

## الفصل الثاني والعشرون العمل مع الملفات

تحتوى مكتبة **MFC** على العديد من الدوال العاملة مع التطبيقات أحادية الوثيقة أو التطبيقات متعددة الوثيقة والتي 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.

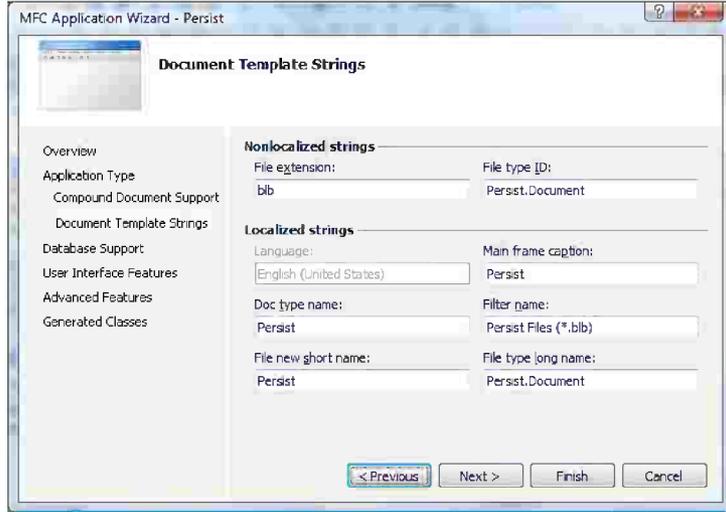
تستخدم تقنية تسلسل البيانات **Serialization** لتحويل بيانات تطبيقك إلى قائمة مرتبة من عناصر البيانات الفردية كي يتم تخزينها على إحدى وحدات التخزين المعروفة مثل القرص المرن أو القرص الصلب أو حتى نقلها إلى إحدى التطبيقات الأخرى، وبمجرد تلقي مجموعة من عناصر البيانات المسلسلة، يقوم البرنامج بقراءة هذه البيانات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

### إنشاء تطبيق أحادي الوثيقة يدعم العمل مع الملفات

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

1. تأكد أنك داخل بيئة التطوير المتكاملة وإلا قم بفتحها من قائمة **Start**.
2. 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**.
3. تأكد من اختيار المجلد الفرعى **MFC** داخل المجلد **Visual C++** بالجانب الأيسر من المربع الحوارى واختيار رمز **MFC Application** من المربع **Templates** بالجانب الأيمن.
4. اكتب اسم مناسب للمشروع الجديد وليكن **Persist** في مربع النص **Name**. قم أيضاً بتحديد المجلد الذى سيحتوى على جميع ملفات المشروع داخل مربع النص **Location**.
5. 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 الذي ترغب في 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 أنواع الملفات المستخدمة بالتطبيق لإضافته إلى قائمة التطبيقات المسجلة بالنظام حتى يتم تنفيذ التطبيق بمجرد فتح ملف يحتوي على الامتداد السابق تعيينه بالخطوة السابقة. اترك النص الافتراضي **Persist.Document** أو قم بكتابة نص آخر؛ إن أردت، داخل مربع النص **File type ID**.
٨. قم بتعيين نوع الملفات التي سيتم عرضها داخل المربعات الحوارية **Open** أو **Save** أو **Save As** داخل مربع النص **Filter name**. اترك النص الافتراضي **Persist Files (\*.blb)** لعرض الملفات ذات الامتداد **blb** فقط.
٩. قم بتعيين الاسم الافتراضي المستخدم عند تخزين الملفات داخل مربع النص **File New Short Name**. اترك النص الافتراضي **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 النصي 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 الأساسي CObject والثاني هو التصنيف CArchive المستول 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 الأساسي CObject، حيث تستخدم هذه الدالة لتحميل عناصر البيانات وتخزينها من وإلى الملفات.

