area sensor
5 - color sequential area sensor
7 - trilinear sensor
8 - color sequential linear sensor
Other - reserved

PropertyTagExifFileSource

The image source. If a DSC recorded the image, the value of this tag is 3.

Tag	0xA300
Type	PropertyTagTypeUndefined
Count	1

PropertyTagExifSceneType

The type of scene. If a DSC recorded the image, the value of this tag must be set to 1, indicating that the image was directly photographed.

Tag	0xA301
Type	PropertyTagTypeUndefined
Count	1

PropertyTagExifCfaPattern

The color filter array (CFA) geometric pattern of the image sensor when a one-chip color area sensor is used. It does not apply to all sensing methods.

Tag	0xA302
Type	PropertyTagTypeUndefined
Count	Any

Related topics

[Image File Format Specifications](#)

[Image Property Tag Constants](#)

[Image Property Tag Type Constants](#)

[Property Tags in Alphabetical Order](#)

[Property Tags in Numerical Order](#)

[Reading and Writing Metadata](#)

Image File Format Specifications

11/2/2020 • 2 minutes to read • [Edit Online](#)

The following table lists several documents that give detailed descriptions of how metadata is stored in image files.

DOCUMENT	LOCATION
Tagged Image File Format (TIFF) Specification	https://www.loc.gov/preservation/digital/formats/fdd/fdd00022.shtml
Exif Specification	https://www.loc.gov/preservation/digital/formats/fdd/fdd000146.shtml
Portable Network Graphics (PNG) Specification	https://www.w3.org/TR/REC-png.html
ISO/IEC 10918-1 (JPEG)	https://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18902
ICC.1:1998-09	https://www.color.org

Related topics

[Image Property Tag Constants](#)

[Image Property Tag Type Constants](#)

[Property Item Descriptions](#)

[Property Tags in Alphabetical Order](#)

[Property Tags in Numerical Order](#)

[Reading and Writing Metadata](#)

GDI+ enumerations

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ defines the following enumerations:

- [BrushType](#)
- [ColorAdjustType](#)
- [ColorChannelFlags](#)
- [ColorMatrixFlags](#)
- [CombineMode](#)
- [CompositingMode](#)
- [CompositingQuality](#)
- [CoordinateSpace](#)
- [CurveAdjustments](#)
- [CurveChannel](#)
- [DashCap](#)
- [DashStyle](#)
- [DitherType](#)
- [DriverStringOptions](#)
- [EmfPlusRecordType](#)
- [EmfToWmfBitsFlags](#)
- [EmfType](#)
- [EncoderParameterValueType](#)
- [EncoderValue](#)
- [FillMode](#)
- [FlushIntention](#)
- [FontStyle](#)
- [HatchStyle](#)
- [HistogramFormat](#)
- [HotkeyPrefix](#)
- [ImageCodecFlags](#)
- [ImageFlags](#)
- [ImageLockMode](#)
- [ImageType](#)
- [InterpolationMode](#)
- [ItemDataPosition](#)
- [LinearGradientMode](#)
- [LineCap](#)
- [LineJoin](#)
- [MatrixOrder](#)
- [MetafileFrameUnit](#)
- [MetafileType](#)
- [ObjectType](#)
- [PaletteFlags](#)

- [PaletteType](#)
- [PathPointType](#)
- [PenAlignment](#)
- [PenType](#)
- [PixelOffsetMode](#)
- [RotateFlipType](#)
- [SmoothingMode](#)
- [Status](#)
- [StringAlignment](#)
- [StringDigitSubstitute](#)
- [StringFormatFlags](#)
- [StringTrimming](#)
- [TextRenderingHint](#)
- [Unit](#)
- [WarpMode](#)
- [WrapMode](#)

GDI+ structures

2/22/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ defines the following structures:

- [BlurParams](#)
- [BrightnessContrastParams](#)
- [ColorBalanceParams](#)
- [ColorCurveParams](#)
- [ColorLUTParams](#)
- [ColorMap](#)
- [ColorMatrix](#)
- [ColorPalette](#)
- [ENHMETAHEADER3](#)
- [GdiplusAbort](#)
- [GdiplusStartupInput](#)
- [GdiplusStartupOutput](#)
- [HueSaturationLightnessParams](#)
- [LevelsParams](#)
- [PWMFRect16](#)
- [RedEyeCorrectionParams](#)
- [SharpenParams](#)
- [TintParams](#)
- [WmfPlaceableFileHeader](#)

GDI+ Flat API

11/2/2020 • 3 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly.

As an alternative to the C++ wrappers, the Microsoft .NET Framework provides a set of managed-code wrapper classes for GDI+. The managed-code wrappers for GDI+ belong to the following namespaces.

- [System.Drawing](#)
- [System.Drawing.Drawing2D](#)
- [System.Drawing.Imaging](#)
- [System.Drawing.Text](#)

Both sets of wrappers (C++ and managed code) use an object-oriented approach, so there are some differences between the way parameters are passed to the wrapper methods and the way parameters are passed to functions in the flat API. For example, one of the C++ wrappers is the [Matrix](#) class. Each **Matrix** object has a field, **nativeMatrix**, that points to an internal variable of type **GpMatrix**. When you pass parameters to a method of a **Matrix** object, that method passes those parameters (or a set of related parameters) along to one of the functions in the GDI+ flat API. But that method also passes the **nativeMatrix** field (as an input parameter) to the flat API function. The following code shows how the [Matrix::Shear](#) method calls the **GdipShearMatrix(GpMatrix *matrix, REAL shearX, REAL shearY, GpMatrixOrder order)** function.

```
Status Shear(
    IN REAL shearX,
    IN REAL shearY,
    IN MatrixOrder order = MatrixOrderPrepend)
{
    ...
    GdipShearMatrix(nativeMatrix, shearX, shearY, order);
    ...
}
```

The [Matrix](#) constructors pass the address of a **GpMatrix** pointer variable (as an output parameter) to the **GdipCreateMatrix(GpMatrix **matrix)** function. **GdipCreateMatrix** creates and initializes an internal **GpMatrix** variable and then assigns the address of the **GpMatrix** to the pointer variable. Then the constructor copies the value of the pointer to the **nativeMatrix** field.

```
Matrix()
{
    GpMatrix *matrix = NULL;
    lastResult = DllExports::GdipCreateMatrix(&matrix);
    SetNativeMatrix(matrix);
}

VOID SetNativeMatrix(GpMatrix *nativeMatrix)
{
    this->nativeMatrix = nativeMatrix;
}
```

Clone methods in the wrapper classes receive no parameters but often pass two parameters to the underlying function in the GDI+ flat API. For example, the [Matrix::Clone](#) method passes **nativeMatrix** (as an input parameter) and the address of a **GpMatrix** pointer variable (as an output parameter) to the **GdipCloneMatrix** function. The following code shows how the **Matrix::Clone** method calls the **GdipCloneMatrix(GpMatrix *matrix, GpMatrix **cloneMatrix)** function.

```
Matrix *Clone() const
{
    GpMatrix *cloneMatrix = NULL;
    ...
    GdipCloneMatrix(nativeMatrix, &cloneMatrix);
    ...
    return new Matrix(cloneMatrix);
}
```

The functions in the flat API return a value of type **GpStatus**. The **GpStatus** enumeration is identical to the [Status](#) enumeration used by the wrapper methods. In **GdiplusGpStubs.h**, **GpStatus** is defined as follows:

```
typedef Status GpStatus;
```

Most of the methods in the wrapper classes return a status value that indicates whether the method succeeded. However, some of the wrapper methods return state values. When you call a wrapper method that returns a state value, the wrapper method passes the appropriate parameters to the underlying function in the GDI+ flat API. For example, the **Matrix** class has an [Matrix::IsInvertible](#) method that passes the **nativeMatrix** field and the address of a **BOOL** variable (as an output parameter) to the **GdipIsMatrixInvertible** function. The following code shows how the **Matrix::IsInvertible** method calls the **GdipIsMatrixInvertible(GDIPCONST GpMatrix *matrix, BOOL *result)** function.

```
BOOL IsInvertible() const
{
    BOOL result = FALSE;
    ...
    GdipIsMatrixInvertible(nativeMatrix, &result);
    return result;
}
```

Another one of the wrappers is the **Color** class. A **Color** object has a single field of type **ARGB**, which is defined as a **DWORD**. When you pass a **Color** object to one of the wrapper methods, that method passes the **ARGB** field along to the underlying function in the GDI+ flat API. The following code shows how the [Pen::SetColor](#) method calls the **GdipSetPenColor(GpPen *pen, ARGB argb)** function. The [Color::GetValue](#) method returns the value of the **ARGB** field.

```
Status SetColor(IN const Color& color)
{
    ...
    GdipSetPenColor(nativePen, color.GetValue());
}
```

The following topics show the relationship between the GDI+ flat API and the C++ wrapper methods.

- [AdjustableArrowCap Functions](#)
- [Bitmap Functions](#)
- [Brush Functions](#)
- [CachedBitmap Functions](#)
- [CustomLineCap Functions](#)

- [Font Functions](#)
- [FontFamilyFunctions](#)
- [Graphics Functions](#)
- [GraphicsPath Functions](#)
- [HatchBrush Functions](#)
- [Image Functions](#)
- [ImageAttributes Functions](#)
- [LinearGradientBrush Functions](#)
- [Matrix Functions](#)
- [Memory Functions](#)
- [Notification Functions](#)
- [PathGradientBrush Functions](#)
- [PathIterator Functions](#)
- [Pen Functions](#)
- [Region Functions](#)
- [SolidBrush Functions](#)
- [String Format Functions](#)
- [Text Functions](#)
- [Texture Brush Functions](#)

AdjustableArrowCap Functions

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [AdjustableArrowCap](#) C++ class.

AdjustableArrowCap Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipCreateAdjustableArrowCap(REAL height, REAL width, BOOL isFilled, GpAdjustableArrowCap **cap)	AdjustableArrowCap::AdjustableArrowCap	Creates an adjustable arrow line cap with the specified height and width. The arrow line cap can be filled or nonfilled. The middle inset defaults to zero.
GpStatus WINGDIPAPI GdipSetAdjustableArrowCapHeight(GpAdjustableArrowCap* cap, REAL height)	AdjustableArrowCap::SetHeight	The AdjustableArrowCap::SetHeight method sets the height of the arrow cap. This is the distance from the base of the arrow to its vertex.
GpStatus WINGDIPAPI GdipGetAdjustableArrowCapHeight(GpAdjustableArrowCap* cap, REAL* height)	AdjustableArrowCap::GetHeight	The AdjustableArrowCap::GetHeight method gets the height of the arrow cap. The height is the distance from the base of the arrow to its vertex.
GpStatus WINGDIPAPI GdipSetAdjustableArrowCapWidth(GpAdjustableArrowCap* cap, REAL width)	AdjustableArrowCap::SetWidth	The AdjustableArrowCap::SetWidth method sets the width of the arrow cap. The width is the distance between the endpoints of the base of the arrow.
GpStatus WINGDIPAPI GdipGetAdjustableArrowCapWidth(GpAdjustableArrowCap* cap, REAL* width)	AdjustableArrowCap::GetWidth	The AdjustableArrowCap::GetWidth method gets the width of the arrow cap. The width is the distance between the endpoints of the base of the arrow.
GpStatus WINGDIPAPI GdipSetAdjustableArrowCapMiddleInset(GpAdjustableArrowCap* cap, REAL middleInset)	AdjustableArrowCap::SetMiddleInset	The AdjustableArrowCap::SetMiddleInset method sets the number of units that the midpoint of the base shifts towards the vertex.

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipGetAdjustableArrowCapMiddleInset(GpAdjustableArrowCap* cap, REAL* middleInset)	AdjustableArrowCap::GetMiddleInset	The AdjustableArrowCap::GetMiddleInset method gets the value of the inset. The middle inset is the number of units that the midpoint of the base shifts towards the vertex.
GpStatus WINGDIPAPI GdipSetAdjustableArrowCapFillState(GpAdjustableArrowCap* cap, BOOL fillState)	AdjustableArrowCap::SetFillState	The AdjustableArrowCap::SetFillState method sets the fill state of the arrow cap. If the arrow cap is not filled, only the outline is drawn.
GpStatus WINGDIPAPI GdipGetAdjustableArrowCapFillState(GpAdjustableArrowCap* cap, BOOL* fillState)	AdjustableArrowCap::IsFilled	The AdjustableArrowCap::IsFilled method determines whether the arrow cap is filled.

Bitmap Functions (GDI+)

12/18/2020 • 6 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [Bitmap](#) C++ class.

Bitmap Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipCreateBitmapFromStream(IStream* stream, GpBitmap **bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on a stream. This function does not use Image Color Management (ICM). It is called when the <i>useEmbeddedColorManagement</i> parameter of the Bitmap::Bitmap constructor is set to FALSE.
GpStatus WINGDIPAPI GdipCreateBitmapFromFile(GDIPCONST WCHAR* filename, GpBitmap **bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on an image file. This function does not use ICM. It is called when the <i>useEmbeddedColorManagement</i> parameter of the Bitmap::Bitmap constructor is set to FALSE.
GpStatus WINGDIPAPI GdipCreateBitmapFromStreamICM(IStream* stream, GpBitmap **bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on a stream. This function uses ICM. It is called when the <i>useEmbeddedColorManagement</i> parameter of the Bitmap::Bitmap constructor is set to TRUE.
GpStatus WINGDIPAPI GdipCreateBitmapFromFileICM(GDIPCONST WCHAR* filename, GpBitmap **bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on an image file. This function uses ICM. It is called when the <i>useEmbeddedColorManagement</i> parameter of the Bitmap::Bitmap constructor is set to TRUE.
GpStatus WINGDIPAPI GdipCreateBitmapFromScan0(INT width, INT height, INT stride, PixelFormat format, BYTE* scan0, GpBitmap** bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on an array of bytes along with size and format information.

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipCreateBitmapFromGraphics(INT width, INT height, GpGraphics* target, GpBitmap** bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on a Graphics object, a width, and a height.
GpStatus WINGDIPAPI GdipCreateBitmapFromDirectDrawSurface(IDirectDrawSurface7* surface, GpBitmap** bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on a DirectDraw surface. The Bitmap::Bitmap object maintains a reference to the DirectDraw surface until the Bitmap::Bitmap object is deleted or goes out of scope.
GpStatus WINGDIPAPI GdipCreateBitmapFromGdiDib(GDIPCONST BITMAPINFO* gdiBitmapInfo, VOID* gdiBitmapData, GpBitmap** bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on a BITMAPINFO structure and an array of pixel data.
GpStatus WINGDIPAPI GdipCreateBitmapFromHBITMAP(HBITMAP hbm, HPALETTE hpal, GpBitmap** bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on a handle to a Windows Windows Graphics Device Interface (GDI) bitmap and a handle to a GDI palette.
GpStatus WINGDIPAPI GdipCreateHBITMAPFromBitmap(GpBitmap* bitmap, HBITMAP* hbmReturn, ARGB background)	Bitmap::GetHBITMAP	The Bitmap::GetHBITMAP method creates a GDI bitmap from this Bitmap object.
GpStatus WINGDIPAPI GdipCreateBitmapFromHICON(HICON hicon, GpBitmap** bitmap)	Bitmap::Bitmap	Creates a Bitmap object based on an icon.
GpStatus WINGDIPAPI GdipCreateHICONFromBitmap(GpBitmap* bitmap, HICON* hbmReturn)	Bitmap::GetHICON	The Bitmap::GetHICON method creates an icon from this Bitmap object.
GpStatus WINGDIPAPI GdipCreateBitmapFromResource(HINSTANCE hInstance, GDIPCONST WCHAR* lpBitmapName, GpBitmap** bitmap)	Bitmap::Bitmap	Creates a Bitmap::Bitmap object based on an application or DLL instance handle and the name of a bitmap resource.
GpStatus WINGDIPAPI GdipCloneBitmapArea(REAL x, REAL y, REAL width, REAL height, PixelFormat format, GpBitmap *srcBitmap, GpBitmap **dstBitmap)	Bitmap::Clone	The Bitmap::Clone method creates a new Bitmap object by copying a portion of this bitmap.
GpStatus WINGDIPAPI GdipCloneBitmapAreal(INT x, INT y, INT width, INT height, PixelFormat format, GpBitmap *srcBitmap, GpBitmap **dstBitmap)	Bitmap::Clone	The Bitmap::Clone method creates a new Bitmap object by copying a portion of this bitmap.

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipBitmapLockBits(GpBitmap* bitmap, GDIPCONST GpRect* rect, UINT flags, PixelFormat format, BitmapData* lockedBitmapData)	Bitmap::LockBits	The Bitmap::LockBits method locks a rectangular portion of this bitmap and provides a temporary buffer that you can use to read or write pixel data in a specified format. Any pixel data that you write to the buffer is copied to the Bitmap object when you call Bitmap::UnlockBits .
GpStatus WINGDIPAPI GdipBitmapUnlockBits(GpBitmap* bitmap, BitmapData* lockedBitmapData)	Bitmap::UnlockBits	The Bitmap::UnlockBits method unlocks a portion of this bitmap that was previously locked by a call to Bitmap::LockBits .
GpStatus WINGDIPAPI GdipBitmapGetPixel(GpBitmap* bitmap, INT x, INT y, ARGB *color)	Bitmap::GetPixel	The Bitmap::GetPixel method gets the color of a specified pixel in this bitmap.
GpStatus WINGDIPAPI GdipBitmapSetPixel(GpBitmap* bitmap, INT x, INT y, ARGB color)	Bitmap::SetPixel	The Bitmap::SetPixel method sets the color of a specified pixel in this bitmap.
GpStatus WINGDIPAPI GdipBitmapSetResolution(GpBitmap* bitmap, REAL xdpi, REAL ydpi)	Bitmap::SetResolution	The Bitmap::SetResolution method sets the resolution of this Bitmap object.
GpStatus WINGDIPAPI GdipBitmapConvertFormat(IN GpBitmap *pInputBitmap, PixelFormat format, DitherType dithertype, PaletteType palettetype, ColorPalette *palette, REAL alphaThresholdPercent)	Bitmap::ConvertFormat	The Bitmap::ConvertFormat method converts a bitmap to a specified pixel format. The original pixel data in the bitmap is replaced by the new pixel data.
GpStatus WINGDIPAPI GdipInitializePalette(OUT ColorPalette *palette, PaletteType palettetype, INT optimalColors, BOOL useTransparentColor, GpBitmap *bitmap)	Bitmap::InitializePalette	The Bitmap::InitializePalette method initializes a standard, optimal, or custom color palette.
GpStatus WINGDIPAPI GdipBitmapApplyEffect(GpBitmap* bitmap, CGpEffect *effect, RECT *roi, BOOL useAuxData, VOID **auxData, INT *auxDataSize)	Bitmap::ApplyEffect	The Bitmap::ApplyEffect method alters this Bitmap object by applying a specified effect.
GpStatus WINGDIPAPI GdipBitmapCreateApplyEffect(GpBitmap **inputBitmaps, INT numInputs, CGpEffect *effect, RECT *roi, RECT *outputRect, GpBitmap **outputBitmap, BOOL useAuxData, VOID **auxData, INT *auxDataSize)	Bitmap::ApplyEffect	The Bitmap::ApplyEffect method creates a new Bitmap object by applying a specified effect to an existing Bitmap object.

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipBitmapGetHistogram(GpBitmap* bitmap, IN HistogramFormat format, IN UInt NumberOfEntries, OUT UInt *channel0, OUT UInt *channel1, OUT UInt *channel2, OUT UInt *channel3)	Bitmap::GetHistogram	The Bitmap::GetHistogram method returns one or more histograms for specified color channels of this Bitmap object.
GpStatus WINGDIPAPI GdipBitmapGetHistogramSize(IN HistogramFormat format, OUT UInt *NumberOfEntries)	Bitmap::GetHistogramSize	The Bitmap::GetHistogramSize returns the number of elements (in an array of UInt s) that you must allocate before you call the Bitmap::GetHistogram method of a Bitmap object.
Status __stdcall GdipCreateEffect(const GUID guid, CGpEffect **effect)	Effect	The constructors of all descendants of the Effect class call GdipCreateEffect. For example, the Blur constructor makes the following call: GdipCreateEffect(BlurEffectGuid, &nativeEffect); BlurEffectGuid is a constant defined in Gdipluseffects.h .
Status __stdcall GdipDeleteEffect(CGpEffect *effect)	virtual ~Effect()	Cleans up resources used by a Bitmap object.
Status __stdcall GdipGetEffectParameterSize(CGpEffect *effect, UInt *size)	Effect::GetParameterSize	The Effect::GetParameterSize method gets the total size, in bytes, of the parameters currently set for this Effect . The Effect::GetParameterSize method is usually called on an object that is an instance of a descendant of the Effect class.
Status __stdcall GdipSetEffectParameters(CGpEffect *effect, const VOID *params, const UInt size)	Effect	Each descendant of the Effect class has a SetParameters method that calls the protected method Effect::SetParameters , which in turn calls GdipSetEffectParameters . For example, the Blur::SetParameters method makes the following call: Effect::SetParameters(parameters, size) .
Status __stdcall GdipGetEffectParameters(CGpEffect *effect, UInt *size, VOID *params)	Effect	Each descendant of the Effect class has a GetParameters method that calls the protected method Effect::GetParameters , which in turn calls GdipSetEffectParameters . For example, the Blur::GetParameters method makes the following call: Effect::GetParameters(parameters, size) .

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipTestControl(GpTestControlEnum control, void * param)	Not called by wrappers	<p>Used for internal testing of GDI+.</p> <p>Specifies changes in image-processing behavior. The meaning of the <i>param</i> parameter varies depending on the value passed in the <i>control</i> parameter.</p> <p>If the <i>control</i> parameter is set to TestControlForceBilinear, <i>param</i> points to a value of type BOOL. If that value is TRUE and the current interpolation mode is anything other than InterpolationModeNearestNeighbor, then the InterpolationModeBilinear algorithm is used. If the value pointed to by <i>param</i> is FALSE or if the current interpolation mode is InterpolationModeNearestNeighbor, then the behavior of GDI+ is unchanged.</p> <p>If the <i>control</i> parameter is set to TestControlNoICM, <i>param</i> points to a value of type BOOL. If that value is TRUE, then ICM profiles are ignored when images are loaded into memory. If the value pointed to by <i>param</i> is FALSE, then the behavior of GDI+ is unchanged.</p> <p>If the <i>control</i> parameter is set to TestControlGetBuildNumber, <i>param</i> is an output parameter that points to a 32-bit integer variable. That variable receives the GDI+ build number.</p> <p>GdipTestControl is not thread-safe; it assumes that there is only one thread using GDI+. If you call GdipTestControl in a multithreaded environment, the results are unpredictable.</p> <p>The GpTestControlEnum enumeration has three elements:</p> <ul style="list-style-type: none"> TestControlForceBilinear = 0, TestControlNoICM = 1, and TestControlGetBuildNumber = 2.

Brush Functions (GDI+)

12/18/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the **Brush** C++ class.

Brush Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipCloneBrush(GpBrush *brush, GpBrush **cloneBrush)	Brush::Clone	The Brush::Clone method creates a new Brush object based on this brush.
GpStatus WINGDIPAPI GdipDeleteBrush(GpBrush *brush)	virtual ~Brush()	Cleans up resources used by a Brush object.
GpStatus WINGDIPAPI GdipGetBrushType(GpBrush *brush, GpBrushType *type)	Brush::GetType	The Brush::GetType method gets the type of this brush.

CachedBitmap Functions

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [CachedBitmap](#) C++ class.

CachedBitmap Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipCreateCachedBitmap(GpBitmap *bitmap, GpGraphics *graphics, GpCachedBitmap **cachedBitmap)	CachedBitmap::CachedBitmap	Creates a CachedBitmap::CachedBitmap object based on a Bitmap object and a Graphics object. The cached bitmap takes the pixel data from the Bitmap object and stores it in a format that is optimized for the display device associated with the Graphics object.
GpStatus WINGDIPAPI GdipDeleteCachedBitmap(GpCachedBit map *cachedBitmap)	<code>~CachedBitmap()</code>	Cleans up resources used by a CachedBitmap object.
GpStatus WINGDIPAPI GdipDrawCachedBitmap(GpGraphics *graphics, GpCachedBitmap *cachedBitmap, INT x, INT y)	Graphics::DrawCachedBitmap	The Graphics::DrawCachedBitmap method draws the image stored in a CachedBitmap object.
UINT WINGDIPAPI GdipEmfToWmfBits(HENHMETAFILE hemf, UINT cbData16, LPBYTE pData16, INT iMapMode, INT eFlags)	Metafile::EmfToWmfBits	Converts an enhanced-format metafile to a Windows Metafile Format (WMF) metafile and stores the converted records in a specified buffer.

CustomLineCap Functions

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [CustomLineCap](#) C++ class.

