الفصل السابع عشر

الرسم داخل بيئة العنصر

Device Context

يتم دائماً الرسم داخل تركيبات مستقلة تحتوى على مواصفات الأداة أو البيئة التى يتم الرسم فيها. لذا يجب أن تقوم أولاً بتحديد هذه البيئة قبل القيام بأى عملية رسومية.

بانتهاء هذا الفصل ستتعرف على:

- *فهم الأنواع المختلفة لبيئات العناصر والتوقيت الصحيح لاستخدامها.*
- *استخدام وإعداد أنماط الرسم المختلفة.*
- *التعرف عى إمكانيات الأداة.*

ربما يبدو لك مصطلح بيئة العنصر أو ما يسمى Device Context (DC) غريباً من الوهلة الأولى، فهو من المصطلحات التى كثيراً ما نسمعها ولا نستطيع تحديد ما هو المطلوب منها. يكفى أن تعرف أن النوافذ لا تعنى شيئاً على الإطلاق بدون وجود بيئة العنصر. يمكنك تعريف بيئة العنصر ببساطة شديدة بأنها اللوحة التى يتم فيها رسم النقاط والخطوط والمربعات والألوان وغيرها. فكلمة Device تعنى أنك تستطيع الرسم على أداة أو جهاز معين مثل الشاشة أو الطابعة أو جهاز عرض ثنائى الأبعاد دون أن تكون ملماً بمواصفات هذا الجهاز، وعلى ذلك يمكن تعريف بيئة العنصر بصورة أخرى على أنها تركيب داخل Windows يحتوى على الإعدادات الافتراضية الخاصة بأى عملية رسومية كرسم خط مثلاً، كما تعرف أيضاً بأنها مجموعة من عناصر الواجهة الرسومية GDI والتى تتكون بدورها من عدة دوال لإنشاء الارتباط بين عمليات استدعاء دوال الرسم والمعرَّفات الأخرى. لا تتعجل الفهم الكامل لبيئة العنصر DC، فما زال أمامنا الكثير فى هذا الفصل، فكن معنا عزيزى القارئ.

أنواع بيئات العناصر

توجد بيئة عنصر افتراضية أساسية، كما يوجد العديد من البيئات الأخرى المستخدمة لأغراض فرعية معينة. لذا تحتوى مكتبة MFC على العديد من تصنيفات بيئات العناصر والتى على رأسها التصنيف الافتراضى CDC الذى يحتوى على عدة دوال للرسم وتمثيل الإحداثيات للحصول على الواجهة الرسومية، كما تحتوى أيضاً على التصنيفات الأخرى والتى تنبثق من التصنيف الافتراضى لأداء المهام الخاصة، وفيما يلى سنقوم بإلقاء الضوء على هذه التصنيفات.

استخدام التصنيف CDC

تحتوى كل نافذة من النوافذ على بيئة العنصر التى تقوم بتغطية النافذة بالكامل، بما فى ذلك نافذة سطح المكتب. للتعرف على قوة وسهولة استخدام بيئة العنصر، سنقوم برسم بعض الأشكال على سطح المكتب. قم بإنشاء مشروع حوارى جديد باسم DCDraw ثم قم

بإضافة زر جديد إلى المربع الحوارى باسم IDC_DRAWIT وعنوان Draw وقم بعد ذلك بإضافة دالة احتواء نقر الزر وذلك بنقر الزر نقراً مزدوجاً.

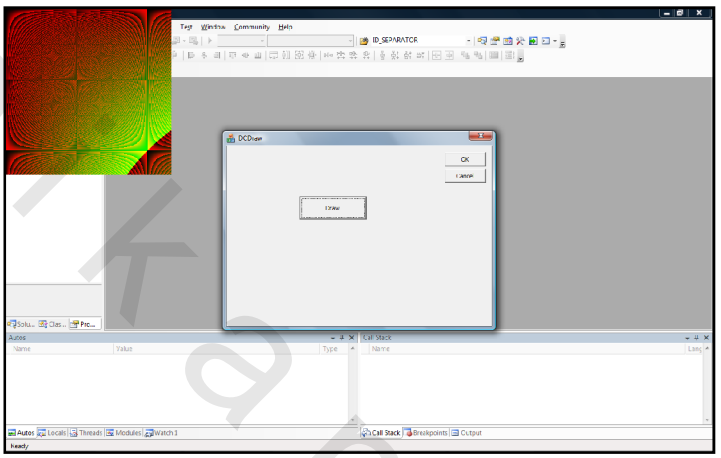
والآن قم بتعديل كود دالة الاحتواء OnBnClickedDrawit() كما يلى:

```
1.    void CDCDrawDlg::OnBnClickedDrawit()
2.    {
3.      // TODO: Add your control notification handler code here
4.      CWnd* pDesktop = GetDesktopWindow();
5.      CDC* pDC = pDesktop->GetWindowDC();
6.      for(int x=0;x<300;x++)
7.      {
8.        for(int y=0;y<300;y++)
9.        {
10.           pDC->SetPixel(x,y,x*y);
11.       }
12.     }
13.     pDesktop->ReleaseDC(pDC);
14.   }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٤ قمنا بتعريف مؤشر لنافذة سطح المكتب باستدعاء الدالة GetDesktopWindow().
- فى السطر رقم ٥ قمنا بتعريف مؤشر لبيئة العنصر الحالية باستدعاء الدالة GetWindowDC().
- فى السطور من ٦ إلى ١٢ قمنا باستخدام دوارتين for لرسم مساحة 300x300 نقطة من الركن الأيسر العلوى لسطح المكتب باستخدام الدالة SetPixel() التى تحتوى على ثلاثة معاملات. الأول والثانى يمثلان الإحداثى الأفقى والرأسى على الترتيب، أما المعامل الثالث والأخير فيعبر عن اللون المستخدم لرسم النقطة (انظر شكل ١٧-١).
- فى السطر رقم ١٣ تم تحرير بيئة العنصر باستدعاء الدالة ReleaseDC().

قم ببناء التطبيق وتنفيذه. انقر زر **Draw** تلاحظ رسم مساحة **300x300** نقطة من الركن الأيسر العلوى لسطح المكتب (انظر شكل ١٧-١)، إلا أنه بإغلاق المربع الحوارى لا يتم حذف الرسم.



شكل ١٧-١رسم جزء من سطح المكتب

لحذف هذه الرسوم وتنظيف الشاشة بمجرد إغلاق المربع الحوارى، قم بإنشاء دالة احتواء رسالة إغلاق المربع الحوارى **WM_DESTROY** وذلك كما يلى:

١. نشط تبويب عرض التصنيفات من نافذة عمل المشروع إذا لم يكن هو التبويب النشط.

٢. قم بفتح قائمة التصنيفات **DCDraw** ثم انقر التصنيف **CDCDrawDlg** لتنشيطه.

٣. انقر زر الرسائل من شريط أدوات مربع الخصائص، تظهر قائمة بالرسائل المتاحة للتصنيف.

٤. اختر الرسالة **WM_DESTROY** ثم اختر **OnDestroy <Add>** من القائمة المصاحبة، يتم إضافة هيكل الدالة **OnDestroy()** إلى نافذة الكود.

والآن قم بتعديل كود الدالة كما يلى:

```
void CDCDrawDlg::OnDestroy()
```

```
{
  CDialog::OnDestroy();
  // TODO: Add your message handler code here
  GetDesktopWindow()->RedrawWindow(NULL,NULL,
  RDW_ERASE+RDW_INVALIDATE+RDW_ALLCHILDREN+
  RDW_ERASENOW);
}
```

قمنا فى هذا الكود باستدعاء الدالة **RedrawWindow()** لإعادة طلاء نافذة سطح المكتب وجميع النوافذ التابعة، حيث تحتوى الدالة على ثلاثة معاملات. الأول والثانى يعبران عن المستطيل والمنطقة التى سيتم إعادة طلائها، وقد قمنا بتمرير القيمة **NULL** لكلا المعاملين لأننا نريد إعادة طلاء المنطقة بالكامل. أما المعامل الثالث فيحتوى على مجموعة من المعرفات، حيث يستخدم المعرف **RDW_ERASE** لمسح أى رسوم على الشاشة والمعرف **RDW_INVALIDATE** لإعادة الطلاء فيما بعد والمعرف **RDW_ALLCHILDREN** لإعادة طلاء جميع النوافذ التابعة.

والآن قم ببناء التطبيق وتنفيذه. انقر زر **Draw** تلاحظ رسم مساحة **300x300** نقطة من الركن الأيسر العلوى لسطح المكتب (راجع شكل ١٧-١)، ويتم حذف الرسم بمجرد إغلاق المربع الحوارى.

فضلاً عن الدالة **SetPixel()**، يوجد العديد من دوال الرسم الأخرى ذات الاستخدامات المتعددة والتى سنقوم بالتعرض لها تفصيلاً فى الفصلين القادمين حتى لا نشتت أفكارك.

استخدام تصنيف منطقة الرسم

يستخدم التصنيف **CClientDC** لتمثيل بيئة أداة منطقة الرسم الطبيعية داخل النافذة حيث يشتق هذا التصنيف من التصنيف الأساسى **CDC** ويختلف عنه فى عدم وجود دالة استدعاء بيئة العنصر أو دالة تحريرها حيث يتم ذلك تلقائياً بمجرد تعريف عنصر من عناصر التصنيف، وذلك لأننا نقوم بتمرير معامل يعبر عن مؤشر للنافذة وبالتالى ربطها تلقائياً ببيئة العنصر المطلوبة، وبمجرد إنهاء التصنيف بإغلاق الدالة التى تحتويه، يتم تلقائياً تحرير بيئة

العنصر المستخدمة دون استدعاء الدالة ReleaseDC() كما كنا نفعل مع التصنيف CDC.

فضلاً عن التصنيف CClientDC، يوجد التصنيف CWindowDC الذى يستخدم للتعامل مع شريط الأدوات وحدود النافذة، إلا أن هذا التصنيف نادراً ما يستخدم لأن فى معظم التطبيقات يتم الرسم داخل النافذة فقط وليس فى شريط العنوان أو حدود النافذة.

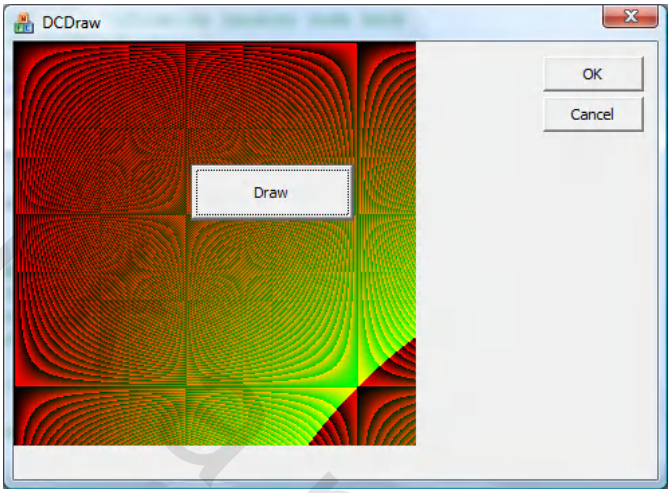
لاستخدام التصنيف CClientDC بدلاً من التصنيف CDC، قم بتعديل كود الدالة OnBnClickedDrawit() كما يلى:

