الفصل الثانى والعشرون
العمل مع الملفات

تحتوى مكتبة MFC على العديد من الـــدوال العاملـــة مـــع التطبيقات أحادية الوثيقة أو التطبيقات متعـــددة الوثيقـــة والـــتى تساعدك على استخدام تقنية تسلسل البيانات والعمل مع الملفات.

بانتهاء هذا الفصل ستتعرف على:

- ***حفظ وتحميل بيانات الوثيقة.***
- ***إنشاء الملفات والقراءة منها أو الكتابة فيها.***
- ***تناقل البيانات من وإلى الحافظة.***

تستخدم تقنية تسلسل البيانات **Serialization** لتحويل بيانات تطبيقك إلى قائمة مرتبة من عناصر البيانات الفردية كى يتم تخزينها على إحدى وحدات التخزين المعروفة مثل القرص المرن أو القرص الصلب أو حتى نقلها إلى إحدى التطبيقات الأخرى، وبمجرد تلقى مجموعة من عناصر البيانات المسلسلة، يقوم البرنامج بقراءة هذه البيانات وإعادة تجميعها فى تركيب واحد.

إنشاء تطبيق أحادى الوثيقة يدعم العمل مع الملفات

يمكنك استخدام معالج التطبيقات لإنشاء تطبيق أحادى الوثيقة يدعم إنشاء واحتواء ونقل عناصر البيانات المخزَّنة داخل تصنيف الوثيقة بالتعديل داخل قالب الوثيقة نفسه. لإنشاء تطبيق أحادى الوثيقة يدعم عمليات نقل البيانات واحتواء الملفات، تابع معنا الخطوات الآتية:

١. تأكد أنك داخل بيئة التطوير المتكاملة وإلا قم بفتحها من قائمة **Start**.

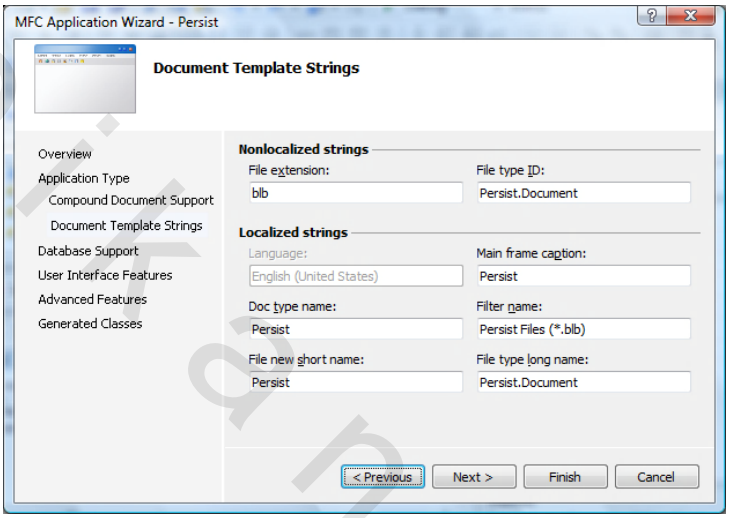
٢. من صفحة البدء، انقر الارتباط **Project** بالسطر **:Create** أو افتح قائمة **File** من شريط القوائم واختر **New** ثم **Project** من القوائم المنسدلة، يظهر المربع الحوارى **New Project**.

٣. تأكد من اختيار المجلد الفرعى **MFC** داخل المجلد **Visual C++** بالجانب الأيسر من المربع الحوارى واختيار رمز **MFC Application** من المربع **Templates** بالجانب الأيمن.

٤. اكتب اسم مناسب للمشروع الجديد وليكن **Persist** فى مربع النص **Name**. قم أيضاً بتحديد المجلد الذى سيحتوى على جميع ملفات المشروع داخل مربع النص **Location**.

٥. انقر زر **Ok** ليبدأ معالج التطبيقات **Application Wizard** فى إنشاء هيكل المشروع باستخدام مكتبة **MFC** وكتابة الكود المطلوب نيابةً عنك.

٦. نشط التبويب **Document Template Strings** ثم قم بتعيين الامتداد الذى ترغب فى استخدامه مع ملفات التطبيق داخل مربع النص **File extension** وليكن **blb** (انظر شكل ٢٢-١).



شكل ٢٢-١ تعيين خيارات ملفات التطبيق

٧. قم بتعيين معرف لنوع الملفات المستخدمة بالتطبيق لإضافته إلى قائمة التطبيقات المسجلة بالنظام حتى يتم تنفيذ التطبيق بمجرد فتح ملف يحتوى على الامتداد السابق تعيينه بالخطوة السابقة. اترك النص الافتراضى **Persist.Document** أو قم بكتابة نص آخر؛ إن أردت، داخل مربع النص **File type ID**.

٨. قم بتعيين نوع الملفات التى سيتم عرضها داخل المربعات الحوارية **Open** أو **Save** أو **Save As** داخل مربع النص **Filter name**. اترك النص الافتراضى **Persist Files(*.blb)** لعرض الملفات ذات الامتداد **blb** فقط.

٩. قم بتعيين الاسم الافتراضى المستخدم عند تخزين الملفات داخل مربع النص **File New Short Name**. اترك النص الافتراضى **Persist** كما هو.

١٠. انقر زر **Finish** لإنهاء المعالج وإنشاء التطبيق الجديد.

يتم تخزين خيارات التبويب Document Template Strings السابق شرحه داخل العنصر IDR_MAINFRAME بالجدول النصى String Table، لذا يمكنك تغيير هذه الخيارات بعد إنشاء التطبيق بتعديل بيانات هذا الجدول من تبويب عرض الموارد بنافذة عمل المشروع.

إنشاء عناصر البيانات المسلسلة

عند العمل مع عناصر البيانات المسلسلة، ستواجه تصنيفين أساسيين. الأول هو التصنيف المنبثق من تصنيف عناصر البيانات الأساسى CObject والثانى هو التصنيف CArchive المسئول عن تسلسل العناصر المنبثقة من التصنيف CObject من وإلى ملف بداخل التصنيف CArchive نفسه. يمكنك إضافة الدالة ()Serialize إلى أى التصنيف المنبثق من تصنيف عناصر البيانات الأساسى CObject، حيث تستخدم هذه الدالة لتحميل عناصر البيانات وتخزينها من وإلى الملفات.

عند حفظ ملف أو تحميله باختيار Save أو Open من قائمة File، يتم استدعاء الدالة ()Serialize بتمرير المرجع ar الذى ينتمى للتصنيف CArchive ويحتوى على متغير الملف المضمن داخل التصنيف والمتصل اتصالاً مباشراً بالملف الحقيقى على وحدة التخزين، ومن ثم يتم استخدام هذا المرجع داخل هيكل الدالة لاستدعائها بعدد مرات عناصر البيانات.

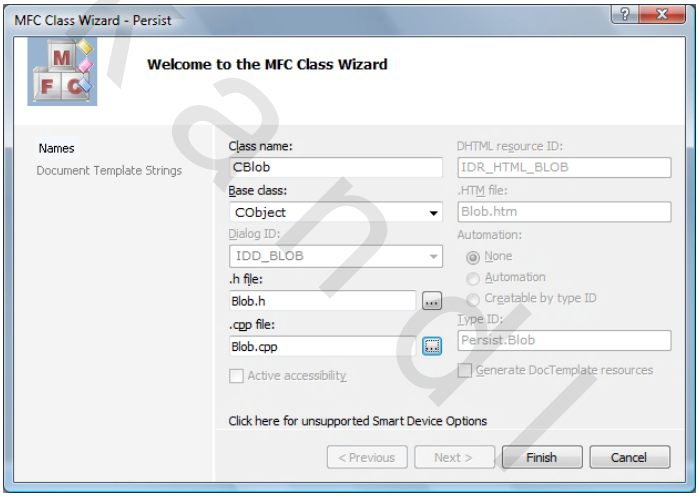
عند حفظ بيانات الوثيقة، يتم قراءة البيانات من جميع العناصر المسلسلة ومن ثم تخزينها داخل الملف المتصل بالأرشيف، لذا يطلق على الملف المنشأ بواسطة عملية تسلسل البيانات ملف الأرشيف المسلسل Serialized Archive File والذى يقوم بتخزين نسخة من بيانات كل عنصر من عناصر التطبيق بالإضافة إلى نوع العنصر وإصداره. أما عند تحميل بيانات الوثيقة، فيتم قراءة البيانات من الملف المتصل بالأرشيف واستدعاء الدالة ()Serialize لإرجاع البيانات إلى عناصرها المسلسلة.

