روقا واجهة برمجة التطبيقات

Application Programming Interface (API)

هذا الملف من إعداد/ محمد فاروق السيد محمد يونس.
لاحظ كلمة "إعداد"، فهو ليس تأليفاً صِرفاً و و لا نقلاً و لا لإقتباس و لا تعريب، و لكن مزيج من كل ذلك، لذا فهو "إعداد".
ما لم يشار فى جزئية لمؤلف محدد أو مرجعية للإقتباس تعود حقوق الملكية الفكرية لمعِد الملف/ محمد فاروق السيد محمد يونس، بما فى ذلك الأمثلة و الأشكال الإيضاحية، فقط لإعطاء كل ذى حق حقه، و ليس زهواً أو تفاخر.

تمهيد:
مقدمة فى التعريف بواجهة برمجة التطبيقات Application Programming Interface (API)
نقلاً عن جزء من الفصل الثانى " فى أعماق 'ويندوز' " من كتاب "إلى القمة مع مايكروسوفت فيجوال بيسيك Microsoft Visual Basic : حتى الإصدار ٣ : الجزء الثانى : الطبعة الثالثة سنة ١٩٩٦" للأستاذ الكبير جمال عمارة.
ISBN: 977-287-001-0
أرجو أن يسامحنى لنقل فقرات مباشرة من كتابه القيم، و ذلك لضيق الوقت و قلة التركيز مما يمنعنى من كتابة المقدمة بنفسى. و كل ما رجوت هو نفع الأخوة، و بالمرة يعتبرها إعلان لكتابه "أمزح طبعاً، لأن الكتاب لم يعد يطبع أو يباع من سنوات لحد علمى". و بالرغم من صدور الكتاب منذ سبع سنوات إلا إن الملعومات النظرية على الأقل أقل عرضه للتقادم و تمتاز بالثبات و لكن تحتاج كل الأمثلة الموجودة بالكتاب لتعديل حيث كان الكتاب يعرض للموجود وقتها و هو Win16 API و لكن الآن نحن بصدد شرح Win32 API أى واجهة برمجة التطبيقات فى ويندوز التى تعمل بمعيارية 32 Bit.

[بدء الاقتباس]
من صفحة ١٥٥

ما هى مكتبات الربط الديناميكى؟

إن الإجابة على هذا السؤال تتطلب شرح معنى المكتبة، و معنى الربط، و لماذا نسميه ربط "ديناميكى"، و هل معنى هذه التسمية أن هناك ربط غير ديناميكى؟

المكتبة Library -فى عالم البرامج- هى ملف مستقل يحتوى على مجموعة من الإجراءات Procedures. و الهدف من استخدام المكتبة هو توفير الوقت و الجهد على المبرمج بحيث يمكنه استدعاء الإجراءات الموجودة فيها بدلاً من كتابتها بنفسه. و توجد مكتبات متخصصة فى أداء وظائف معينة مثل مكتبات الاتصالات Communication التى تُسهل على المبرمج كتابة برامج الاتصالات، و مكتبات الرسومات Graphics التى تسمح للمبرمج بأداء عمليات الرسم المعقدة. و بعض الشركات تقوم بتصميم هذه المكتبات و بيعها. و يمكن للمبرمج شراء هذه المكتبات و الاستفادة من الإجراءات الموجودة فيها بدلاً من كتابة كل ما يحتاج إليه بنفسه.

و لكن بعد أن ينتهى المبرمج من استخدام الإجراءات الموجودة فى المكتبة، كيف يضمها إلى برنامجه بحيث يمكن عمل برنامج قابل للتنفيذ Executable file يقوم بتوزيعه أو بيعه للمستخدمين؟

من صفحة ١٥٦

مع معظم لغات البرمجة (من اللغات التى تسمح بعمل ملفات قابلة للتنفيذ) يأتى برنامج يُسمى "الرابط" Linker تكون وظيفته هى قراءة البرنامج و البحث عن الإجراءات الخارجية التى يستخدمها. بعد ذلك ينسخ هذه الإجراءات من المكتبة التى توجد بها و "يربطها" بالبرنامج. و عند عمل الملف النهائى القابل للتنفيذ، تصبح هذه الإجراءات جزءاً لا يتجزأ من البرنامج.

و تسمى هذه العملية بالربط الساكن (الستاتيكى) Static Linking لأن الربط يتم عند صنع البرنامج و يبقى بدون تغيير (ساكناً) فى كل مرة يتم فيها تشغيل البرنامج.