```
1.  void CDCDrawDlg:: OnBnClickedDrawit()
2.  {
3.    // TODO: Add your control notification handler code here
4.    CClientDC dlgDC(this);
5.    for(int x=0;x<300;x++)
6.    {
7.      for(int y=0;y<300;y++)
8.      {
9.         dlgDC.SetPixel(x,y,x*y);
10.     }
11.   }
12. }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٤ قمنا بتعريف عنصر ينتمى إلى التصنيف CClientDc بتمرير المعامل this ليعبر عن النافذة الحالية وليس سطح المكتب كما فى المثال السابق.
- السطور من ٥ إلى ١١ لا تختلف كثيراً عن الكود السابق إلا فى السطر رقم ٩ حيث تم استدعاء الدالة SetPixel() كعنصر وليس مؤشر.
- لاحظ عدم استدعاء الدالة GetWindowDC() أو الدالة ReleaseDC() حيث يتم تضمينها تلقائياً كما ذكرنا من قبل.

قم ببناء التطبيق وتنفيذه. انقر زر **Draw** تلاحظ رسم مساحة **300x300** نقطة من الركن الأيسر العلوى للنافذة (انظر شكل ١٧-٢).



شكل ١٧-٢ الرسم داخل النافذة

استخدام تصنيف بيئة الطلاء

يستخدم التصنيف **CPaintDC** لتمثيل بيئة أداة الطلاء **Paint Device Context** حيث ينبثق من التصنيف الأساسى **CDC** ويمتاز باحتوائه على رسالة **WM_PAINT** التى يقوم نظام التشغيل بإرسالها حينما يتم كشف النافذة أو جزءٍ منها عن طريق نافذة أخرى لإخبار التطبيق بإعادة طلاء المنطقة المكشوفة. وبدلاً من إعادة طلاء النافذة بالكامل كلما تم كشف جزء معين منها، يقوم نظام التشغيل بتمرير مستطيل يحتوى على إحداثيات الجزء الذى تم كشفه، وفى هذه الحالة يمكنك استخدام هذه المعلومات لإعادة طلاء الجزء المكشوف فقط بدلاً من ضياع الكثير من الوقت فى عمل غير مفيد ولن يشعر به المستخدم مما يتسبب فى سرعة عمل التطبيق ككل.

للتعرف على طريقة استخدام التصنيف **CPaintDC**، قم بفتح الدالة **OnPaint()** داخل نافذة المحرر ثم قم بتعديل كود السطر رقم ١٩ كما يلى:

```
1.    void CDCDrawDlg::OnPaint()
2.    {
```

```
3.      if (IsIconic())
4.      {
5.        CPaintDC dc(this); // device context for painting
6.        SendMessage(WM_ICONERASEBKGND,
                      (WPARAM) dc.GetSafeHdc(), 0);
7.         //Center icon in client rectangle
8.         int cxIcon = GetSystemMetrics(SM_CXICON);
9.         int cyIcon = GetSystemMetrics(SM_CYICON);
10.        CRect rect;
11.        GetClientRect(&rect);
12.        int x = (rect.Width() - cxIcon + 1) / 2;
13.        int y = (rect.Height() - cyIcon + 1) / 2;
14.        //Draw the icon
15.        dc.DrawIcon(x, y, m_hIcon);
16.     }
17.     else
18.     {
19.        OnBnClickedDrawit();
20.     }
21.  }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٣ تم استدعاء الدالة IsIconic() التى تقوم بإرجاع القيمة TRUE فى حالة تقليص النافذة (أى تظهر كرمز على شريط المهام) حيث يتم رسم رمز التطبيق فى منتصف مستطيل النافذة المتقلصة كما فى السطر رقم ١٥.
- فى السطر رقم ١٩قمنا باستدعاء الدالة OnDrawit() لتنفيذ كودها فى حالة إرجاع الدالة IsIconic() للقيمة FALSE.

والآن قم بتعديل كود الدالة OnBnClickedDrawit() لاستخدام التصنيف CPaintDC وذلك كما يلى:

```
1.   void CDCDrawDlg:: OnBnClickedDrawit()
2.   {
3.     // TODO: Add your control notification handler code here
4.     CPaintDC paintDC(this);
5.     RECT* pRect = &paintDC.m_ps.rcPaint;
6.     for(int x=pRect->left;x<pRect->right;x++)
7.     {
```