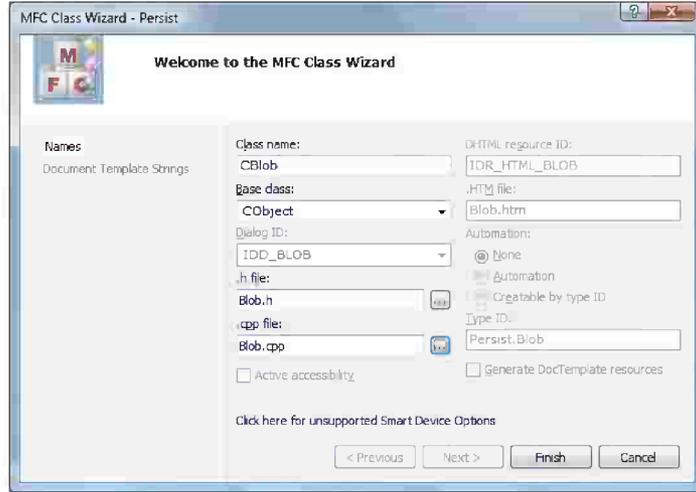
عند حفظ ملف أو تحميله باختيار Save أو Open من قائمة File، يتم استدعاء الدالة Serialize() بتمرير المرجع ar الذى ينتمى للتصنيف CArchive ويحتوى على متغير الملف المضمن داخل التصنيف والمتصل اتصالاً مباشراً بالملف الحقيقى على وحدة التخزين، ومن ثم يتم استخدام هذا المرجع داخل هيكل الدالة لاستدعائها بعدد مرات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

عند حفظ بيانات الوثيقة، يتم قراءة البيانات من جميع العناصر المسلسلة ومن ثم تخزينها داخل الملف المتصل بالأرشفيف، لذا يطلق على الملف المنشأ بواسطة عملية تسلسل البيانات ملف الأرشفيف 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 بالأرشفيف 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 التالي 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 ptPosition);
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 يلي:

- فى السطور رقم ١ و ٢ و ١٥ يتم التأكد من عدم تعريف التصنيف مرتين باستخدام التركيب `#ifndef` `#endif`.
  - فى السطر رقم ٣ تم تعريف التصنيف الجديد `CBlob` المنبثق من التصنيف `CObject` ونصح 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 ٧ و ٨ 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 يلي:

```
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
20.                                     (NULL_PEN);
21.     srand(m_nShape);
22.     for(int n=0;n<3;n++)
23.     {
24.         CPoint ptBlob(m_ptPosition);
25.         ptBlob+=CPoint(rand()%m_nSize, rand()%m_nSize);
26.         CRect rcBlob(ptBlob,ptBlob);
27.         rcBlob.InflateRect(m_nSize,m_nSize);
28.         pDC->Ellipse(rcBlob);
29.     }
30.     pDC->SelectObject(pOldBrush);
31.     pDC->SelectObject(pOldPen);
32. }
```

وعن هذا الكود، نوضح ما يلي:

- في السطور ١ و ٢ تم تضمين الملفات الرئيسية التي تحتوى على تعريف التصنيف الجديد والتعريفات الأساسية لتصنيفات مكتبة MFC.
- في 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 والتي 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 التي يتم استدعاؤها فى حالة تمرير إحداثى للعنصر الجديد للتصنيف. وفى هذه الدالة يتم استخدام الدالة **srand()** والدالة **rand()** لتخصيص قيم عشوائية لمغيرات 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 بيانها 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 التالي 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** في 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 يلي:

```
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 كل منها باستخدام الدالة **GetAt()** وأخيراً حذف بيانات مصفوفة العنصر **m\_BlobArray**.

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

```
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 ثم إضافتها إلى الوثيقة الحالية. تختلف هذه الطريقة تبعاً للتطبيق وتعتمد اعتماداً كبيراً على واجهة المستخدم. في المثال الذي بين أيدينا، سيقوم المستخدم بإنشاء عنصر ينتمي للتصنيف **CBlob** بمجرد النقر بزر الفأرة الأيسر في أي مكان داخل العرض. قم بإضافة دالة احتواء حدث نقر الزر الأيسر **OnLButtonDown()** ثم قم بتعديل كودها كما يلي:

```

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