فإذا فرضنا أنك تكتب برنامجاً يستخدم الإجراءين MoveText و ShowResult من إحدى المكتبات، فإن برنامج "الرابط" سيقوم بنسخ هذين الإجراءين من المكتبة إلى البرنامج القابل للتنفيذ.
{يوجد شكل توضيحى رقم (١-٢) بالكتاب الأصلى}

من صفحة ١٥٧

و طريقة الربط الساكن طريقة جيدة لأنها تُقلل من عدد الملفات المطلوب توزيعها مع برنامجك. و لكن لها بعض العيوب، أهمها: لنفرض أن هناك عدة برامج تستخدم نفس المكتبة و تستدعى نفس الإجراءات، و قام مستخدم ما بشراء هذه البرامج، فمعنى ذلك أن لديه عدة نسخ من هذه الإجراءات على القرص الصلب فى جهازه.

و قد كانت طريقة الربط الساكن مقبولة عندما كانت الأجهزة تُشغِل نظام "دوس"، ذلك أن الخسارة من تلك الطريقة كانت محصورة فى القرص الصلب فقط. أما مع قدوم "ويندوز" و معها إمكانية تشغيل أكثر من برنامج فى أكثر من نفس الوقت، ظهر عيب طريقة الربط الساكن بطريقة جلية. فالإجراءات المتكررة فى البرامج المختلفة لا تستهلك مساحة من القرص الصلب فقط، و إنما من الذاكرة كذلك.

فإذا فرضنا أن المستخدم يُشغِل البرامج (أ) و (ب) و (ج) فى نفس الوقت و أن هذه البرامج جميعاً تستخدم الإجراء ShowResult، فإن معنى ذلك أن هناك جزء من التعليمات Code متكرر فى الذاكرة بدون داع. فإذا فرضنا أن الإجراء ShowResult يحتل ٢٠ كيلو بايت، فإن معنى ذلك أن هناك ٤٠ كيلو بايت مهدرة ليس على القرص الصلب فقط، و إنما فى الذاكرة أيضاً.
{يوجد شكل توضيحى رقم (٢-٢) بالكتاب الأصلى}

من صفحة ١٥٨

و إذا نظرت إلى أى من تطبيقات "ويندوز"، ستجد أنها جميعاً تفعل أشياء كثيرة متشابهة: مثل صنع النوافذ و القوائم و أشرطة التمرير و الأزرار... و خلافه. فلو قام كل مبرمج بوضع الإجراءات التى تفعل ذلك فى داخل برنامجه، فإن ذلك سيؤدى إلى إهدار كبير فى موارد النظام System Resources. من هنا جاءت فكرة الربط الديناميكى Dynamic Linking. فعند تصميم "ويندوز" وضع المبرمجون الوظائف الأساسية التى يحتاجها البرنامج فى مكتبات ربط ديناميكى Dynamic Link Libraries و تم توزيعها مع "ويندوز" بحيث تكون متوفرة لجميع المبرمجين.

و عند كتابة برنامجك و استدعاءك لأحد هذه الإجراءات، فإن برنامج الرابط Linker لا ينسخ تلك الإجراءات من المكتبة إلى برنامجك كما كان يحدث عند الربط الساكن، و إنما يضع معلومات داخل برنامجك توضح اسم المكتبة التى تستخدمها و اسم الإجراءات داخلها.

و عند تشغيل برنامجك النهائى داخل "ويندوز"، و عندما يستدعى البرنامج أياً من الإجراءات الموجودة فى المكتبة، تقوم "ويندوز" بتحميل المكتبة فى الذاكرة بحيث تصبح متوفرة لبرنامجك، و تقوم بتحديد عناوين الإجراءات التى يستدعيها البرنامج (على الطائر) و من ثم تسمح له باستدعاء تلك الإجراءات. من هنا جاءت تسمية هذه الطريقة بالربط الديناميكى، لأنها لا تتم إلا أثناء تشغيل البرنامج.

و تظهر فائدة هذه الطريقة جلية إذا قام أكثر من برنامج فى نفس الوقت باستدعاء إجراءات مختلفة من نفس المكتبة. فـ "ويندوز" تُحَمِّل نسخة واحدة من هذه المكتبة فقط فى الذاكرة و تسمح للبرامج المختلفة باستدعاء الإجراءات الموجودة فيها. و بالتالى لا يكون هناك أى إهدار لموارد النظام: فهناك نسخة واحدة من الإجراءات المستخدمة على القرص الصلب و فى الذاكرة.

من صفحة ١٥٩

{يوجد شكل توضيحى رقم (٣-٢) بالكتاب الأصلى}
و مكتبات الربط الديناميكى ليست مقصورة على ويندوز فقط؛ بل يمكن لأى مبرمج أن يصنع لنفسه عدة مكتبات خاصة به تحتوى على الإجراءات التى يكتبها هو و يمكنه فى أى وقت ضم هذه المكتبات إلى برنامجه عن طريق استدعاء الإجراءات الموجودة بها.