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipCreateCustomLineCap(GpPath* fillPath, GpPath* strokePath, GpLineCap baseCap, REAL baseInset, GpCustomLineCap **customCap)	CustomLineCap::CustomLineCap	Creates a CustomLineCap::CustomLineCap object.
GpStatus WINGDIPAPI GdipDeleteCustomLineCap(GpCustomLineCap* customCap)	virtual ~CustomLineCap()	Cleans up resources used by a CustomLineCap::CustomLineCap object.
GpStatus WINGDIPAPI GdipCloneCustomLineCap(GpCustomLineCap* customCap, GpCustomLineCap** clonedCap)	CustomLineCap::Clone	The CustomLineCap::Clone method copies the contents of the existing object into a new CustomLineCap object.
GpStatus WINGDIPAPI GdipGetCustomLineCapType(GpCustomLineCap* customCap, CustomLineCapType* capType)	Not called by wrapper methods.	When this function is called, the <i>capType</i> parameter receives the type of the CustomLineCap specified by <i>customCap</i> . The CustomLineCapType enumeration (defined in GdiplusEnums.h) has two elements: CustomLineCapTypeDefault = 0 and CustomLineCapTypeAdjustableArrow = 1.
GpStatus WINGDIPAPI GdipSetCustomLineCapStrokeCaps(GpCustomLineCap* customCap, GpLineCap startCap, GpLineCap endCap)	CustomLineCap::SetStrokeCap	The CustomLineCap::SetStrokeCap method sets the LineCap object used to start and end lines within the GraphicsPath object that defines this CustomLineCap object.
GpStatus WINGDIPAPI GdipGetCustomLineCapStrokeCaps(GpCustomLineCap* customCap, GpLineCap* startCap, GpLineCap* endCap)	CustomLineCap::GetStrokeCaps	The CustomLineCap::GetStrokeCaps method gets the end cap styles for both the start line cap and the end line cap. Line caps are LineCap objects that end the individual lines within a path.

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipSetCustomLineCapStrokeJoin(GpCustomLineCap* customCap, GpLineJoin lineJoin)	CustomLineCap::SetStrokeJoin	The CustomLineCap::SetStrokeJoin method sets the style of line join for the stroke. The line join specifies how two lines that intersect within the GraphicsPath object that makes up the custom line cap are joined.
GpStatus WINGDIPAPI GdipGetCustomLineCapStrokeJoin(GpCustomLineCap* customCap, GpLineJoin* lineJoin)	CustomLineCap::GetStrokeJoin	The CustomLineCap::GetStrokeJoin method returns the style of LineJoin used to join multiple lines in the same GraphicsPath object.
GpStatus WINGDIPAPI GdipSetCustomLineCapBaseCap(GpCustomLineCap* customCap, GpLineCap baseCap)	CustomLineCap::SetBaseCap	The CustomLineCap::SetBaseCap method sets the LineCap that appears as part of this CustomLineCap at the end of a line.
GpStatus WINGDIPAPI GdipGetCustomLineCapBaseCap(GpCustomLineCap* customCap, GpLineCap* baseCap)	CustomLineCap::GetBaseCap	The CustomLineCap::GetBaseCap method gets the style of the base cap. The base cap is a LineCap object used as a cap at the end of a line along with this CustomLineCap object.
GpStatus WINGDIPAPI GdipSetCustomLineCapBaseInset(GpCustomLineCap* customCap, REAL inset)	CustomLineCap::SetBaseInset	The CustomLineCap::SetBaseInset method sets the base inset value of this custom line cap. This is the distance between the end of a line and the base cap.
GpStatus WINGDIPAPI GdipGetCustomLineCapBaseInset(GpCustomLineCap* customCap, REAL* inset)	CustomLineCap::GetBaseInset	The CustomLineCap::GetBaseInset method gets the distance between the base cap to the start of the line.
GpStatus WINGDIPAPI GdipSetCustomLineCapWidthScale(GpCustomLineCap* customCap, REAL widthScale)	CustomLineCap::SetWidthScale	The CustomLineCap::SetWidthScale method sets the value of the scale width. This is the amount to scale the custom line cap relative to the width of the Pen used to draw lines. The default value of 1.0 does not scale the line cap.
GpStatus WINGDIPAPI GdipGetCustomLineCapWidthScale(GpCustomLineCap* customCap, REAL* widthScale)	CustomLineCap::GetWidthScale	The CustomLineCap::GetWidthScale method gets the value of the scale width. This is the amount to scale the custom line cap relative to the width of the Pen object used to draw a line. The default value of 1.0 does not scale the line cap.

Font Functions

11/2/2020 • 3 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [Font](#) C++ class.

Font Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreateFontFromDC(HDC hdc, GpFont **font)	Font::Font(IN HDC hdc)	Creates a Font object based on the GDI font object that is currently selected into a specified device context. This constructor is provided for compatibility with GDI.
GpStatus WINGDIPAPI GdipCreateFontFromLogfontA(HDC hdc, GDIPCONST LOGFONTA *logfont, GpFont **font)	Font::Font(IN HDC hdc, IN const LOGFONTA* logfont)	Creates a Font object directly from a GDI logical font. The GDI logical font is a LOGFONTA structure, which is the one-byte character version of a logical font. This constructor is provided for compatibility with GDI.
GpStatus WINGDIPAPI GdipCreateFontFromLogfontW(HDC hdc, GDIPCONST LOGFONTW *logfont, GpFont **font)	Font::Font(IN HDC hdc, IN const LOGFONTW* logfont)	Creates a Font object directly from a GDI logical font. The GDI logical font is a LOGFONTW structure, which is the one-byte character version of a logical font. This constructor is provided for compatibility with GDI.
GpStatus WINGDIPAPI GdipCloneFont(GpFont* font, GpFont** cloneFont)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipCreateFont(GDIPCONST GpFontFamily *fontFamily, REAL emSize, INT style, Unit unit, GpFont **font)	Font::Font(IN const FontFamily * family, IN REAL emSize, IN INT style, IN Unit unit)	Creates a Font object based on a font family, a size, a font style, a unit of measurement, and a FontCollection object.
GpStatus WINGDIPAPI GdipCloneFont(GpFont* font, GpFont** cloneFont)	Font::Font* Clone() const	Creates a new Font object based on this Font object.
GpStatus WINGDIPAPI GdipDeleteFont(GpFont* font)	Not called by wrapper methods.	Not implemented.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetFamily(GpFont *font, GpFontFamily **family)	Status Font::GetFamily(OUT FontFamily *family) const	Gets the font family on which this font is based.
GpStatus WINGDIPAPI GdipGetFontStyle(GpFont *font, INT *style)	INT Font::GetStyle() const	Gets the style of this font's typeface
GpStatus WINGDIPAPI GdipGetFontSize(GpFont *font, REAL *size)	REAL Font::GetSize() const	Returns the font size (commonly called the em size) of this Font object. The size is in the units of this Font object.
GpStatus WINGDIPAPI GdipGetFontUnit(GpFont *font, Unit *unit)	Unit Font::GetUnit() const	Returns the unit of measure of this Font object.
GpStatus WINGDIPAPI GdipGetFontHeight(GDIPCONST GpFont *font, GDIPCONST GpGraphics *graphics, REAL *height)	REAL Font::GetHeight(IN const Graphics *graphics) const	Gets the line spacing of this font in the current unit of a specified Graphics object. The line spacing is the vertical distance between the base lines of two consecutive lines of text. Thus, the line spacing includes the blank space between lines along with the height of the character itself.
GpStatus WINGDIPAPI GdipGetFontHeightGivenDPI(GDIPCON ST GpFont *font, REAL dpi, REAL *height)	REAL Font::GetHeight(IN REAL dpi) const	Gets the line spacing, in pixels, of this font. The line spacing is the vertical distance between the base lines of two consecutive lines of text. Thus, the line spacing includes the blank space between lines along with the height of the character itself.
GpStatus WINGDIPAPI GdipGetLogFontA(GpFont * font, GpGraphics *graphics, LOGFONTA * logfontA)	Status Font::GetLogFontA(IN const Graphics *g, OUT LOGFONTA *logfontA) const	Uses a LOGFONTA structure to get the attributes of this Font object.
GpStatus WINGDIPAPI GdipGetLogFontW(GpFont * font, GpGraphics *graphics, LOGFONTW * logfontW)	Status Font::GetLogFontW(IN const Graphics *g, OUT LOGFONTW *logfontW) const	Uses a LOGFONTW structure to get the attributes of this Font object.
GpStatus WINGDIPAPI GdipNewInstalledFontCollection(GpFont Collection** fontCollection)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipNewPrivateFontCollection(GpFontC ollection** fontCollection)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipDeletePrivateFontCollection(GpFon tCollection** fontCollection)	Not called by wrapper methods.	Not implemented.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetFontCollectionFamilyCount(GpFontCollection* fontCollection, INT * numFound)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipGetFontCollectionFamilyList(GpFontCollection* fontCollection, INT numSought, GpFontFamily* gpfamilies[], INT* numFound)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipPrivateAddFontFile(GpFontCollection* fontCollection, GDIPCONST WCHAR* filename)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipPrivateAddMemoryFont(GpFontCollection* fontCollection, GDIPCONST void* memory, INT length)	Not called by wrapper methods.	Not implemented.

FontFamilyFunctions

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdipplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [FontFamily](#) C++ class.

FontFamily Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipDeleteFontFamily(GpFontFamily *fontFamily)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipCloneFontFamily(GpFontFamily *fontFamily, GpFontFamily **clonedFontFamily)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipCreateFontFamilyFromName(GDIP CONST WCHAR *name, GpFontCollection *fontCollection, GpFontFamily **FontFamily)	FontFamily::FontFamily(IN const WCHAR* name, IN const FontCollection* fontCollection)	Creates a FontFamily::FontFamily object based on a specified font family.
GpStatus WINGDIPAPI GdipCloneFontFamily(GpFontFamily *FontFamily, GpFontFamily **clonedFontFamily)	FontFamily* FontFamily::Clone()	Creates a new FontFamily::FontFamily object based on this FontFamily::FontFamily object.
GpStatus WINGDIPAPI GdipGetGenericFontFamilySansSerif(Gp FontFamily **nativeFamily)	FontFamily * FontFamily::GenericSansSerif()	Gets a FontFamily::FontFamily object that specifies a generic sans serif typeface.
GpStatus WINGDIPAPI GdipGetGenericFontFamilySerif(GpFont Family **nativeFamily)	const FontFamily * FontFamily::GenericSerif()	Gets a FontFamily::FontFamily object that specifies a generic serif typeface.
GpStatus WINGDIPAPI GdipGetGenericFontFamilyMonospace(GpFontFamily **nativeFamily)	const FontFamily * FontFamily::GenericMonospace()	Gets a FontFamily::FontFamily object that specifies a generic monospace typeface.
GpStatus WINGDIPAPI GdipGetFamilyName(GDIPCONST GpFontFamily *family, WCHAR name[LF_FACESIZE], LANGID language)	Status FontFamily::GetFamilyName(IN WCHAR name[LF_FACESIZE], IN LANGID language) const	Gets the name of this font family.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdiplIsStyleAvailable(GDIPCONST GpFontFamily *family, INT style, BOOL * IsStyleAvailable)	BOOL FontFamily::IsStyleAvailable(IN INT style) const	Determines whether the specified style is available for this font family.
GpStatus WINGDIPAPI GdipFontCollectionEnumerable(GpFontCollection* fontCollection, GpGraphics* graphics, INT * numFound)	Not called by wrapper methods.	Not implemented
GpStatus WINGDIPAPI GdipFontCollectionEnumerate(GpFontCollection* fontCollection, INT numSought, GpFontFamily* gpfamilies[], INT* numFound, GpGraphics* graphics)	Not called by wrapper methods.	Not implemented
GpStatus WINGDIPAPI GdipGetEmHeight(GDIPCONST GpFontFamily *family, INT style, UINT16 * EmHeight)	UINT16 FontFamily::GetEmHeight(IN INT style) const	Gets the size (commonly called em size or em height), in design units, of this font family.
GpStatus WINGDIPAPI GdipGetCellAscent(GDIPCONST GpFontFamily *family, INT style, UINT16 * CellAscent)	UINT16 FontFamily::GetCellAscent(IN INT style) const	Gets the cell ascent, in design units, of this font family for the specified style or style combination.
GpStatus WINGDIPAPI GdipGetCellDescent(GDIPCONST GpFontFamily *family, INT style, UINT16 * CellDescent)	UINT16 FontFamily::GetCellDescent(IN INT style) const	Gets the cell descent, in design units, of this font family for the specified style or style combination.
GpStatus WINGDIPAPI GdipGetLineSpacing(GDIPCONST GpFontFamily *family, INT style, UINT16 * LineSpacing)	UINT16 FontFamily::GetLineSpacing(IN INT style) const	Gets the line spacing, in design units, of this font family for the specified style or style combination. The line spacing is the vertical distance between the base lines of two consecutive lines of text.

Graphics Functions

11/2/2020 • 33 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the **Graphics** C++ class.