إنشاء تصنيف التسلسل

إذا كان تطبيقك يحتاج إلى عناصر بيانات معينة، يجب أن تقوم بإنشاء تصنيفات جديدة لوصف واحتواء هذه البيانات. لإنشاء التصنيف الجديد، تابع معنا الخطوات الآتية:

١. قم بتنشيط التبويب **Class View** ثم انقر اسم المشروع بزر الفأرة الأيمن واختر **Add⮞Class** من القائمة الموضعية، تظهر نافذة إضافة تصنيف جديد.

٢. اختر رمز **MFC Class** ثم انقر زر **Add**، تظهر نافذة معالج إضافة تصنيف جديد (انظر شكل ٢٢-٢).



شكل ٢٢-٢ نافذة معالج إضافة تصنيف جديد

٣. اكتب اسم التصنيف وليكن **CBlob** داخل مربع النص **Class Name** ثم اختر **CObject** من مربع السرد والتحرير **Base Class**.

٤. انقر زر **Finish** لإغلاق نافذة المعالج وإضافة التصنيف الجديد.

٥. قم بتنشيط تبويب مستكشف الحل **Solution Explorer** ثم قم بفتح الملف **blob.h**.

٦. قم بإدخال الكود التالى إلى الملف ثم قم بحفظه:

```
1.  #ifndef _BLOB_H
2.  #define _BLOB_H
```

```
3.    class CBlob:   public CObject
4.    {
5.      DECLARE_SERIAL(CBlob);
6.      public:
7.           CBlob();
8.           CBlob(CPoint ptPosision);
9.           void Draw(CDC* pDC);
10.          CPoint      m_ptPosition;
11.          COLORREF    m_crColor;
12.          int           m_nSize;
13.          unsigned    m_nShape;
14.   };
15.   #endif
```

وعن هذا الكود، نوضح ما يلى:

- فى السطور رقم ١و٢و١٥ يتم التأكد من عدم تعريف التصنيف مرتين باستخدام التركيب **#ifndef #endif**.
- فى السطر رقم ٣ تم تعريف التصنيف الجديد **CBlob** المنبثق من التصنيف **CObject** وننصح دائماً بأن تنبثق جميع تصنيفاتك من هذا التصنيف لاحتوائه على معظم الوظائف الشهيرة التى تحتاج إليها.
- فى السطر رقم ٥ تم استخدام المختزل **DECLARE_SERIAL** لإضافة دوال تسلسل البيانات إلى التصنيف الجديد **CBlob**.
- فى السطور رقم ٧و٨ تم تعريف دوال إنشاء التصنيف التى يتم استدعاء أى منها بمجرد إنشاء عنصر جديد من عناصر التصنيف تبعاً لمعاملات تعريف العنصر.
- فى السطر رقم ٩ تم تعريف الدالة **()Draw** التى ستستخدم للرسم فيما بعد.
- فى السطور من ١٠ إلى ١٣ تم تعريف المتغيرات التى سيتم استخدامها من قبل التصنيف.

٧. الخطوة التالية بعد إنشاء الملف الرئيسى هى تعديل ملف المصدر **Source File** الخاص باستخدام التصنيف وتعريف دواله. افتح الملف **blob.cpp** من داخل التبويب **Solution Explorer** ثم قم بتعديل كود الملف كما يلى:

```
1.    #include"stdafx.h"
2.    #include "blob.h"
3.    IMPLEMENT_SERIAL(CBlob,CObject,1) CBlob::CBlob()
4.    {
5.
6.    }
7.    CBlob::CBlob(CPoint ptPosition)
8.    {
9.      srand(GetTickCount());
10.     m_ptPosition = ptPosition;
11.     m_crColor= RGB(rand()%505,rand()%505,rand()%505);
12.     m_nSize = 10 + rand()%30;
13.     m_nShape = rand();
14.   }
15.   void CBlob::Draw(CDC *pDC)
16.   {
17.     CBrush brDraw(m_crColor);
18.     CBrush* pOldBrush = pDC->SelectObject(&brDraw);
19.     CPen* pOldPen = (CPen*)pDC->SelectStockObject
                                              (NULL_PEN);
20.     srand(m_nShape);
21.     for(int n=0;n<3;n++)
22.     {
23.       CPoint ptBlob(m_ptPosition);
24.       ptBlob+=CPoint(rand()%m_nSize, rand()%m_nSize);
25.       CRect rcBlob(ptBlob,ptBlob);
26.       rcBlob.InflateRect(m_nSize,m_nSize);
27.       pDC->Ellipse(rcBlob);
28.     }
29.     pDC->SelectObject(pOldBrush);
30.     pDC->SelectObject(pOldPen);
31.   }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطور ١و٢ تم تضمين الملفات الرئيسية التى تحتوى على تعريف التصنيف الجديد والتعريفات الأساسية لتصنيفات مكتبة **MFC**.
- فى السطر رقم ٣ تم استدعاء المختزل **IMPLEMENT_SERIAL** لإضافة وظائف تمثيل تسلسل البيانات السابق تعريفها باستدعاء المختزل

DECLARE_SERIAL بالملف blob.h. يحتوى المختزل على ثلاثة معاملات، يحتوى المعامل الأول على اسم التصنيف الجديد، بينما يحتوى المعامل الثانى على التصنيف الأساسى للتصنيف الجديد، أما المعامل الثالث والأخير فيحتوى على رقم الإصدار.

- فى السطور من ٤ إلى ٦ تم تعريف دالة الإنشاء الأولى للتصنيف والتى لا تحتوى على أى كود ويتم استدعاؤها فى حالة عدم تمرير أى معاملات للعنصر الجديد للتصنيف.
- فى السطور من ٧ إلى ١٤ تم تعريف دالة الإنشاء الثانية للتصنيف التى يتم استدعاؤها فى حالة تمرير إحداثى للعنصر الجديد للتصنيف. وفى هذه الدالة يتم استخدام الدالة srand() والدالة rand() لتخصيص قيم عشوائية لمتغيرات التصنيف مع تخزين الإحداثى الجديد فى المتغير m_ptPosition. سيتم استخدام هذه المتغيرات فيما بعد داخل الدالة Draw() لرسم مجموعة من البصمات.
- فى السطور من ١٥ إلى ٢٨ تم تعريف دالة الرسم Draw() المستخدمة لرسم مجموعة البصمات.
- فى السطور من ١٧ إلى ١٩ يتم تعريف واختيار قلم الرسم وفرشاة الألوان.
- فى السطور من ٢١ إلى ٢٨ يتم استخدام الدوارة for لرسم ثلاث مستطيلات.

احتواء بيانات الوثيقة

بعد أن قمنا بتعريف تصنيفات التطبيق، يجب أن نقوم بتعريف عناصر تنتمى لهذه التصنيفات لتمثيل بيانات المستخدم كعناصر مستقلة. وحينما تقوم بإنشاء هذه العناصر، يمكنك التعديل فى بياناتها وحفظها أو تحميلها أو حتى حذفها نهائياً. وعندما تقوم بالحفظ أو التحميل، يجب أن تقوم بتسلسل جميع البيانات واحداً بعد الآخر باستخدام التصنيف CObArray وهو أحد تصنيفات مكتبة MFC ويحتوى على مصفوفة قابلة للنمو تحتوى على التصنيفات المنبثقة من التصنيف CObject وعناصر البيانات المرتبطة به. لإضافة

عنصر التصنيف **CObArray** إلى تصنيف الوثيقة **CPersistDoc**، قم بإضافة السطر التالى داخل تعريف التصنيف:

```
// Attributes
public:
	CObArray m_BlobArray;
```

وبذلك يمكنك استخدام العنصر الجديد **m_BlobArray** عند العمل مع عناصر التصنيف المنبثق من التصنيف **CObject** (عناصر التصنيف **CBlob** فى هذا المثال).

من الأشياء الهامة عند العمل مع الوثيقة أن تقوم الوثيقة بتدمير نفسها بمجرد إغلاق التطبيق. قم بإضافة دالة حذف الوثيقة **()DeleteBlobs** إلى تصنيف الوثيقة ثم قم بتعديل كود الدالة كما يلى:

```
void CPersistDoc::DeleteBlobs()
{
	for(int i=0;i<m_BlobArray.GetSize();i++)
		delete m_BlobArray.GetAt(i);
	m_BlobArray.RemoveAll();
}
```

حيث يتم استخدام الدوارة **for** للبحث عن كل عنصر من عناصر بيانات الوثيقة باستخدام الدالة **()GetSize** لمعرفة عدد العناصر ثم حذف كلٍ منها باستخدام الدالة **()GetAt** وأخيراً حذف بيانات مصفوفة العنصر **m_BlobArray**.

يتم بعد ذلك استدعاء هذه الدالة داخل دالة هدم التصنيف **()CpersistDoc~** التى يتم استدعاؤها بمجرد حذف أحد عناصر التصنيف، كما يتم استدعاؤها أيضاً داخل دالة إنشاء وثيقة جديدة **()OnNewDocument** لحذف الوثيقة الحالية من الذاكرة وذلك كما يلى:

```
CPersistDoc::~CPersistDoc()
{
	DeleteBlobs();
}
BOOL CPersistDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;
	// TODO: add reinitialization code here
```

```
        // (SDI documents will reuse this document)
        DeleteBlobs();
        return TRUE;
}
```

وبذلك نكون قد قمنا باحتواء عناصر البيانات وحذفها، لكن يجب أن يكون هناك طريقة تمكن المستخدم من إنشاء عناصر بيانات جديدة ومن ثمَّ إضافتها إلى الوثيقة الحالية. تختلف هذه الطريقة تبعاً للتطبيق وتعتمد اعتماداً كبيراً على واجهة المستخدم. فى المثال الذى بين أيدينا، سيقوم المستخدم بإنشاء عنصر ينتمى للتصنيف **CBlob** بمجرد النقر بزر الفأرة الأيسر فى أى مكان داخل العرض. قم بإضافة دالة احتواء حدث نقر الزر الأيسر **OnLButtonDown()** ثم قم بتعديل كودها كما يلى:

```
void CPersistView::OnLButtonDown(UINT nFlags, CPoint point)
{
        GetDocument()->m_BlobArray.Add(new CBlob(point));
        Invalidate();
}
```

وعن هذا الكود، نوضح ما يلى:

- يتم الحصول على الوثيقة الحالية باستخدام الدالة **GetDocument()**.
- يتم إضافة عنصر جديد ينتمى إلى التصنيف **CBlob** إلى المصفوفة **m_BlobArray** السابق تعريفها مع تمرير إحداثى النقر.
- يتم استدعاء الدالة **Invalidate()** لإعادة رسم العرض باستدعاء الدالة **OnDraw()**.

قم بعد ذلك بتعديل كود الدالة **OnDraw()** لرسم عناصر البيانات المخزَّنة داخل المصفوفة **m_BlobArray** باستدعاء الدالة **Draw()** السابق تعريفها، وذلك كما يلى:

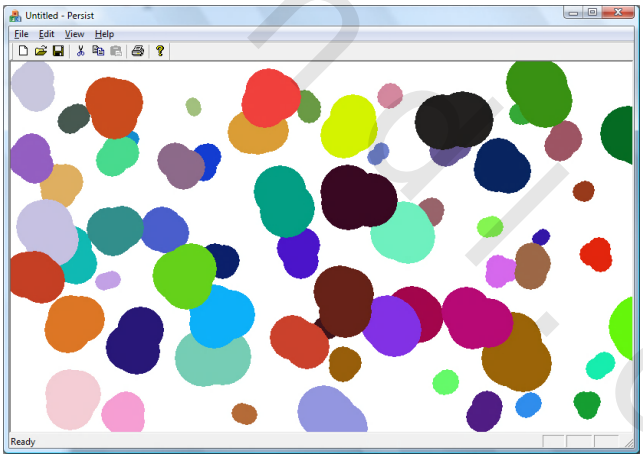
```
1.      void CPersistView::OnDraw(CDC* pDC)
2.      {
3.      CPersistDoc* pDoc = GetDocument();
4.      ASSERT_VALID(pDoc);
5.      // TODO: add draw code for native data here
6.      for(int i=0;i<pDoc->m_BlobArray.GetSize();i++)
7.      {
8.      CBlob* pBlob=(CBlob*)pDoc->m_BlobArray.GetAt(i);
```

```
9.      pBlob->Draw(pDC);
10.     }
11.     }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٦ يتم استخدام الدوارة **for** للمرور على جميع عناصر بيانات المصفوفة **m_BlobArray** كلٍ على حده.
- فى السطر رقم ٨ يتم تخزين عنصر البيانات داخل مؤشر للتصنيف **CBlob**.
- فى السطر رقم ٩ يتم استدعاء الدالة **()Draw** العضو فى التصنيف **CBlob** لرسم عنصر البيانات داخل العرض.

قم بتضمين الملف الرئيسى للتصنيف **CBlob** وهو **blob.h** فى بداية ملف **PersistView.cpp**، ثم قم ببناء التطبيق وتنفيذه. انقر فى أماكن متفرقة من نافذة العرض، تحصل على نتيجة مشابهة لشكل ٢٢-٣.



شكل ٢٢-٣ إظهار البصمات داخل نافذة العرض

تسلسل عناصر البيانات

كى تتمكن من تسلسل عناصر بياناتك، يجب أن تقوم بإضافة نسخة جديدة من الدالة **()Serialize** إلى تصنيفك لتحويل متغيراتك إلى قائمة مرتبة وبالتالى يمكنك تخزينها فى ملف أو تحميلها منه. لإضافة الدالة الجديدة، تابع معنا الخطوات الآتية:

١. نشط التبويب **Class View** إذا لم يكن هو التبويب النشط.
٢. انقر التصنيف **CBlob** بزر الفأرة الأيمن ثم اختر **Add Function**⇐**Add** من القائمة الموضعية، تظهر نافذة معالج إضافة دالة جديدة.
٣. قم بإدخال نوع الدالة فى مربع النص **Return Type** وهو **void** فى هذه الحالة.
٤. قم بإدخال اسم الدالة فى مربع النص **Function name** وهو **Serialize(CArchive& ar)** فى هذه الحالة.
٥. لتعيين معامل الدالة، قم بكتابة اسم المعامل داخل مربع النص **Parameter name** وهو **ar** فى هذه الحالة ثم اكتب نوع المعامل داخل مربع السرد والتحرير **Parameter Type** وهو **&CArchive** فى هذه الحالة وأخيراً انقر زر **Add** لإضافة المعامل إلى الدالة.
٦. نشط مربع الاختيار **Virtual**، تلاحظ إضافة كلمة **virtual** داخل مربع النص **Function signature**.
٧. انقر زر **Finish** لإغلاق نافذة المعالج وإضافة الدالة الجديدة، تلاحظ إضافة الدالة إلى ملف التصنيف الرئيسى **Blob.h** وملف المصدر **Blob.cpp**.