```

وعن هذا الكود، نوضح ما يلي:

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

قم بعد ذلك بتعديل كود الدالة **OnDraw()** لرسم عناصر البيانات المخزنة داخل المصفوفة **m\_BlobArray** باستدعاء الدالة **Draw()** السابق تعريفها، وذلك كما يلي:

```

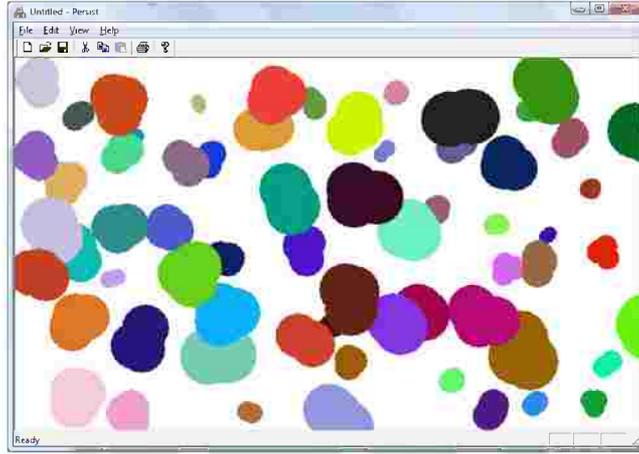
1. void CPersistView::OnDraw(CDC* pDC)
2. {
3.     CPersistDoc* pDoc = GetDocument();
4.     ASSERT_VALID(pDoc);
5.     // TODO: add draw code for native data here
6.     for(int i=0;i<pDoc->m_BlobArray.GetSize();i++)
7.     {
8.         CBlob* pBlob=(CBlob*)pDoc->m_BlobArray.GetAt(i);

```

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

وعن هذا الكود، نوضح ما يلي:

- في السطر رقم ٦ يتم استخدام الدوارة `for` للمرور على جميع عناصر بيانات المصفوفة `m_BlobArray` كل على حده.
  - في السطر رقم ٨ يتم تخزين عنصر البيانات داخل مؤشر للتصنيف `CBlob`.
  - في السطر رقم ٩ يتم استدعاء الدالة `Draw()` العضو في التصنيف `CBlob` لرسم عنصر البيانات داخل العرض.
- قم بتضمين الملف الرئيسي للتصنيف `CBlob` وهو `blob.h` في بداية ملف `PersistView.cpp`، ثم قم ببناء التطبيق وتنفيذه. انقر في أماكن متفرقة من نافذة العرض، تحصل على نتيجة مشابهة لشكل ٢٢-٣.



شكل ٢٢-٣ إظهار البصمات داخل نافذة العرض

### تسلسل عناصر البيانات

كحي تتمكن من تسلسل عناصر بياناتك، يجب أن تقوم بإضافة نسخة جديدة من الدالة `Serialize()` إلى تصنيفك لتحويل متغيراتك إلى قائمة مرتبة وبالتالي يمكنك تخزينها في 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** من القائمة الموضوعية، تظهر نافذة معالج إضافة دالة جديدة.
٣. قم بإدخال نوع الدالة في مربع النص **Return Type** وهو **void** في هذه الحالة.
٤. قم بإدخال اسم الدالة في مربع النص **Function name** وهو **Serialize(CArchive& ar)** في هذه الحالة.
٥. لتعيين معامل الدالة، قم بكتابة اسم المعامل داخل مربع النص **Parameter name** وهو **ar** في هذه الحالة ثم اكتب نوع المعامل داخل مربع السرد والتحرير **Parameter Type** وهو **CArchive&** في 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 الرئيسي **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 **Serialize <Add>** 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 يحتوي 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 **IsStoring()** و **IsLoading()** للتعرف على اتجاه نقل البيانات.