Graphics Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipFlush(GpGraphics *graphics, GpFlushIntention intention)	VOID Graphics::Flush(IN FlushIntention intention = FlushIntentionFlush)	Flushes all pending graphics operations.
GpStatus WINGDIPAPI GdipCreateFromHDC(HDC hdc, GpGraphics **graphics)	Graphics::Graphics(IN HDC hdc)	Creates a Graphics object that is associated with a specified device context.
GpStatus WINGDIPAPI GdipCreateFromHDC2(HDC hdc, HANDLE hDevice, GpGraphics **graphics)	Graphics::Graphics(IN HDC hdc, IN HANDLE hdevice)	Creates a Graphics object that is associated with a specified device context and a specified device.
GpStatus WINGDIPAPI GdipCreateFromHWND(HWND hwnd, GpGraphics **graphics)	Graphics::Graphics(IN HWND hwnd, IN BOOL icm = FALSE)	Creates a Graphics object that is associated with a specified window.
GpStatus WINGDIPAPI GdipCreateFromHWNDICM(HWND hwnd, GpGraphics **graphics)	Graphics::Graphics(IN HWND hwnd, IN BOOL icm = FALSE)	This function uses Image Color Management (ICM). It is called when the <i>icm</i> parameter of the Graphics::Graphics constructor is set to TRUE.
GpStatus WINGDIPAPI GdipDeleteGraphics(GpGraphics *graphics)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipGetDC(GpGraphics* graphics, HDC * hdc)	HDC Graphics::GetHDC()	Gets a handle to the device context associated with this Graphics object.
GpStatus WINGDIPAPI GdipReleaseDC(GpGraphics* graphics, HDC hdc)	VOID Graphics::ReleaseHDC(IN HDC hdc)	Releases a device context handle obtained by a previous call to the Graphics::GetHDC method of this Graphics object.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSetCompositingMode(GpGraphics *graphics, CompositingMode compositingMode)	Status Graphics::SetCompositingMode(IN CompositingMode compositingMode)	Sets the compositing mode of this Graphics object.
GpStatus WINGDIPAPI GdipGetCompositingMode(GpGraphics *graphics, CompositingMode *compositingMode)	Status Graphics::GetCompositingMode() const	Gets the compositing mode currently set for this Graphics object.
GpStatus WINGDIPAPI GdipSetRenderingOrigin(GpGraphics *graphics, INT x, INT y)	Status Graphics::SetRenderingOrigin(IN INT x, IN INT y)	Sets the rendering origin of this Graphics object. The rendering origin is used to set the dither origin for 8-bits-per-pixel and 16-bits-per-pixel dithering and is also used to set the origin for hatch brushes. Syntax
GpStatus WINGDIPAPI GdipGetRenderingOrigin(GpGraphics *graphics, INT *x, INT *y)	Status Graphics::GetRenderingOrigin(OUT INT *x, OUT INT *y) const	Gets the rendering origin currently set for this Graphics object. The rendering origin is used to set the dither origin for 8-bits per pixel and 16-bits per pixel dithering and is also used to set the origin for hatch brushes.
GpStatus WINGDIPAPI GdipSetCompositingQuality(GpGraphics *graphics, CompositingQuality compositingQuality)	Status Graphics::SetCompositingQuality(IN CompositingQuality compositingQuality)	Sets the compositing quality of this Graphics object.
GpStatus WINGDIPAPI GdipGetCompositingQuality(GpGraphics *graphics, CompositingQuality *compositingQuality)	CompositingQuality Graphics::GetCompositingQuality() const	Gets the compositing quality currently set for this Graphics object.
GpStatus WINGDIPAPI GdipSetSmoothingMode(GpGraphics *graphics, SmoothingMode smoothingMode)	Status Graphics::SetSmoothingMode(IN SmoothingMode smoothingMode)	Sets the rendering quality of the Graphics object.
GpStatus WINGDIPAPI GdipGetSmoothingMode(GpGraphics *graphics, SmoothingMode *smoothingMode)	SmoothingMode Graphics::GetSmoothingMode() const	Determines whether smoothing (antialiasing) is applied to the Graphics object.
GpStatus WINGDIPAPI GdipSetPixelOffsetMode(GpGraphics *graphics, PixelOffsetMode pixelOffsetMode)	Status Graphics::SetPixelOffsetMode(IN PixelOffsetMode pixelOffsetMode)	Sets the pixel offset mode of this Graphics object.
GpStatus WINGDIPAPI GdipGetPixelOffsetMode(GpGraphics *graphics, PixelOffsetMode *pixelOffsetMode)	PixelOffsetMode Graphics::GetPixelOffsetMode() const	Gets the pixel offset mode currently set for this Graphics object.
GpStatus WINGDIPAPI GdipSetTextRenderingHint(GpGraphics *graphics, TextRenderingHint mode)	Status Graphics::SetTextRenderingHint(IN TextRenderingHint newMode)	Sets the text rendering mode of this Graphics object.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetTextRenderingHint(GpGraphics *graphics, TextRenderingHint *mode)	TextRenderingHint Graphics::GetTextRenderingHint() const	Gets the text rendering mode currently set for this Graphics object.
GpStatus WINGDIPAPI GdipSetTextContrast(GpGraphics *graphics, UINT contrast)	Status Graphics::SetTextContrast(IN UINT contrast)	Sets the contrast value of this Graphics object. The contrast value is used for antialiasing text.
GpStatus WINGDIPAPI GdipGetTextContrast(GpGraphics *graphics, UINT * contrast)	UINT Graphics::GetTextContrast() const	Gets the contrast value currently set for this Graphics object. The contrast value is used for antialiasing text.
GpStatus WINGDIPAPI GdipSetInterpolationMode(GpGraphics *graphics, InterpolationMode interpolationMode)	Status Graphics::SetInterpolationMode(IN InterpolationMode interpolationMode)	Sets the interpolation mode of this Graphics object. The interpolation mode determines the algorithm that is used when images are scaled or rotated.
GpStatus WINGDIPAPI GdipGraphicsSetAbort(GpGraphics *pGraphics, GdiplusAbort *pAbort)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipGetInterpolationMode(GpGraphics *graphics, InterpolationMode *interpolationMode)	InterpolationMode Graphics::GetInterpolationMode() const	Gets the interpolation mode currently set for this Graphics object. The interpolation mode determines the algorithm that is used when images are scaled or rotated.
GpStatus WINGDIPAPI GdipSetWorldTransform(GpGraphics *graphics, GpMatrix *matrix)	Status Graphics::SetTransform(IN const Matrix* matrix)	Sets the world transformation of this Graphics object.
GpStatus WINGDIPAPI GdipResetWorldTransform(GpGraphics *graphics)	Status Graphics::ResetTransform()	Sets the world transformation matrix of this Graphics object to the identity matrix.
GpStatus WINGDIPAPI GdipMultiplyWorldTransform(GpGraphics *graphics, GDIPCONST GpMatrix *matrix, GpMatrixOrder order)	Status Graphics::MultiplyTransform(IN const Matrix* matrix, IN MatrixOrder order = MatrixOrderPrepend)	Updates this Graphics object's world transformation matrix with the product of itself and another matrix.
GpStatus WINGDIPAPI GdipTranslateWorldTransform(GpGraphics *graphics, REAL dx, REAL dy, GpMatrixOrder order)	Status Graphics::TranslateTransform(IN REAL dx, IN REAL dy, IN MatrixOrder order = MatrixOrderPrepend)	Updates this Graphics object's world transformation matrix with the product of itself and a translation matrix.
GpStatus WINGDIPAPI GdipScaleWorldTransform(GpGraphics *graphics, REAL sx, REAL sy, GpMatrixOrder order)	Status Graphics::ScaleTransform(IN REAL sx, IN REAL sy, IN MatrixOrder order = MatrixOrderPrepend)	Updates this Graphics object's world transformation matrix with the product of itself and a scaling matrix.
GpStatus WINGDIPAPI GdipRotateWorldTransform(GpGraphics *graphics, REAL angle, GpMatrixOrder order)	Status Graphics::RotateTransform(IN REAL angle, IN MatrixOrder order = MatrixOrderPrepend)	Updates the world transformation matrix of this Graphics object with the product of itself and a rotation matrix.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetWorldTransform(GpGraphics *graphics, GpMatrix *matrix)	Status Graphics::GetTransform(OUT Matrix* matrix) const	Gets the world transformation matrix of this Graphics object.
GpStatus WINGDIPAPI GdipResetPageTransform(GpGraphics *graphics)	Not called by wrapper methods.	This function resets the page transform matrix to identity.
GpStatus WINGDIPAPI GdipGetPageUnit(GpGraphics *graphics, GpUnit *unit)	Unit Graphics::GetPageUnit() const	Gets the unit of measure currently set for this Graphics object.
GpStatus WINGDIPAPI GdipGetPageScale(GpGraphics *graphics, REAL *scale)	REAL Graphics::GetPageScale() const	Gets the scaling factor currently set for the page transformation of this Graphics object. The page transformation converts page coordinates to device coordinates.
GpStatus WINGDIPAPI GdipSetPageUnit(GpGraphics *graphics, GpUnit unit)	Status Graphics::SetPageUnit(IN Unit unit)	Sets the unit of measure for this Graphics object. The page unit belongs to the page transformation, which converts page coordinates to device coordinates.
GpStatus WINGDIPAPI GdipSetPageScale(GpGraphics *graphics, REAL scale)	Status Graphics::SetPageScale(IN REAL scale)	Sets the scaling factor for the page transformation of this Graphics object. The page transformation converts page coordinates to device coordinates.
GpStatus WINGDIPAPI GdipGetDpiX(GpGraphics *graphics, REAL* dpi)	REAL Graphics::GetDpiX() const	Gets the horizontal resolution, in dots per inch, of the display device associated with this Graphics object.
GpStatus WINGDIPAPI GdipGetDpiY(GpGraphics *graphics, REAL* dpi)	REAL Graphics::GetDpiY() const	Gets the vertical resolution, in dots per inch, of the display device associated with this Graphics object.
GpStatus WINGDIPAPI GdipTransformPoints(GpGraphics *graphics, GpCoordinateSpace destSpace, GpCoordinateSpace srcSpace, GpPointF *points, INT count)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipTransformPointsI(GpGraphics *graphics, GpCoordinateSpace destSpace, GpCoordinateSpace srcSpace, GpPoint *points, INT count)	Status Graphics::TransformPoints(IN CoordinateSpace destSpace, IN CoordinateSpace srcSpace, IN OUT Point* pts, IN INT count) const	Converts an array of points from one coordinate space to another. The conversion is based on the current world and page transformations of this Graphics object.
GpStatus WINGDIPAPI GdipGetNearestColor(GpGraphics *graphics, ARGB* argb)	Status Graphics::GetNearestColor(IN OUT Color* color) const	Gets the nearest color to the color that is passed in. This method works on 8-bits per pixel or lower display devices for which there is an 8-bit color palette.
HPALETTE WINGDIPAPI GdipCreateHalftonePalette()	static HPALETTE Graphics::GetHalftonePalette()	Gets a Windows halftone palette.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipDrawLine(GpGraphics *graphics, GpPen *pen, REAL x1, REAL y1, REAL x2, REAL y2)	Status Graphics::DrawLine(IN const Pen* pen, IN REAL x1, IN REAL y1, IN REAL x2, IN REAL y2)	Draws a line that connects two points.
GpStatus WINGDIPAPI GdipDrawLineI(GpGraphics *graphics, GpPen *pen, INT x1, INT y1, INT x2, INT y2)	Status Graphics::DrawLine(IN const Pen* pen, IN INT x1, IN INT y1, IN INT x2, IN INT y2)	Draws a line that connects two points.
GpStatus WINGDIPAPI GdipDrawLines(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPointF *points, INT count)	Status Graphics::DrawLines(IN const Pen* pen, IN const PointF* points, IN INT count)	Draws a sequence of connected lines.
GpStatus WINGDIPAPI GdipDrawLinesI(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPoint *points, INT count)	Status Graphics::DrawLines(IN const Pen* pen, IN const Point* points, IN INT count)	Draws a sequence of connected lines.
GpStatus WINGDIPAPI GdipDrawArc(GpGraphics *graphics, GpPen *pen, REAL x, REAL y, REAL width, REAL height, REAL startAngle, REAL sweepAngle)	Status Graphics::DrawArc(IN const Pen* pen, IN REAL x, IN REAL y, IN REAL width, IN REAL height, IN REAL startAngle, IN REAL sweepAngle)	Draws an arc. The arc is part of an ellipse.
GpStatus WINGDIPAPI GdipDrawArcI(GpGraphics *graphics, GpPen *pen, INT x, INT y, INT width, INT height, REAL startAngle, REAL sweepAngle)	Status Graphics::DrawArc(IN const Pen* pen, IN INT x, IN INT y, IN INT width, IN INT height, IN REAL startAngle, IN REAL sweepAngle)	Draws an arc. The arc is part of an ellipse.
GpStatus WINGDIPAPI GdipDrawBezier(GpGraphics *graphics, GpPen *pen, REAL x1, REAL y1, REAL x2, REAL y2, REAL x3, REAL y3, REAL x4, REAL y4)	Status Graphics::DrawBezier(IN const Pen* pen, IN REAL x1, IN REAL y1, IN REAL x2, IN REAL y2, IN REAL x3, IN REAL y3, IN REAL x4, IN REAL y4)	Draws a Bézier spline.
GpStatus WINGDIPAPI GdipDrawBezierI(GpGraphics *graphics, GpPen *pen, INT x1, INT y1, INT x2, INT y2, INT x3, INT y3, INT x4, INT y4)	Status Graphics::DrawBezier(IN const Pen* pen, IN INT x1, IN INT y1, IN INT x2, IN INT y2, IN INT x3, IN INT y3, IN INT x4, IN INT y4)	Draws a Bézier spline.
GpStatus WINGDIPAPI GdipDrawBeziers(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPointF *points, INT count)	Status Graphics::DrawBeziers(IN const Pen* pen, IN const PointF* points, IN INT count)	Draws a sequence of connected Bézier splines.
GpStatus WINGDIPAPI GdipDrawBeziersI(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPoint *points, INT count)	Status Graphics::DrawBeziers(IN const Pen* pen, IN const Point* points, IN INT count)	Draws a sequence of connected Bézier splines.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipDrawRectangle(GpGraphics *graphics, GpPen *pen, REAL x, REAL y, REAL width, REAL height)	Status Graphics::DrawRectangle(IN const Pen* pen, IN REAL x, IN REAL y, IN REAL width, IN REAL height)	Draws a rectangle.
GpStatus WINGDIPAPI GdipDrawRectangleI(GpGraphics *graphics, GpPen *pen, INT x, INT y, INT width, INT height)	Status Graphics::DrawRectangle(IN const Pen* pen, IN INT x, IN INT y, IN INT width, IN INT height)	Draws a rectangle.
GpStatus WINGDIPAPI GdipDrawRectangles(GpGraphics *graphics, GpPen *pen, GDIPCONST GpRectF *rects, INT count)	Status Graphics::DrawRectangles(IN const Pen* pen, IN const RectF* rects, IN INT count)	Draws a sequence of rectangles.
GpStatus WINGDIPAPI GdipDrawRectanglesI(GpGraphics *graphics, GpPen *pen, GDIPCONST GpRect *rects, INT count)	Status Graphics::DrawRectangles(IN const Pen* pen, IN const Rect* rects, IN INT count)	Draws a sequence of rectangles.
GpStatus WINGDIPAPI GdipDrawEllipse(GpGraphics *graphics, GpPen *pen, REAL x, REAL y, REAL width, REAL height)	Status Graphics::DrawEllipse(IN const Pen* pen, IN REAL x, IN REAL y, IN REAL width, IN REAL height)	Draws an ellipse.
GpStatus WINGDIPAPI GdipDrawEllipsel(GpGraphics *graphics, GpPen *pen, INT x, INT y, INT width, INT height)	Status Graphics::DrawEllipse(IN const Pen* pen, IN INT x, IN INT y, IN INT width, IN INT height)	Draws an ellipse.
GpStatus WINGDIPAPI GdipDrawPie(GpGraphics *graphics, GpPen *pen, REAL x, REAL y, REAL width, REAL height, REAL startAngle, REAL sweepAngle)	Status Graphics::DrawPie(IN const Pen* pen, IN REAL x, IN REAL y, IN REAL width, IN REAL height, IN REAL startAngle, IN REAL sweepAngle)	Draws a pie.
GpStatus WINGDIPAPI GdipDrawPiel(GpGraphics *graphics, GpPen *pen, INT x, INT y, INT width, INT height, REAL startAngle, REAL sweepAngle)	Status Graphics::DrawPie(IN const Pen* pen, IN INT x, IN INT y, IN INT width, IN INT height, IN REAL startAngle, IN REAL sweepAngle)	Draws a pie.
GpStatus WINGDIPAPI GdipDrawPolygon(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPointF *points, INT count)	Status Graphics::DrawPolygon(IN const Pen* pen, IN const PointF* points, IN INT count)	Draws a polygon.
GpStatus WINGDIPAPI GdipDrawPolyoni(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPoint *points, INT count)	Status Graphics::DrawPolygon(IN const Pen* pen, IN const Point* points, IN INT count)	Draws a polygon.
GpStatus WINGDIPAPI GdipDrawPath(GpGraphics *graphics, GpPen *pen, GpPath *path)	Status Graphics::DrawPath(IN const Pen* pen, IN const GraphicsPath* path)	Draws a sequence of lines and curves defined by a GraphicsPath object.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipDrawCurve(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPointF *points, INT count)	Status Graphics::DrawCurve(IN const Pen* pen, IN const PointF* points, IN INT count)	Draws a closed cardinal spline.
GpStatus WINGDIPAPI GdipDrawCurve1(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPoint *points, INT count)	Status Graphics::DrawCurve(IN const Pen* pen, IN const Point* points, IN INT count)	Draws a closed cardinal spline.
GpStatus WINGDIPAPI GdipDrawCurve2(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPointF *points, INT count, REAL tension)	Status Graphics::DrawCurve(IN const Pen* pen, IN const PointF* points, IN INT count, IN REAL tension)	Draws a closed cardinal spline.
GpStatus WINGDIPAPI GdipDrawCurve2I(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPoint *points, INT count, REAL tension)	Status Graphics::DrawCurve(IN const Pen* pen, IN const Point* points, IN INT count, IN REAL tension)	Draws a closed cardinal spline.
GpStatus WINGDIPAPI GdipDrawCurve3(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPointF *points, INT count, INT offset, INT numberOfSegments, REAL tension)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipDrawCurve3I(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPoint *points, INT count, INT offset, INT numberOfSegments, REAL tension)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipDrawClosedCurve(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPointF *points, INT count)	Status Graphics::DrawClosedCurve(IN const Pen* pen, IN const PointF* points, IN INT count)	Draws a closed cardinal spline.
GpStatus WINGDIPAPI GdipDrawClosedCurve1(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPoint *points, INT count)	Status Graphics::DrawClosedCurve(IN const Pen* pen, IN const Point* points, IN INT count)	Draws a closed cardinal spline.
GpStatus WINGDIPAPI GdipDrawClosedCurve2(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPointF *points, INT count, REAL tension)	Status Graphics::DrawClosedCurve(IN const Pen *pen, IN const PointF* points, IN INT count, IN REAL tension)	Draws a closed cardinal spline.
GpStatus WINGDIPAPI GdipDrawClosedCurve2I(GpGraphics *graphics, GpPen *pen, GDIPCONST GpPoint *points, INT count, REAL tension)	Status Graphics::DrawClosedCurve(IN const Pen *pen, IN const Point* points, IN INT count, IN REAL tension)	Draws a closed cardinal spline.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGraphicsClear(GpGraphics *graphics, ARGB color)	Status Graphics::Clear(IN const Color &color)	Clears a Graphics object to a specified color.
GpStatus WINGDIPAPI GdipFillRectangle(GpGraphics *graphics, GpBrush *brush, REAL x, REAL y, REAL width, REAL height)	Status Graphics::FillRectangle(IN const Brush* brush, IN REAL x, IN REAL y, IN REAL width, IN REAL height)	Uses a brush to fill the interior of a rectangle.
GpStatus WINGDIPAPI GdipFillRectangleI(GpGraphics *graphics, GpBrush *brush, INT x, INT y, INT width, INT height)	Status Graphics::FillRectangle(IN const Brush* brush, IN INT x, IN INT y, IN INT width, IN INT height)	Uses a brush to fill the interior of a rectangle.
GpStatus WINGDIPAPI GdipFillRectangles(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpRectF *rects, INT count)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipFillRectanglesI(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpRect *rects, INT count)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipFillPolygon(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpPointF *points, INT count, GpFillMode fillMode)	Status Graphics::FillPolygon(IN const Brush* brush, IN const PointF* points, IN INT count, IN FillMode fillMode)	Uses a brush to fill the interior of a polygon.
GpStatus WINGDIPAPI GdipFillPolygonI(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpPoint *points, INT count, GpFillMode fillMode)	Status Graphics::FillPolygon(IN const Brush* brush, IN const Point* points, IN INT count, IN FillMode fillMode)	Uses a brush to fill the interior of a polygon.
GpStatus WINGDIPAPI GdipFillPolygon2(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpPointF *points, INT count)	Not called by wrapper methods.	This function fills a polygon with a brush. The <i>points</i> parameter specifies the vertices of the polygon. The <i>count</i> parameter specifies the number of vertices. The <i>brush</i> parameter specifies the brush object used to fill the polygon. The fill mode is FillModeAlternate .
GpStatus WINGDIPAPI GdipFillPolygon2I(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpPoint *points, INT count)	Not called by wrapper methods.	This function fills a polygon with a brush. The <i>points</i> parameter specifies the vertices of the polygon. The <i>count</i> parameter specifies the number of vertices. The <i>brush</i> parameter specifies the brush object used to fill the polygon. The fill mode is FillModeAlternate .
GpStatus WINGDIPAPI GdipFillEllipse(GpGraphics *graphics, GpBrush *brush, REAL x, REAL y, REAL width, REAL height)	Status Graphics::FillEllipse(IN const Brush* brush, IN REAL x, IN REAL y, IN REAL width, IN REAL height)	Uses a brush to fill the interior of an ellipse that is specified by coordinates and dimensions.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipFillEllipse(GpGraphics *graphics, GpBrush *brush, INT x, INT y, INT width, INT height)	Status Graphics::FillEllipse(IN const Brush* brush, IN INT x, IN INT y, IN INT width, IN INT height)	Uses a brush to fill the interior of an ellipse that is specified by coordinates and dimensions.
GpStatus WINGDIPAPI GdipFillPie(GpGraphics *graphics, GpBrush *brush, REAL x, REAL y, REAL width, REAL height, REAL startAngle, REAL sweepAngle)	Status Graphics::FillPie(IN const Brush* brush, IN REAL x, IN REAL y, IN REAL width, IN REAL height, IN REAL startAngle, IN REAL sweepAngle)	Uses a brush to fill the interior of a pie.
GpStatus WINGDIPAPI GdipFillPiel(GpGraphics *graphics, GpBrush *brush, INT x, INT y, INT width, INT height, REAL startAngle, REAL sweepAngle)	Status Graphics:: FillPie(IN const Brush* brush, IN INT x, IN INT y, IN INT width, IN INT height, IN REAL startAngle, IN REAL sweepAngle)	Uses a brush to fill the interior of a pie.
GpStatus WINGDIPAPI GdipFillPath(GpGraphics *graphics, GpBrush *brush, GpPath *path)	Status Graphics::FillPath(IN const Brush* brush, IN const GraphicsPath* path)	Uses a brush to fill the interior of a path. If a figure in the path is not closed, this method treats the nonclosed figure as if it were closed by a straight line that connects the figure's starting and ending points.
GpStatus WINGDIPAPI GdipFillClosedCurve(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpPointF *points, INT count)	Status Graphics::FillClosedCurve(IN const Brush* brush, IN const PointF* points, IN INT count)	Creates a closed cardinal spline from an array of points and uses a brush to fill the interior of the spline.
GpStatus WINGDIPAPI GdipFillClosedCurve1(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpPoint *points, INT count)	Status Graphics::FillClosedCurve(IN const Brush* brush, IN const Point* points, IN INT count)	Creates a closed cardinal spline from an array of points and uses a brush to fill the interior of the spline.
GpStatus WINGDIPAPI GdipFillClosedCurve2(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpPointF *points, INT count, REAL tension, GpFillMode fillMode)	Status Graphics::FillClosedCurve(IN const Brush* brush, IN const PointF* points, IN INT count, IN FillMode fillMode, IN REAL tension = 0.5f)	Creates a closed cardinal spline from an array of points and uses a brush to fill, according to a specified mode, the interior of the spline.
GpStatus WINGDIPAPI GdipFillClosedCurve2I(GpGraphics *graphics, GpBrush *brush, GDIPCONST GpPoint *points, INT count, REAL tension, GpFillMode fillMode)	Status Graphics::FillClosedCurve(IN const Brush* brush, IN const Point* points, IN INT count, IN FillMode fillMode, IN REAL tension = 0.5f)	Creates a closed cardinal spline from an array of points and uses a brush to fill, according to a specified mode, the interior of the spline.
GpStatus WINGDIPAPI GdipFillRegion(GpGraphics *graphics, GpBrush *brush, GpRegion *region)	Status Graphics::FillRegion(IN const Brush* brush, IN const Region* region)	Uses a brush to fill a specified region.
GpStatus WINGDIPAPI GdipDrawImage(GpGraphics *graphics, GpImage *image, REAL x, REAL y)	Status Graphics::DrawImage(IN Image* image, IN REAL x, IN REAL y)	Draws an image at a specified location.
GpStatus WINGDIPAPI GdipDrawImage1(GpGraphics *graphics, GpImage *image, INT x, INT y)	Status Graphics::DrawImage(IN Image* image, IN INT x, IN INT y)	Draws an image at a specified location.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipDrawImageRect(GpGraphics *graphics, GpImage *image, REAL x, REAL y, REAL width, REAL height)	Status Graphics::DrawImage(IN Image* image, IN REAL x, IN REAL y, IN REAL width, IN REAL height)	Draws an image.
GpStatus WINGDIPAPI GdipDrawImageRectI(GpGraphics *graphics, GpImage *image, INT x, INT y, INT width, INT height)	Status Graphics::DrawImage(IN Image* image, IN INT x, IN INT y, IN INT width, IN INT height)	Draws an image.
GpStatus WINGDIPAPI GdipDrawImagePoints(GpGraphics *graphics, GpImage *image, GDIPCONST GpPointF *dstpoints, INT count)	Status Graphics::DrawImage(IN Image* image, IN const PointF* destPoints, IN INT count)	Draws an image.
GpStatus WINGDIPAPI GdipDrawImagePointSI(GpGraphics *graphics, GpImage *image, GDIPCONST GpPoint *dstpoints, INT count)	Status Graphics::DrawImage(IN Image* image, IN const Point* destPoints, IN INT count)	Draws an image.
GpStatus WINGDIPAPI GdipDrawImagePointRect(GpGraphics *graphics, GpImage *image, REAL x, REAL y, REAL srcx, REAL srcy, REAL srcwidth, REAL srcheight, GpUnit srcUnit)	Status Graphics::DrawImage(IN Image* image, IN REAL x, IN REAL y, IN REAL srcx, IN REAL srcy, IN REAL srcwidth, IN REAL srcheight, IN Unit srcUnit)	Draws an image.
GpStatus WINGDIPAPI GdipDrawImagePointRectI(GpGraphics *graphics, GpImage *image, INT x, INT y, INT srcx, INT srcy, INT srcwidth, INT srcheight, GpUnit srcUnit)	Status Graphics::DrawImage(IN Image* image, IN INT x, IN INT y, IN INT srcx, IN INT srcy, IN INT srcwidth, IN INT srcheight, IN Unit srcUnit)	Draws an image.
GpStatus WINGDIPAPI GdipDrawImageRectRect(GpGraphics *graphics, GpImage *image, REAL dstx, REAL dsty, REAL dstwidth, REAL dstheight, REAL srcx, REAL srcy, REAL srcwidth, REAL srcheight, GpUnit srcUnit, GDIPCONST GpImageAttributes* imageAttributes, DrawImageAbort callback, VOID * callbackData)	Status Graphics::DrawImage(IN Image* image, IN const RectF& destRect, IN REAL srcx, IN REAL srcy, IN REAL srcwidth, IN REAL srcheight, IN Unit srcUnit, IN const ImageAttributes* imageAttributes = NULL, IN DrawImageAbort callback = NULL, IN VOID* callbackData = NULL)	Draws an image. In the flat function, the <i>dstx</i> , <i>dsty</i> , <i>dstwidth</i> , and <i>dstheight</i> parameters specify a rectangle that corresponds to the <i>dstRect</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdipDrawImageRectRectI(GpGraphics *graphics, GpImage *image, INT dstx, INT dsty, INT dstwidth, INT dstheight, INT srcx, INT srcy, INT srcwidth, INT srcheight, GpUnit srcUnit, GDIPCONST GpImageAttributes* imageAttributes, DrawImageAbort callback, VOID * callbackData)	Status Graphics::DrawImage(IN Image* image, IN const Rect& destRect, IN INT srcx, IN INT srcy, IN INT srcwidth, IN INT srcheight, IN Unit srcUnit, IN const ImageAttributes* imageAttributes = NULL, IN DrawImageAbort callback = NULL, IN VOID* callbackData = NULL)	Draws an image. In the flat function, the <i>dstx</i> , <i>dsty</i> , <i>dstwidth</i> , and <i>dstheight</i> parameters specify a rectangle that corresponds to the <i>dstRect</i> parameter in the wrapper method.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipDrawImagePointsRect(GpGraphics *graphics, GpImage *image, GDIPCONST GpPointF *points, INT count, REAL srcx, REAL srcy, REAL srcwidth, REAL srcheight, GpUnit srcUnit, GDIPCONST GpImageAttributes* imageAttributes, DrawImageAbort callback, VOID * callbackData)	Status Graphics::DrawImage (IN Image* image, IN const PointF* destPoints, IN INT count, IN REAL srcx, IN REAL srcy, IN REAL srcwidth, IN REAL srcheight, IN Unit srcUnit, IN const ImageAttributes* imageAttributes = NULL, IN DrawImageAbort callback = NULL, IN VOID* callbackData = NULL)	Draws an image.
GpStatus WINGDIPAPI GdipDrawImagePointsRectI(GpGraphics *graphics, GpImage *image, GDIPCONST GpPoint *points, INT count, INT srcx, INT srcy, INT srcwidth, INT srcheight, GpUnit srcUnit, GDIPCONST GpImageAttributes* imageAttributes, DrawImageAbort callback, VOID * callbackData)	Status Graphics::DrawImage (IN Image* image, IN const Point* destPoints, IN INT count, IN INT srcx, IN INT srcy, IN INT srcwidth, IN INT srcheight, IN Unit srcUnit, IN const ImageAttributes* imageAttributes = NULL, IN DrawImageAbort callback = NULL, IN VOID* callbackData = NULL)	Draws an image.
GpStatus WINGDIPAPI GdipDrawImageFX(GpGraphics *graphics, GpImage *image, GpRectF *source, GpMatrix *xForm, CGpEffect *effect, GpImageAttributes *imageAttributes, GpUnit srcUnit)	Status Graphics::DrawImage (IN Image *image, IN RectF *sourceRect, IN Matrix *xForm, IN Effect *effect, IN ImageAttributes *imageAttributes, IN Unit srcUnit)	Draws a portion of an image after applying a specified effect.
GpStatus WINGDIPAPI GdipEnumerateMetafileDestPoint(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST PointF & destPoint, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile (IN const Metafile * metafile, IN const PointF & destPoint, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.
GpStatus WINGDIPAPI GdipEnumerateMetafileDestPointI(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST Point & destPoint, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile (IN const Metafile * metafile, IN const Point & destPoint, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.
GpStatus WINGDIPAPI GdipEnumerateMetafileDestRect(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST RectF & destRect, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile (IN const Metafile * metafile, IN const RectF & destRect, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipEnumerateMetafileDestRectI(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST Rect & destRect, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile(IN const Metafile * metafile, IN const Rect & destRect, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.
GpStatus WINGDIPAPI GdipEnumerateMetafileDestPoints(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST PointF * destPoints, INT count, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipEnumerateMetafileDestPointsI(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST Point * destPoints, INT count, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile(IN const Metafile * metafile, IN const Point * destPoints, IN INT count, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.
GpStatus WINGDIPAPI GdipEnumerateMetafileSrcRectDestPoin t(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST PointF & destPoint, GDIPCONST RectF & srcRect, Unit srcUnit, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipEnumerateMetafileSrcRectDestPoin tI(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST Point & destPoint, GDIPCONST Rect & srcRect, Unit srcUnit, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile(IN const Metafile * metafile, IN const Point & destPoint, IN const Rect & srcRect, IN Unit srcUnit, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.
GpStatus WINGDIPAPI GdipEnumerateMetafileSrcRectDestRect(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST RectF & destRect, GDIPCONST RectF & srcRect, Unit srcUnit, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile(IN const Metafile * metafile, IN const RectF & destRect, IN const RectF & srcRect, IN Unit srcUnit, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipEnumerateMetafileSrcRectDestRect(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST Rect & destRect, GDIPCONST Rect & srcRect, Unit srcUnit, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile(IN const Metafile * metafile, IN const Rect & destRect, IN const Rect & srcRect, IN Unit srcUnit, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.
GpStatus WINGDIPAPI GdipEnumerateMetafileSrcRectDestPoints(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST PointF * destPoints, INT count, GDIPCONST RectF & srcRect, Unit srcUnit, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile(IN const Metafile * metafile, IN const PointF * destPoints, IN INT count, IN const RectF & srcRect, IN Unit srcUnit, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.
GpStatus WINGDIPAPI GdipEnumerateMetafileSrcRectDestPointsI(GpGraphics * graphics, GDIPCONST GpMetafile * metafile, GDIPCONST Point * destPoints, INT count, GDIPCONST Rect & srcRect, Unit srcUnit, EnumerateMetafileProc callback, VOID * callbackData, GDIPCONST GpImageAttributes * imageAttributes)	Status Graphics::EnumerateMetafile(IN const Metafile * metafile, IN const Point * destPoints, IN INT count, IN const Rect & srcRect, IN Unit srcUnit, IN EnumerateMetafileProc callback, IN VOID * callbackData = NULL, IN const ImageAttributes * imageAttributes = NULL)	Calls an application-defined callback function for each record in a specified metafile. You can use this method to display a metafile by calling PlayRecord in the callback function.
GpStatus WINGDIPAPI GdipPlayMetafileRecord(GDIPCONST GpMetafile * metafile, EmfPlusRecordType recordType, UINT flags, UINT dataSize, GDIPCONST BYTE * data)	Status Metafile::PlayRecord(IN EmfPlusRecordType recordType, IN UINT flags, IN UINT dataSize, IN const BYTE * data) const	Plays a metafile record.
GpStatus WINGDIPAPI GdipSetClipGraphics(GpGraphics *graphics, GpGraphics *srcgraphics, CombineMode combineMode)	Status Graphics::SetClip(IN const Graphics* g, IN CombineMode combineMode = CombineModeReplace)	Updates the clipping region of this Graphics object to a region that is the combination of itself and the clipping region of another Graphics object. The <i>g</i> parameter in the wrapper method corresponds to the <i>srcgraphics</i> parameter in the flat function.
GpStatus WINGDIPAPI GdipSetClipRect(GpGraphics *graphics, REAL x, REAL y, REAL width, REAL height, CombineMode combineMode)	Status Graphics::SetClip(IN const RectF& rect, IN CombineMode combineMode = CombineModeReplace)	Updates the clipping region of this Graphics object to a region that is the combination of itself and a rectangle. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters in the flat function specify a rectangle that corresponds to the <i>rect</i> parameter in the wrapper method.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSetClipRectI(GpGraphics *graphics, INT x, INT y, INT width, INT height, CombineMode combineMode)	Status Graphics::SetClip(IN const Rect& rect, IN CombineMode combineMode = CombineModeReplace)	Updates the clipping region of this Graphics object to a region that is the combination of itself and a rectangle. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters in the flat function specify a rectangle that corresponds to the <i>rect</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdipSetClipPath(GpGraphics *graphics, GpPath *path, CombineMode combineMode)	Status Graphics::SetClip(IN const GraphicsPath* path, IN CombineMode combineMode = CombineModeReplace)	Updates the clipping region of this Graphics object to a region that is the combination of itself and the region specified by a graphics path. If a figure in the path is not closed, this method treats the nonclosed figure as if it were closed by a straight line that connects the figure's starting and ending points.
GpStatus WINGDIPAPI GdipSetClipRegion(GpGraphics *graphics, GpRegion *region, CombineMode combineMode)	Status Graphics::SetClip(IN const Region* region, IN CombineMode combineMode = CombineModeReplace)	Updates the clipping region of this Graphics object to a region that is the combination of itself and the region specified by a Region object.
GpStatus WINGDIPAPI GdipSetClipHrgn(GpGraphics *graphics, HRGN hRgn, CombineMode combineMode)	Status Graphics::SetClip(IN HRGN hRgn, IN CombineMode combineMode = CombineModeReplace)	Updates the clipping region of this Graphics object to a region that is the combination of itself and a Windows Graphics Device Interface (GDI) region
GpStatus WINGDIPAPI GdipResetClip(GpGraphics *graphics)	Status Graphics::ResetClip()	Sets the clipping region of this Graphics object to an infinite region.
GpStatus WINGDIPAPI GdipTranslateClip(GpGraphics *graphics, REAL dx, REAL dy)	Status Graphics::TranslateClip(IN REAL dx, IN REAL dy)	Translates the clipping region of this Graphics object.
GpStatus WINGDIPAPI GdipTranslateClipI(GpGraphics *graphics, INT dx, INT dy)	Status Graphics::TranslateClip(IN INT dx, IN INT dy)	Translates the clipping region of this Graphics object.
GpStatus WINGDIPAPI GdipGetClip(GpGraphics *graphics, GpRegion *region)	Status Graphics::GetClip(OUT Region* region) const	Gets the clipping region of this Graphics object.
GpStatus WINGDIPAPI GdipGetClipBounds(GpGraphics *graphics, GpRectF *rect)	Status Graphics::GetClipBounds(OUT RectF* rect) const	Gets a rectangle that encloses the clipping region of this Graphics object.
GpStatus WINGDIPAPI GdipGetClipBoundsI(GpGraphics *graphics, GpRect *rect)	Status Graphics::GetClipBounds(OUT Rect* rect) const	Gets a rectangle that encloses the clipping region of this Graphics object.
GpStatus WINGDIPAPI GdipIsClipEmpty(GpGraphics *graphics, BOOL *result)	BOOL Graphics::IsClipEmpty() const	Determines whether the clipping region of this Graphics object is empty.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetVisibleClipBounds(GpGraphics *graphics, GpRectF *rect)	Status Graphics::GetVisibleClipBounds(OUT RectF *rect) const	Gets a rectangle that encloses the visible clipping region of this Graphics object. The visible clipping region is the intersection of the clipping region of this Graphics object and the clipping region of the window.
GpStatus WINGDIPAPI GdipGetVisibleClipBoundsI(GpGraphics *graphics, GpRect *rect)	Status Graphics::GetVisibleClipBounds(OUT Rect *rect) const	Gets a rectangle that encloses the visible clipping region of this Graphics object. The visible clipping region is the intersection of the clipping region of this Graphics object and the clipping region of the window.
GpStatus WINGDIPAPI GdiplIsVisibleClipEmpty(GpGraphics *graphics, BOOL *result)	BOOL Graphics::IsVisibleClipEmpty() const	Determines whether the visible clipping region of this Graphics object is empty. The visible clipping region is the intersection of the clipping region of this Graphics object and the clipping region of the window.
GpStatus WINGDIPAPI GdiplIsVisiblePoint(GpGraphics *graphics, REAL x, REAL y, BOOL *result)	BOOL Graphics::IsVisible(IN const PointF& point) const	Determines whether the specified point is inside the visible clipping region of this Graphics object. The visible clipping region is the intersection of the clipping region of this Graphics object and the clipping region of the window. The <i>x</i> and <i>y</i> parameters in the flat function represent the <i>x</i> and <i>y</i> coordinates of a point that corresponds to the <i>point</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdiplIsVisiblePointI(GpGraphics *graphics, INT x, INT y, BOOL *result)	BOOL Graphics::IsVisible(IN const Point& point) const	Determines whether the specified point is inside the visible clipping region of this Graphics object. The visible clipping region is the intersection of the clipping region of this Graphics object and the clipping region of the window. The <i>x</i> and <i>y</i> parameters in the flat function represent the <i>x</i> and <i>y</i> coordinates of a point that corresponds to the <i>point</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdiplIsVisibleRect(GpGraphics *graphics, REAL x, REAL y, REAL width, REAL height, BOOL *result)	BOOL Graphics::IsVisible(IN const RectF& rect) const	Determines whether the specified rectangle intersects the visible clipping region of this Graphics object. The visible clipping region is the intersection of the clipping region of this Graphics object and the clipping region of the window. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters in the flat function specify a rectangle that corresponds to the <i>rect</i> parameter in the wrapper method.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdiplIsVisibleRect(GpGraphics *graphics, INT x, INT y, INT width, INT height, BOOL *result)	BOOL Graphics::IsVisible(IN const Rect& rect) const	Determines whether the specified rectangle intersects the visible clipping region of this Graphics object. The visible clipping region is the intersection of the clipping region of this Graphics object and the clipping region of the window. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters in the flat function specify a rectangle that corresponds to the <i>rect</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdipSaveGraphics(GpGraphics *graphics, GraphicsState *state)	GraphicsState Graphics::Save() const	Saves the current state (transformations, clipping region, and quality settings) of this Graphics object. You can restore the state later by calling the Graphics::Restore method.
GpStatus WINGDIPAPI GdipRestoreGraphics(GpGraphics *graphics, GraphicsState state)	Status Graphics::Restore(IN GraphicsState gstate)	Sets the state of this Graphics object to the state stored by a previous call to the Graphics::Save method of this Graphics object.
GpStatus WINGDIPAPI GdipBeginContainer(GpGraphics *graphics, GDIPCONST GpRectF* dstrect, GDIPCONST GpRectF *srcrect, GpUnit unit, GraphicsContainer *state)	Graphics::GraphicsContainer BeginContainer(IN const RectF &dstrect, IN const RectF &srcrect, IN Unit unit)	Begins a new graphics container.
GpStatus WINGDIPAPI GdipBeginContainerI(GpGraphics *graphics, GDIPCONST GpRect* dstrect, GDIPCONST GpRect *srcrect, GpUnit unit, GraphicsContainer *state)	Graphics::GraphicsContainer BeginContainer(IN const Rect &dstrect, IN const Rect &srcrect, IN Unit unit)	Begins a new graphics container.
GpStatus WINGDIPAPI GdipBeginContainer2(GpGraphics *graphics, GraphicsContainer* state)	Graphics::GraphicsContainer BeginContainer()	Begins a new graphics container.
GpStatus WINGDIPAPI GdipEndContainer(GpGraphics *graphics, GraphicsContainer state)	Status Graphics::EndContainer(IN GraphicsContainer state)	Closes a graphics container that was previously opened by the Graphics::BeginContainer method.
GpStatus WINGDIPAPI GdipGetMetafileHeaderFromEmf(HENHMETAFILE hEmf, MetafileHeader * header)	static Status Metafile::GetMetafileHeader(IN HENHMETAFILE hEmf, OUT MetafileHeader * header)	Gets the header.
GpStatus WINGDIPAPI GdipGetMetafileHeaderFromFile(GDIPCONST WCHAR* filename, MetafileHeader * header)	static Status Metafile::GetMetafileHeader(IN const WCHAR* filename, OUT MetafileHeader * header)	Gets the header.
GpStatus WINGDIPAPI GdipGetMetafileHeaderFromStream(IStream * stream, MetafileHeader * header)	static Status Metafile::GetMetafileHeader(IN IStream * stream, OUT MetafileHeader * header)	Gets the header.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetMetafileHeaderFromMetafile(GpMetafile * metafile, MetafileHeader * header)	Status Metafile::GetMetafileHeader(OUT MetafileHeader * header) const	Gets the header.
GpStatus WINGDIPAPI GdipGetHemfFromMetafile(GpMetafile * metafile, HENHMETAFILE * hEmf)	HENHMETAFILE Metafile::GetHENHMETAFILE()	Gets a Windows handle to an Enhanced Metafile (EMF) file.
GpStatus WINGDIPAPI GdipCreateStreamOnFile(GDIPCONST WCHAR * filename, UINT access, IStream **stream)	Not called by wrapper methods.	Returns a pointer to an IStream interface based on a file. The filename parameter specifies the file. The access parameter is a set of flags that must include GENERIC_READ or GENERIC_WRITE. The stream parameter receives a pointer to the IStream interface.
GpStatus WINGDIPAPI GdipCreateMetafileFromWmf(HMETAFILE hWmf, BOOL deleteWmf, GDIPCONST WmfPlaceableFileHeader * wmfPlaceableFileHeader, GpMetafile **metafile)	Metafile::Metafile(IN HMETAFILE hWmf, IN const WmfPlaceableFileHeader * wmfPlaceableFileHeader, IN BOOL deleteWmf = FALSE)	Creates a Windows GDI+ Metafile::Metafile object for recording. The format will be placeable metafile.
GpStatus WINGDIPAPI GdipCreateMetafileFromEmf(HENHMETFILE hEmf, BOOL deleteEmf, GpMetafile **metafile)	Metafile::Metafile(IN HENHMETAFILE hEmf, IN BOOL deleteEmf = FALSE)	Creates a Windows GDI+ Metafile::Metafile object for playback based on a Windows Graphics Device Interface (GDI) Enhanced Metafile (EMF) file.
GpStatus WINGDIPAPI GdipCreateMetafileFromFile(GDIPCONST WCHAR* file, GpMetafile **metafile)	Metafile::Metafile(IN const WCHAR* filename)	Creates a Metafile::Metafile object for playback.
GpStatus WINGDIPAPI GdipCreateMetafileFromWmfFile(GDIPCONST WCHAR* file, GDIPCONST WmfPlaceableFileHeader * wmfPlaceableFileHeader, GpMetafile **metafile)	Not called by wrapper methods.	Not implemented.
GpStatus WINGDIPAPI GdipCreateMetafileFromStream(IStream * stream, GpMetafile **metafile)	Metafile::Metafile(IN IStream* stream)	Creates a Metafile::Metafile object from an IStream interface for playback.
GpStatus WINGDIPAPI GdipRecordMetafile(HDC referenceHdc, EmfType type, GDIPCONST GpRectF * frameRect, MetafileFrameUnit frameUnit, GDIPCONST WCHAR * description, GpMetafile ** metafile)	Metafile::Metafile(IN HDC referenceHdc, IN const RectF & frameRect, IN MetafileFrameUnit frameUnit = MetafileFrameUnitGdi, IN EmfType type = EmfTypeEmfPlusDual, IN const WCHAR * description = NULL)	Creates a Metafile::Metafile object for recording.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipRecordMetafile(HDC referenceHdc, EmfType type, GDIPCONST GpRect * frameRect, MetafileFrameUnit frameUnit, GDIPCONST WCHAR * description, GpMetafile ** metafile)	Metafile::Metafile(IN HDC referenceHdc, IN const Rect & frameRect, IN MetafileFrameUnit frameUnit = MetafileFrameUnitGdi, IN EmfType type = EmfTypeEmfPlusDual, IN const WCHAR * description = NULL)	Creates a Metafile::Metafile object for recording.
GpStatus WINGDIPAPI GdipRecordMetafileFileName(GDIPCONST WCHAR* fileName, HDC referenceHdc, EmfType type, GDIPCONST GpRectF * frameRect, MetafileFrameUnit frameUnit, GDIPCONST WCHAR * description, GpMetafile ** metafile)	Metafile::Metafile(IN const WCHAR* fileName, IN HDC referenceHdc, IN const RectF & frameRect, IN MetafileFrameUnit frameUnit = MetafileFrameUnitGdi, IN EmfType type = EmfTypeEmfPlusDual, IN const WCHAR * description = NULL)	Creates a Metafile::Metafile object for recording.
GpStatus WINGDIPAPI GdipRecordMetafileFileName(GDIPCONST WCHAR* fileName, HDC referenceHdc, EmfType type, GDIPCONST GpRect * frameRect, MetafileFrameUnit frameUnit, GDIPCONST WCHAR * description, GpMetafile ** metafile)	Metafile::Metafile(IN const WCHAR* fileName, IN HDC referenceHdc, IN const Rect & frameRect, IN MetafileFrameUnit frameUnit = MetafileFrameUnitGdi, IN EmfType type = EmfTypeEmfPlusDual, IN const WCHAR * description = NULL)	Creates a Metafile::Metafile object for recording.
GpStatus WINGDIPAPI GdipRecordMetafileStream(IStream * stream, HDC referenceHdc, EmfType type, GDIPCONST GpRectF * frameRect, MetafileFrameUnit frameUnit, GDIPCONST WCHAR * description, GpMetafile ** metafile)	Metafile::Metafile(IN IStream * stream, IN HDC referenceHdc, IN const RectF & frameRect, IN MetafileFrameUnit frameUnit = MetafileFrameUnitGdi, IN EmfType type = EmfTypeEmfPlusDual, IN const WCHAR * description = NULL)	Creates a Metafile::Metafile object for recording to an IStream interface.
GpStatus WINGDIPAPI GdipRecordMetafileStreamI(IStream * stream, HDC referenceHdc, EmfType type, GDIPCONST GpRect * frameRect, MetafileFrameUnit frameUnit, GDIPCONST WCHAR * description, GpMetafile ** metafile)	Metafile::Metafile(IN IStream * stream, IN HDC referenceHdc, IN const Rect & frameRect, IN MetafileFrameUnit frameUnit = MetafileFrameUnitGdi, IN EmfType type = EmfTypeEmfPlusDual, IN const WCHAR * description = NULL)	Creates a Metafile::Metafile object for recording to an IStream interface.
GpStatus WINGDIPAPI GdipSetMetafileDownLevelRasterizationLimit(GpMetafile * metafile, UINT metafileRasterizationLimitDpi)	Status Metafile::SetDownLevelRasterizationLimit(IN UINT metafileRasterizationLimitDpi)	Sets the resolution for certain brush bitmaps that are stored in this metafile.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetMetafileDownLevelRasterizationLimit(GDIPCONST GpMetafile * metafile, UINT * metafileRasterizationLimitDpi)	UINT Metafile::GetDownLevelRasterizationLimit() const	Gets the rasterization limit currently set for this metafile. The rasterization limit is the resolution used for certain brush bitmaps that are stored in the metafile. For a detailed explanation of the rasterization limit, see Metafile::SetDownLevelRasterizationLimit .
GpStatus WINGDIPAPI GdipGetImageDecodersSize(UINT *numDecoders, UINT *size)	Status GetImageDecodersSize(OUT UINT *numDecoders, OUT UINT *size)	Gets the number of available image decoders and the total size of the array of ImageCodecInfo objects that is returned by the GetImageDecoders function.
GpStatus WINGDIPAPI GdipGetImageDecoders(UINT numDecoders, UINT size, ImageCodecInfo *decoders)	Status GetImageDecoders(IN UINT numDecoders, IN UINT size, OUT ImageCodecInfo *decoders)	Gets an array of ImageCodecInfo objects that contain information about the available image decoders.
GpStatus WINGDIPAPI GdipGetImageEncodersSize(UINT *numEncoders, UINT *size)	Status GetImageEncodersSize(OUT UINT *numEncoders, OUT UINT *size)	Gets the number of available image encoders and the total size of the array of ImageCodecInfo objects that is returned by the GetImageEncoders function.
GpStatus WINGDIPAPI GdipGetImageEncoders(UINT numEncoders, UINT size, ImageCodecInfo *encoders)	Status GetImageEncoders(IN UINT numEncoders, IN UINT size, OUT ImageCodecInfo *encoders)	Gets an array of ImageCodecInfo objects that contain information about the available image encoders.
GpStatus WINGDIPAPI GdipComment(GpGraphics* graphics, UINT sizeData, GDIPCONST BYTE * data)	Status Graphics::AddMetafileComment(IN const BYTE * data, IN UINT sizeData)	Adds a text comment to an existing metafile.