يوجد طريقة أخرى لتعريف هذه الدالة من خلال مربع الخصائص، تابع معنا الخطوات الآتية:

١. انقر التصنيف **CBlob** من داخل تبويب عرض التصنيفات لتنشيطه.
٢. تأكد من ظهور مربع الخصائص أمامك ثم انقر زر **Overrides** من شريط الأدوات.
٣. انقر الدالة **Serialize** ثم اختر **Add <Serialize>** من القائمة المصاحبة، يتم إضافة هيكل الدالة إلى كود التصنيف **CBlob**.

كما ذكرنا من قبل، يتم ربط الملف الموجود على وحدة التخزين بعنصر التصنيف **CArchive** المار للدالة **()Serialize**، حيث يحتوى هذا التصنيف على مجموعة من

الدوال المعرفة كمعاملات ، مثل المعامل >> المستخدم لتخزين قيمة معينة إلى الملف أو المعامل << المستخدم لتحميل قيمة من الملف. كما يمكنك استخدام الدوال IsLoading() و IsStoring() للتعرف على اتجاه نقل البيانات.

قم بتعديل كود الدالة Serialize() التى قمت بإضافتها منذ قليل كما يلى:

```
1.  void CBlob::Serialize(CArchive& ar)
2.  {
3.      if (ar.IsStoring())
4.      {       // storing code
5.          ar << m_ptPosition;
6.          ar << m_crColor;
7.          ar << m_nSize;
8.          ar << m_nShape;
9.      }
10.     else
11.     {       // loading code
12.         ar >> m_ptPosition;
13.         ar >> m_crColor;
14.         ar >> m_nSize;
15.         ar >> m_nShape;
16.     }
17. }
```

ففى السطر رقم ٣ تم استدعاء الدالة IsStoring() داخل عبارة if للتأكد من أن عملية نقل البيانات تتم من العرض إلى الملف الموجود على وحدة التخزين، وبالتالى يتم استخدام المعامل >> لنقل البيانات إلى الملف فى السطور من ٥ إلى ٨، وإلا يتم عكس العملية باستخدام المعامل << لتحميل البيانات من الملف إلى نافذة العرض فى السطور من ١٢ إلى ١٥.

قم بتعديل كود الدالة Serialize() الأساسية كما يلى:

```
void CPersistDoc::Serialize(CArchive& ar)
{
        m_BlobArray.Serialize(ar);
}
```

حيث يتم استدعاء الدالة ()Serialize الجديدة بتمرير المرجع ar الذى يتصل اتصالاً مباشراً بالملف الموجود على وحدة التخزين.

والآن قم ببناء التطبيق وتنفيذه، تلاحظ أنه يمكنك حفظ الملف وإعادة تحميله مرةً أخرى باستخدام خيارات قائمة File مثل Save وSave As وOpen.

استخدام قائمة الملفات المستخدمة حديثًا

عندما تقوم بحفظ عدة ملفات بأسماء مختلفة بالامتداد blb، تلاحظ ظهور قائمة بأسماء هذه الملفات داخل قائمة File والتى يمكنك من خلالها فتح أيٍ من هذه الملفات مباشرةً دون اللجوء للمربع الحوارى Open.

تسجيل أنواع الوثائق

من خلال تعاملك مع التطبيقات العاملة تحت نظام التشغيل Windows، لعلك لاحظت فتح تطبيق معين بمجرد النقر المزدوج على أى ملف ينتمى لهذا التطبيق، وذلك لأن هذا التطبيق ضمن التطبيقات المعرفة لنظام التشغيل. لإضافة التطبيق الحالى إلى قائمة التطبيقات المعرفة، يجب وجود سطرى الكود التاليين للدالة ()InitInstance بعد السطر AddDocTemplate(pDocTemplate); كما يلى:

```
// Enable DDE Execute open
EnableShellOpen();
RegisterShellFileTypes(TRUE);
```

العمل مع الملفات

تستخدم عملية تسلسل البيانات غالباً مع التطبيقات أحادية الوثيقة أو التطبيقات متعددة الوثيقة التى تحتاج لتخزين بياناتها، لكن هناك الكثير من المواقف التى تحتاج فيها إلى إنشاء الملفات مباشرةً أو قراءتها أو حتى كتابتها. يمكنك أداء كل ذلك من خلال التصنيف CFile الذى من خلاله يمكنك أداء العديد من وظائف الملفات.

استخدام التصنيف CFile

يمكنك إنشاء عناصر التصنيف CFile باستخدام إحدى الصيغ الثلاث الآتية:

- لا تحتوى الصيغة الأولى على أية معاملات حيث تقوم بإنشاء عنصر لملف غير مفتوح والذى يمكنك من خلاله استدعاء الدالة **()Open** لإنشاء وفتح ملف موجود على وحدة التخزين.
- تحتوى الصيغة الثانية على معامل واحد من النوع **hFile** يعبر عن ملف مفتوح. يمكنك استخدام هذه الطريقة لربط عنصر **CFile** بملف مفتوح.
- تحتوى الصيغة الثالثة على معاملين، الأول يحتوى على اسم ومسار الملف الذى تريد فتحه أو إنشاءه، أما الثانى فيحتوى على توليفة من معرفات الملف.

فتح الملفات

يعتبر فتح الملفات من العمليات الهامة عند العمل مع الملفات، حيث يمكنك اختيار توليفة من المعرفات التى تتناسب مع الغرض الذى من أجله تم فتح الملف. فقد تكون فى حاجة إلى إنشاء ملف جديد أو فتح ملف موجود للقراءة فقط أو فتح ملف موجود للقراءة والكتابة فى آنٍ واحد، كل هذا يكون باستدعاء الدالة **()Open** وتمرير المعرفات المناسبة للمعامل الثانى من الجدول ٢٢-١ التالى، حيث يحتوى المعامل الأول على اسم الملف المراد إنشاؤه أو فتحه.

جدول ٢٢-١ معرفات فتح الملف باستخدام الدالة **()Open**

القيمة	*الوصف*
CFile::modeCreate	إنشاء ملف جديد دائماً حتى ولو كان اسم الملف موجود
CFile::modeNoTruncate	يستخدم هذا المعرف مع المعرف السابق لإنشاء ملف جديد إذا كان الملف غير موجود أو فتح الملف إذا كان موجود
CFile::modeRead	فتح الملف للقراءة فقط
CFile::modeWrite	فتح الملف للكتابة فقط

القيمة	الوصف
CFile::modeReadWrite	فتح الملف للقراءة والكتابة
CFile::ShareDenyNone	يمكن للعمليات الأخرى أن تقوم بالقراءة من الملف أو الكتابة فيه
CFile::ShareExclusive	لا يمكن للعمليات الأخرى أن تقوم بالقراءة من الملف أو الكتابة فيه طالما كان الملف مفتوحاً فى العمليات الحالية
CFile::ShareDenyRead	لا يمكن للعمليات الأخرى أن تقوم بالقراءة من الملف طالما كان مفتوحاً
CFile::ShareDenyWrite	لا يمكن للعمليات الأخرى أن تقوم بالكتابة فى الملف طالما كان مفتوحاً
CFile::typeText	يستخدم مع تصنيفات منبثقة معينة مع أنواع معينة من معالجات النصوص
CFile::typeBinary	يستخدم فقط مع بعض التصنيفات المنبثقة

فمثلاً، الكود التالى يقوم بفتح الملف **waleed.txt** إذا كان موجوداً (لوجود المعرف **CFile::modeCreate**)، وإلا يتم إنشاء ملف جديد بنفس الاسم (لوجود المعرف **CFile::modeNoTruncate** بجانب المعرف السابق).

```
CFile MyFile;
MyFile.Open("Waleed.txt" , CFile::modeCreate +
   CFile::modeNoTruncate);
```

كما يمكنك استبدال المعامل + بالمعامل | هكذا:

```
CFile MyFile;
MyFile.Open("Waleed.txt",CFile::modeCreate |
   CFile::modeNoTruncate);
```

تقوم الدالة **()Open** بإرجاع القيمة **TRUE** فى حالة نجاح عملية فتح الملف أو إنشاءه أو القيمة **FALSE** فى حالة فشل العملية، لذا يمكنك تمرير معامل ثالث اختيارى عبارة عن مؤشر للتصنيف **CFileException** والذى من خلاله يمكنك التعرف على أسباب فشل عملية فتح الملف أو إنشاءه.