```
8.          for(int y=pRect->top;y<pRect->bottom;y++)
9.          {
10.             paintDC.SetPixel(x,y,x*y);
11.             CDialog::OnPaint();
12.         }
13.     }
14. }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٤ قمنا بتعريف متغير ينتمى للتصنيف **CPaintDC**.
- فى السطر رقم ٥ قمنا بتعريف المتغير **pRect** كمؤشر لعنصر ينتمى للتصنيف **RECT** ليحتوى على المنطقة التى سيتم إعادة طلائها وتم إعطاءه عنوان **m_ps** كقيمة ابتدائية. و**m_ps** عبارة عن تركيب **PAINTSTRUCTURE** عضو فى التصنيف **CPaintDC** بالتعريف التالى:

```
typedef struct tagPAINTSTRUCT {
   HDC         hdc;
   BOOL        fErase;
   RECT        rcPaint;
   BOOL        fRestore;
   BOOL        fIncUpdate;
   BYTE        rgbReserved[32];
} PAINTSTRUCT;
```

وقد استخدمنا **rcPaint** لتحديد إحداثيات المنطقة المراد إعادة طلائها.

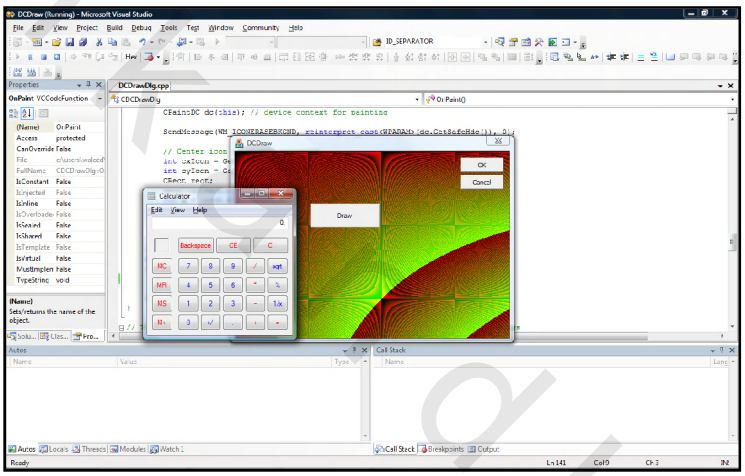
- فى السطور من ٦ إلى ١٢ قمنا باستخدام المتغير **pRect** داخل الدوارتين **for** لرسم الشكل بألوان مختلفة باستدعاء الدالة **()SetPixel** فى السطر رقم ١٠.
- بدلاً من استدعاء الدالة ٩٠.٠٠٠ مرة قبل ذلك **(300x300 pixels)** ، يتم استدعاؤها تبعاً للجزء المكشوف فقط من النافذة مما يزيد من سرعة التطبيق.

والآن قم ببناء التطبيق وتنفيذه. يظهر المربع الحوارى مغطى بمجموعة من الألوان عدا أزرار التحكم، حيث يقوم نظام التشغيل بإرسال الرسالة **WM_PAINT** إلى المربع الحوارى بمجرد ظهوره لرسمه بالكامل وبعد ذلك يتم رسم الأزرار على المنطقة الملونة. لاختبار سلوك بيئة أداة الطلاء **Paint DC**، تابع معنا الخطوات الآتية:

١. قم بتغطية منتصف المربع الحوارى بنافذة تطبيق آخر وليكن برنامج الحاسبة من البرامج الملحقة (انظر شكل ١٧-٣).

٢. انقر المربع الحوارى DCDraw لتنشيطه، يظهر المربع الحوارى فى مقدمة الشاشة ويتم إعادة طلاء الجزء الذى تم حجبه عن طريق نافذة الحاسبة.

٣. انقر نافذة الحاسبة ثم قم بسحبها قريباً وبعيداً عن المربع الحوارى، تلاحظ إعادة طلاء أى جزء من المربع الحوارى تم حجبه بواسطة نافذة الحاسبة.



شكل ١٧-٣ تغطية المربع الحوارى بنافذة الحاسبة

استخدام تصنيف بيئة الذاكرة

تختلف بيئة الذاكرة Memory DC عن الأنواع السابقة فى أنها لا ترتبط بأداة معينة أو عنصر معين، حيث يمكنك استخدامها دون أن يظهر أى شيء. يمكنك أيضاً إنشاء بيئة ذاكرة متوافقة مع بيئة عرض أخرى، وحينئذٍ يمكنك نسخ الصور إلى النافذة إذا ما أردت عدم إظهارها ومن ثم تحميلها من الذاكرة إلى تصنيف العرض مرة أخرى.

لا يوجد داخل مكتبة MFC تصنيف خاص ببيئة الذاكرة وذلك لوجود تصنيف CDC الذى يمكنك استخدامه للرسم داخل الذاكرة بدلاً من النوافذ والمربعات الحوارية.