GraphicsPath Functions

11/2/2020 • 12 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the **GraphicsPath** C++ class.

GraphicsPath Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreatePath(GpFillMode brushMode, GpPath **path)	GraphicsPath::GraphicsPath(IN FillMode fillMode = FillModeAlternate)	Creates a GraphicsPath object and initializes the fill mode. This is the default constructor.
GpStatus WINGDIPAPI GdipCreatePath2(GDIPCONST GpPointF* points, GDIPCONST BYTE* types, INT count, GpFillMode fillMode, GpPath **path)	GraphicsPath::GraphicsPath(IN const PointF* points, IN const BYTE* types, IN INT count, IN FillMode fillMode = FillModeAlternate)	Creates a GraphicsPath object based on an array of points, an array of types, and a fill mode.
GpStatus WINGDIPAPI GdipCreatePath2I(GDIPCONST GpPoint* points, GDIPCONST BYTE* types, INT count, GpFillMode fillMode, GpPath **path)	GraphicsPath::GraphicsPath(IN const Point* points, IN const BYTE* types, IN INT count, IN FillMode fillMode = FillModeAlternate)	Creates a GraphicsPath object based on an array of points, an array of types, and a fill mode.
GpStatus WINGDIPAPI GdipClonePath(GpPath* path, GpPath **clonePath)	GraphicsPath::GraphicsPath* Clone() const	Creates a new GraphicsPath object, and initializes it with the contents of this GraphicsPath object.
GpStatus WINGDIPAPI GdipDeletePath(GpPath* path)	GraphicsPath::~GraphicsPath()	Releases resources used by the GraphicsPath object.
GpStatus WINGDIPAPI GdipResetPath(GpPath* path)	GraphicsPath::Reset	Empties the path and sets the fill mode to FillModeAlternate.
GpStatus WINGDIPAPI GdipGetPointCount(GpPath* path, INT* count)	INT GraphicsPath::GetPointCount	Gets the number of points in this path's array of data points. This is the same as the number of types in the path's array of point types.
GpStatus WINGDIPAPI GdipGetPathTypes(GpPath* path, BYTE* types, INT count)	Status GraphicsPath::GetPathTypes(OUT BYTE* types, IN INT count) const	Gets this path's array of point types.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetPathPoints(GpPath*, GpPointF* points, INT count)	Status GraphicsPath::GetPathPoints(OUT PointF* points, IN INT count) const	Gets this path's array of points. The array contains the endpoints and control points of the lines and Bézier splines that are used to draw the path.
GpStatus WINGDIPAPI GdipGetPathPointsl(GpPath*, GpPoint* points, INT count)	Status GraphicsPath::GetPathPoints(OUT Point* points, IN INT count) const	Gets this path's array of points. The array contains the endpoints and control points of the lines and Bézier splines that are used to draw the path.
GpStatus WINGDIPAPI GdipGetPathFillMode(GpPath *path, GpFillMode *fillmode)	FillMode GraphicsPath::GetFillMode() const	Gets the fill mode of this path.
GpStatus WINGDIPAPI GdipSetPathFillMode(GpPath *path, GpFillMode fillmode)	Status GraphicsPath::SetFillMode(IN FillMode fillmode)	Sets the fill mode of this path.
GpStatus WINGDIPAPI GdipGetPathData(GpPath *path, GpPathData* pathData)	Status GraphicsPath::GetPathData(OUT PathData* pathData) const	Gets an array of points and an array of point types from this path. Together, these two arrays define the lines, curves, figures, and markers of this path.
GpStatus WINGDIPAPI GdipStartPathFigure(GpPath *path)	Status GraphicsPath::StartFigure()	Starts a new figure without closing the current figure. Subsequent points added to this path are added to the new figure.
GpStatus WINGDIPAPI GdipClosePathFigure(GpPath *path)	Status GraphicsPath::CloseFigure()	Closes the current figure of this path.
GpStatus WINGDIPAPI GdipClosePathFigures(GpPath *path)	Status GraphicsPath::CloseAllFigures()	Closes all open figures in this path.
GpStatus WINGDIPAPI GdipSetPathMarker(GpPath* path)	Status GraphicsPath::SetMarker()	Designates the last point in this path as a marker point.
GpStatus WINGDIPAPI GdipClearPathMarkers(GpPath* path)	Status GraphicsPath::ClearMarkers()	Clears the markers from this path.
GpStatus WINGDIPAPI GdipReversePath(GpPath* path)	Status GraphicsPath::Reverse()	Reverses the order of the points that define this path's lines and curves.
GpStatus WINGDIPAPI GdipGetPathLastPoint(GpPath* path, GpPointF* lastPoint)	Status GraphicsPath::GetLastPoint(OUT PointF* lastPoint) const	Gets the ending point of the last figure in this path.
GpStatus WINGDIPAPI GdipAddPathLine(GpPath *path, REAL x1, REAL y1, REAL x2, REAL y2)	Status GraphicsPath::AddLine(IN REAL x1, IN REAL y1, IN REAL x2, IN REAL y2)	Adds a line to the current figure of this path.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipAddPathLine2(GpPath *path, GDIPCONST GpPointF *points, INT count)	Status GraphicsPath::AddLines(IN const PointF* points, IN INT count)	Adds a sequence of connected lines to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathArc(GpPath *path, REAL x, REAL y, REAL width, REAL height, REAL startAngle, REAL sweepAngle)	Status GraphicsPath::AddArc(IN REAL x, IN REAL y, IN REAL width, IN REAL height, IN REAL startAngle, IN REAL sweepAngle)	Adds an elliptical arc to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathBezier(GpPath *path, REAL x1, REAL y1, REAL x2, REAL y2, REAL x3, REAL y3, REAL x4, REAL y4)	Status GraphicsPath::AddBezier(IN REAL x1, IN REAL y1, IN REAL x2, IN REAL y2, IN REAL x3, IN REAL y3, IN REAL x4, IN REAL y4)	Adds a Bézier spline to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathBeziers(GpPath *path, GDIPCONST GpPointF *points, INT count)	Status GraphicsPath::AddBeziers(IN const PointF* points, IN INT count)	Adds a sequence of connected Bézier splines to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathCurve(GpPath *path, GDIPCONST GpPointF *points, INT count)	Status GraphicsPath::AddCurve(IN const PointF* points, IN INT count)	Adds a cardinal spline to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathCurve2(GpPath *path, GDIPCONST GpPointF *points, INT count, REAL tension)	Status GraphicsPath::AddCurve(IN const PointF* points, IN INT count, IN REAL tension)	Adds a cardinal spline to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathCurve3(GpPath *path, GDIPCONST GpPointF *points, INT count, INT offset, INT numberOfSegments, REAL tension)	Status GraphicsPath::AddCurve(IN const PointF* points, IN INT count, IN INT offset, IN INT numberOfSegments, IN REAL tension)	Adds a cardinal spline to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathClosedCurve(GpPath *path, GDIPCONST GpPointF *points, INT count)	Status GraphicsPath::AddClosedCurve(IN const PointF* points, IN INT count)	Adds a closed cardinal spline to this path.
GpStatus WINGDIPAPI GdipAddPathClosedCurve2(GpPath *path, GDIPCONST GpPointF *points, INT count, REAL tension)	Status GraphicsPath::AddClosedCurve(IN const PointF* points, IN INT count, IN REAL tension)	Adds a closed cardinal spline to this path.
GpStatus WINGDIPAPI GdipAddPathRectangle(GpPath *path, REAL x, REAL y, REAL width, REAL height)	Status GraphicsPath::AddRectangle(IN const RectF& rect)	Adds a rectangle to this path. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters in the flat function specify a rectangle that corresponds to the <i>rect</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdipAddPathRectangles(GpPath *path, GDIPCONST GpRectF *rects, INT count)	Status GraphicsPath::AddRectangles(IN const RectF* rects, IN INT count)	Adds a sequence of rectangles to this path.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipAddPathEllipse(GpPath *path, REAL x, REAL y, REAL width, REAL height)	Status GraphicsPath::AddEllipse(IN REAL x, IN REAL y, IN REAL width, IN REAL height)	Adds an ellipse to this path.
GpStatus WINGDIPAPI GdipAddPathPie(GpPath *path, REAL x, REAL y, REAL width, REAL height, REAL startAngle, REAL sweepAngle)	Status GraphicsPath::AddPie(IN REAL x, IN REAL y, IN REAL width, IN REAL height, IN REAL startAngle, IN REAL sweepAngle)	Adds a pie to this path. An arc is a portion of an ellipse, and a pie is a portion of the area enclosed by an ellipse. A pie is bounded by an arc and two lines (edges) that go from the center of the ellipse to the endpoints of the arc.
GpStatus WINGDIPAPI GdipAddPathPolygon(GpPath *path, GDIPCONST GpPointF *points, INT count)	Status GraphicsPath::AddPolygon(IN const PointF* points, IN INT count)	Adds a polygon to this path.
GpStatus WINGDIPAPI GdipAddPathPath(GpPath *path, GDIPCONST GpPath* addingPath, BOOL connect)	Status GraphicsPath::AddPath(IN const GraphicsPath* addingPath, IN BOOL connect)	Adds a path to this path.
GpStatus WINGDIPAPI GdipAddPathString(GpPath *path, GDIPCONST WCHAR *string, INT length, GDIPCONST GpFontFamily *family, INT style, REAL emSize, GDIPCONST RectF *layoutRect, GDIPCONST GpStringFormat *format)	Status GraphicsPath::AddString(IN const WCHAR *string, IN INT length, IN const FontFamily *family, IN INT style, IN REAL emSize, IN const RectF &layoutRect, IN const StringFormat *format)	Adds the outline of a string to this path.
GpStatus WINGDIPAPI GdipAddPathStringI(GpPath *path, GDIPCONST WCHAR *string, INT length, GDIPCONST GpFontFamily *family, INT style, REAL emSize, GDIPCONST Rect *layoutRect, GDIPCONST GpStringFormat *format)	Status GraphicsPath::AddString(IN const WCHAR *string, IN INT length, IN const FontFamily *family, IN INT style, IN REAL emSize, IN const Rect &layoutRect, IN const StringFormat *format)	Adds the outline of a string to this path.
GpStatus WINGDIPAPI GdipAddPathLineI(GpPath *path, INT x1, INT y1, INT x2, INT y2)	Status GraphicsPath::AddLine(IN INT x1, IN INT y1, IN INT x2, IN INT y2)	Adds a line to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathLine2I(GpPath *path, GDIPCONST GpPoint *points, INT count)	Status GraphicsPath::AddLines(IN const Point* points, IN INT count)	Adds a sequence of connected lines to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathArcI(GpPath *path, INT x, INT y, INT width, INT height, REAL startAngle, REAL sweepAngle)	Status GraphicsPath::AddArc(IN INT x, IN INT y, IN INT width, IN INT height, IN REAL startAngle, IN REAL sweepAngle)	Adds an elliptical arc to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathBezierI(GpPath *path, INT x1, INT y1, INT x2, INT y2, INT x3, INT y3, INT x4, INT y4)	Status GraphicsPath::AddBezier(IN INT x1, IN INT y1, IN INT x2, IN INT y2, IN INT x3, IN INT y3, IN INT x4, IN INT y4)	Adds a Bézier spline to the current figure of this path.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipAddPathBeziersI(GpPath *path, GDIPCONST GpPoint *points, INT count)	Status GraphicsPath::AddBeziers(IN const Point* points, IN INT count)	Adds a Bézier spline to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathCurvel(GpPath *path, GDIPCONST GpPoint *points, INT count)	Status GraphicsPath::AddCurve(IN const Point* points, IN INT count)	Adds a cardinal spline to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathCurve2I(GpPath *path, GDIPCONST GpPoint *points, INT count, REAL tension)	Status GraphicsPath::AddCurve(IN const Point* points, IN INT count, IN REAL tension)	Adds a cardinal spline to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathCurve3I(GpPath *path, GDIPCONST GpPoint *points, INT count, INT offset, INT numberOfSegments, REAL tension)	Status GraphicsPath::AddCurve(IN const Point* points, IN INT count, IN INT offset, IN INT numberOfSegments, IN REAL tension)	Adds a cardinal spline to the current figure of this path.
GpStatus WINGDIPAPI GdipAddPathClosedCurvel(GpPath *path, GDIPCONST GpPoint *points, INT count)	Status GraphicsPath::AddClosedCurve(IN const Point* points, IN INT count)	Adds a closed cardinal spline to this path.
GpStatus WINGDIPAPI GdipAddPathClosedCurve2I(GpPath *path, GDIPCONST GpPoint *points, INT count, REAL tension)	Status GraphicsPath::AddClosedCurve(IN const Point* points, IN INT count, IN REAL tension)	Adds a closed cardinal spline to this path.
GpStatus WINGDIPAPI GdipAddPathRectangleI(GpPath *path, INT x, INT y, INT width, INT height)	Status GraphicsPath::AddRectangle(IN const Rect& rect)	Adds a rectangle to this path. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters in the flat function specify a rectangle that corresponds to the <i>rect</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdipAddPathRectanglesI(GpPath *path, GDIPCONST GpRect *rects, INT count)	Status GraphicsPath::AddRectangles(IN const Rect* rects, INT count)	Adds a sequence of rectangles to this path
GpStatus WINGDIPAPI GdipAddPathEllipsel(GpPath *path, INT x, INT y, INT width, INT height)	Status GraphicsPath::AddEllipse(IN INT x, IN INT y, IN INT width, IN INT height)	Adds an ellipse to this path.
GpStatus WINGDIPAPI GdipAddPathPieI(GpPath *path, INT x, INT y, INT width, INT height, REAL startAngle, REAL sweepAngle)	Status GraphicsPath::AddPie(IN INT x, IN INT y, IN INT width, IN INT height, IN REAL startAngle, IN REAL sweepAngle)	Adds a pie to this path. An arc is a portion of an ellipse, and a pie is a portion of the area enclosed by an ellipse. A pie is bounded by an arc and two lines (edges) that go from the center of the ellipse to the endpoints of the arc.
GpStatus WINGDIPAPI GdipAddPathPolygonI(GpPath *path, GDIPCONST GpPoint *points, INT count)	Status GraphicsPath::AddPolygon(IN const Point* points, IN INT count)	Adds a polygon to this path.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipFlattenPath(GpPath *path, GpMatrix* matrix, REAL flatness)	Status GraphicsPath::Flatten(IN const Matrix* matrix = NULL, IN REAL flatness = FlatnessDefault)	Applies a transformation to this path and converts each curve in the path to a sequence of connected lines.
GpStatus WINGDIPAPI GdipWindingModeOutline(GpPath *path, GpMatrix *matrix, REAL flatness)	Status GraphicsPath::Outline(IN const Matrix *matrix = NULL, IN REAL flatness = FlatnessDefault)	Transforms and flattens this path, and then converts this path's data points so that they represent only the outline of the path.
GpStatus WINGDIPAPI GdipWidenPath(GpPath *nativePath, GpPen *pen, GpMatrix *matrix, REAL flatness)	Status GraphicsPath::Widen(IN const Pen* pen, IN const Matrix* matrix = NULL, IN REAL flatness = FlatnessDefault)	Replaces this path with curves that enclose the area that is filled when this path is drawn by a specified pen. This method also flattens the path.
GpStatus WINGDIPAPI GdipWarpPath(GpPath *path, GpMatrix* matrix, GDIPCONST GpPointF *points, INT count, REAL srcx, REAL srcy, REAL srcwidth, REAL srcheight, WarpMode warpMode, REAL flatness)	Status GraphicsPath::Warp(IN const PointF* destPoints, IN INT count, IN const RectF& srcRect, IN const Matrix* matrix = NULL, IN WarpMode warpMode = WarpModePerspective, IN REAL flatness = FlatnessDefault)	Applies a warp transformation to this path. This method also flattens (converts to a sequence of straight lines) the path. The <i>srcx</i> , <i>srcy</i> , <i>srcwidth</i> , and <i>srcheight</i> parameters in the flat function specify a rectangle that corresponds to the <i>srcRect</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdipTransformPath(GpPath* path, GpMatrix* matrix)	Status GraphicsPath::Transform(IN const Matrix* matrix)	Multiplies each of this path's data points by a specified matrix.
GpStatus WINGDIPAPI GdipGetPathWorldBounds(GpPath* path, GpRectF* bounds, GDIPCONST GpMatrix *matrix, GDIPCONST GpPen *pen)	Status GraphicsPath::GetBounds(OUT RectF* bounds, IN const Matrix* matrix, IN const Pen* pen) const	Gets a bounding rectangle for this path.
GpStatus WINGDIPAPI GdipGetPathWorldBoundsI(GpPath* path, GpRect* bounds, GDIPCONST GpMatrix *matrix, GDIPCONST GpPen *pen)	Status GraphicsPath::GetBounds(OUT Rect* bounds, IN const Matrix* matrix, IN const Pen* pen) const	Gets a bounding rectangle for this path.
GpStatus WINGDIPAPI GdipIsVisiblePathPoint(GpPath* path, REAL x, REAL y, GpGraphics *graphics, BOOL *result)	BOOL GraphicsPath::IsVisible(IN REAL x, IN REAL y, IN const Graphics* g) const	Determines whether a specified point lies in the area that is filled when this path is filled by a specified Graphics object.
GpStatus WINGDIPAPI GdipIsVisiblePathPointI(GpPath* path, INT x, INT y, GpGraphics *graphics, BOOL *result)	BOOL GraphicsPath::IsVisible(IN INT x, IN INT y, IN const Graphics* g) const	Determines whether a specified point lies in the area that is filled when this path is filled by a specified Graphics object.
GpStatus WINGDIPAPI GdipIsOutlineVisiblePathPoint(GpPath* path, REAL x, REAL y, GpPen *pen, GpGraphics *graphics, BOOL *result)	BOOL GraphicsPath::IsOutlineVisible(IN REAL x, IN REAL y, IN const Pen* pen, IN const Graphics* g) const	Determines whether a specified point touches the outline of this path when the path is drawn by a specified Graphics object and a specified pen.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdiplIsOutlineVisiblePathPointI(GpPath* path, INT x, INT y, GpPen *pen, GpGraphics *graphics, BOOL *result)	BOOL GraphicsPath::IsOutlineVisible(IN INT x, IN INT y, IN const Pen* pen, IN const Graphics* g) const	Determines whether a specified point touches the outline of this path when the path is drawn by a specified Graphics object and a specified pen.