تحرير الملفات

بمجرد فتح الملف يمكنك تحريره بالقراءة منه أو الكتابة فيه تبعاً لخيارات الفتح التى قمت بتحديدها بالمعامل الثانى للدالة **()Open**. حيث يحتوى التصنيف **CFile** على الدوال **()Read** و**()write** التى تقوم بالقراءة من الملف أو الكتابة فيه تبعاً للموضع الحالى للملف. فبمجرد فتح الملف يتم تعيين موضع الملف بحيث يشير لبدايته، فإذا قمت بقراءة ٢٠٠ بايت من الملف، يتم تغيير موضع الملف ليشير إلى البايت رقم ٢٠١.

تحتوى الدالة **()Read** على معاملين، المعامل الأول عبارة عن مؤشر للمخزن المؤقت الذى سيتم وضع البيانات المقروءة فيه، أما المعامل الثانى فيحتوى على كمية البيانات التى سيتم قراءتها. وتقوم الدالة بإرجاع كمية البيانات الحقيقية التى تمت قراءتها، والتى قد تقل عن الكمية التى تم تحديدها إذا كانت كمية البيانات المتبقية فى الملف أقل من التى تم تحديدها، أو قد تكون صفر إذا كان موضع الملف يشير إلى نهايته. أما الدالة **()write** فتحتوى على معاملين أيضاً، الأول عبارة عن المتغير النصى الذى سيتم نسخ بياناته للملف ، والثانى عبارة عن كمية البيانات التى ستتم قراءتها.

فمثلاً، يقوم الكود التالى بفتح الملف **Waleed.txt** للقراءة فقط ثم قراءة ٢٠٠ بايت من بداية الملف:

```
CFile MyFile;
MyFile.Open("Waleed.txt" , CFile::modeRead);
Char arMyReadBuffer[200];
UINT uBytesRead = MyFile.Read(arMyReadBuffer,sizeof
(arMyReadBuffer);
```

للتعرف على طريقة استخدام دوال التصنيف **CFile** للعمل مع الملفات، تابع معنا الخطوات الآتية:

١. قم بإنشاء تطبيق حوارى جديد باسم **FileEdit**.

٢. قم بإضافة مربع نص كبير للمربع الحوارى الرئيسى ثم قم بتخصيص القيمة **True** للخاصية **MultiLine** والخاصية **want return** من مربع الخصائص.

٣. استخدم معالج التصنيفات لربط مربع النص بمتغير نصى من النوع **CString** وليكن **m_EditBox**.

٤. قم بتعديل كود الدالة **()OnInitDialog** لقراءة البيانات من ملف **Waleed.txt** بمجرد تشغيل التطبيق هكذا:

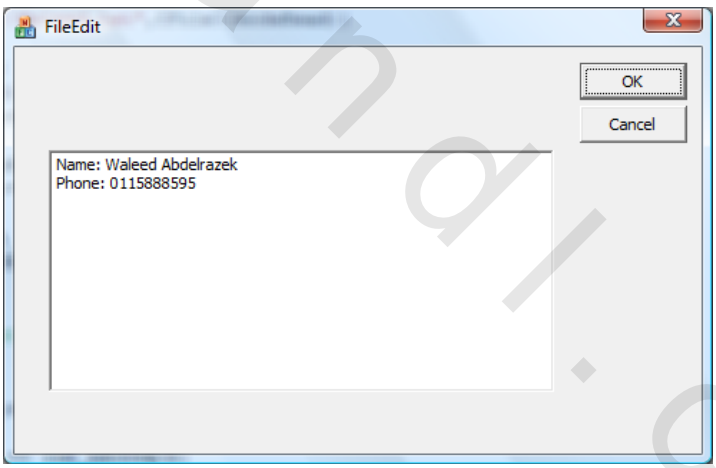
```
1.    // TODO: Add extra initialization here
2.    CFile MyFile;
3.    if(MyFile.Open("C:\\Waleed.txt",CFile::modeRead))
4.    {
5.      char cBuf[512];
6.      UINT uBytesRead;
7.      while(uBytesRead= MyFile.Read(cBuf,sizeof(cBuf)-1))
8.      {
9.        cBuf[uBytesRead] = NULL;
10.       m_EditBox += CString(cBuf);
11.     }
12.     MyFile.Close();
13.     UpdateData(FALSE);
14.   }
15.   return TRUE;  // return TRUE  unless you set the focus to a
16.   control
```

وعن هذا الكود، نوضح ما يلى:

- يتم إضافة الكود السابق للدالة **()OnInitDialog** بعد سطر التعليق رقم ١.
- فى السطر رقم ٣ تم استدعاء الدالة **()Open** لفتح ملف **Waleed.txt** للقراءة فقط (يمكنك إنشاء هذا الملف باستخدام برنامج **Notepad**).

- فى السطر رقم ٧ تم استخدام الدوارة **while** لإضافة بيانات الملف لمربع الـــنص الموجود بالمربع الحوارى.
- فى السطر رقم ٩ يتم تخزين القيمة **NULL** فى البايت الأخير من المصفوفة حتى يتم فصل البيانات الحالية عن البيانات القادمة.
- فى السطر رقم ١٠ يتم إضافة بيانات المصفوفة إلى مربع النص.
- فى السطر رقم ١٢ يتم استدعاء الدالة **()Close** لإغلاق الملف.
- فى السطر رقم ١٣ يتم استدعاء الدالة **()UpdateData** مـــع تمريـــر المعامـــل **FALSE** لإرسال بيانات المتغير النصى إلى مربع النص.

قم ببناء التطبيق وتنفيذه، تلاحظ ظهور بيانات الملف **Waleed.txt** داخل مربع الـــنص بالمربع الحوارى (انظر شكل ٢٢-٤).



شكل ٢٢-٤ إظهار محتويات الملف بمجرد تشغيل التطبيق

ربما أردت تعديل بيانات الملف من خلال مربع النص، وفى هذه الحالة من الأفضل كتابـــة بيانات مربع النص إلى الملف مرةً أخرى بمجرد نقر زر **Ok** فى المربع الحوارى. قم بإنشـــاء دالة احتواء الزر **Ok** ثم قم بتعديل كود الدالة كما يلى:

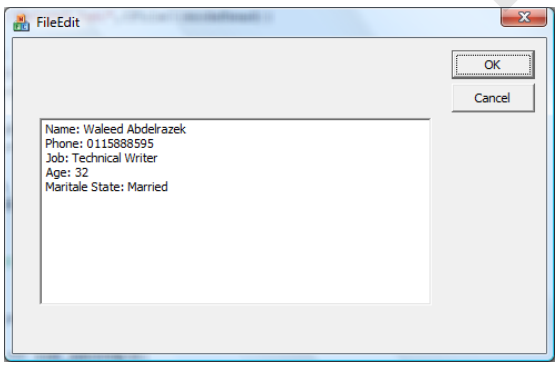
```
1.    void CFileEditDlg::OnBnClickedOK()
2.    {
3.      // TODO: Add extra validation here
```

```
4.      UpdateData(TRUE);
5.      CFile MyFile;
6.      if(MyFile.Open("C:\\Waleed.txt",CFile::modeCreate +
                                          CFile::modeWrite))
7.      {
8.         MyFile.Write((LPCSTR)m_EditBox,m_EditBox.
                                                GetLength());
9.         MyFile.Close();
10.     }
11.     CDialog::OnOK();
12.  }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٤ تم استدعاء الدالة **UpdateData()** مع تمرير القيمة **TRUE** لنقل البيانات من مربع النص إلى المتغير المرتبط به **m_EditBox**.
- فى السطر رقم ٨ يتم استدعاء الدالة **write()** لنسخ بيانات المتغير **m_EditBox**؛ بعد تحويلها إلى صيغة المخزن المؤقت **LPCSTR**، إلى الملف **Waleed.txt** حيث تم تمرير معاملين، الأول يحتوى على اسم المتغير الذى سيتم نسخ البيانات منه، أما الثانى فيحتوى على كمية البيانات المراد نسخها.

قم ببناء التطبيق وتنفيذه ثم قم بتعديل محتويات مربع النص وانقر زر **Ok**. قم بتنفيذ التطبيق مرةً أخرى، تلاحظ ظهور التغييرات التى قمت بإجرائها دلالةً على كتابتها إلى الملف بمجرد نقر زر **Ok** فى المرة السابقة (انظر شكل ٢٢-٥).



شكل ٢٢-٥ تعديل الملف من خلال مربع النص

التحكم فى الموضع الحالى للملف

كما ذكرنا من قبل تتم القراءة من الملف أو الكتابة فيه تبعاً لموضعه الحالى، لذا يمكنك التحكم فى هذا الموضع باستخدام مجموعة من دوال التصنيف **CFile**. فيمكنك مثلاً استخدام الدالة **()GetPosition** للحصول على الموضع الحالى للملف، حيث تقوم الدالة بإرجاع قيمة من النوع **DWORD** تحتوى على الموضع الحالى لمؤشر موضع الملف. يمكنك أيضاً تغيير موضع الملف باستخدام الدوال **()Seek** للذهاب إلى مكان معين أو **()SeekToBegin** للذهاب إلى أول الملف أو **()SeekToEnd** للذهاب إلى نهايته. حيث لا تحتوى الدالتان الأخيرتان إلى معاملات، بينما تحتوى الدالة **()Seek** على معاملين، الأول يعبر عن عدد البايت التى سيتم تجاوزها، والثانى أحد المعرفات الموضحة بالجدول ٢٢-٢ التالى لتحديد اتجاه الانتقال.

جدول ٢٢-٢ معرفات الدالة **()Seek**

القيمة	الوصف
CFile::begin	يكون الموضع الجديد تبعاً لبداية الملف
CFile::end	يكون الموضع الجديد تبعاً لنهاية الملف
CFile::current	يكون الموضع الجديد تبعاً للموضع الحالى للملف

يمكنك تمرير قيم سالبة للدالة **()Seek** إذا أردت الانتقال إلى مكان فى بداية الملف تبعاً لنهاية الملف أو الموضع الحالى له. فإذا أردت مثلاً الانتقال بمقدار ٥٠ بايت للخلف من الموضع الحالى للملف، يمكنك استخدام الكود التالى:

```
LONG lOffSet = MyFile.Seek(-50,CFile::current);
```

كما تقوم الدالة **()Seek** بإرجاع قيمة تعبر عن الموضع الجديد لمؤشر الملف تبعاً لبدايته.

إيجاد معلومات عن الملف

هناك العديد من الدوال التى تقوم بإرجاع معلومات عن الملف المفتوح حالياً والتى يمكن بيانها كما يلى:

- تقوم الدالة **()GetFileName** بإرجاع اسم الملف فقط ولا تقوم بإرجاع المسار.
- تقوم الدالة **()GetFilePath** بإرجاع الاسم الكامل للملف الحالى وكذلك المسار.
- تقوم الدالة **()GetFileTitle** بإرجاع اسم الملف فقط ولا تقوم بإرجاع المسار أو الامتداد.
- تقوم الدالة **()GetLength** بإرجاع قيمة من النوع **DWORD** تحتوى على الطول الحالى للملف، كما يمكنك من خلال الدالة المصاحبة **()SetLength** التحكم فى طول الملف.
- تقوم الدالة **()GetStatus** بإرجاع معلومات عن إنشاء الملف وتعديله أثناء عمليات القراءة والكتابة المختلفة. وتحتوى هذه الدالة على معاملين، الأول اختيارى ويعبر عن اسم الملف، فإن تجاهلته سيتم استخدام الملف المفتوح حالياً. أما الثانى فعبارة عن عنصر ينتمى للتصنيف **CFileStatus** ليحتوى على معلومات الملف. فمثلاً إذا أردت الحصول على بيانات عن الملف **Waleed.txt**، قم بإضافة الكود التالى للدالة **()OnBnClickedOk**:

```
1.  CFileStatus FileStatus;
2.  CFile::GetStatus("C:\\Waleed.txt",FileStatus);
3.  AfxMessageBox(_T("Last Modified on ") +
     FileStatus.m_mtime.Format("%A, %B %d,%Y"));
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ١ تم تعريف عنصر جديد **FileStatus** ينتمى للتصنيف **CFileStatus**.
- فى السطر رقم ٢ تم استدعاء الدالة **()GetStatus** بتمرير اسم الملف والعنصر الذى سيحتوى على معلوماته.
- فى السطر رقم ٣ تم استخدام الدالة **()AfxMessageBox** لإظهار جملة **Last Modified on** ثم استخدام المعامل **m_mtime** العضو فى التصنيف **CFileStatus** والمستخدم للحصول على تاريخ آخر تعديل للملف.

يحتوى الجدول ٢٢-٣ على المعاملات الأخرى المستخدمة لمعرفة بيانات الملف.

جدول ٢٢-٣ معرفات خصائص الملف الأعضاء فى التصنيف CFileStatus

المعامل	الوصف
CTime m_mtime	تاريخ ووقت آخر تعديل
CTime m_atime	تاريخ ووقت آخر قراءة
CTime m_ctime	تاريخ ووقت إنشاء الملف
LONG m_size	حجم الملف بالبايت
BYTE m_attribute	خصائص القراءة والكتابة
char m_szFullName	الاسم الكامل للملف ومساره

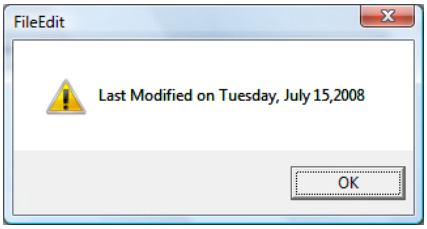
يحتوى المعامل m_attribute على عدة معرفات، يتم اختبار أيٍ منها باستخدام المعامل المنطقى & والتى يمكن بيانها بالجدول ٢٢-٤. فمثلاً الكود التالى يقوم باختبار الملف لنعرف هل هو للقراءة فقط أم لا:

```
if(FileStatus.m_attribute & CFile::readOnly)
        AfxMessageBox("File is Read Only ! ");
```

جدول ٢٢-٤ معرفات المعامل m_attribute

اسم المعرف	القيمة	الوصف
Normal	0x00	ملف عادى
Readonly	0x01	الملف للقراءة فقط
Hidden	0x02	الملف مخفى
System	0x04	الملف نظامى
Volume	0x08	الملف Disk Volume
Directory	0x10	الملف disk directory
Archive	0x20	الملف أرشيفى

والآن قم ببناء التطبيق وتنفيذه. انقر زر Ok تحصل على رسالة مشابهة لشكل ٢٢-٦ التالى.



شكل ٢٢-٦ إظهار تاريخ آخر تعديل للملف

إعادة تسمية الملفات وحذفها

يمكنك استخدام الدالة الساكنة Rename() فى إعادة تسمية الملف. حيث تحتوى هذه الدالة على معاملين، أحدهما الاسم القديم للملف والآخر اسمه الجديد. فمثلاً إذا أردت تغيير اسم الملف Waleed.txt إلى Ali.txt، قم باستخدام الكود التالى:

```
CFile::Rename("C:\\Waleed.txt","C:\\Ali.txt");
```

يمكنك أيضاً استخدام الدالة الثابتة Remove() لحذف الملف نهائياً. فإذا أردت مثلاً حذف الملف Ali.txt، قم باستخدام الكود التالى:

```
CFile::Remove("C:\\Ali.txt");
```

لاحظ استخدام مدى التصنيف CFile قبل كلٍ من الدالتين وذلك لأن كل منهما من النوع Static مثلهما مثل الدالة GetStatus().