للتعرف على طريقة إنشاء واستخدام تصنيف بيئة الذاكرة، قم بتعديل كود الدالة OnBnClickedDrawit() كما يلى:

```
1.    void CDCDrawDlg::OnBnClickedDrawit()
2.    {
3.      // TODO: Add your control notification handler code here
4.      CClientDC clientDC(this);
5.      CDC memDC;
6.      memDC.CreateCompatibleDC(&clientDC);
7.      CRect rcClient;
8.      GetClientRect(&rcClient);
9.      CBitmap memBitmap;
10.     memBitmap.CreateCompatibleBitmap(&clientDC,
                              rcClient.Width(),rcClient.Height());
11.     memDC.SelectObject(&memBitmap);
12.     for(int x=0;x<rcClient.Width();x++)
13.     {
14.       for(int y=0;y<rcClient.Height();y++)
15.       {
16.         memDC.SetPixel(x,y,x*y);
17.       }
18.     }
19.     clientDC.BitBlt(0,0,rcClient.Width(),rcClient.Height(),
                                  &memDC,0,0,SRCINVERT);
20.   }
```

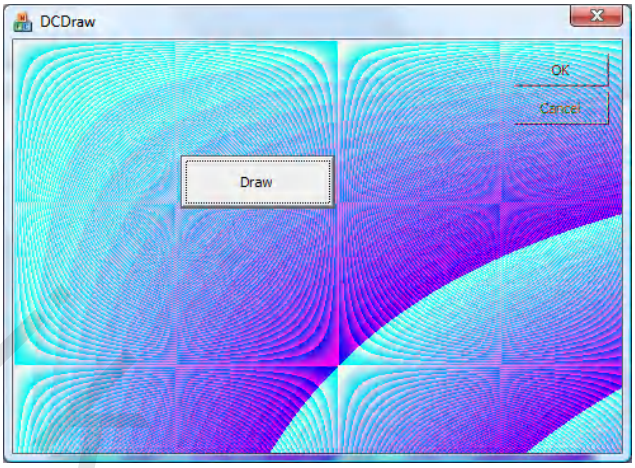
وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٤ تم تعريف متغير ينتمى للتصنيف CClientDC.
- فى السطر رقم ٥ تم تعريف متغير ينتمى للتصنيف CDC.
- فى السطر رقم ٦ تم استدعاء الدالة CreaetCompatibleDC() لربط تصنيف الطلاء بتصنيف الذاكرة حتى يتوافق مع نافذة الشاشة.
- فى السطر رقم ٩ تم تعريف متغير صورة نقطية لاستخدامه كبيئة ذاكرة.
- فى السطر رقم ١٠ تم استدعاء الدالة CreatCompatibleBitmap() لإنشاء الصورة النقطية. تحتوى هذه الدالة على ثلاثة معاملات، الأول عبارة عن نوع بيئة

العنصر المراد استخدامها والثانى يمثل عرض المنطقة المراد رسمها داخل الصورة النقطية، ويعبر الثالث عن طول هذه المنطقة.

- فى السطر رقم ١١ تم استدعاء الدالة **SelectObject()** لإضافة الصورة النقطية إلى بيئة العنصر، وبالتالى سيتم إضافة جميع رسوم الذاكرة إلى هذه الصورة.
- فى السطور من ١٢ إلى ١٨ يتم رسم الخلفية كما تعودنا ولكن داخل الذاكرة وليس على الشاشة كما اعتدنا قبل ذلك.
- فى السطر رقم ١٩ يتم استدعاء الدالة **BitBlt()** للصق الصورة النقطية من الذاكرة إلى الشاشة وتحتوى على ثمانية معاملات.الأول والثانى يمثلان الإحداثى الأفقى والرأسى للجزء الذى سيتم لصقه من الصورة. الثالث والرابع يمثلان عرض وارتفاع هذا الجزء. الخامس يحتوى على نوع بيئة العنصر. السادس والسابع يمثلان الإحداثى الأفقى والرأسى للمكان الذى سيتم اللصق فيه على الشاشة. أما المعامل الثامن والأخير فيعبر عن الألوان التى سيتم استخدامها لرسم الصورة على الشاشة، وقد اخترنا **SRCINVERT** لعكس الألوان حتى تكون مميزة عن بيئات العناصر السابقة. يمكنك تغيير هذا المعامل إلى **SRCCOPY** إذا أردت إظهار الصورة على الشاشة بنفس ألوانها الأساسية.

والآن قم بحذف سطر استدعاء الدالة **OnBnClickedDrawit()** من كود الدالة **OnPaint()** (وإلا ستحصل على تأثيرات غريبة جداً داخل النافذة) ثم قم ببناء التطبيق وتنفيذه. انقر الزر **Draw**، تلاحظ تأخير قصير نتيجة لرسم الصورة إلى الذاكرة وبعد ذلك تظهر داخل النافذة (انظر شكل ١٧-٤).



شكل ١٧-٤ استخدام بيئة الذاكرة Memory DC

استخدام أنماط الرسم

تعتبر أنماط الرسم Mapping Modes من الأدوات الأساسية فى أى بيئة عنصر، والتى من خلالها يمكنك الرسم على الشاشة أو الطابعة بوحدات قياس أخرى غير النقاط Pixels مثل البوصة inch أو الملى متر millimeters. عند العمل مع أنماط الرسم ، ستجد بعض المصطلحات مثل وحدة العنصر Device unit والتى تعبر عن أصغر وحدة يمكن للعنصر تمثيلها (وهى النقطة Pixel) والوحدة المنطقية Logical unit والتى تعبر عن أنواع القياسات الأخرى مثل البوصة أو السنتيمتر.

يمكنك تغيير نمط الرسم الافتراضى MM_TEXT باستدعاء الدالة SetMapMode() بتمرير أحد أنماط الرسم الموضحة بالجدول ١٧-١ التالى.