HatchBrush Functions

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the **HatchBrush** C++ class.

HatchBrush Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreateHatchBrush(GpHatchStyle hatchstyle, ARGB forecol, ARGB backcol, GpHatch **brush)	HatchBrush::HatchBrush(IN HatchStyle hatchStyle, IN const Color& foreColor, IN const Color& backColor = Color())	Creates a HatchBrush object based on a hatch style, a foreground color, and a background color.
GpStatus WINGDIPAPI GdipGetHatchStyle(GpHatch *brush, GpHatchStyle *hatchstyle)	HatchStyle HatchBrush::GetHatchStyle() const	Gets the hatch style of this hatch brush.
GpStatus WINGDIPAPI GdipGetHatchForegroundColor(GpHatch *brush, ARGB* forecol)	Status HatchBrush::GetForegroundColor(OUT Color* color) const	Gets the foreground color of this hatch brush.
GpStatus WINGDIPAPI GdipGetHatchBackgroundColor(GpHatch *brush, ARGB* backcol)	Status HatchBrush::GetBackgroundColor(OUT Color *color) const	Gets the background color of this hatch brush.

Image Functions

11/2/2020 • 7 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [Image](#) C++ class.

HatchBrush Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipLoadImageFromStream(IStream* stream, GpImage **image)	Image::Image(IN IStream* stream, IN BOOL useEmbeddedColorManagement)	Creates an Image object based on a stream. This flat function does not use Image Color Management (ICM).
GpStatus WINGDIPAPI GdipLoadImageFromFile(GDIPCONST WCHAR* filename, GpImage **image)	Image::Image(IN const WCHAR* filename, IN BOOL useEmbeddedColorManagement)	Creates an Image object based on a file. This flat function does not use ICM.
GpStatus WINGDIPAPI GdipLoadImageFromStreamICM(IStream* stream, GpImage **image)	Image::Image(IN IStream* stream, IN BOOL useEmbeddedColorManagement)	Creates an Image object based on a stream. This flat function does not use ICM.
GpStatus WINGDIPAPI GdipLoadImageFromFileICM(GDIPCONST WCHAR* filename, GpImage **image)	Image::Image(IN const WCHAR* filename, IN BOOL useEmbeddedColorManagement)	Creates an Image object based on a file. This flat function does not use ICM.
GpStatus WINGDIPAPI GdipCloneImage(GpImage *image, GpImage **cloneImage)	Image* Image::Clone()	Creates a new Image object and initializes it with the contents of this Image object.
GpStatus WINGDIPAPI GdipDisposeImage(GpImage *image)	Image::~Image()	Releases resources used by the Image object.
GpStatus WINGDIPAPI GdipSaveImageToFile(GpImage *image, GDIPCONST WCHAR* filename, GDIPCONST CLSID* clsidEncoder, GDIPCONST EncoderParameters* encoderParams)	Status Image::Save(IN const WCHAR* filename, IN const CLSID* clsidEncoder, IN const EncoderParameters *encoderParams)	Saves this image to a file.
GpStatus WINGDIPAPI GdipSaveImageToStream(GpImage *image, IStream* stream, GDIPCONST CLSID* clsidEncoder, GDIPCONST EncoderParameters* encoderParams)	Status Image::Save(IN IStream* stream, IN const CLSID* clsidEncoder, IN const EncoderParameters *encoderParams)	Saves this image to a stream.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSaveAdd(GpImage *image, GDIPCONST EncoderParameters* encoderParams)	Status Image::SaveAdd(IN const EncoderParameters *encoderParams)	Adds a frame to a file or stream specified in a previous call to the Save method. Use this method to save selected frames from a multiple-frame image to another multiple-frame image.
GpStatus WINGDIPAPI GdipSaveAddImage(GpImage *image, GpImage* newImage, GDIPCONST EncoderParameters* encoderParams)	Status Image::SaveAdd(IN Image* newImage, IN const EncoderParameters *encoderParams)	Adds a frame to a file or stream specified in a previous call to the Save method.
GpStatus WINGDIPAPI GdipGetImageGraphicsContext(GpImage *image, GpGraphics **graphics)	Graphics::Graphics(IN Image* image)	Creates a Graphics object that is associated with an Image object.
GpStatus WINGDIPAPI GdipGetImageBounds(GpImage *image, GpRectF *srcRect, GpUnit *srcUnit)	Status Image::GetBounds(OUT RectF *srcRect, OUT Unit *srcUnit)	Gets the bounding rectangle for this image.
GpStatus WINGDIPAPI GdipGetImageDimension(GpImage *image, REAL *width, REAL *height)	Status Image::GetPhysicalDimension(OUT SizeF* size)	Gets the width and height of this image. In the flat function, the <i>width</i> and <i>height</i> parameters together correspond to the <i>size</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdipGetImageType(GpImage *image, ImageType *type)	ImageType Image::GetType() const	Gets the type (bitmap or metafile) of this Image object.
GpStatus WINGDIPAPI GdipGetImageWidth(GpImage *image, UINT *width)	UINT Image::GetWidth()	Gets the width, in pixels, of this image.
GpStatus WINGDIPAPI GdipGetImageHeight(GpImage *image, UINT *height)	UINT Image::GetHeight()	Gets the image height, in pixels, of this image.
GpStatus WINGDIPAPI GdipGetImageHorizontalResolution(GpImage *image, REAL *resolution)	REAL Image::GetHorizontalResolution()	Gets the horizontal resolution, in dots per inch, of this image.
GpStatus WINGDIPAPI GdipGetImageVerticalResolution(GpImage *image, REAL *resolution)	REAL Image::GetVerticalResolution()	Gets the vertical resolution, in dots per inch, of this image.
GpStatus WINGDIPAPI GdipGetImageFlags(GpImage *image, UINT *flags)	UINT Image::GetFlags()	Gets a set of flags that indicate certain attributes of this Image object.
GpStatus WINGDIPAPI GdipGetImageRawFormat(GpImage *image, GUID *format)	Status Image::GetRawFormat(OUT GUID *format)	Gets a globally unique identifier (GUID) that identifies the format of this Image object. GUIDs that identify various file formats are defined in <i>Gdiplusimaging.h</i> .

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetImagePixelFormat(GpImage *image, PixelFormat *format)	PixelFormat Image::GetPixelFormat()	Gets the pixel format of this Image object.
GpStatus WINGDIPAPI GdipGetImageThumbnail(GpImage *image, UINT thumbWidth, UINT thumbHeight, GpImage **thumbImage, GetThumbnailImageAbort callback, VOID * callbackData)	Image* Image::GetThumbnailImage(IN UINT thumbWidth, IN UINT thumbHeight, IN GetThumbnailImageAbort callback, IN VOID* callbackData)	Gets a thumbnail image from this Image object.
GpStatus WINGDIPAPI GdipGetEncoderParameterListSize(GpImage *image, GDIPCONST CLSID* clsidEncoder, UINT* size)	UINT Image::GetEncoderParameterListSize(IN const CLSID* clsidEncoder)	Gets the size, in bytes, of the parameter list for a specified image encoder.
GpStatus WINGDIPAPI GdipGetEncoderParameterList(GpImage *image, GDIPCONST CLSID* clsidEncoder, UINT size, EncoderParameters* buffer)	Status Image::GetEncoderParameterList(IN const CLSID* clsidEncoder, IN UINT size, OUT EncoderParameters* buffer)	Gets a list of the parameters supported by a specified image encoder.
GpStatus WINGDIPAPI GdipImageGetFrameDimensionsCount(GpImage* image, UINT* count)	UINT Image::GetFrameDimensionsCount()	Gets the number of frame dimensions in this Image object.
GpStatus WINGDIPAPI GdipImageGetFrameDimensionsList(GpImage* image, GUID* dimensionIDs, UINT count)	Status Image::GetFrameDimensionsList(OUT GUID* dimensionIDs, IN UINT count)	Gets the identifiers for the frame dimensions of this Image object.
GpStatus WINGDIPAPI GdipImageGetFrameCount(GpImage *image, GDIPCONST GUID* dimensionID, UINT* count)	UINT Image::GetFrameCount(IN const GUID* dimensionID)	Gets the number of frames in a specified dimension of this Image object.
GpStatus WINGDIPAPI GdipImageSelectActiveFrame(GpImage *image, GDIPCONST GUID* dimensionID, UINT frameIndex)	Status Image::SelectActiveFrame(IN const GUID *dimensionID, IN UINT frameIndex)	Selects the frame in this Image object specified by a dimension and an index.
GpStatus WINGDIPAPI GdipImageRotateFlip(GpImage *image, RotateFlipType rfType)	Status Image::RotateFlip(IN RotateFlipType rotateFlipType)	Rotates and flips this image.
GpStatus WINGDIPAPI GdipGetImagePalette(GpImage *image, ColorPalette *palette, INT size)	Status Image::GetPalette(OUT ColorPalette *palette, IN INT size)	Gets the ColorPalette of this Image object.
GpStatus WINGDIPAPI GdipSetImagePalette(GpImage *image, GDIPCONST ColorPalette *palette)	Status Image::SetPalette(IN const ColorPalette *palette)	Sets the color palette of this Image object.
GpStatus WINGDIPAPI GdipGetImagePaletteSize(GpImage *image, INT *size)	INT Image::GetPaletteSize()	Gets the size, in bytes, of the color palette of this Image object.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetPropertyCount(GpImage *image, UINT* numOfProperty)	UINT Image::GetPropertyCount()	Gets the pixel format of this Image object.
GpStatus WINGDIPAPI GdipGetPropertyIdList(GpImage *image, UINT numOfProperty, PROPID* list)	Status Image::GetPropertyIdList(IN UINT numOfProperty, OUT PROPID* list)	Gets a list of the property identifiers used in the metadata of this Image object.
GpStatus WINGDIPAPI GdipGetPropertyItemSize(GpImage *image, PROPID propId, UINT* size)	UINT Image::GetPropertyItemSize(IN PROPID propId)	Gets the size, in bytes, of a specified property item of this Image object.
GpStatus WINGDIPAPI GdipGetPropertyItem(GpImage *image, PROPID propId,UINT propSize, PropertyItem* buffer)	Status Image::GetPropertyItem(IN PROPID propId, IN UINT propSize, OUT PropertyItem* buffer)	Gets a specified property item (piece of metadata) from this Image object.
GpStatus WINGDIPAPI GdipGetPropertySize(GpImage *image, UINT* totalBufferSize, UINT* numProperties)	Status Image::GetPropertySize(OUT UINT* totalBufferSize, OUT UINT* numProperties)	Gets the total size, in bytes, of all the property items stored in this Image object. This method also gets the number of property items stored in this Image object.
GpStatus WINGDIPAPI Gdip GetAllPropertyItems(GpImage *image, UINT totalBufferSize, UINT numProperties, PropertyItem* allItems)	Status Image::GetAllPropertyItems(IN UINT totalBufferSize, IN UINT numProperties, OUT PropertyItem* allItems)	Gets all the property items (metadata) stored in this Image object.
GpStatus WINGDIPAPI GdipRemovePropertyItem(GpImage *image, PROPID propId)	Status Image::RemovePropertyItem(IN PROPID propId)	Removes a property item (piece of metadata) from this Image object.
GpStatus WINGDIPAPI GdipSetPropertyItem(GpImage *image, GDIPCONST PropertyItem* item)	Status Image::SetPropertyItem	Sets a property item (piece of metadata) for this Image object. If the item already exists, then its contents are updated; otherwise, a new item is added.
GpStatus WINGDIPAPI GdipFindFirstImageItem(GpImage *image, ImageItemData* item)	Status Image::FindFirstItem(IN ImageItemData* item)	Retrieves the description and the data size of the first metadata item in this Image object.
GpStatus WINGDIPAPI GdipFindNextImageItem(GpImage *image, ImageItemData* item)	Status Image::FindNextItem(IN ImageItemData* item)	Retrieves the description and the data size of the next metadata item in this Image object. This method is used along with the Image::FindFirstItem method to enumerate the metadata items stored in this Image object.
GpStatus WINGDIPAPI GdipGetImageItemData(GpImage *image, ImageItemData* item)	Status Image::GetImageData(IN ImageItemData* item)	Gets one piece of metadata from this Image object.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdiplusImageSetAbort(GpImage *plImage, GdiplusAbort *pAbort)	Status Image::SetAbort(GdiplusAbort *pAbort)	Sets the object whose Abort method is called periodically during time-consuming rendering operation.
GpStatus WINGDIPAPI GdipConvertToEmfPlus(const GpGraphics* refGraphics, GpMetafile* metafile, BOOL* conversionSuccess, EmfType emfType, const WCHAR* description, GpMetafile** out_metafile)	Status Metafile::ConvertToEmfPlus(const Graphics* refGraphics, BOOL* conversionSuccess, EmfType emfType, const WCHAR* description)	Converts this Metafile object to the EMF+ format.
GpStatus WINGDIPAPI GdipConvertToEmfPlusToFile(const GpGraphics* refGraphics, GpMetafile* metafile, BOOL* conversionSuccess, const WCHAR* filename, EmfType emfType, const WCHAR* description, GpMetafile** out_metafile)	Status Metafile::ConvertToEmfPlus(const Graphics* refGraphics, const WCHAR* filename, BOOL* conversionSuccess, EmfType emfType, const WCHAR* description)	Converts this Metafile object to the EMF+ format.
GpStatus WINGDIPAPI GdipConvertToEmfPlusToStream(const GpGraphics* refGraphics, GpMetafile* metafile, BOOL* conversionSuccess, IStream* stream, EmfType emfType, const WCHAR* description, GpMetafile** out_metafile)	Status Metafile::ConvertToEmfPlus(const Graphics* refGraphics, IStream* stream, BOOL* conversionSuccess, EmfType emfType, const WCHAR* description)	Converts this Metafile object to the EMF+ format.
GpStatus WINGDIPAPI GdiplusImageForceValidation(GpImage *image)	Not called by wrapper methods.	This function forces validation of the image.

ImageAttributes Functions

11/2/2020 • 5 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [ImageAttributes](#) C++ class.