قم بتعديل كود الدالة **Serialize()** التي قمت بإضافتها منذ قليل كما يلي:

```
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. }
```

ففي 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، وبالتالي يتم استخدام المعامل << لنقل البيانات إلى الملف في السطور من ٥ إلى ٨، وإلا يتم عكس العملية باستخدام المعامل >> لتحميل البيانات من الملف إلى نافذة العرض في السطور من ١٢ إلى

.١٥

قم بتعديل كود الدالة **Serialize()** الأساسية كما يلي:

```
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** و **Save As** و **Open**.

### استخدام قائمة الملفات المستخدمة حديثاً

عندما تقوم بحفظ عدة ملفات بأسماء مختلفة بالامتداد **blb**، تلاحظ ظهور قائمة بأسماء هذه الملفات داخل قائمة **File** والتي يمكنك من خلالها فتح أى 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 المعروفة لنظام التشغيل. لإضافة التطبيق الحالى إلى قائمة التطبيقات المعروفة، يجب وجود سطرى الكود التاليين للدالة **InitInstance()** بعد السطر **AddDocTemplate(pDocTemplate)** كما يلي:

```
// 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 تحتوي الصيغة الأولى على أية معاملات حيث تقوم بإنشاء عنصر الملف غير مفتوح والذي يمكنك من خلاله استدعاء الدالة **Open()** لإنشاء وفتح ملف موجود على وحدة التخزين.
- تحتوي الصيغة الثانية على معامل واحد من النوع **hFile** يعبر عن ملف مفتوح. يمكنك استخدام هذه الطريقة لربط عنصر **CFile** بملف مفتوح.
- تحتوي الصيغة الثالثة على معاملين، الأول يحتوي على اسم ومسار الملف الذي تريد فتحه أو إنشائه، أما الثاني فيحتوي على توليفة من معرفات الملف.

### فتح الملفات

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

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

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

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

فمثلاً، الكود التالي 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** في حالة نجاح عملية فتح الملف أو إنشائه أو القيمة **FALSE** في حالة فشل العملية، لذا يمكنك تمرير معامل ثالث اختياري عبارة عن مؤشر للتصنيف **CFileException** والذي من خلاله يمكنك التعرف على أسباب فشل عملية فتح الملف أو إنشائه.



### تحرير الملفات

بمجرد فتح الملف يمكنك تحريره بالقراءة منه أو الكتابة فيه تبعاً لخيارات الفتح التي قمت بتحديدتها بالمعامل الثاني للدالة **Open()**. حيث يحتوي التصنيف **CFile** على الدوال **Read()** و **write()** التي تقوم بالقراءة من الملف أو الكتابة فيه تبعاً للموضع الحالي 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.

تحتوي الدالة **Read()** على معاملين، المعامل الأول عبارة عن مؤشر للمخزن المؤقت الذي سيتم وضع البيانات المقروءة فيه، أما المعامل الثاني فيحتوي على كمية البيانات التي سيتم قراءتها. وتقوم الدالة بإرجاع كمية البيانات الحقيقية التي تمت قراءتها، والتي قد تقل عن الكمية التي تم تحديدها إذا كانت كمية البيانات المتبقية في الملف أقل من التي تم تحديدها، أو قد تكون صفر إذا كان موضع الملف يشير إلى نهايته. أما الدالة **write()** فتحتوي على معاملين أيضاً، الأول عبارة عن المتغير النصي الذي سيتم نسخ بياناته للملف ، والثاني عبارة عن كمية البيانات التي سيتم قراءتها.

فمثلاً، يقوم الكود التالي 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:

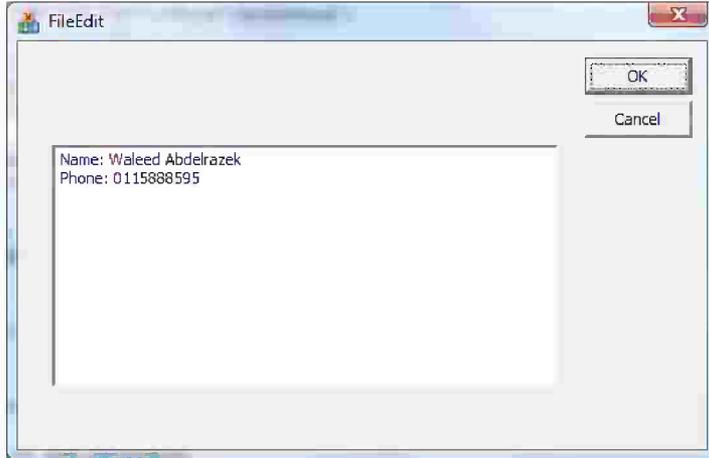
```

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).

- في السطر رقم ٧ تم استخدام الدوارة **while** لإضافة بيانات الملف لمربع النص الموجود بالمربع الحوارى.
  - في السطر رقم ٩ يتم تخزين القيمة **NULL** فى البايٲ 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()** مع تمرير المعامل **FALSE** لإرسال بيانات المتغير النصى إلى مربع النص.
- قم ببناء التطبيق وتنفيذه، تلاحظ ظهور بيانات الملف **Waleed.txt** داخل مربع النص 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 كتابة بيانات مربع النص إلى الملف مرة أخرى بمجرد نقر زر **Ok** فى المربع الحوارى. قم بإنشاء 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 يلي:

- في السطر رقم ٤ تم استدعاء الدالة UpdateData() مع تمرير القيمة TRUE لنقل البيانات من مربع النص إلى المتغير المرتبط به m\_EditBox.
  - في 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 يحتوي على اسم المتغير الذي سيتم نسخ البيانات منه، أما الثاني فيحتوي على كمية البيانات المراد نسخها.
- قم ببناء التطبيق وتنفيذه ثم قم بتعديل محتويات مربع النص وانقر زر Ok. قم بتنفيذ التطبيق مرة أخرى، تلاحظ ظهور التغييرات التي قمت بإجرائها دلالةً على كتابتها إلى الملف بمجرد نقر زر Ok في المرة السابقة (انظر شكل ٥-٢٢).



شكل ٥-٢٢ تعديل الملف من خلال مربع النص

### التحكم في الموضع الحالي للملف

كما ذكرنا من قبل تتم القراءة من الملف أو الكتابة فيه تبعاً لموضعه الحالي، لذا يمكنك التحكم في هذا الموضع باستخدام مجموعة من دوال التصنيف **CFile**. فيمكنك مثلاً استخدام الدالة **GetPosition()** للحصول على الموضع الحالي للملف، حيث تقوم الدالة بإرجاع قيمة من النوع **DWORD** تحتوى على الموضع الحالي 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 معلمين، الأول يعبر عن عدد البايت التي سيتم تجاوزها، والثاني أحد المعرفات الموضحة بالجدول ٢-٢٢ التالي لتحديد اتجاه الانتقال.

جدول ٢-٢٢ معرفات الدالة **Seek()**

الوصف	القيمة
يكون الموضع الجديد تبعاً لبداية الملف	<b>CFile::begin</b>
يكون الموضع الجديد تبعاً لنهاية الملف	<b>CFile::end</b>
يكون الموضع الجديد تبعاً للموضع الحالي للملف	<b>CFile::current</b>

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

```
LONG IOffset = 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 التي تقوم بإرجاع معلومات عن الملف المفتوح حالياً والتي يمكن بيانها كما يلي:

- تقوم الدالة **GetFileName()** بإرجاع اسم الملف فقط ولا تقوم بإرجاع المسار.
- تقوم الدالة **GetFilePath()** بإرجاع الاسم الكامل للملف الحالي 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 الحالي للملف، كما يمكنك من خلال الدالة المصاحبة **SetLength()** التحكم في طول الملف.
- تقوم الدالة **GetStatus()** بإرجاع معلومات عن إنشاء الملف وتعديله أثناء عمليات القراءة والكتابة المختلفة. وتحتوى هذه الدالة على معاملين، الأول اختياري ويعبر عن اسم الملف، فإن تجاهلته سيتم استخدام الملف المفتوح حالياً. أما الثاني فعبارة عن عنصر ينتمى للتصنيف **CFileStatus** ليحتوى على معلومات الملف. فمثلاً إذا أردت الحصول على بيانات عن الملف **Waleed.txt**، قم بإضافة الكود التالي 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 يلي:

- في السطر رقم ١ تم تعريف عنصر جديد **FileStatus** ينتمى للتصنيف **CFileStatus**.
- في السطر رقم ٢ تم استدعاء الدالة **GetStatus()** بتمرير اسم الملف والعنصر الذى سيحتوى على معلوماته.
- في السطر رقم ٣ تم استخدام الدالة **AfxMessageBox()** لإظهار جملة **Last Modified on** ثم استخدام المعامل **m\_mtime** العضو في 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 في 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 اختيار أى منها باستخدام المعامل المنطقي & والتي يمكن بيانها بالجدول ٢٢-٤. فمثلاً الكود التالى يقوم باختبار الملف لمعرفة 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 ٦-٢٢ التالي.



شكل ٦-٢٢ إظهار تاريخ آخر تعديل للملف

### إعادة تسمية الملفات وحذفها

يمكنك استخدام الدالة الساكنة **Rename()** في إعادة تسمية الملف. حيث تحتوي 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 التالي:

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

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

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

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



### التصنيفات المنبثقة من التصنيف **CFile**

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

- يستخدم التصنيف **CMemFile** لأداء بعض وظائف الملفات التي 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 وستعرض 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 **CDataPathProperty** و **CAsyncMonikerFile** و **CMonikerFile** 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 ينتمي للتصنيف **CArchive**.
- التصنيف **CResentFileList** غير منبثق من التصنيف **CFile** ولكن يستخدم بكثرة عند العمل مع الملفات لاسترجاع وتعيين الملفات التي 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 لعملية النسخ واللصق نفس التنسيق كما سنرى بعد قليل.

#### تعيين تنسيقات بيانات الحافظة

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

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

جدول ٢٢-٥ تنسيقات الحافظة القياسية

الوصف	معرف التنسيق
هو التنسيق الأكثر شيوعاً ويستخدم لنقل النصوص التي تحتوي على علامات <code>&lt;CR&gt;</code> و <code>&lt;LF&gt;</code>	<code>CF_TEXT</code>
يستخدم في حالة نقل بيانات صورة نقطية	<code>CF_BITMAP</code>
يستخدم في حالة نقل صورة نقطية لا تعتمد على جهاز معين <code>DIB</code>	<code>CF_DIB</code>
يستخدم في حالة نقل صورة من النوع <code>TIFF</code>	<code>CF_TIFF</code>
يستخدم في حالة نقل بيانات صوتية	<code>CF_WAVE</code>
يستخدم في حالة نقل صورة ملونة <code>Palette</code>	<code>CF_PALETTE</code>
يستخدم في حالة نقل صور من النوع <code>metafile</code>	<code>CF_METAFILEPICT</code>

بالرجوع إلى التطبيق `Persist` الذي قمنا بإنشائه في بداية هذا الفصل، سنقوم بإضافة تنسيق خاص للقص واللصق باستخدام الدالة `RegisterClipboardFormat()`. قم بتعديل كود دالة إنشاء التصنيف `CPersistDoc` كما يلي:

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

حيث تم تخزين اسم التنسيق الجديد `PersistClipFormat` في 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 في 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** وهي **OnEditCopy()**.

٢. قم بتعديل كود الدالة **OnEditCopy()** كما يلي:

```
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  
12.            (m_uClipFormat,hGlobalMem);  
13.        pDataSource->SetClipboard();  
14.    }  
15.    else  
16.        AfxMessageBox("Can't alloc memory for copy");  
17. }
```

وعن هذا الكود، نوضح ما يلي:

- في السطر رقم ٣ تم إنشاء عنصر **COleDataSource** كي 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.
- في السطر رقم ٤ تم تعريف عنصر ملف من النوع **CSharedFile** السابق شرحه، حيث من خلاله يتم تلقائياً فتح ملف داخل الذاكرة وحجز جزء منها لتحتوياته.
- في السطور ٥ و ٦ يتم استخدام نفس عملية تسلسل البيانات التي 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 في إرجاع الدالة **IsStoring()** للقيمة **TRUE** ومن ثم تخزين بيانات الأرشيف بالملف الموجود بالذاكرة.
- في السطر رقم ٧ يتم استدعاء الدالة **Close()** لإغلاق ملف الأرشيف الموجود على وحدة التخزين لعدم الحاجة إليه.
- في السطر رقم ٨ يتم فصل الملف الموجود بالذاكرة المشتركة عن ملف الأرشيف.
- في السطر رقم ١١ يتم تخزين البيانات؛ ذات التنسيق الذى قمنا بتعريفه، داخل مصدر البيانات **OLE**.
- في السطر رقم ١٢ يتم استدعاء الدالة **SetClipboard()** لتخزين محتويات مصدر البيانات داخل الحافظة.
- في حالة عدم نجاح عملية الفصل بالسطر رقم ٨، يتم إظهار رسالة تفيد بعدم القدرة على حجز الذاكرة كما في سطر ١٥.
- لا تنسى تضمين الملفات الرئيسية الآتية والتي 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 يلي:
```

```
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 يلي:

```
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 يلي:

```
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 يلي:

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

- في السطر رقم ١٢ يتم تعريف ملف الأرشيف وربطه بالملف الموجود بالذاكرة مع تمرير المعامل **load** مما يتسبب في إرجاع الدالة **IsLoading()** للقيمة **TRUE** ومن ثم تخزين بيانات الملف الموجود بالذاكرة بملف الأرشيف.
  - في السطر رقم ١٣ تم استدعاء الدالة **DeleteBlobs()** لحذف البيانات الحالية للوثيقة.
  - في السطر رقم ١٤ تم استدعاء الدالة **Serialize()** لتحميل بيانات ملف الذاكرة.
  - في 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 يلي:

```
1. void CPersistDoc::OnUpdateEditPaste(CCmdUI* pCmdUI)
2. {
3.     COleDataObject oleTarget;
4.     oleTarget.AttachClipboard();
5.     pCmdUI->Enable(oleTarget.IsDataAvailable
6.                   (m_uClipFormat));
7. }
```

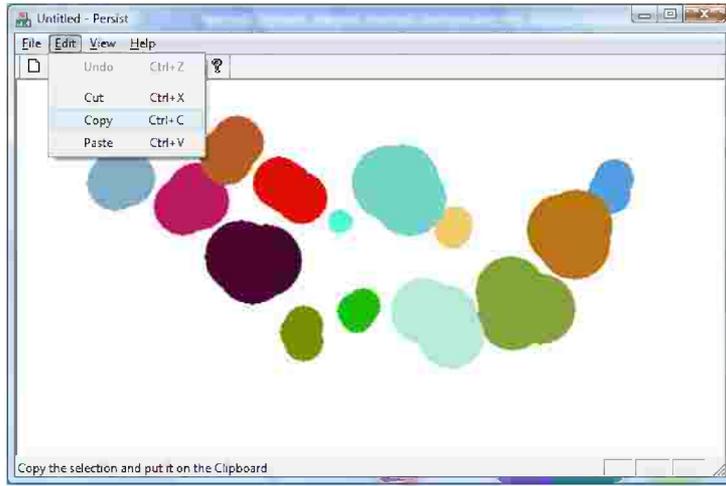
وعن هذا الكود، نوضح ما يلي:

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

عدم وجود بيانات بالحافظة وبالتالي تعطيل الخيار **Paste**. يتم تعيين هذه القيمة من

خلال الدالة **IsDataAvailable()**.

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



شكل ٧-٢٢ قص ولصق بيانات نافذة العرض باستخدام الحافظة

تذكر أنك لإنشاء دالة احتواء أحد عناصر القائمة أو دالة واجهة المستخدم، تقوم بنقر العنصر داخل نافذة تحرير القائمة بزر الفأرة الأيمن ثم اختيار **Add Handler Function** من القائمة الموضعية، وفي هذه الحالة تظهر نافذة المعالج التي يمكنك من خلالها اختيار **Command** لدالة الاحتواء أو **Update\_Command\_UI** لدالة واجهة المستخدم.