جدول ١٧-١ أنماط الرسم

النمط	القيمة المكافئة
MM_TEXT	نقطة واحدة
MM_LOMETRIC	٠.١ ملليمتر
MM_HIMETRIC	٠.٠١ ملليمتر

النمط	القيمة المكافئة
MM_LOENGLISH	٠.٠١ بوصة
MM_HIENGLISH	٠.٠٠١ بوصة
MM_TWIPS	١/١٤٤٠ من البوصة أو ١/٢٠ من النقطة
MM_ISOTROPIC	تعرف بواسطة المبرمج وغالباً يتساوى فيها الإحداثيات الأفقية والرأسية
MM_ANISOTROPIC	تعرف بواسطة المبرمج

من الجدول السابق نلاحظ أن النمط **MM_TEXT** يعنى أن كل وحدة منطقية يتم تمثيلها بنقطة واحدة **Pixel** وبالتالى لا يوجد أى تحويلات. أما النمط **MM_LOMETRIC** فيعنى أن كل وحدة منطقية تقابل ٠.١ من المليمتر، فإذا قمت مثلاً بتمرير القيمة ١٠٠، فإن القيمة المناظرة تكون ١٠٠ X ٠.١ مليمتر = ١ سم، وبالتالى ستقوم الدالة بحساب عدد النقاط التى ستقوم الشاشة أو الطابعة بعرضها فى السنتيمتر الواحد.

للتعرف على طريقة العمل مع أنماط الرسم، قم بإنشاء تطبيق أحادى الوثيقة وذلك بتنشيط زر الاختيار **Single document** بالتبويب **Application Type** بنافذة معالج التطبيقات باسم **MapeMode** ثم قم بفتح الدالة **OnDraw()** من داخل تصنيف العرض **CMapModeView** وقم بتعديل كود الدالة كما يلى:

```
1.   void CMapModeView::OnDraw(CDC* pDC)
2.   {
3.     CMapModeDoc* pDoc = GetDocument();
4.     ASSERT_VALID(pDoc);
5.     //TODO: add draw code for native data here
6.     pDC->SetMapMode(MM_LOMETRIC);
7.     CRect rcClient;
8.     GetClientRect(&rcClient);
9.     pDC->DPtoLP(&rcClient);
10.    for(int x=rcClient.TopLeft().x; x<rcClient.BottomRight().x;
                                                    x+= 100)
11.    {
```

```
12.         pDC->SetPixel(x,rcClient.CenterPoint().y-1,0);
13.         pDC->SetPixel(x,rcClient.CenterPoint().y,0);
14.         pDC->SetPixel(x,rcClient.CenterPoint().y+1,0);
15.     }
16.  }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٦ تم تغيير نمط الرسم من النمط الافتراضى **MM_TEXT** إلى النمط **MM_LOMETRIC** والذى يعنى أنه سيتم ضرب قيمة كل إحداثى فى ٠.١ ملليمتر.
- فى السطر رقم ٩ تم استدعاء الدالة **()DptoLp** للتحويل من النقطة إلى الإحداثيات المنطقية.
- فى السطر رقم ١٠ تم استخدام الدوارة **for** لاستخدام المتغير **x** من يسار منطقة الرسم إلى يمينها كل ١٠٠ وحدة منطقية (أى كل ١ سنتيمتر فى حالة النمط **MM_LOMETRIC**).
- فى السطور من ١١إلى ١٥ تم استخدام الدالة **()SetPixel** داخل الدوارة **for** لرسم سطر من النقاط فى منتصف النافذة بفاصل سنتيمتر واحد.

والآن قم ببناء التطبيق وتنفيذه، تلاحظ وجود صف أفقى من النقاط بعرض النافذة وبفاصل سنتيمتر واحد بين كل نقطة والأخرى.

حاول تغيير نمط الرسم إلى **MM_LOENGLISH** ثم قم بإعادة بناء التطبيق وتنفيذه، تلاحظ زيادة الفواصل لتصل إلى واحد بوصة بدلاً من واحد سنتيمتر.

أنماط الرسم المعرَّفة

يمكنك التحكم فى خصائص وإعدادات نمط الرسم بإنشاء نمط معرف بتمرير المعامل **MM_ANISOTROPIC** للدالة **()SetMapMode** وبذلك يمكنك تعيين إحداثيات أفقية وأخرى رأسية.

للتعرف على طريقة استخدام الأنماط المعرفة، قم بتعديل كود الدالة () OnDraw كما يلى:

```
1.    void CMapModeView::OnDraw(CDC* pDC)
2.    {
3.      CMapModeDoc* pDoc = GetDocument();
4.      ASSERT_VALID(pDoc);
5.      // TODO: add draw code for native data here
6.      pDC->SetMapMode(MM_ANISOTROPIC);
7.      CRect rcClient;
8.      GetClientRect(&rcClient);
9.      pDC->SetViewportExt (CSize(rcClient.BottomRight().x,
                                                    500));
10.     pDC->SetWindowExt(CSize(10000,500));
11.     pDC->DPtoLP(&rcClient);
12.     for(int x=rcClient.TopLeft().x;x<rcClient.BottomRight().x;
                                                    x+=100)
13.     {
14.         pDC->SetPixel(x,rcClient.CenterPoint().y-1,0);
15.         pDC->SetPixel(x,rcClient.CenterPoint().y,0);
16.         pDC->SetPixel(x,rcClient.CenterPoint().y+1,0);
17.     }
18.   }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٦ تم تغيير نمط الرسم إلى MM_ANISOTROPIC.
- فى السطر رقم ٩ تم استدعاء الدالة SetViewPortExt() لتعيين الطول والعرض بالنقاط، بحيث يكون العرض هو مقدار عرض المنطقة المختارة والطول ٥٠٠ نقطة لأعلى.
- فى السطر رقم ١٠ تم استدعاء الدالة SetWindowExt() لتعيين العرض المنطقى ليكون ١٠٠٠٠ نقطة والطول المنطقى ليكون ٥٠٠ نقطة لأعلى.
- فى السطور من ١٢ إلى ١٧تم استخدام الدوارة for لرسم صف من النقاط الأفقية عبر النافذة.

