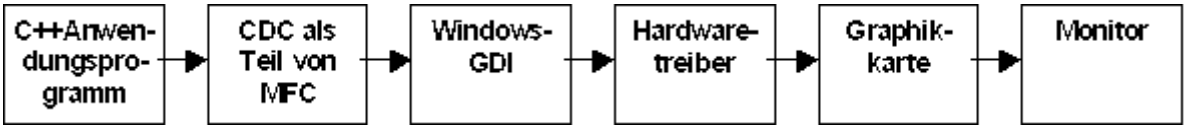
# Courses 2DCx, C2: Draw, FAQs


(in Anlehnung an Jeff Prosise: Windows-Programmierung mit MFC 2.Aufl., MP 1999, pp.35-41)



## Was hat das Betriebssystem mit Graphikprogrammierung zu tun ?

Grafikprogrammierung ist sehr schwer, wenn das Betriebssystem keine oder wenig Graphikbefehle anbietet. Man braucht mindestens eine Grafikbibliothek mit Routinen im Stile von DrawLine und DrawCircle. Andernfalls muss man Ausgaberoutinen selbst schreiben und deren Laufzeiten einzeln optimieren. Jede Graphikkarte und jeder Drucker will unterstützt werden und man muss jede Graphikkartenänderung direkt oder mittels einer neuen Bibliothek in seine Programme einarbeiten. Aus Sicht des Grafikprogrammierers verhält sich Graphikkarte und Drucker wie ein schnell bewegliches Ziel: Kaum hat man sich darauf eingestellt, ändert sich die Richtung und nichts funktioniert mehr.
MacOS, Linux, Windows und OS2 mit ihren Modellen der geräteunabhängigen Grafikausgabe erlösen den Programmierer von diesem Dauerproblem. Unter Windows arbeitet der Grafikcode eines Programms ohne weiteres Zutun des Programmierers mit jeder Grafikkarte zusammen, für die ein Windows-Treiber verfügbar ist. Da fast jeder Hersteller von Graphikkarten und Printern Windowstreiber für sein Produkt zur Verfügung stellt, läßt sich behaupten, dass fast jeder Grafikcode mit fast jeder Grafikhardware zurecht kommt. Der Code für die Bildschirmausgabe läßt sich sogar für die Ausgabe auf einem Drucker einsetzen. Diese geräteunabhängige Grafikprogrammierung hat also den wichtigen Vorzug, dass sich der Anwendungsprogrammierer nicht mehr mit hardwarespezifischen Problemen beschäftigen muss. Windows stellt als Betriebssystem von sich aus eine umfangreiche Sammlung von Grafikroutinen bereit, die den Programmierer von den Hardwaretreibern abschirmen. Das für die Grafikausgabe zuständige Modul von Windows trägt den Namen GDI (Graphics Device Interface verpackt in GDI32.DLL) und verkörpert eine Sammlung von API (Application Programming Interface)-Funktionen bzw. Diensten, derer sich der Programmierer bedienen kann. Die MFC vereinfacht weiter den Verkehr zwischen der Sprache C++ und der GDI, indem sie eine Vielfalt von komfortablen Grafikklassen anbietet, deren wichtigste die Klasse `CDC (Class for Device Contexts)` ist.




see also: http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/gdi/wingdistart_9ezp.htm

## Wozu dient die Klasse ClassDeviceContext CDC ?

Kurzantwort: CDC ist ein Dach über DC-Daten plus GDI-Methoden.
Die Fensterorientierung von Ein- und Ausgabeoperationen ist das zentrale Konzept von Windows zur Durchsetzung der Regel, dass kein Programm ein anderes stören oder zerstören kann. Ein Programm muss zur Kommunikation mit dem Benutzer (mindestens) ein eigenes exklusives Fenster anmelden, und die GDI gewährleistet, dass alle textuellen und grafischen Ausgaben des Programms auf dieses Fenster beschränkt bleiben. Der Mechanismus, den die GDI dabei verwendet, ist einfach aber wirksam - es ist der Gerätekontext = Device Context = DC.
Wenn ein Programm unter Windows die Bildschirmanzeige, einen Drucker oder irgend ein anderes Ausgabegerät anspricht, zeichnet es nicht die Bildpunkte, die das Gerät letzten Endes ausgibt (diese kennt das Programm in der Regel überhaupt nicht), sondern es zeichnet auf ein logisches Gerät mit abstrakten Eigenschaften, den sogenannten Device Context DC. Ein DC ist nichts weiter als eine ziemlich komplizierte Datenstruktur, auf die sowohl das Anwendungsprogramm als auch der Graphikkartentreiber Lese- und Schreibzugriff hat. Das Betriebssystem erstellt zur Laufzeit für jedes sich öffnende grafikfähige Fenster (mindestens) einen eigenen DC exklusiv für dieses Fenster her und verwaltet diesen über die gesamte Lebensdauer des Fensters.