ImageAttributes Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreateImageAttributes(GpImageAttributes **imageattr)	ImageAttributes::ImageAttributes()	Creates an ImageAttributes object.
GpStatus WINGDIPAPI GdipCloneImageAttributes(GDIPCONST GpImageAttributes *imageattr, GpImageAttributes **cloneImageattr)	ImageAttributes* ImageAttributes::Clone() const	Makes a copy of this ImageAttributes object.
GpStatus WINGDIPAPI GdipDisposeImageAttributes(GpImageAttributes *imageattr)	ImageAttributes::~ImageAttributes()	Releases resources used by the ImageAttributes object.
GpStatus WINGDIPAPI GdipSetImageAttributesToIdentity(GpImageAttributes *imageattr, ColorAdjustType type)	Status ImageAttributes::SetToIdentity(IN ColorAdjustType type = ColorAdjustTypeDefault)	Sets the color-adjustment matrix of a specified category to identity matrix.
GpStatus WINGDIPAPI GdipResetImageAttributes(GpImageAttributes *imageattr, ColorAdjustType type)	Status ImageAttributes::Reset(IN ColorAdjustType type = ColorAdjustTypeDefault)	Sets the color-adjustment matrix of a specified category to identity matrix.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSetImageAttributesColorMatrix(GpImageAttributes *imageattr, ColorAdjustType type, BOOL enableFlag, GDIPCONST ColorMatrix* colorMatrix, GDIPCONST ColorMatrix* grayMatrix, ColorMatrixFlags flags)	Status ImageAttributes::SetColorMatrix(IN const ColorMatrix *colorMatrix, IN ColorMatrixFlags mode = ColorMatrixFlagsDefault, IN ColorAdjustType type = ColorAdjustTypeDefault)Status ImageAttributes::ClearColorMatrix(IN ColorAdjustType type = ColorAdjustTypeDefault)	Sets the color-adjustment matrix for a specified category. The <i>enableFlag</i> parameter in the flat function is a Boolean value that specifies whether a separate color adjustment is enabled for the category specified by the <i>type</i> parameter. ImageAttributes::SetColorMatrix sets <i>enableFlag</i> to TRUE, and ImageAttributes::ClearColorMatrix sets <i>enableFlag</i> to FALSE. Clears the color-adjustment matrix for a specified category. The <i>grayMatrix</i> parameter specifies a matrix to be used for adjusting gray shades when the value of the <i>flags</i> parameter is ColorMatrixFlagsAltGray .
GpStatus WINGDIPAPI GdipSetImageAttributesThreshold(GpImageAttributes *imageattr, ColorAdjustType type, BOOL enableFlag, REAL threshold)	Status ImageAttributes::SetThreshold(IN REAL threshold, IN ColorAdjustType type = ColorAdjustTypeDefault)Status ImageAttributes::ClearThreshold(IN ColorAdjustType type = ColorAdjustTypeDefault)	Sets the threshold (transparency range) for a specified category. The <i>enableFlag</i> parameter in the flat function is a Boolean value that specifies whether a separate threshold is enabled for the category specified by the <i>type</i> parameter. ImageAttributes::SetThreshold sets <i>enableFlag</i> to TRUE, and ImageAttributes::ClearThreshold sets <i>enableFlag</i> to FALSE. Clears the threshold value for a specified category.
GpStatus WINGDIPAPI GdipSetImageAttributesGamma(GpImageAttributes *imageattr, ColorAdjustType type, BOOL enableFlag, REAL gamma)	Status ImageAttributes::SetGamma(IN REAL gamma, IN ColorAdjustType type = ColorAdjustTypeDefault)Status ImageAttributes::ClearGamma(IN ColorAdjustType type = ColorAdjustTypeDefault)	Sets the gamma value for a specified category. The <i>enableFlag</i> parameter in the flat function is a Boolean value that specifies whether a separate gamma is enabled for the category specified by the <i>type</i> parameter. ImageAttributes::SetGamma sets <i>enableFlag</i> to TRUE, and ImageAttributes::ClearGamma sets <i>enableFlag</i> to FALSE. Disables gamma correction for a specified category.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSetImageAttributesNoOp(GpImageAttributes *imageattr, ColorAdjustType type, BOOL enableFlag)	Status ImageAttributes::SetNoOp(IN ColorAdjustType type = ColorAdjustTypeDefault) Status ImageAttributes::ClearNoOp(IN ColorAdjustType type = ColorAdjustTypeDefault)	Turns off color adjustment for a specified category. You can call the ImageAttributes::ClearNoOp method to reinstate the color-adjustment settings that were in place before the call to ImageAttributes::SetNoOp method . The <i>enableFlag</i> parameter in the flat function is a Boolean value that specifies whether a color adjustment is enabled for the category specified by the <i>type</i> parameter. ImageAttributes::SetNoOp sets <i>enableFlag</i> to TRUE, and ImageAttributes::ClearNoOp sets <i>enableFlag</i> to FALSE. Clears the NoOp setting for a specified category.
GpStatus WINGDIPAPI GdipSetImageAttributesColorKeys(GpImageAttributes *imageattr, ColorAdjustType type, BOOL enableFlag, ARGB colorLow, ARGB colorHigh)	Status ImageAttributes::SetColorKey(IN const Color& colorLow, IN const Color& colorHigh, IN ColorAdjustType type = ColorAdjustTypeDefault) Status ImageAttributes::ClearColorKey(IN ColorAdjustType type = ColorAdjustTypeDefault)	Sets the color key (transparency range) for a specified category. The <i>enableFlag</i> parameter in the flat function is a Boolean value that specifies whether a separate transparency range is enabled for the category specified by the <i>type</i> parameter. ImageAttributes::SetColorKey sets <i>enableFlag</i> to TRUE, and ImageAttributes::ClearColorKey sets <i>enableFlag</i> to FALSE. Clears the color key (transparency range) for a specified category.
GpStatus WINGDIPAPI GdipSetImageAttributesOutputChannel (GpImageAttributes *imageattr, ColorAdjustType type, BOOL enableFlag, ColorChannelFlags channelFlags)	Status ImageAttributes::SetOutputChannel(IN ColorChannelFlags channelFlags, IN ColorAdjustType type = ColorAdjustTypeDefault) Status ImageAttributes::ClearOutputChannel(IN ColorAdjustType type = ColorAdjustTypeDefault)	Sets the cyan-magenta-yellow-black (CMYK) output channel for a specified category. The <i>enableFlag</i> parameter in the flat function is a Boolean value that specifies whether a separate output channel is enabled for the category specified by the <i>type</i> parameter. ImageAttributes::SetOutputChannel sets <i>enableFlag</i> to TRUE, and ImageAttributes::ClearOutputChannel sets <i>enableFlag</i> to FALSE. Clears the cyan-magenta-yellow-black (CMYK) output channel setting for a specified category.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSetImageAttributesOutputChannel ColorProfile(GpImageAttributes *imageattr, ColorAdjustType type, BOOL enableFlag, GDIPCONST WCHAR *colorProfileFilename)	Status ImageAttributes::SetOutputChann elColorProfile(IN const WCHAR *colorProfileFilename, IN ColorAdjustType type = ColorAdjustTypeDefault) Status ImageAttributes::ClearOutputChan nelColorProfile(IN ColorAdjustType type = ColorAdjustTypeDefault)	Sets the output channel color-profile file for a specified category. The <i>enableFlag</i> parameter in the flat function is a Boolean value that specifies whether a separate output channel color profile is enabled for the category specified by the <i>type</i> parameter. ImageAttributes::SetOutputChann elColorProfile sets <i>enableFlag</i> to TRUE, and ImageAttributes::ClearOutputChan nelColorProfile sets <i>enableFlag</i> to FALSE. Clears the output channel color profile setting for a specified category.
GpStatus WINGDIPAPI GdipSetImageAttributesRemapTable(Gp imageAttributes *imageattr, ColorAdjustType type, BOOL enableFlag, UINT mapSize, GDIPCONST ColorMap *map)	Status ImageAttributes::SetRemapTable(IN UINT mapSize, IN const ColorMap *map, IN ColorAdjustType type = ColorAdjustTypeDefault) Status ImageAttributes::ClearRemapTable(IN ColorAdjustType type = ColorAdjustTypeDefault)	Sets the color-remap table for a specified category. The <i>enableFlag</i> parameter in the flat function is a Boolean value that specifies whether a separate color remap table is enabled for the category specified by the <i>type</i> parameter. ImageAttributes::SetRemapTable sets <i>enableFlag</i> to TRUE, and ImageAttributes::ClearRemapTable sets <i>enableFlag</i> to FALSE. Clears the color-remap table for a specified category.
GpStatus WINGDIPAPI GdipSetImageAttributesWrapMode(Gp imageAttributes *imageAttr, WrapMode wrap, ARGB argb, BOOL clamp)	Status ImageAttributes::SetWrapMode(IN WrapMode wrap, IN const Color& color = Color(), IN BOOL clamp = FALSE)	Sets the wrap mode of this ImageAttributes object
GpStatus WINGDIPAPI GdipSetImageAttributesICMMode(Gp imageAttributes *imageAttr, BOOL on)	Not called by wrapper methods.	This function sets an internal state variable to the value specified by the <i>on</i> parameter. If this value is TRUE, Image Color Management (ICM) is used for all color adjustment. If the value is FALSE, ICM is not used.
GpStatus WINGDIPAPI GdipGetImageAttributesAdjustedPalette (GpImageAttributes *imageAttr, ColorPalette * colorPalette, ColorAdjustType colorAdjustType)	Status ImageAttributes::GetAdjustedPalet te(IN OUT ColorPalette*
 colorPalette, IN ColorAdjustType colorAdjustType) const	Adjusts the colors in a palette according to the adjustment settings of a specified category.
GpStatus WINGDIPAPI GdipSetImageAttributesCachedBackgro und(GpImageAttributes *imageattr, BOOL enableFlag)	Not called by wrapper methods.	Sets or clears the CachedBackground member of a specified GpImageAttributes object. GDI+ does not use the CachedBackground member, so calling this function has no effect. The <i>imageattr</i> parameter specifies the GpImageAttributes object. The <i>enableFlag</i> parameter specifies whether the CachedBackground member is set (TRUE) or cleared (FALSE).

LinearGradientBrush Functions

11/2/2020 • 6 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [LinearGradientBrush](#) C++ class.

LinearGradientBrush Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreateLineBrush(GDIPCONST GpPointF* point1, GDIPCONST GpPointF* point2, ARGB color1, ARGB color2, GpWrapMode wrapMode, GpLineGradient **lineGradient)	LinearGradientBrush::LinearGradientBrush(IN const PointF& point1, IN const PointF& point2, IN const Color& color1, IN const Color& color2)	Creates a LinearGradientBrush object from a set of boundary points and boundary colors. The <i>wrapMode</i> parameter in the flat function is a member of the WrapMode enumeration that specifies how areas filled with the brush are tiled.
GpStatus WINGDIPAPI GdipCreateLineBrush1(GDIPCONST GpPoint* point1, GDIPCONST GpPoint* point2, ARGB color1, ARGB color2, GpWrapMode wrapMode, GpLineGradient **lineGradient)	LinearGradientBrush::LinearGradientBrush(IN const Point& point1, IN const Point& point2, IN const Color& color1, IN const Color& color2)	Creates a LinearGradientBrush object from a set of boundary points and boundary colors. The <i>wrapMode</i> parameter in the flat function is a member of the WrapMode enumeration that specifies how areas filled with the brush are tiled.
GpStatus WINGDIPAPI GdipCreateLineBrushFromRect(GDIPCONST GpRectF* rect, ARGB color1, ARGB color2, LinearGradientMode mode, GpWrapMode wrapMode, GpLineGradient **lineGradient)	LinearGradientBrush::LinearGradientBrush(IN const RectF& rect, IN const Color& color1, IN const Color& color2, IN LinearGradientMode mode)	Creates a LinearGradientBrush object based on a rectangle and mode of direction. The <i>wrapMode</i> parameter in the flat function is a member of the WrapMode enumeration that specifies how areas filled with the brush are tiled.
GpStatus WINGDIPAPI GdipCreateLineBrushFromRect1(GDIPCONST GpRect* rect, ARGB color1, ARGB color2, LinearGradientMode mode, GpWrapMode wrapMode, GpLineGradient **lineGradient)	LinearGradientBrush::LinearGradientBrush(IN const Rect& rect, IN const Color& color1, IN const Color& color2, IN LinearGradientMode mode)	Creates a LinearGradientBrush object based on a rectangle and mode of direction. The <i>wrapMode</i> parameter in the flat function is a member of the WrapMode enumeration that specifies how areas filled with the brush are tiled.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreateLineBrushFromRectWithAngle(GDIPCONST GpRectF* rect, ARGB color1, ARGB color2, REAL angle, BOOL isAngleScalable, GpWrapMode wrapMode, GpLineGradient **lineGradient)	<code>LinearGradientBrush::LinearGradientBrush(IN const RectF& rect, IN const Color& color1, IN const Color& color2, IN REAL angle, IN BOOL isAngleScalable = FALSE)</code>	Creates a LinearGradientBrush object from a rectangle and angle of direction. The <i>wrapMode</i> parameter in the flat function is a member of the WrapMode enumeration that specifies how areas filled with the brush are tiled.
GpStatus WINGDIPAPI GdipCreateLineBrushFromRectWithAngle(GDIPCONST GpRect* rect, ARGB color1, ARGB color2, REAL angle, BOOL isAngleScalable, GpWrapMode wrapMode, GpLineGradient **lineGradient)	<code>LinearGradientBrush::LinearGradientBrush(IN const Rect& rect, IN const Color& color1, IN const Color& color2, IN REAL angle, IN BOOL isAngleScalable = FALSE)</code>	Creates a LinearGradientBrush object from a rectangle and angle of direction. The <i>wrapMode</i> parameter in the flat function is a member of the WrapMode enumeration that specifies how areas filled with the brush are tiled.
GpStatus WINGDIPAPI GdipSetLineColors(GpLineGradient *brush, ARGB color1, ARGB color2)	<code>Status LinearGradientBrush::SetLinearColors(IN const Color& color1, IN const Color& color2)</code>	Sets the starting color and ending color of this linear gradient brush.
GpStatus WINGDIPAPI GdipGetLineColors(GpLineGradient *brush, ARGB* colors)	<code>Status LinearGradientBrush::GetLinearColors(OUT Color* colors) const</code>	Gets the starting color and ending color of this linear gradient brush.
GpStatus WINGDIPAPI GdipGetLineRect(GpLineGradient *brush, GpRectF *rect)	<code>Status LinearGradientBrush::GetRectangle(OUT RectF* rect) const</code>	Gets the rectangle that defines the boundaries of the gradient.
GpStatus WINGDIPAPI GdipGetLineRectI(GpLineGradient *brush, GpRect *rect)	<code>Status LinearGradientBrush::GetRectangle(OUT Rect* rect) const</code>	Gets the rectangle that defines the boundaries of the gradient.
GpStatus WINGDIPAPI GdipSetLineGammaCorrection(GpLineGradient *brush, BOOL useGammaCorrection)	<code>Status LinearGradientBrush::SetGammaCorrection(IN BOOL useGammaCorrection)</code>	Specifies whether gamma correction is enabled for this linear gradient brush.
GpStatus WINGDIPAPI GdipGetLineGammaCorrection(GpLineGradient *brush, BOOL *useGammaCorrection)	<code>BOOL LinearGradientBrush::GetGammaCorrection() const</code>	Determines whether gamma correction is enabled.
GpStatus WINGDIPAPI GdipGetLineBlendCount(GpLineGradient *brush, INT *count)	<code>INT LinearGradientBrush::GetBlendCount() const</code>	Gets the number of blend factors currently set .
GpStatus WINGDIPAPI GdipGetLineBlend(GpLineGradient *brush, REAL *blend, REAL* positions, INT count)	<code>Status LinearGradientBrush::GetBlend(OUT REAL* blendFactors, OUT REAL* blendPositions, IN INT count) const</code>	Gets the blend factors and their corresponding blend positions from a LinearGradientBrush object.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSetLineBlend(GpLineGradient *brush, GDIPCONST REAL *blend, GDIPCONST REAL* positions, INT count)	Status LinearGradientBrush::SetBlend(IN const REAL* blendFactors, IN const REAL* blendPositions, IN INT count)	Sets the blend factors and the blend positions of this linear gradient brush to create a custom blend.
GpStatus WINGDIPAPI GdipGetLinePresetBlendCount(GpLineGradient *brush, INT *count)	INT LinearGradientBrush::GetInterpolationColorCount() const	Gets the number of colors currently set to be interpolated for this linear gradient brush.
GpStatus WINGDIPAPI GdipGetLinePresetBlend(GpLineGradient *brush, ARGB *blend, REAL* positions, INT count)	Status LinearGradientBrush::GetInterpolationColors(OUT Color* presetColors, OUT REAL* blendPositions, IN INT count) const	Gets the colors currently set to be interpolated for this linear gradient brush and their corresponding blend positions.
GpStatus WINGDIPAPI GdipSetLinePresetBlend(GpLineGradient *brush, GDIPCONST ARGB *blend, GDIPCONST REAL* positions, INT count)	Status LinearGradientBrush::SetInterpolationColors(IN const Color* presetColors, IN const REAL* blendPositions, IN INT count)	Sets the colors to be interpolated for this linear gradient brush and their corresponding blend positions.
GpStatus WINGDIPAPI GdipSetLineSigmaBlend(GpLineGradient *brush, REAL focus, REAL scale)	Status LinearGradientBrush::SetBlendBellShape(IN REAL focus, IN REAL scale = 1.0f)	Sets the blend shape of this linear gradient brush to create a custom blend based on a bell-shaped curve.
GpStatus WINGDIPAPI GdipSetLineLinearBlend(GpLineGradient *brush, REAL focus, REAL scale)	Status LinearGradientBrush::SetBlendTriangularShape(IN REAL focus, IN REAL scale = 1.0f)	Sets the blend shape of this linear gradient brush to create a custom blend based on a triangular shape.
GpStatus WINGDIPAPI GdipSetLineWrapMode(GpLineGradient *brush, GpWrapMode wrapmode)	Status LinearGradientBrush::SetWrapMode(IN WrapMode wrapMode)	Sets the wrap mode of this linear gradient brush.
GpStatus WINGDIPAPI GdipGetLineWrapMode(GpLineGradient *brush, GpWrapMode *wrapmode)	WrapMode LinearGradientBrush::GetWrapMode() const	Gets the wrap mode for this brush. The wrap mode determines how an area is tiled when it is painted with a brush.
GpStatus WINGDIPAPI GdipGetLineTransform(GpLineGradient *brush, GpMatrix *matrix)	Status LinearGradientBrush::GetTransform(OUT Matrix *matrix) const	Gets the transformation matrix of this linear gradient brush.
GpStatus WINGDIPAPI GdipSetLineTransform(GpLineGradient *brush, GDIPCONST GpMatrix *matrix)	Status LinearGradientBrush::SetTransform (IN const Matrix* matrix)	Sets the transformation matrix of this linear gradient brush.
GpStatus WINGDIPAPI GdipResetLineTransform(GpLineGradient *brush)	Status LinearGradientBrush::ResetTransform()	Resets the transformation matrix of this linear gradient brush to the identity matrix. This means that no transformation takes place.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipMultiplyLineTransform(GpLineGradient* brush, GDIPCONST GpMatrix *matrix, GpMatrixOrder order)	Status LinearGradientBrush::MultiplyTransform (IN const Matrix* matrix, IN MatrixOrder order = MatrixOrderPrepend)	Updates this brush's transformation matrix with the product of itself and another matrix.
GpStatus WINGDIPAPI GdipTranslateLineTransform(GpLineGradient* brush, REAL dx, REAL dy, GpMatrixOrder order)	Status LinearGradientBrush::TranslateTransform (IN REAL dx, IN REAL dy, IN MatrixOrder order = MatrixOrderPrepend)	Updates this brush's current transformation matrix with the product of itself and a translation matrix.
GpStatus WINGDIPAPI GdipScaleLineTransform(GpLineGradient* brush, REAL sx, REAL sy, GpMatrixOrder order)	Status LinearGradientBrush::ScaleTransform (IN REAL sx, IN REAL sy, IN MatrixOrder order = MatrixOrderPrepend)	Updates this brush's current transformation matrix with the product of itself and a scaling matrix.
GpStatus WINGDIPAPI GdipRotateLineTransform(GpLineGradient* brush, REAL angle, GpMatrixOrder order)	Status LinearGradientBrush::RotateTransform (IN REAL angle, IN MatrixOrder order = MatrixOrderPrepend)	Updates this brush's current transformation matrix with the product of itself and a rotation matrix.

Matrix Functions

11/2/2020 • 3 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [Matrix](#) C++ class.

Matrix Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipCreateMatrix(GpMatrix **matrix)	Matrix::Matrix	Creates and initializes a Matrix::Matrix object that represents the identity matrix.
GpStatus WINGDIPAPI GdipCreateMatrix2(REAL m11, REAL m12, REAL m21, REAL m22, REAL dx, REAL dy, GpMatrix **matrix)	Matrix::Matrix	Creates and initializes a Matrix::Matrix object based on six numbers that define an affine transformation.
GpStatus WINGDIPAPI GdipCreateMatrix3(GDIPCONST GpRectF *rect, GDIPCONST GpPointF *dstplg, GpMatrix **matrix)	Matrix::Matrix	Creates a Matrix::Matrix object based on a rectangle and a point.
GpStatus WINGDIPAPI GdipCreateMatrix3I(GDIPCONST GpRect *rect, GDIPCONST GpPoint *dstplg, GpMatrix **matrix)	Matrix::Matrix	Creates a Matrix::Matrix object based on a rectangle and a point.
GpStatus WINGDIPAPI GdipCloneMatrix(GpMatrix *matrix, GpMatrix **cloneMatrix)	Matrix::Clone	The Matrix::Clone method creates a new Matrix object that is a copy of this Matrix object.
GpStatus WINGDIPAPI GdipDeleteMatrix(GpMatrix *matrix)	<code>~Matrix()</code>	Cleans up resources used by a Matrix::Matrix object.
GpStatus WINGDIPAPI GdipSetMatrixElements(GpMatrix *matrix, REAL m11, REAL m12, REAL m21, REAL m22, REAL dx, REAL dy)	Matrix::SetElements	The Matrix::SetElements method sets the elements of this matrix.
GpStatus WINGDIPAPI GdipMultiplyMatrix(GpMatrix *matrix, GpMatrix* matrix2, GpMatrixOrder order)	Matrix::Multiply	The Matrix::Multiply method updates this matrix with the product of itself and another matrix.

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipTranslateMatrix(GpMatrix *matrix, REAL offsetX, REAL offsetY, GpMatrixOrder order)	Matrix::Translate	The Matrix::Translate method updates this matrix with the product of itself and a translation matrix.
GpStatus WINGDIPAPI GdipScaleMatrix(GpMatrix *matrix, REAL scaleX, REAL scaleY, GpMatrixOrder order)	Matrix::Scale	The Matrix::Scale method updates this matrix with the product of itself and a scaling matrix.
GpStatus WINGDIPAPI GdipRotateMatrix(GpMatrix *matrix, REAL angle, GpMatrixOrder order)	Matrix::Rotate	The Matrix::Rotate method updates this matrix with the product of itself and a rotation matrix.
GpStatus WINGDIPAPI GdipShearMatrix(GpMatrix *matrix, REAL shearX, REAL shearY, GpMatrixOrder order)	Matrix::Shear	The Matrix::Shear method updates this matrix with the product of itself and a shearing matrix.
GpStatus WINGDIPAPI GdipInvertMatrix(GpMatrix *matrix)	Matrix::Invert	If this matrix is invertible, the Matrix::Invert method replaces the elements of this matrix with the elements of its inverse.
GpStatus WINGDIPAPI GdipTransformMatrixPoints(GpMatrix *matrix, GpPointF *pts, INT count)	Matrix::TransformPoints	The Matrix::TransformPoints method multiplies each point in an array by this matrix. Each point is treated as a row matrix. The multiplication is performed with the row matrix on the left and this matrix on the right.
GpStatus WINGDIPAPI GdipTransformMatrixPointsI(GpMatrix *matrix, GpPoint *pts, INT count)	Matrix::TransformPoints	The Matrix::TransformPoints method multiplies each point in an array by this matrix. Each point is treated as a row matrix. The multiplication is performed with the row matrix on the left and this matrix on the right.
GpStatus WINGDIPAPI GdipVectorTransformMatrixPoints(GpMatrix *matrix, GpPointF *pts, INT count)	Matrix::TransformVectors	The Matrix::TransformVectors method multiplies each vector in an array by this matrix. The translation elements of this matrix (third row) are ignored. Each vector is treated as a row matrix. The multiplication is performed with the row matrix on the left and this matrix on the right.
GpStatus WINGDIPAPI GdipVectorTransformMatrixPointsI(GpMatrix *matrix, GpPoint *pts, INT count)	Matrix::TransformVectors	The Matrix::TransformVectors method multiplies each vector in an array by this matrix. The translation elements of this matrix (third row) are ignored. Each vector is treated as a row matrix. The multiplication is performed with the row matrix on the left and this matrix on the right.

FLAT FUNCTION	WRAPPER METHOD	DESCRIPTION
GpStatus WINGDIPAPI GdipGetMatrixElements(GDIPCONST GpMatrix *matrix, REAL *matrixOut)	Matrix::GetElements	The Matrix::GetElements method gets the elements of this matrix. The elements are placed in an array in the order m11, m12, m21, m22, m31, m32, where mij denotes the element in row i, column j.
GpStatus WINGDIPAPI GdipIsMatrixInvertible(GDIPCONST GpMatrix *matrix, BOOL *result)	Matrix::IsInvertible	The Matrix::IsInvertible method determines whether this matrix is invertible.
GpStatus WINGDIPAPI GdipIsMatrixIdentity(GDIPCONST GpMatrix *matrix, BOOL *result)	Matrix::IsIdentity	The Matrix::IsIdentity method determines whether this matrix is the identity matrix.
GpStatus WINGDIPAPI GdipIsMatrixEqual(GDIPCONST GpMatrix *matrix, GDIPCONST GpMatrix *matrix2, BOOL *result)	Matrix::Equals	The Matrix::Equals method determines whether the elements of this matrix are equal to the elements of another matrix.

Memory Functions

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [GdiplusBase](#) C++ class.

Memory Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipAlloc(size_t size)	GpStatus WINGDIPAPI GdiplusBase void* (operator new) (size_t in_size)	Allocates memory for one Windows GDI+ object. GdipAlloc is declared in GdiplusMem.h.
GpStatus WINGDIPAPI GdipFree(void* ptr);	GpStatus WINGDIPAPI GdiplusBase void (operator delete) (void* in_pVoid)	Deallocates memory for one Windows GDI+ object. GdipFree is declared in GdiplusMem.h.

Notification Functions

2/22/2020 • 2 minutes to read • [Edit Online](#)

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdiplusNotificationHook(OUT ULONG_PTR *token)	Not called by wrapper methods.	<p>The GdiplusStartup function returns (in its output parameter) a pointer to a GdiplusStartupOutput structure. One of the members of the structure is a pointer to a notification hook function that has the same signature as GdiplusNotificationHook. There are two ways you can call the notification hook function; you can use the pointer returned by GdiplusStartup or you can call GdiplusNotificationHook. In fact, GdiplusNotificationHook simply verifies that you have suppressed the background thread and then calls the notification hook function that is returned by GdiplusStartup. The token parameter receives an identifier that you should later pass in a corresponding call to the notification unhook function.</p>
VOID WINGDIPAPI GdiplusNotificationUnhook(ULONG_PTR token)	Not called by wrapper methods.	<p>The GdiplusStartup function returns (in its output parameter) a pointer to a GdiplusStartupOutput structure. One of the members of the structure is a pointer to a notification unhook function that has the same signature as GdiplusNotificationUnhook. There are two ways you can call the notification unhook function; you can use the pointer returned by GdiplusStartup or you can call GdiplusNotificationUnhook. In fact, GdiplusNotificationUnhook simply verifies that you have suppressed the background thread and then calls the notification unhook function that is returned by GdiplusStartup. When you call the notification unhook function, pass the token that you previously received from a corresponding call to the notification hook function. If you do not do this, there will be resource leaks that won't be cleaned up until the process exits.</p>

PathGradientBrush Functions

11/2/2020 • 6 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [PathGradientBrush](#) C++ class.