التصنيفات المنبثقة من التصنيف CFile

يحتوى التصنيف CFile على الوظائف الأساسية للعمل مع الملفات، ورغم ذلك هناك العديد من التصنيفات المنبثقة عن هذا التصنيف والتى يمكنك استخدامها لأداء وظائف خاصة ومحددة والتى يمكننا إلقاء نظرة سريعة عليها كما يلى:

- يستخدم التصنيف CMemFile لأداء بعض وظائف الملفات التى تتطلب سرعة عالية، حيث يكون التعامل مع الذاكرة بدلاً من وحدة التخزين.

- التصنيف **CSharedFile** حالة خاصة من التصنيف **CMemFile** حيث يقوم بالتعامل مع الذاكرة المشتركة وسنتعرض له لاحقاً عند العمل مع الحافظة **Clipboard**.
- يستخدم التصنيف **COleStreamFile** مع العمليات المتعلقة بكائنات الربط والتضمين **OLE**. وهناك العديد من التصنيفات المنبثقة عن هذا التصنيف مثل **CMonikerFile** و**CAsyncMonikerFile** و**CDataPathProperty** وأخيراً التصنيف **CCachedDataPathProperty**.
- يتيح لك التصنيف **CSocketFile** التعامل مع الشبكات كما لو كانت ملفات لنقل الوثائق المختلفة فيما بينها من خلال تسلسل البيانات **Serialization** باستخدام عنصر ينتمى للتصنيف **CArchive**.
- التصنيف **CResentFileList** غير منبثق من التصنيف **CFile** ولكن يستخدم بكثرة عند العمل مع الملفات لاسترجاع وتعيين الملفات التى تم فتحها مؤخراً.

تناقل البيانات من خلال الحافظة

لعلك لاحظت من خلال دراستك لنظام التشغيل **Windows** والتطبيقات العاملة تحته، كيف أن نسخ البيانات يعنى أخذ نسخة منها إلى الحافظة **Clipboard** ولصقها يعنى لصق البيانات من الحافظة إلى المكان المحدد. ولضمان نسخ البيانات بطريقة صحيحة يجب أن يكون لعمليتى النسخ واللصق نفس التنسيق كما سنرى بعد قليل.

تعيين تنسيقات بيانات الحافظة

أول ما يجب أن تقوم به قبل البدء فى توجيه البيانات إلى الحافظة أو استرجاع البيانات منها أن تجعل تطبيقك يدعم تنسيقات البيانات المختلفة المستخدمة من قِبَل هذا التطبيق. وقد تكون التنسيقات التى يستخدمها تطبيقك إحدى التنسيقات القياسية المعروفة والموضحة بالجدول ٢٢-٥، وإلا كان عليك إنشاء تنسيقات خاصة تتلاءم مع تطبيقك باستخدام

الدالة **RegisterClipboardFormat()**، وفى جميع الحالات يكون التنسيق عبارة عن قيمة من النوع **UINT**.

جدول ٢٢-٥ تنسيقات الحافظة القياسية

معرف التنسيق	الوصف
CF_TEXT	هو التنسيق الأكثر شيوعاً ويستخدم لنقل النصوص التى تحتوى على علامات <CR> و <LF>
CF_BITMAP	يستخدم فى حالة نقل بيانات صورة نقطية
CF_DIB	يستخدم فى حالة نقل صورة نقطية لا تعتمد على جهاز معين DIB
CF_TIFF	يستخدم فى حالة نقل صورة من النوع TIFF
CF_WAVE	يستخدم فى حالة نقل بيانات صوتية
CF_PALETTE	يستخدم فى حالة نقل صورة ملونة Palette
CF_METAFILEPICT	يستخدم فى حالة نقل صور من النوع metafile

بالرجوع إلى التطبيق **Persist** الذى قمنا بإنشائه فى بداية هذا الفصل، سنقوم بإضافة تنسيق خاص للقص واللصق باستخدام الدالة **RegisterClipboardFormat()**. قم بتعديل كود دالة إنشاء التصنيف **CPersistDoc** كما يلى:

```
CPersistDoc::CPersistDoc()
{
// TODO: add member initialization code here
        m_uClipFormat =
RegisterClipboardFormat("PersistClipFormat");
}
```

حيث تم تخزين اسم التنسيق الجديد **PersistClipFormat** فى المتغير **m_uClipFormat** حتى يمكنك استخدامه لاحقاً مع عنصر الوثيقة. قم بإضافة تعريف

المتغير داخل تعريف التصنيف CPersistDoc بعد تعريف المتغير m_BlobArray هكذا:

```
public:
CObArray m_BlobArray;
UINT m_uClipFormat;
```

نسخ البيانات إلى الحافظة

الخطوة التالية بعد تعريف التنسيق المستخدم من قبل الحافظة هو تمثيل عمليات النسخ والقص، وكما تعلم فإن عملية القص حالة خاصة من عملية النسخ وتختلف عنها فى حذف البيانات بعد نسخها. للتعرف على طريقة تمثيل عمليات النسخ والقص، تابع معنا الخطوات الآتية:

١. من نافذة محرر الموارد، قم بإضافة دالة احتواء الخيار Copy داخل قائمة Edit وهى OnEditCopy().

٢. قم بتعديل كود الدالة OnEditCopy() كما يلى:

```
1.  void CPersistDoc::OnEditCopy()
2.  {
3.    COleDataSource* pDataSource = new COleDataSource;
4.    CSharedFile fileClipCopy;
5.    CArchive arClipCopy(&fileClipCopy,CArchive::store);
6.    Serialize(arClipCopy);
7.    arClipCopy.Close();
8.    HANDLE hGlobalMem = fileClipCopy.Detach();
9.    if(hGlobalMem)
10.   {
11.     pDataSource->CacheGlobalData
                          (m_uClipFormat,hGlobalMem);
12.     pDataSource->SetClipboard();
13.   }
14.   else
15.     AfxMessageBox("Can't alloc memory for copy");
16. }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٣ تم إنشاء عنصر **COleDataSource** كى يمكنك من خلاله العمل مع جميع أنواع البيانات لنقل البيانات إلى التطبيقات المختلفة، ليس فقط باستخدام الحافظة ولكن باستخدام تقنية السحب والإفلات وتقنيات نقل البيانات الأخرى.
- فى السطر رقم ٤ تم تعريف عنصر ملف من النوع **CSharedFile** السابق شرحه، حيث من خلاله يتم تلقائياً فتح ملف داخل الذاكرة وحجز جزء منها لمحتوياته.
- فى السطور ٥ و ٦ يتم استخدام نفس عملية تسلسل البيانات التى قمنا باستخدامها من قبل مع التصنيف **CBlob** لتخزين البيانات بملف بالذاكرة المشتركة. ففى السطر رقم ٥ تم تعريف عنصر ينتمى للتصنيف **CArchive** وربطه بالملف الموجود بالذاكرة المشتركة، مع تمرير المعرف **store** مما يتسبب فى إرجاع الدالة **()IsStoring** للقيمة **TRUE** ومن ثمَّ تخزين بيانات الأرشيف بالملف الموجود بالذاكرة.
- فى السطر رقم ٧ يتم استدعاء الدالة **()Close** لإغلاق ملف الأرشيف الموجود على وحدة التخزين لعدم الحاجة إليه.
- فى السطر رقم ٨ يتم فصل الملف الموجود بالذاكرة المشتركة عن ملف الأرشيف.
- فى السطر رقم ١١ يتم تخزين البيانات؛ ذات التنسيق الذى قمنا بتعريفه، داخل مصدر البيانات **OLE**.
- فى السطر رقم ١٢يتم استدعاء الدالة **()SetClipboard** لتخزين محتويات مصدر البيانات داخل الحافظة.
- فى حالة عدم نجاح عملية الفصل بالسطر رقم ٨، يتم إظهار رسالة تفيد بعدم القدرة على حجز الذاكرة كما فى سطر ١٥.
- لا تنسى تضمين الملفات الرئيسية الآتية والتى تحتوى على بعض وظائف كائنات **OLE** والحافظة داخل ملف **PersistDoc.h**:

```
#include "afxadv.h"
```

```
#include "afxole.h"
class CPersistDoc : public CDocument
```

لأننا نقوم باستخدام كائن OLE، يجب إعطاء قيمة ابتدائية لمكتبة OLE باستدعاء الدالة AfxOleInit() داخل الدالة InitInstance() كما يلى:

```
BOOL CPersistApp::InitInstance()
{
AfxEnableControlContainer();
AfxOleInit();
.....
}
```

٣. من نافذة محرر الموارد، قم بإضافة دالة احتواء الخيار Cut من قائمة Edit وهى OnEditCut().

٤. قم بتعديل كود الدالة OnEditCut() كما يلى:

```
void CPersistDoc::OnEditCut()
{
        OnEditCopy();
        DeleteBlobs();
        UpdateAllViews(NULL);
}
```

وكما ترى يتم استدعاء دالة احتواء النسخ ثم بعد ذلك يتم حذف بيانات الوثيقة باستخدام الدالة DeleteBlobs() ثم تحديث العرض باستدعاء الدالة UpdateAllViews() مع تمرير المعامل NULL لمسح نافذة العرض.

لصق البيانات من الحافظة

الخطوة الأخيرة هى تمثيل عملية لصق البيانات من الحافظة والتى تم نسخها أو قصها قبل ذلك باستخدام الدالة OnEditCopy() أو الدالة OnEditCut(). تابع معنا الخطوات الآتية:

١. من نافذة محرر الموارد، قم بإضافة دالة احتواء الخيار Paste من قائمة Edit وهى OnEditPaste().

٢. قم بتعديل كود الدالة OnEditPaste() كما يلى:

```
1.    void CPersistDoc::OnEditPaste()
```

```
2.    {
3.      COleDataObject oleTarget;
4.      oleTarget.AttachClipboard();
5.      if(oleTarget.IsDataAvailable(m_uClipFormat))
6.      {
7.        HANDLE hGlobalMem = oleTarget.GetGlobalData
                                        (m_uClipFormat);
8.        if(hGlobalMem)
9.        {
10.           CSharedFile fileTarget;
11.           fileTarget.SetHandle(hGlobalMem);
12.           CArchive arTarget(&fileTarget,CArchive::load);
13.           DeleteBlobs();
14.           Serialize(arTarget);
15.           UpdateAllViews(NULL);
16.       }
17.     }
18.   }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٣ تم تعريف العنصر **oleTarget** الذى ينتمى للتصنيف **COleDataObject** والذى سيستخدم للتأكد من وجود التنسيق المطلوب داخل الحافظة.
- فى السطر رقم ٤ تم استدعاء الدالة **()AttachClipboard** لربط عنصر **OLE** بالحافظة.
- فى السطر رقم ٥ تم استدعاء الدالة **()IsDataAvailable** مع تمرير اسم التنسيق **m_uClipFormat** للتأكد من وجود هذا التنسيق داخل الحافظة.
- فى السطر رقم ٧ تم استدعاء الدالة **()GetGlobalData** للحصول على بيانات الحافظة ذات التنسيق **m_uClipFormat**.
- فى السطر رقم ١٠ تم تعريف متغير ملف من النوع **CSharedFile** والذى من خلاله يتم تلقائياً فتح ملف داخل الذاكرة.
- فى السطر رقم ١١ تم توجيه مؤشر الملف إلى المكان المحجوز بالذاكرة المشتركة.

- فى السطر رقم ١٢ يتم تعريف ملف الأرشيف وربطه بالملف الموجود بالذاكرة مع تمرير المعامل load مما يتسبب فى إرجاع الدالة ()IsLoading للقيمة TRUE ومن ثمَّ تخزين بيانات الملف الموجود بالذاكرة بملف الأرشيف.
- فى السطر رقم ١٣ تم استدعاء الدالة ()DeleteBlobs لحذف البيانات الحالية للوثيقة.
- فى السطر رقم ١٤ تم استدعاء الدالة ()Serialize لتحميل بيانات ملف الذاكرة.
- فى السطر رقم ١٥ يتم استدعاء الدالة ()UpdateAllViews مع تمرير المعامل NULL لتحديث العرض بالبيانات الجديدة.

يمكنك إضافة دالة واجهة المستخدم للخيار Paste حتى تتمكن من تنشيط الخيار أو تعطيله تبعاً لمحتويات الحافظة. قم بإنشاء دالة واجهة المستخدم ()OnUpdateEditPaste ثم قم بتعديل كود الدالة كما يلى:

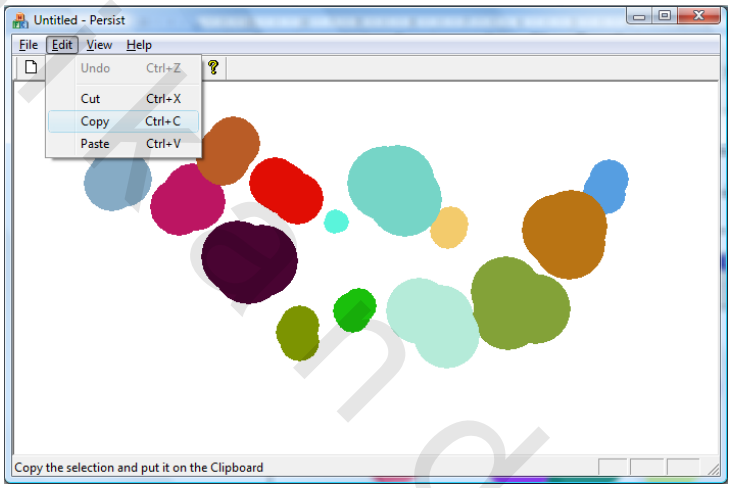
```
1.    void CPersistDoc::OnUpdateEditPaste(CCmdUI* pCmdUI)
2.    {
3.      COleDataObject      oleTarget;
4.      oleTarget.AttachClipboard();
5.      pCmdUI->Enable(oleTarget.IsDataAvailable
                                    (m_uClipFormat));
6.    }
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ٣ تم تعريف العنصر oleTarget الذى ينتمى للتصنيف COleDataObject.
- فى السطر رقم ٤ تم استدعاء الدالة ()AttachClipboard لربط عنصر OLE بالحافظة.
- فى السطر رقم ٥ تم استدعاء الدالة ()Enable مع تمرير القيمة TRUE فى حالة وجود بيانات بالحافظة وبالتالى تمكين الخيار Paste، أو القيمة FALSE فى حالة

عدم وجود بيانات بالحافظة وبالتالى تعطيل الخيار **Paste**. يتم تعيين هذه القيمة من خلال الدالة **IsDataAvailable()**.

والآن قم ببناء التطبيق وتنفيذه. انقر عدة مرات داخل نافذة العرض ثم اختـــر **Copy** أو **Cut** ثم بعد ذلك اختر **Paste**، تلاحظ لصق بيانات العرض التى قمت بقصها مرةً أخرى (انظر شكل ٢٢-٧).



شكل ٢٢-٧ قص ولصق بيانات نافذة العرض باستخدام الحافظة

تذكر أنك لإنشاء دالة احتواء أحد عناصر القائمة أو دالة واجهة المســـتخدم، تقوم بنقر العنصر داخل نافذة تحرير القائمة بزر الفأرة الأيمن ثم اختيـــار **Add Handler Function** من القائمة الموضعية، وفى هذه الحالة تظهر نافذة المعالج الـــتى يمكنـــك مـــن خلالهـــا اختيـــار **Command** لدالـــة الاحتـــواء أو **Update_Command_UI** لدالة واجهة المستخدم.