Ein Gerätekontext ist also eine irgendwo innerhalb von Windows existierende Datenstruktur, deren Felder eine Beschreibung davon liefern, was die GDI über die Beschaffenheit der Ausgabefläche sowie über den aktuellen Zustand des zugeordneten physikalischen Ausgabegerätes weiß. Bevor ein Programm irgend etwas graphisch ausgeben kann, muss es sich von der GDI einen DC für das Ausgabegerät in Form eines Handles (oder eines Pointers) zuteilen lassen. Diesen Handle( = nummerierter Schlüssel) muss das Programm dann jedes Mal vorzeigen, wenn es eine Ausgabefunktion der GDI aufruft.
Dem Objektgedanken folgend generalisiert die MFC nun den Device Context zum Objekt, bestehend aus der DC-Datenstruktur und zusätzlichen Methoden, die nichts anderes als verkapselte GDI-Funktionen sind. Mit anderen Worten, die CDC-Klasse (und ihre Ableitungen `CClientDC, CPaintDC, CWindowDC, CMetaFileDC`) verhüllt komfortabel den Windows-DC und die GDI-Funktionen unter einem einzigen geschlossenen Dach.

C++Anwendungsprogramm <-> CDC <-> Hardwaretreiber,
wobei CDC die Windows-Datenstruktur DC mit der Windows-Funktionsbibliothek GDI unter einem Dach vereinigt.
Beispiel: `CClientDC dc(this); dc.TextOut(0,0,"Hurra");` bedeutet:
Gib mir einen Schlüssel für den DC des inneren zum Zeichnen nutzbaren Rechteckgebiets des am Bildschirm gerade aktiven Fensters und nenne ihn dc. Mit diesem Schlüssel schreibe Hurra ab der Stelle 0,0.
Vorsicht: Der Inhalt des DC kann sich schnell ändern (z.B. wenn ein Teil des Fensters von einem anderen Programm abgedeckt wird). Der Schlüssel wird in diesem Fall wertlos. Er muss deshalb bei jedem Eintritt in die `OnPaint()`-Funktion vom Betriebssystem neu angefordert werden.

# Was sind die wichtigsten Attribute eines Gerätekontextes DC ?

Der DC ist ein logisches Gerät, keine physikalische Graphikkarte. Trotzdem verhält er sich nicht viel anders als eine physikalische Grafikkarte. Alle Gerätezustände (=Attribute) bleiben solange zufällig, bis jemand sie einstellt. Der Konstruktor der Klasse CDC automatisiert diese Aufgabe. Jedesmal, wenn Sie von Windows einen Gerätekontext anfordern, erhalten Sie einen mit Standardattributen, die solange gültig bleiben, bis jemand sie verstellt:

Standardwerte, die CDC- (oder CClientDC- oder CPaintDC-) Klasse soll dc heissen.

| | | | |
|---|---|---|---|
| **Attribut** | | | |
| **Standardwert** | **Methode zum Setzen** | **Methode zum Abfragen** | |
| Textfarbe | schwarz | dc.SetTextColor | dc.GetTextColor |
| Hintergrundfarbe | weiss | dc.SetBkColor | dc.GetBkColor |
| Koordinatensytem | MM_TEXT | dc.SetMapMode | dc.GetMapMode |
| Zeichenmodus | R2_COPYPEN | dc.SetROP2 | dc.GetROP2 |
| Aktuelle Position | (0,0) | dc.MoveTo | dc.GetCurrentPosition |
| Stift = Pen | BLACK_PEN | dc.SelectObject | dc.SelectObject |
| Pinsel = Brush | WHITE_BRUSH | dc.SelectObject | dc.SelectObject |
| Schrift = Font | SYSTEM_FONT | dc.SelectObject | dc.SelectObject |

Ein Attribut kann Unterattribute enthalten wie z.B. Stift. Es hat Unterattribute wie Strichbreite und Strichart ( durchgezogen, gepunktet, strich, strichpunktstrich etc. ). Deshalb bietet MFC z.B. für Stifte, Pinsel und Schriften eigene kleine Klassen `CPen, CBrush, CFont` an.