PathGradientBrush Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreatePathGradient(GDIPCONST GpPointF* points, INT count, GpWrapMode wrapMode, GpPathGradient **polyGradient)	PathGradientBrush::PathGradientBrush(IN const PointF* points, IN INT count, IN WrapMode wrapMode = WrapModeClamp)	Creates a PathGradientBrush object based on an array of points. Initializes the wrap mode of the path gradient brush.
GpStatus WINGDIPAPI GdipCreatePathGradientI(GDIPCONST GpPoint* points, INT count, GpWrapMode wrapMode, GpPathGradient **polyGradient)	PathGradientBrush::PathGradientBrush(IN const Point* points, IN INT count, IN WrapMode wrapMode = WrapModeClamp)	Creates a PathGradientBrush object based on an array of points. Initializes the wrap mode of the path gradient brush.
GpStatus WINGDIPAPI GdipCreatePathGradientFromPath(GDIP CONST GpPath* path, GpPathGradient **polyGradient)	PathGradientBrush::PathGradientBrush(IN const GraphicsPath* path)	Creates a PathGradientBrush object based on a GraphicsPath object.
GpStatus WINGDIPAPI GdipGetPathGradientCenterColor(GpPathGradient *brush, ARGB* colors)	Status PathGradientBrush::GetCenterColor(OUT Color* color) const	Gets the color of the center point of this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientCenterColor(GpPathGradient *brush, ARGB colors)	Status PathGradientBrush::SetCenterColor(IN const Color& color)	Sets the center color of this path gradient brush. The center color is the color that appears at the brush's center point.
GpStatus WINGDIPAPI GdipGetPathGradientSurroundColorsWi thCount(GpPathGradient *brush, ARGB* color, INT* count)	Status PathGradientBrush::GetSurroundColors(OUT Color* colors, IN OUT INT* count) const	Gets the surround colors currently specified for this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientSurroundColorsWit hCount(GpPathGradient *brush, GDIPCONST ARGB* color, INT* count)	Status PathGradientBrush::SetSurroundColors(IN const Color* colors, IN OUT INT* count)	Sets the surround colors of this path gradient brush. The surround colors are colors specified for discrete points on the brush's boundary path.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetPathGradientPath(GpPathGradient *brush, GpPath *path)	Status PathGradientBrush::GetGraphicsPath(OUT GraphicsPath* path) const	Is not implemented in GDI+ version 1.0.
GpStatus WINGDIPAPI GdipSetPathGradientPath(GpPathGradient *brush, GDIPCONST GpPath *path)	Status PathGradientBrush::SetGraphicsPath(IN const GraphicsPath* path)	Is not implemented in GDI+ version 1.0.
GpStatus WINGDIPAPI GdipGetPathGradientCenterPoint(GpPathGradient *brush, GpPointF* points)	Status PathGradientBrush::GetCenterPoint(OUT PointF* point) const	Gets the center point of this path gradient brush.
GpStatus WINGDIPAPI GdipGetPathGradientCenterPointI(GpPathGradient *brush, GpPoint* points)	Status PathGradientBrush::GetCenterPoint(OUT Point* point) const	Gets the center point of this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientCenterPoint(GpPathGradient *brush, GDIPCONST GpPointF* points)	Status PathGradientBrush::SetCenterPoint(IN const PointF& point)	Sets the center point of this path gradient brush. By default, the center point is at the centroid of the brush's boundary path, but you can set the center point to any location inside or outside the path.
GpStatus WINGDIPAPI GdipSetPathGradientCenterPointI(GpPathGradient *brush, GDIPCONST GpPoint* points)	Status PathGradientBrush::SetCenterPoint(IN const Point& point)	Sets the center point of this path gradient brush. By default, the center point is at the centroid of the brush's boundary path, but you can set the center point to any location inside or outside the path.
GpStatus WINGDIPAPI GdipGetPathGradientRect(GpPathGradient *brush, GpRectF *rect)	Status PathGradientBrush::GetRectangle(OUT RectF* rect) const	Gets the smallest rectangle that encloses the boundary path of this path gradient brush.
GpStatus WINGDIPAPI GdipGetPathGradientRectI(GpPathGradient *brush, GpRect *rect)	Status PathGradientBrush::GetRectangle(OUT Rect* rect) const	Gets the smallest rectangle that encloses the boundary path of this path gradient brush.
GpStatus WINGDIPAPI GdipGetPathGradientPointCount(GpPathGradient *brush, INT* count)	INT PathGradientBrush::GetPointCount() const	Gets the number of points in the array of points that defines this brush's boundary path.
GpStatus WINGDIPAPI GdipGetPathGradientSurroundColorCount(GpPathGradient *brush, INT* count)	INT PathGradientBrush::GetSurroundColorCount() const	Gets the number of colors that have been specified for the boundary path of this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientGammaCorrection(GpPathGradient *brush, BOOL useGammaCorrection)	Status PathGradientBrush::SetGammaCorrection(IN BOOL useGammaCorrection)	Specifies whether gamma correction is enabled for this path gradient brush.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetPathGradientGammaCorrection(GpPathGradient *brush, BOOL *useGammaCorrection)	BOOL PathGradientBrush::GetGammaCorrection() const	Determines whether gamma correction is enabled for this path gradient brush.
GpStatus WINGDIPAPI GdipGetPathGradientBlendCount(GpPathGradient *brush, INT *count)	INT PathGradientBrush::GetBlendCount() const	Gets the number of blend factors currently set for this path gradient brush.
GpStatus WINGDIPAPI GdipGetPathGradientBlend(GpPathGradient *brush, REAL *blend, REAL *positions, INT count)	PathGradientBrush::GetBlend(OUT REAL* blendFactors, OUT REAL* blendPositions, IN INT count) const	Gets the blend factors and the corresponding blend positions currently set for this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientBlend(GpPathGradient *brush, GDIPCONST REAL *blend, GDIPCONST REAL *positions, INT count)	Status PathGradientBrush::SetBlend(IN const REAL* blendFactors, IN const REAL* blendPositions, IN INT count)	Sets the blend factors and the blend positions of this path gradient brush.
GpStatus WINGDIPAPI GdipGetPathGradientPresetBlendCount(GpPathGradient *brush, INT *count)	INT PathGradientBrush::GetInterpolationColorCount() const	Gets the number of preset colors currently specified for this path gradient brush.
GpStatus WINGDIPAPI GdipGetPathGradientPresetBlend(GpPathGradient *brush, ARGB *blend, REAL* positions, INT count)	Status PathGradientBrush::GetInterpolationColors(OUT Color* presetColors, OUT REAL* blendPositions, IN INT count) const	Gets the preset colors and blend positions currently specified for this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientPresetBlend(GpPathGradient *brush, GDIPCONST ARGB *blend, GDIPCONST REAL* positions, INT count)	Status PathGradientBrush::SetInterpolationColors(IN const Color* presetColors, IN const REAL* blendPositions, IN INT count)	Sets the preset colors and the blend positions of this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientSigmaBlend(GpPathGradient *brush, REAL focus, REAL scale)	Status PathGradientBrush::SetBlendBellShape(IN REAL focus, IN REAL scale = 1.0)	Sets the blend shape of this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientLinearBlend(GpPathGradient *brush, REAL focus, REAL scale)	Status PathGradientBrush::SetBlendTriangularShape(IN REAL focus, IN REAL scale = 1.0)	Sets the blend shape of this path gradient brush.
GpStatus WINGDIPAPI GdipGetPathGradientWrapMode(GpPathGradient *brush, GpWrapMode *wrapmode)	WrapMode PathGradientBrush::GetWrapMode() const	Gets the wrap mode currently set for this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientWrapMode(GpPathGradient *brush, GpWrapMode wrapmode)	Status PathGradientBrush::SetWrapMode(IN WrapMode wrapMode)	Sets the wrap mode of this path gradient brush.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetPathGradientTransform(GpPath Gradient *brush, GpMatrix *matrix)	Status PathGradientBrush::GetTransform(OUT Matrix *matrix) const	Gets transformation matrix of this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientTransform(GpPath Gradient *brush, GpMatrix *matrix)	Status PathGradientBrush::SetTransform(IN const Matrix* matrix)	Sets the transformation matrix of this path gradient brush.
GpStatus WINGDIPAPI GdipResetPathGradientTransform(GpPath Gradient* brush)	Status PathGradientBrush::ResetTransform()	Resets the transformation matrix of this path gradient brush to the identity matrix. This means that no transformation will take place.
GpStatus WINGDIPAPI GdipMultiplyPathGradientTransform(Gp PathGradient* brush, GDIPCONST GpMatrix *matrix, GpMatrixOrder order)	Status PathGradientBrush::MultiplyTransform(IN const Matrix* matrix, IN MatrixOrder order = MatrixOrderPrepend)	Updates the brush's transformation matrix with the product of itself and another matrix.
GpStatus WINGDIPAPI GdipTranslatePathGradientTransform(Gp PathGradient* brush, REAL dx, REAL dy, GpMatrixOrder order)	Status PathGradientBrush::TranslateTransform(IN REAL dx, IN REAL dy, IN MatrixOrder order = MatrixOrderPrepend)	Updates this brush's current transformation matrix with the product of itself and a translation matrix.
GpStatus WINGDIPAPI GdipScalePathGradientTransform(GpPathGradient* brush, REAL sx, REAL sy, GpMatrixOrder order)	Status PathGradientBrush::ScaleTransform(IN REAL sx, IN REAL sy, IN MatrixOrder order = MatrixOrderPrepend)	Updates this brush's current transformation matrix with the product of itself and a scaling matrix.
GpStatus WINGDIPAPI GdipRotatePathGradientTransform(GpPathGradient* brush, REAL angle, GpMatrixOrder order)	Status PathGradientBrush::RotateTransform(IN REAL angle, IN MatrixOrder order = MatrixOrderPrepend)	Updates this brush's current transformation matrix with the product of itself and a rotation matrix.
GpStatus WINGDIPAPI GdipGetPathGradientFocusScales(GpPathGradient *brush, REAL* xScale, REAL* yScale)	Status PathGradientBrush::GetFocusScales(OUT REAL* xScale, OUT REAL* yScale) const	Gets the focus scales of this path gradient brush.
GpStatus WINGDIPAPI GdipSetPathGradientFocusScales(GpPathGradient *brush, REAL xScale, REAL yScale)	Status PathGradientBrush::SetFocusScales(IN REAL xScale, IN REAL yScale)	Sets the focus scales of this path gradient brush

PathIterator Functions

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [GraphicsPathIterator](#) C++ class.

GraphicsPathIterator Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreatePathIter(GpPathIterator **iterator, GpPath* path)	GraphicsPathIterator::GraphicsPathIterator(IN const GraphicsPath* path)	Creates a new GraphicsPathIterator object and associates it with a GraphicsPath object.
GpStatus WINGDIPAPI GdipDeletePathIter(GpPathIterator *iterator)	GraphicsPathIterator::~GraphicsPathIterator()	Releases resources used by the GraphicsPathIterator object.
GpStatus WINGDIPAPI GdipPathIterNextSubpath(GpPathIterator* iterator, INT *resultCount, INT* startIndex, INT* endIndex, BOOL* isClosed)	INT GraphicsPathIterator::NextSubpath(OUT INT* startIndex, OUT INT* endIndex, OUT BOOL* isClosed)	Gets the starting index and the ending index of the next subpath (figure) in this iterator's associated path.
GpStatus WINGDIPAPI GdipPathIterNextSubpathPath(GpPathIterator* iterator, INT* resultCount, GpPath* path, BOOL* isClosed)	INT GraphicsPathIterator::NextSubpath(OUT const GraphicsPath* path, OUT BOOL* isClosed)	Gets the next figure (subpath) from this iterator's associated path.
GpStatus WINGDIPAPI GdipPathIterNextPathType(GpPathIterator* iterator, INT* resultCount, BYTE* pathType, INT* startIndex, INT* endIndex)	INT GraphicsPathIterator::NextPathType(OUT BYTE* pathType, OUT INT* startIndex, OUT INT* endIndex)	Gets the starting index and the ending index of the next group of data points that all have the same type.
GpStatus WINGDIPAPI GdipPathIterNextMarker(GpPathIterator * iterator, INT *resultCount, INT* startIndex, INT* endIndex)	INT GraphicsPathIterator::NextMarker(OUT INT* startIndex, OUT INT* endIndex)	Gets the starting index and the ending index of the next marker-delimited section in this iterator's associated path.
GpStatus WINGDIPAPI GdipPathIterNextMarkerPath(GpPathIterator* iterator, INT* resultCount, GpPath* path)	INT GraphicsPathIterator::NextMarker(OUT const GraphicsPath* path)	Gets the next marker-delimited section of this iterator's associated path.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipPathIterGetCount(GpPathIterator* iterator, INT* count)	INT GraphicsPathIterator::GetCount() const	Returns the number of data points in the path.
GpStatus WINGDIPAPI GdipPathIterGetSubpathCount(GpPathIterator* iterator, INT* count)	INT GraphicsPathIterator::GetSubpathCount() const	Returns the number of subpaths (also called figures) in the path.
GpStatus WINGDIPAPI GdipPathIterIsValid(GpPathIterator* iterator, BOOL* valid)	Not called by wrapper methods.	This function passes a Boolean value that indicates whether the path iterator specified by the <i>iterator</i> parameter is valid. The output parameter <i>valid</i> receives the result.
GpStatus WINGDIPAPI GdipPathIterHasCurve(GpPathIterator* iterator, BOOL* hasCurve)	BOOL GraphicsPathIterator::HasCurve() const	Determines whether the path has any curves.
GpStatus WINGDIPAPI GdipPathIterRewind(GpPathIterator* iterator)	VOID GraphicsPathIterator::Rewind()	Rewinds this iterator to the beginning of its associated path.
GpStatus WINGDIPAPI GdipPathIterEnumerate(GpPathIterator* iterator, INT* resultCount, GpPointF* points, BYTE* types, INT count)	INT GraphicsPathIterator::Enumerate(OUT PointF *points, OUT BYTE *types, IN INT count)	Copies the path's data points to a PointF array and copies the path's point types to a BYTE array.
GpStatus WINGDIPAPI GdipPathIterCopyData(GpPathIterator* iterator, INT* resultCount, GpPointF* points, BYTE* types, INT startIndex, INT endIndex)	INT GraphicsPathIterator::CopyData(OUT PointF* points, OUT BYTE* types, IN INT startIndex, IN INT endIndex)	Copies a subset of the path's data points to a PointF array and copies a subset of the path's point types to a BYTE array.

Pen Functions (GDI+)

12/18/2020 • 6 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [Pen](#) C++ class.

Pen Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreatePen1(ARGB color, REAL width, GpUnit unit, GpPen **pen)	Pen::Pen(IN const Color& color, IN REAL width = 1.0f)	Creates a Pen object that uses a specified color and width. The <i>unit</i> parameter of the flat function is a member of the Unit enumeration that specifies the unit of measure for the width of the pen.
GpStatus WINGDIPAPI GdipCreatePen2(GpBrush *brush, REAL width, GpUnit unit, GpPen **pen)	Pen::Pen(IN const Brush* brush, IN REAL width = 1.0f)	Creates a Pen object that uses the attributes of a brush and a real number to set the width of this Pen object. The <i>unit</i> parameter of the flat function is a member of the Unit enumeration that specifies the unit of measure for the width of the pen.
GpStatus WINGDIPAPI GdipClonePen(GpPen *pen, GpPen **clonepen)	Pen* Pen::Clone() const	Copies a Pen object.
GpStatus WINGDIPAPI GdipDeletePen(GpPen *pen)	Pen::~Pen()	Releases resources used by the Pen object.
GpStatus WINGDIPAPI GdipSetPenWidth(GpPen *pen, REAL width)	Status Pen::SetWidth(IN REAL width)	Sets the width for this Pen object.
GpStatus WINGDIPAPI GdipGetPenWidth(GpPen *pen, REAL *width)	REAL Pen::GetWidth() const	Gets the width currently set for this Pen object.
GpStatus WINGDIPAPI GdipSetPenUnit(GpPen *pen, GpUnit unit)	Not called by wrapper methods.	This function sets the unit of measure for the pen specified by the <i>pen</i> parameter to the value specified by the <i>unit</i> parameter. The <i>unit</i> parameter is a member of the Unit enumeration that specifies the unit of measure for the width of the pen.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetPenUnit(GpPen *pen, GpUnit *unit)	Not called by wrapper methods.	This function gets the unit of measure for the pen specified by the <i>pen</i> parameter. The <i>unit</i> parameter receives a member of the Unit enumeration that indicates the unit of measure for the width of the pen.
GpStatus WINGDIPAPI GdipSetPenLineCap197819(GpPen *pen, GpLineCap startCap, GpLineCap endCap, GpDashCap dashCap)	Status Pen::SetLineCap(IN LineCap startCap, IN LineCap endCap, IN DashCap dashCap)	Sets the cap styles for the start, end, and dashes in a line drawn with this pen.
GpStatus WINGDIPAPI GdipSetPenStartCap(GpPen *pen, GpLineCap startCap)	Status Pen::SetStartCap(IN LineCap startCap)	Sets the start cap for this Pen object.
GpStatus WINGDIPAPI GdipSetPenEndCap(GpPen *pen, GpLineCap endCap)	Status Pen::SetEndCap(IN LineCap endCap)	Sets the end cap for this Pen object.
GpStatus WINGDIPAPI GdipSetPenDashCap197819(GpPen *pen, GpDashCap dashCap)	Status Pen::SetDashCap(IN DashCap dashCap)	Sets the dash cap style for this Pen object.
GpStatus WINGDIPAPI GdipGetPenStartCap(GpPen *pen, GpLineCap *startCap)	LineCap Pen::GetStartCap()const	Gets the start cap currently set for this Pen object.
GpStatus WINGDIPAPI GdipGetPenEndCap(GpPen *pen, GpLineCap *endCap)	LineCap Pen::GetEndCap()const	Gets the end cap currently set for this Pen object.
GpStatus WINGDIPAPI GdipGetPenDashCap197819(GpPen *pen, GpDashCap *dashCap)	DashCap Pen::GetDashCap() const	Gets the dash cap style currently set for this Pen object.
GpStatus WINGDIPAPI GdipSetPenLineJoin(GpPen *pen, GpLineJoin lineJoin)	Status Pen::SetLineJoin(IN LineJoin lineJoin)	Sets the line join for this Pen object.
GpStatus WINGDIPAPI GdipGetPenLineJoin(GpPen *pen, GpLineJoin *lineJoin)	LineJoin Pen::GetLineJoin() const	Gets the line join for this Pen object.
GpStatus WINGDIPAPI GdipSetPenCustomStartCap(GpPen *pen, GpCustomLineCap* customCap)	Status Pen::SetCustomStartCap(IN const CustomLineCap* customCap)	Sets the custom start cap for this Pen object.
GpStatus WINGDIPAPI GdipGetPenCustomStartCap(GpPen *pen, GpCustomLineCap** customCap)	Pen::GetCustomStartCap(OUT CustomLineCap* customCap) const	Gets the custom start cap for this Pen object.
GpStatus WINGDIPAPI GdipSetPenCustomEndCap(GpPen *pen, GpCustomLineCap* customCap)	Status Pen::SetCustomEndCap(IN const CustomLineCap* customCap)	Sets the custom end cap currently set for this Pen object

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetPenCustomEndCap(GpPen *pen, GpCustomLineCap** customCap)	Status Pen::GetCustomEndCap(OUT CustomLineCap* customCap) const	Gets the custom end cap currently set for this Pen object.
GpStatus WINGDIPAPI GdipSetPenMiterLimit(GpPen *pen, REAL miterLimit)	Status Pen::SetMiterLimit(IN REAL miterLimit)	Sets the miter length currently set for this Pen object.
GpStatus WINGDIPAPI GdipGetPenMiterLimit(GpPen *pen, REAL *miterLimit)	REAL Pen::GetMiterLimit() const	Gets the miter length currently set for this Pen object.
GpStatus WINGDIPAPI GdipSetPenMode(GpPen *pen, GpPenAlignment penMode)	Status Pen::SetAlignment(IN PenAlignment penAlignment)	Sets the alignment currently set for this Pen object.
GpStatus WINGDIPAPI GdipGetPenMode(GpPen *pen, GpPenAlignment *penMode)	PenAlignment Pen::GetAlignment() const	Sets the alignment currently set for this Pen object.
GpStatus WINGDIPAPI GdipSetPenTransform(GpPen *pen, GpMatrix *matrix)	Status Pen::SetTransform(IN const Matrix* matrix)	Sets the world transformation matrix currently set for this Pen object.
GpStatus WINGDIPAPI GdipGetPenTransform(GpPen *pen, GpMatrix *matrix)	Pen::GetTransform(OUT Matrix* matrix) const	Gets the world transformation matrix currently set for this Pen object.
GpStatus WINGDIPAPI GdipResetPenTransform(GpPen *pen)	Status Pen::ResetTransform()	Sets the world transformation matrix of this Pen object to the identity matrix.
GpStatus WINGDIPAPI GdipMultiplyPenTransform(GpPen *pen, GDIPCONST GpMatrix *matrix, GpMatrixOrder order)	Status Pen::MultiplyTransform(IN const Matrix* matrix, IN MatrixOrder order = MatrixOrderPrepend)	Updates the world transformation matrix of this Pen object with the product of itself and another matrix.
GpStatus WINGDIPAPI GdipTranslatePenTransform(GpPen *pen, REAL dx, REAL dy, GpMatrixOrder order)	Not called by wrapper methods.	
GpStatus WINGDIPAPI GdipScalePenTransform(GpPen *pen, REAL sx, REAL sy, GpMatrixOrder order)	Status Pen::ScaleTransform(IN REAL sx, IN REAL sy, IN MatrixOrder order = MatrixOrderPrepend)	Sets the Pen object's world transformation matrix equal to the product of itself and a scaling matrix.
GpStatus WINGDIPAPI GdipRotatePenTransform(GpPen *pen, REAL angle, GpMatrixOrder order)	Status Pen::RotateTransform(IN REAL angle, IN MatrixOrder order = MatrixOrderPrepend)	Updates the world transformation matrix of this Pen object with the product of itself and a rotation matrix.
GpStatus WINGDIPAPI GdipSetColor(GpPen *pen, ARGB argb)	Status Pen::SetColor(IN const Color& color)	Sets the color for this Pen object.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetPenColor(GpPen *pen, ARGB *argb)	Status Pen::GetColor(OUT Color* color) const	Gets the color for this Pen object.
GpStatus WINGDIPAPI GdipSetPenBrushFill(GpPen *pen, GpBrush *brush)	Status Pen::SetBrush(IN const Brush* brush)	Sets the Brush object that a pen uses to fill a line.
GpStatus WINGDIPAPI GdipGetPenBrushFill(GpPen *pen, GpBrush **brush)	Brush* Pen::GetBrush() const	Gets the Brush object that a pen uses to fill a line.
GpStatus WINGDIPAPI GdipGetPenFillType(GpPen *pen, GpPenType* type)	PenType Pen::GetPenType() const	Gets the type currently set for this Pen object.
GpStatus WINGDIPAPI GdipGetPenDashStyle(GpPen *pen, GpDashStyle *dashstyle)	DashStyle Pen::GetDashStyle() const	Gets the dash style currently set for this Pen object.
GpStatus WINGDIPAPI GdipSetPenDashStyle(GpPen *pen, GpDashStyle dashstyle)	Status Pen::SetDashStyle(IN DashStyle dashStyle)	Sets the dash style currently set for this Pen object.
GpStatus WINGDIPAPI GdipGetPenDashOffset(GpPen *pen, REAL *offset)	REAL Pen::GetDashOffset() const	Gets the distance from the start of the line to the start of the first space in a dashed line.
GpStatus WINGDIPAPI GdipSetPenDashOffset(GpPen *pen, REAL offset)	Status Pen::SetDashOffset(IN REAL dashOffset)	Sets the distance from the start of the line to the start of the first space in a dashed line.
GpStatus WINGDIPAPI GdipGetPenDashCount(GpPen *pen, INT *count)	INT Pen::GetDashPatternCount() const	Gets the number of elements in a dash pattern array.
GpStatus WINGDIPAPI GdipSetPenDashArray(GpPen *pen, GDIPCONST REAL *dash, INT count)	Status Pen::SetDashPattern(IN const REAL* dashArray, IN INT count)	Sets an array of custom dashes and spaces currently set for this Pen object.
GpStatus WINGDIPAPI GdipGetPenDashArray(GpPen *pen, REAL *dash, INT count)	Status Pen::GetDashPattern(OUT REAL* dashArray, IN INT count) const	Gets an array of custom dashes and spaces currently set for this Pen object.
GpStatus WINGDIPAPI GdipGetPenCompoundCount(GpPen *pen, INT *count)	INT Pen::GetCompoundArrayCount() const	Gets the number of elements in a compound array.
GpStatus WINGDIPAPI GdipSetPenCompoundArray(GpPen *pen, GDIPCONST REAL *dash, INT count)	Status Pen::SetCompoundArray(IN const REAL* compoundArray, IN INT count)	Sets the compound array currently set for this Pen object.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetPenCompoundArray(GpPen *pen, REAL *dash, INT count)	Status Pen::GetCompoundArray(OUT REAL* compoundArray, IN INT count) const	Gets the compound array currently set for this Pen object.