قم ببناء التطبيق وتنفيذه، تلاحظ ظهور صف من النقاط بعرض النافذة.

التعرف على إمكانيات الجهاز

يمكنك استخدام بيئة العنصر للتعرف على إمكانيات الجهاز المصاحب باستخدام الدالة **GetDeviceCaps()**. فمثلاً هناك طابعات ملونة وأخرى أبيض وأسود. يمكنك من خلال هذه الدالة التعرف على أى معلومة عن الجهاز بتمرير قيمة من القيم الموجودة بالجداول اللاحقة. فهناك القيم المتعلقة بتقنية الجهاز كما بالجدول ١٧-٢.

جدول ١٧-٢ القيم المتعلقة بتقنية الجهاز

القيمة المرجعة	الوصف
DT_RASDISPLAY	كارت شاشة عادى وشاشة
DT_RASPRINTER	طابعة عادية
DT_PLOTTER	جهاز Plotter عادى
DT_RASCAMERA	جهاز إدخال صورة نقطية
DT_CHARSTREAM	مجموعة من الأحرف مثل لوحة المفاتيح
DT_METAFILE	ملف يحتوى على تعليمات طريقة رسم مخطط
DT_DISPFILE	ملف عرض

هناك أيضاً القيم المتعلقة بإصدار معرف الجهاز والموضحة بالجدول ١٧-٣.

جدول ١٧-٣ القيم المتعلقة بإصدار معرف الجهاز

القيمة الممررة	وصف القيمة المرجعة
HORZRES	عرض العنصر بالنقاط
VERTRES	ارتفاع العنصر بالنقاط
HORZSIZE	عرض العنصر بالملليمتر
VERTSIZE	ارتفاع العنصر بالملليمتر
ASPECTX	نسبة الطول إلى العرض بالنقاط

القيمة الممررة	وصف القيمة المرجعة
ASPECTY	نسبة العرض إلى الطول بالنقاط
ASPECTXY	طول قطر العنصر بالنقاط

كما يوجد العديد من القيم الأخرى المتعلقة بالألوان وإمكانيات الرسم والموضحة بالجدول ١٧-٤.

جدول ١٧-٤ القيم المتعلقة بالألوان وإمكانيات الرسم

القيمة الممررة	وصف القيمة المرجعة
NUMCOLORS	عدد الألوان التى يدعمها العنصر
PLANES	عدد مستويات الألوان التى يدعمها العنصر
BITSPIXEL	عدد البت المستخدمة لتمثيل النقطة
NUMPENS	عدد أقلام الرسم التى يدعمها العنصر
NUMBRUSHES	عدد فرش الرسم التى يدعمها العنصر
NUMFONTS	عدد الخطوط التى يدعمها العنصر
COLORRES	درجة نقاء ألوان العنصر
SIZEPALETTE	عدد مجموعات الألوان التى يدعمها النظام

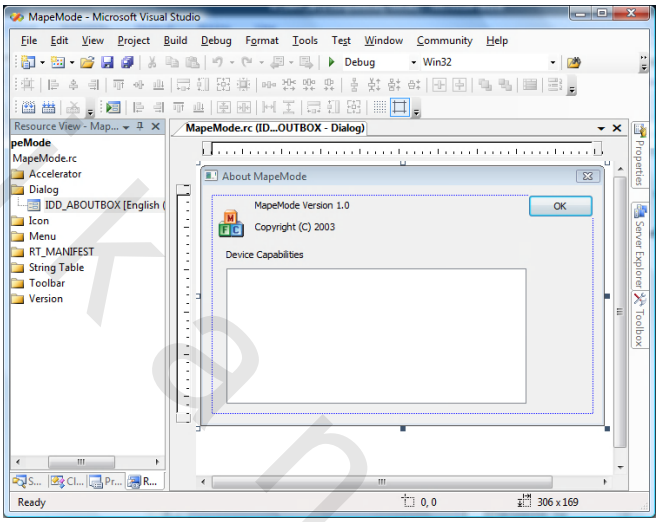
للتعرف على طريقة عمل الدالة **GetDeviceCaps()**، سنقوم بالتعرف على بعض خصائص كارت الشاشة المثبت بجهازك داخل المربع الحوارى **About** بالتطبيق الحالى **MapMode**. سنقوم أولاً بإضافة مربع سرد للمربع الحوارى **About**، تابع معنا الخطوات الآتية:

١. نشط تبويب عرض الموارد من نافذة عمل المشروع.
٢. انقر المربع الحوارى **IDD_ABOUTBOX** نقراً مزدوجاً لفتحه داخل نافذة المحرر.
٣. قم بزيادة مساحة المربع الحوارى ثم قم بإضافة مربع سرد إليه.
٤. من مربع الخصائص المصاحب لمربع السرد، قم بتغيير اسمه إلى **IDC_DEVCAPS**.