Region Functions (GDI+)

12/18/2020 • 5 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [Region](#) C++ class.

Region Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreateRegion(GpRegion **region)	Region::Region()	Creates a region that is infinite. This is the default constructor.
GpStatus WINGDIPAPI GdipCreateRegionRect(GDIPCONST GpRectF *rect, GpRegion **region)	Region::Region(IN const RectF& rect)	Creates a region that is defined by a rectangle.
GpStatus WINGDIPAPI GdipCreateRegionRectI(GDIPCONST GpRect *rect, GpRegion **region)	Region::Region(IN const Rect& rect)	Creates a region that is defined by a rectangle.
GpStatus WINGDIPAPI GdipCreateRegionPath(GpPath *path, GpRegion **region)	Region::Region(IN const GraphicsPath* path)	Creates a region that is defined by a GraphicsPath object and has a fill mode that is contained in the GraphicsPath object.
GpStatus WINGDIPAPI GdipCreateRegionRgnData(GDIPCONST BYTE *regionData, INT size, GpRegion **region)	Region::Region(IN const BYTE* regionData, IN INT size)	Creates a region that is defined by data obtained from another region.
GpStatus WINGDIPAPI GdipCreateRegionHrgn(HRGN hRgn, GpRegion **region)	Region::Region(IN Hrgn hRgn)	Creates a region that is identical to the region that is specified by a handle to a GDI region.
GpStatus WINGDIPAPI GdipCloneRegion(GpRegion *region, GpRegion **cloneRegion)	Region* Region::Clone() const	Makes a copy of this Region object and returns the address of the new Region object.
GpStatus WINGDIPAPI GdipDeleteRegion(GpRegion *region)	Region::~Region()	Releases resources used by the Region object.
GpStatus WINGDIPAPI GdipSetInfinite(GpRegion *region)	Status Region::MakeInfinite()	Updates this region to an infinite region.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSetEmpty(GpRegion *region)	Status Region::MakeEmpty()	Updates this region to an empty region. In other words, the region occupies no space on the display device.
GpStatus WINGDIPAPI GdipCombineRegionRect(GpRegion *region, GDIPCONST GpRectF *rect, CombineMode combineMode)	Status Region::Intersect(IN const RectF& rect)	Updates this region to the portion of itself that intersects the specified rectangle's interior. The <i>combineMode</i> parameter in the flat function is a member of the CombineMode enumeration that specifies how the region and rectangle are combined.
GpStatus WINGDIPAPI GdipCombineRegionRectI(GpRegion *region, GDIPCONST GpRect *rect, CombineMode combineMode)	Status Region::Intersect(IN const Rect& rect)	Updates this region to the portion of itself that intersects the specified rectangle's interior. The <i>combineMode</i> parameter in the flat function is a member of the CombineMode enumeration that specifies how the region and rectangle are combined.
GpStatus WINGDIPAPI GdipCombineRegionPath(GpRegion *region, GpPath *path, CombineMode combineMode)	Status Region::Intersect(IN const GraphicsPath* path)	Updates this region to the portion of itself that intersects the specified path's interior. The <i>combineMode</i> parameter in the flat function is a member of the CombineMode enumeration that specifies how the region and path are combined.
GpStatus WINGDIPAPI GdipCombineRegionRegion(GpRegion *region, GpRegion *region2, CombineMode combineMode)	Region::Intersect(IN const Region* region)	Updates this region to the portion of itself that intersects another region. The <i>combineMode</i> parameter in the flat function is a member of the CombineMode enumeration that specifies how the regions are combined.
GpStatus WINGDIPAPI GdipTranslateRegion(GpRegion *region, REAL dx, REAL dy)	Region::Translate(IN REAL dx, IN REAL dy)	Offsets this region by specified amounts in the horizontal and vertical directions.
GpStatus WINGDIPAPI GdipTranslateRegionI(GpRegion *region, INT dx, INT dy)	Status Region::Translate(IN INT dx, IN INT dy)	Offsets this region by specified amounts in the horizontal and vertical directions.
GpStatus WINGDIPAPI GdipTransformRegion(GpRegion *region, GpMatrix *matrix)	Region::Transform(IN const Matrix* matrix)	Transforms this region by multiplying each of its data points by a specified matrix.
GpStatus WINGDIPAPI GdipGetRegionBounds(GpRegion *region, GpGraphics *graphics, GpRectF *rect)	Status Region::GetBounds(OUT RectF* rect, IN const Graphics* g) const	Gets a rectangle that encloses this region.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetRegionBoundsI(GpRegion *region, GpGraphics *graphics, GpRect *rect)	Status Region::GetBounds(OUT Rect* rect, IN const Graphics* g) const	Gets a rectangle that encloses this region.
GpStatus WINGDIPAPI GdipGetRegionHrgn(GpRegion *region, GpGraphics *graphics, HRGN *hRgn)	HRGN Region::GetHrgn(IN const Graphics* g) const	Creates a GDI region from this region.
GpStatus WINGDIPAPI GdiplIsEmptyRegion(GpRegion *region, GpGraphics *graphics, BOOL *result)	BOOL Region::IsEmpty(IN const Graphics *g) const	Determines whether this region is empty.
GpStatus WINGDIPAPI GdiplIsInfiniteRegion(GpRegion *region, GpGraphics *graphics, BOOL *result)	BOOL Region::IsInfinite(IN const Graphics *g) const	Determines whether this region is infinite.
GpStatus WINGDIPAPI GdiplIsEqualRegion(GpRegion *region, GpRegion *region2, GpGraphics *graphics, BOOL *result)	BOOL Region::Equals(IN const Region* region, IN const Graphics* g) const	Determines whether this region is equal to a specified region.
GpStatus WINGDIPAPI GdipGetRegionDataSize(GpRegion *region, UINT *bufferSize)	UINT Region::GetDataSize() const	Gets the number of bytes of data that describes this region.
GpStatus WINGDIPAPI GdipGetRegionData(GpRegion *region, BYTE * buffer, UINT bufferSize, UINT *sizeFilled)	Status Region::GetData(OUT BYTE* buffer, IN UINT bufferSize, OUT UINT* sizeFilled) const	Gets data that describes this region.
GpStatus WINGDIPAPI GdiplIsVisibleRegionPoint(GpRegion *region, REAL x, REAL y, GpGraphics *graphics, BOOL *result)	BOOL Region::IsVisible(IN const PointF& point, IN const Graphics* g) const	Determines whether a point is inside this region. The <i>x</i> and <i>y</i> parameters in the flat function specify the <i>x</i> and <i>y</i> coordinates of a point that corresponds to the <i>point</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdiplIsVisibleRegionPointI(GpRegion *region, INT x, INT y, GpGraphics *graphics, BOOL *result)	BOOL Region::IsVisible(IN const Point& point, IN const Graphics* g) const	Determines whether a point is inside this region. The <i>x</i> and <i>y</i> parameters in the flat function specify the <i>x</i> and <i>y</i> coordinates of a point that corresponds to the <i>point</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdiplIsVisibleRegionRect(GpRegion *region, REAL x, REAL y, REAL width, REAL height, GpGraphics *graphics, BOOL *result)	BOOL Region::IsVisible(IN const RectF& rect, IN const Graphics* g) const	Determines whether a rectangle intersects this region. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters in the flat function specify a rectangle that corresponds to the <i>rect</i> parameter in the wrapper method.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdiplIsVisibleRegionRectI(GpRegion *region, INT x, INT y, INT width, INT height, GpGraphics *graphics, BOOL *result)	BOOL Region::IsVisible(IN const Rect& rect, IN const Graphics* g) const	Determines whether a rectangle intersects this region. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters in the flat function specify a rectangle that corresponds to the <i>rect</i> parameter in the wrapper method.
GpStatus WINGDIPAPI GdipGetRegionScansCount(GpRegion *region, UINT* count, GpMatrix* matrix)	UINT Region::GetRegionScansCount(IN const Matrix* matrix) const	Gets the number of rectangles that approximate this region. The region is transformed by a specified matrix before the rectangles are calculated.
GpStatus WINGDIPAPI GdipGetRegionScans(GpRegion *region, GpRectF* rects, INT* count, GpMatrix* matrix)	Status Region::GetRegionScans(IN const Matrix* matrix, OUT RectF* rects, IN OUT INT* count) const	Gets an array of rectangles that approximate this region. The region is transformed by a specified matrix before the rectangles are calculated.
GpStatus WINGDIPAPI GdipGetRegionScansI(GpRegion *region, GpRect* rects, INT* count, GpMatrix* matrix)	Status Region::GetRegionScans(IN const Matrix* matrix, OUT Rect* rects, IN OUT INT* count) const	Gets an array of rectangles that approximate this region. The region is transformed by a specified matrix before the rectangles are calculated.

SolidBrush Functions

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. We recommend not to call the functions in the flat API directly. Whenever you make calls to GDI+, we recommend that you do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more info on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the **SolidBrush** C++ class.

Brush Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreateSolidFill(ARGB color, GpSolidFill **brush)	SolidBrush::SolidBrush(IN const Color& color)	Creates a SolidBrush object based on a color
GpStatus WINGDIPAPI GdipSetSolidFillColor(GpSolidFill *brush, ARGB color)	SolidBrush::SetColor(IN const Color& color)	Sets the color of this solid brush
GpStatus WINGDIPAPI GdipGetSolidFillColor(GpSolidFill *brush, ARGB *color)	SolidBrush::GetColor(OUT Color* color) const	Gets the color of this solid brush

String Format Functions

11/2/2020 • 4 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdipplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [StringFormat](#) C++ class.

StringFormat Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreateStringFormat(INT formatAttributes, LANGID language, GpStringFormat **format)	StringFormat::StringFormat(IN INT formatFlags = 0, IN LANGID language = LANG_NEUTRAL)	Creates a StringFormat object based on string format flags and a language.
GpStatus WINGDIPAPI GdipStringFormatGetGenericDefault(Gp StringFormat **format)	StringFormat* StringFormat::GenericDefault()	Creates a generic, default StringFormat object.
GpStatus WINGDIPAPI GdipStringFormatGetGenericTypographi c(GpStringFormat **format)	StringFormat* StringFormat::GenericTypographic()	Creates a generic, typographic StringFormat object.
GpStatus WINGDIPAPI GdipDeleteStringFormat(GpStringForma t *format)	StringFormat::~StringFormat()	Releases resources used by the StringFormat object.
GpStatus WINGDIPAPI GdipCloneStringFormat(GDIPCONST GpStringFormat *format, GpStringFormat **newFormat)	StringFormat::StringFormat(IN const StringFormat *format)	Creates a StringFormat object from another StringFormat object.
GpStatus WINGDIPAPI GdipSetStringFormatFlags(GpStringFor mat *format, INT flags)	Status StringFormat::SetFormatFlags(IN INT flags)	Sets the format flags for this StringFormat object. The format flags determine most of the characteristics of a StringFormat object.
GpStatus WINGDIPAPI GdipGetStringFormatFlags(GDIPCONST GpStringFormat *format, INT *flags)	INT StringFormat::GetFormatFlags() const	Gets the string format flags for this StringFormat object.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSetStringFormatAlign(GpStringFormat *format, StringAlignment align)	Status StringFormat::SetAlignment(IN StringAlignment align)	Sets the line alignment of this StringFormat object in relation to the origin of the layout rectangle. The line alignment setting specifies how to align the string vertically in the layout rectangle. The layout rectangle is used to position the displayed string.
GpStatus WINGDIPAPI GdipGetStringFormatAlign(GDIPCONST GpStringFormat *format, StringAlignment *align)	StringAlignment StringFormat::GetAlignment() const	Gets an element of the StringAlignment enumeration that indicates the character alignment of this StringFormat object in relation to the origin of the layout rectangle. A layout rectangle is used to position the displayed string.
GpStatus WINGDIPAPI GdipSetStringFormatLineAlign(GpStringFormat *format, StringAlignment align)	Status StringFormat::SetLineAlignment(IN StringAlignment align)	Sets the line alignment of this StringFormat object in relation to the origin of the layout rectangle. The line alignment setting specifies how to align the string vertically in the layout rectangle. The layout rectangle is used to position the displayed string.
GpStatus WINGDIPAPI GdipGetStringFormatLineAlign(GDIPCONST GpStringFormat *format, StringAlignment *align)	StringAlignment StringFormat::GetLineAlignment() const	Gets an element of the StringAlignment enumeration that indicates the line alignment of this StringFormat object in relation to the origin of the layout rectangle. The line alignment setting specifies how to align the string vertically in the layout rectangle. The layout rectangle is used to position the displayed string.
GpStatus WINGDIPAPI GdipSetStringFormatTrimming(GpStringFormat *format, StringTrimming trimming)	Status StringFormat::SetTrimming(IN StringTrimming trimming)	Sets the trimming style for this StringFormat object. The trimming style determines how to trim a string so that it fits into the layout rectangle.
GpStatus WINGDIPAPI GdipGetStringFormatTrimming(GDIPCONST GpStringFormat *format, StringTrimming *trimming)	StringTrimming StringFormat::GetTrimming() const	Gets an element of the StringTrimming enumeration that indicates the trimming style of this StringFormat object. The trimming style determines how to trim characters from a string that is too large to fit in the layout rectangle.
GpStatus WINGDIPAPI GdipSetStringFormatHotkeyPrefix(GpStringFormat *format, INT hotkeyPrefix)	Status StringFormat::SetHotkeyPrefix(IN HotkeyPrefix hotkeyPrefix)	Sets the type of processing that is performed on a string when the hot key prefix, an ampersand (&), is encountered. The ampersand is called the hot key prefix and can be used to designate certain keys as hot keys.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipGetStringFormatHotkeyPrefix(GDIPCONST GpStringFormat *format, INT *hotkeyPrefix)	HotkeyPrefix StringFormat::GetHotkeyPrefix() const	Gets an element of the HotkeyPrefix enumeration that indicates the type of processing that is performed on a string when a hot key prefix, an ampersand (&), is encountered.
GpStatus WINGDIPAPI GdipSetStringFormatTabStops(GpStringFormat *format, REAL firstTabOffset, INT count, GDIPCONST REAL *tabStops)	Status StringFormat::SetTabStops(IN REAL firstTabOffset, IN INT count, IN const REAL *tabStops)	Sets the offsets for tab stops in this StringFormat object.
GpStatus WINGDIPAPI GdipGetStringFormatTabStops(GDIPCONST GpStringFormat *format, INT count, REAL *firstTabOffset, REAL *tabStops)	Status StringFormat::GetTabStops(IN INT count, OUT REAL *firstTabOffset, OUT REAL *tabStops) const	Gets the offsets of the tab stops in this StringFormat object.
GpStatus WINGDIPAPI GdipGetStringFormatTabStopCount(GDIPCONST GpStringFormat *format, INT *count)	INT StringFormat::GetTabStopCount() const	Gets the number of tab-stop offsets in this StringFormat object.
GpStatus WINGDIPAPI GdipSetStringFormatDigitSubstitution(GpStringFormat *format, LANGID language, StringDigitSubstitute substitute)	Status StringFormat::SetDigitSubstitution(IN LANGID language, IN StringDigitSubstitute substitute)	Sets the digit substitution method and the language that corresponds to the digit substitutes.
GpStatus WINGDIPAPI GdipGetStringFormatDigitSubstitution(GDIPCONST GpStringFormat *format, LANGID *language, StringDigitSubstitute *substitute)	StringDigitSubstitute StringFormat::GetDigitSubstitutionMethod() const	gets an element of the StringDigitSubstitute enumeration that indicates the digit substitution method that is used by this StringFormat object. The <i>language</i> parameter in the flat function is a 16-bit value that specifies the language to use.
GpStatus WINGDIPAPI GdipGetStringFormatMeasurableCharacterRangeCount(GDIPCONST GpStringFormat *format, INT *count)	INT StringFormat::GetMeasurableCharacterRangeCount()	gets the number of measurable character ranges that are currently set. The character ranges that are set can be measured in a string by using the Graphics::MeasureCharacterRanges method.
GpStatus WINGDIPAPI GdipSetStringFormatMeasurableCharacterRanges(GpStringFormat *format, INT rangeCount, GDIPCONST CharacterRange *ranges)	Status StringFormat::SetMeasurableCharacterRanges(IN INT rangeCount, IN const CharacterRange *ranges)	Sets a series of character ranges for this StringFormat object that, when in a string, can be measured by the Graphics::MeasureCharacterRanges method.

Text Functions

11/2/2020 • 2 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#). The following flat API functions are wrapped by the **Graphics** Text C++ class.

Text Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipDrawString(GpGraphics *graphics, GDIPCONST WCHAR *string, INT length, GDIPCONST GpFont *font, GDIPCONST RectF *layoutRect, GDIPCONST GpStringFormat *stringFormat, GDIPCONST GpBrush *brush)	Status Graphics::DrawString(IN const WCHAR *string, IN INT length, IN const Font *font, IN const RectF &layoutRect, IN const StringFormat *stringFormat, IN const Brush *brush)	Draws a string based on a font, a layout rectangle, and a format.
GpStatus WINGDIPAPI GdipMeasureString(GpGraphics *graphics, GDIPCONST WCHAR *string, INT length, GDIPCONST GpFont *font, GDIPCONST RectF *layoutRect, GDIPCONST GpStringFormat *stringFormat, RectF *boundingBox, INT *codepointsFitted, INT *linesFilled)	Status Graphics::MeasureString(IN const WCHAR *string, IN INT length, IN const Font *font, IN const RectF &layoutRect, IN const StringFormat *stringFormat, OUT RectF *boundingBox, OUT INT *codepointsFitted = 0, OUT INT *linesFilled = 0) const	Measures the extent of the string in the specified font, format, and layout rectangle.
GpStatus WINGDIPAPI GdipMeasureCharacterRanges(GpGraphics *graphics, GDIPCONST WCHAR *string, INT length, GDIPCONST GpFont *font, GDIPCONST RectF &layoutRect, GDIPCONST GpStringFormat *stringFormat, INT regionCount, GpRegion **regions)	Status Graphics::MeasureCharacterRanges(IN const WCHAR *string, IN INT length, IN const Font *font, IN const RectF &layoutRect, IN const StringFormat *stringFormat, IN INT regionCount, OUT Region *regions) const	Gets a set of regions each of which bounds a range of character positions within a string.
GpStatus WINGDIPAPI GdipDrawDriverString(GpGraphics *graphics, GDIPCONST UINT16 *text, INT length, GDIPCONST GpFont *font, GDIPCONST GpBrush *brush, GDIPCONST PointF *positions, INT flags, GDIPCONST GpMatrix *matrix)	Status Graphics::DrawDriverString(IN const UINT16 *text, IN INT length, IN const Font *font, IN const Brush *brush, IN const PointF *positions, IN INT flags, IN const Matrix *matrix)	Draws characters at the specified positions. The method gives the client complete control over the appearance of text. The method assumes that the client has already set up the format and layout to be applied.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipMeasureDriverString(GpGraphics *graphics, GDIPCONST UINT16 *text, INT length, GDIPCONST GpFont *font, GDIPCONST PointF *positions, INT flags, GDIPCONST GpMatrix *matrix, RectF *boundingBox)	Status Graphics::MeasureDriverString(IN const UINT16 *text, IN INT length, IN const Font *font, IN const PointF *positions, IN INT flags, IN const Matrix *matrix, OUT RectF *boundingBox) const	Measures the bounding box for the specified characters and their corresponding positions.

Texture Brush Functions

11/2/2020 • 3 minutes to read • [Edit Online](#)

Windows GDI+ exposes a flat API that consists of about 600 functions, which are implemented in Gdiplus.dll and declared in Gdiplusflat.h. The functions in the GDI+ flat API are wrapped by a collection of about 40 C++ classes. It is recommended that you do not directly call the functions in the flat API. Whenever you make calls to GDI+, you should do so by calling the methods and functions provided by the C++ wrappers. Microsoft Product Support Services will not provide support for code that calls the flat API directly. For more information on using these wrapper methods, see [GDI+ Flat API](#).

The following flat API functions are wrapped by the [TextureBrush](#) C++ class.

TextureBrush Functions and Corresponding Wrapper Methods

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipCreateTexture(GpImage *image, GpWrapMode wrapmode, GpTexture **texture)	TextureBrush::TextureBrush(IN Image* image, IN WrapMode wrapMode = WrapModeTile)	Creates a TextureBrush object based on an image and a wrap mode. The size of the brush defaults to the size of the image, so the entire image is used by the brush.
GpStatus WINGDIPAPI GdipCreateTexture2(GpImage *image, GpWrapMode wrapmode, REAL x, REAL y, REAL width, REAL height, GpTexture **texture)	TextureBrush::TextureBrush(IN Image* image, IN WrapMode wrapMode, IN REAL dstX, IN REAL dstY, IN REAL dstWidth, IN REAL dstHeight)	Creates a TextureBrush object based on an image, a wrap mode, and a defining set of coordinates.
GpStatus WINGDIPAPI GdipCreateTexture1A(GpImage *image, GDIPCONST GpImageAttributes *imageAttributes, REAL x, REAL y, REAL width, REAL height, GpTexture **texture)	TextureBrush::TextureBrush(IN Image *image, IN const RectF &dstRect, IN const ImageAttributes *imageAttributes = NULL)	Creates a TextureBrush object based on an image, a defining rectangle, and a set of image properties. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters of the flat function define a rectangle that corresponds to the <i>dstRect</i> parameter of the wrapper method.
GpStatus WINGDIPAPI GdipCreateTexture2I(GpImage *image, GpWrapMode wrapmode, INT x, INT y, INT width, INT height, GpTexture **texture)	TextureBrush::TextureBrush(IN Image* image, IN WrapMode wrapMode, IN INT dstX, IN INT dstY, IN INT dstWidth, IN INT dstHeight)	Creates a TextureBrush object based on an image, a wrap mode, and a defining set of coordinates.
GpStatus WINGDIPAPI GdipCreateTexture1AI(GpImage *image, GDIPCONST GpImageAttributes *imageAttributes, INT x, INT y, INT width, INT height, GpTexture **texture)	TextureBrush::TextureBrush(IN Image *image, IN const Rect &dstRect, IN const ImageAttributes *imageAttributes = NULL)	Creates a TextureBrush object based on an image, a defining rectangle, and a set of image properties. The <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> parameters of the flat function define a rectangle that corresponds to the <i>dstRect</i> parameter of the wrapper method.
GpStatus WINGDIPAPI GdipGetTextureTransform(GpTexture *brush, GpMatrix *matrix)	StatusTextureBrush::GetTransform(OUT Matrix* matrix) const	Gets the transformation matrix of this texture brush.

FLAT FUNCTION	WRAPPER METHOD	REMARKS
GpStatus WINGDIPAPI GdipSetTextureTransform(GpTexture *brush, GDIPCONST GpMatrix *matrix)	Status TextureBrush::SetTransform(IN const Matrix* matrix)	Sets the transformation matrix of this texture brush.
GpStatus WINGDIPAPI GdipResetTextureTransform(GpTexture* brush)	Status TextureBrush::ResetTransform()	Resets the transformation matrix of this texture brush to the identity matrix. This means that no transformation takes place.
GpStatus WINGDIPAPI GdipMultiplyTextureTransform(GpTexture* brush, GDIPCONST GpMatrix *matrix, GpMatrixOrder order)	Status TextureBrush::MultiplyTransform(IN const Matrix* matrix, IN MatrixOrder order = MatrixOrderPrepend)	Updates this brush's transformation matrix with the product of itself and another matrix.
GpStatus WINGDIPAPI GdipTranslateTextureTransform(GpTexture* brush, REAL dx, REAL dy, GpMatrixOrder order)	Status TextureBrush::TranslateTransform(IN REAL dx, IN REAL dy, IN MatrixOrder order = MatrixOrderPrepend)	Updates this brush's current transformation matrix with the product of itself and a translation matrix.
GpStatus WINGDIPAPI GdipScaleTextureTransform(GpTexture* brush, REAL sx, REAL sy, GpMatrixOrder order)	Status TextureBrush::ScaleTransform(IN REAL sx, IN REAL sy, IN MatrixOrder order = MatrixOrderPrepend)	Updates this texture brush's current transformation matrix with the product of itself and a scaling matrix.
GpStatus WINGDIPAPI GdipRotateTextureTransform(GpTexture* brush, REAL angle, GpMatrixOrder order)	Status TextureBrush::RotateTransform(IN REAL angle, IN MatrixOrder order = MatrixOrderPrepend)	Updates this texture brush's current transformation matrix with the product of itself and a rotation matrix.
GpStatus WINGDIPAPI GdipSetTextureWrapMode(GpTexture *brush, GpWrapMode wrapmode)	Status TextureBrush::SetWrapMode(IN WrapMode wrapMode)	Sets the wrap mode of this texture brush.
GpStatus WINGDIPAPI GdipGetTextureWrapMode(GpTexture *brush, GpWrapMode *wrapmode)	WrapMode TextureBrush::GetWrapMode() const	Gets the wrap mode currently set for this texture brush.
GpStatus WINGDIPAPI GdipGetTextureImage(GpTexture *brush, GpImage **image)	Image * TextureBrush::GetImage() const	Gets a pointer to the Image object that is defined by this texture brush.