٥. قم أيضاً بتخصيص القيمة **False** للخاصية **Sort**.

٦. قم بإضافة أداة عنوان لمربع السرد باسم **Device Capabilities**.

يجب أن يظهر المربع الحوارى **About** الآن كما فى شكل ١٧-٥.



شكل ١٧-٥ إضافة مربع السرد للمربع الحوارى **About**

لإضافة النصوص لمربع السرد، يجب أن نقوم أولاً بربطه بمتغير. تابع معنا الخطوات الآتية:

١. نشط التبويب **Class View** من نافذة عمل المشروع.

٢. انقر التصنيف **CAboutDlg** بزر الفأرة الأيمن ثم اختر **Add➢Add Variable** من القائمة الموضعية، يظهر معالج إضافة متغير جديد.

٣. قم بتنشيط مربع الاختيار **Control variable** لأننا نرغب فى إنشاء متغير مرتبط بإحدى أدوات التحكم الموجودة بالمربع الحوارى.

٤. اختر مربع السرد **IDC_DEVCAPS** من مربع السرد **Control ID**.

٥. اختر **Control** من مربع السرد **Category**.

٦. تأكد من اختيار نوع البيانات **CListBox** بمربع السرد والتحريـــر **Variable Type**.

٧. اكتب اسم المتغير فى خانة **Variable name** وليكن **m_listDevCaps** واختر درجة المتغير من مربع السرد **Access** وهو **Public** فى هذه الحالة.

٨. انقر زر **Finish** لإغلاق نافذة المعالج.

كى تبدأ فى كتابة كود الدالة **()GetDeviceCaps** لعرض النتائج داخل مربع السرد، يجب أن نقوم بإضافة دالة الاحتواء **()OnInitDialog**. تابع معنا الخطوات الآتية:

١. نشط التبويب **ClassView** من نافذة عمل المشروع إذا لم يكن هو التبويب النشط.

٢. انقر التصنيف **CAboutDlg** بزر الفأرة الأيمن لتنشيطه.

٣. انقر زر **Overrides** من شريط أدوات مربع الخصائص ثم اختر الدالة **OnInitDialog** واختر **OnInitDialog <Add>** من القائمة المصاحبة، يتم إضافة هيكل الدالة **()OnInitDialog** للتصنيف **CAboutDlg**.

٤. قم بتعديل كود الدالة كما يلى:

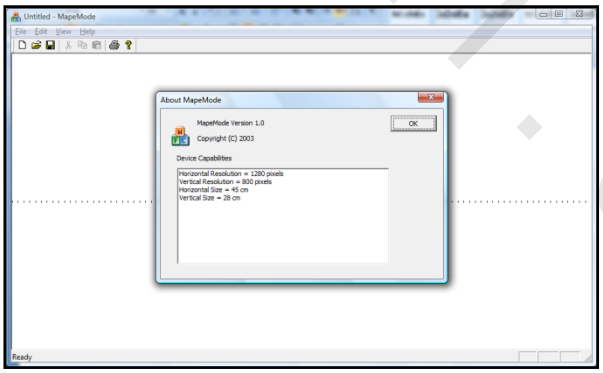
```
1.  BOOL CAboutDlg::OnInitDialog()
2.  {
3.    CDialog::OnInitDialog();
4.    //TODO: Add extra initialization here
5.    CClientDC dcDev(this);
6.    CString strCap;
7.    strCap.Format("Horizontal Resolution = %d pixels" ,
                        dcDev.GetDeviceCaps(HORZRES));
8.    m_listDevCaps.AddString(strCap);
9.    strCap.Format("Vertical Resolution = %d pixels",
                      dcDev.GetDeviceCaps(VERTRES));
10.   m_listDevCaps.AddString(strCap);
11.   strCap.Format("Horizontal Size = %d cm",
                     dcDev.GetDeviceCaps(HORZSIZE)/10);
12.   m_listDevCaps.AddString(strCap);
13.   strCap.Format("Vertical Size = %d cm",
                        dcDev.GetDeviceCaps(VERTSIZE)/10);
14.   m_listDevCaps.AddString(strCap);
15.   return TRUE;  // return TRUE unless you set the focus to a
                        control
```

```
16.     //EXCEPTION: OCX Property Pages should return FALSE
17.   }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٥ تم تعريف المتغير dcDev كى يحتوى على بيانات بيئة المربع الحوارى.
- فى السطر رقم ٧ تم استدعاء الدالة GetDeviceCaps() بتمرير المعامل HORZRES للتعرف على درجة النقاء الأفقية للعرض (راجع جدول ١٥-٣) وتخزين القيمة المرجعة داخل المتغير النصى strCap.
- فى السطر رقم ٨ تم إضافة نص المتغير strCap إلى مربع السرد داخل المربع الحوارى About.
- يتم تكرار نفس الترتيب فى باقى سطور الدالة لإضافة درجة النقاء الرأسية والطول الأفقى والرأسى لكارت الشاشة.

والآن قم ببناء التطبيق وتنفيذه. افتح قائمة Help من شريط القوائم ثم اختر About MapMode من القائمة المنسدلة، تظهر الخصائص السابقة لكارت الشاشة داخل المربع الحوارى About (انظر شكل ١٧-٦).



شكل ١٧-٦ إظهار معلومات عن كارت الشاشة المثبت بجهازك