واجهة برمجة التطبيقات

يبدو مصطلح واجهة برمجة التطبيقات Application Programming Interface (أو API للاختصار) و كأن الهدف منه هو إخافة الناس. و لكن معنى واجهة برمجة التطبيقات –ببساطة- هو أن أحد التطبيقات يحتوى على مجموعة إجراءات "يُصدِّرها" للتطبيقات الأخرى بحيث يمكن لهذه التطبيقات أن تستدعيها. و إذا أشرنا لواجهة برمجة "ويندوز" فإن المصطلح فى هذه الحالة يشير إلى مكتبات الربط الديناميكى التى توزَّع مع كل نسخة من "ويندوز" و التى تحتوى على مئات الإجراءات التى يستخدمها المبرمجون عند كتابة برامج لـ "ويندوز".

و قد كان الهدف من "فيجوَل بيسيك" هو عزل المبرمج عن هذه الواجهة و توفير بيئة سهلة لكتابة تطبيقات لـ "ويندوز". ذلك أن إجراءات واجهة البرمجة متشعبة و متنوعة و تحتاج إلى وقت كبير فى تعلّمها، كما أن طريقة استخدامها أصعب –إلى حد كبير- من

من صفحة ١٦٠

استخدام الأوامر المبنية داخل "فيجوَل بيسيك". أضف إلى ذلك أن بعض هذه الإجراءات يعتمد فى عمله على إجراءات أخرى بحيث يجب استخدامه فى سياق و ترتيب معين.

لماذا تحتاج إلى مكتبات الربط الديناميكى؟

فإذا كان الهدف من "فيجوَل بيسيك" هو عزل المبرمج به عن تلك الواجهة فلماذا نتكلم عنها إذن؟ و ما علاقة كل الحديث السابق عن الربط الساكن و الربط الديناميكى بـ "فيجوَل بيسيك"؟

السبب واضح: و هو أنك كمبرمج -محترف- لا بد أن تصطدم إن آجلاً أو عاجلاً بوظيفة تريد أن تؤديها فى برنامجك و لا تجد وسيلة لأدائها من "فيجوَل بيسيك". ذلك أن بساطة و سهولة "فيجوَل بيسيك" كان لا بد أن تأتى على حساب نقطة أخرى. لقد كان على مصممى لغة "فيجوَل بيسيك" أن يتجنبوا بعض الأمور لكى يحتفظوا باللغة فى حيّز السهولة و البساطة. فإذا احتجت إلى أداء وظيفة ما فى "فيجوَل بيسيك" و بحثت فى الأوامر المبنية داخله عن أمر يقوم بما تريد و لم تجده، فماذا تفعل؟ لا بد أن ذلك سيثير غضبك، و مما قد يزيد من ذلك الغضب أنك ترى بعض التطبيقات الأخرى تفعل نفس الشيء الذى تريد فعله! إذن فما تريد أن تفعله "ممكن" فى بيئة "ويندوز" بشكل عام، فلماذا لا يوجد فى "فيجوَل بيسيك"؟

و قد كان لدى مصممى "فيجوَل بيسيك" بعد النظر الكافى ليتوقعوا حدوث شيء مثل هذا، لذلك وضعوا فى "فيجوَل بيسيك" آلية استدعاء الإجراءات الخارجية الموجودة فى مكتبات الربط الديناميكى سواءً الموجودة فى "ويندوز" أو التى تكتبها أنت. و قد أعطى ذلك "فيجوَل بيسيك" ميزة كبرى، و هى قابلية التوسع.

فإذا أردت أداء وظيفة معينة فى "فيجوَل بيسيك" و لم تجد أمراً مناسباً لها، عليك بالإجراءات الموجودة فى "ويندوز"، فإن لم تجدها هناك، يمكنك كتابتها بنفسك و وضعها فى مكتبة ربط ديناميكى ثم استخدامها فى برنامجك.

من صفحة ١٦١ "بتصرف::محمد فاروق:بسبب تقادم المعلومات"

توجد مكتبات الربط الديناميكى -فى الغالب- فى ملفات تنتهى بالامتداد dll. و تحتوى "ويندوز" على عدة مكتبات ربط ديناميكى توزع مع كل نسخة من "ويندوز". و ميزة هذه المكتبات أنك لا تحتاج إلى توزيعها على المستخدمين لأنها موجودة لديهم. و يوضح التالى أهم هذه المكتبات و استخدام كل منها:

Kernel32: الإجراءات المتعلقة بتشغيل البرامج، و تنظيم الذاكرة، و الانتقال بين البرامج المختلفة، و التعامل مع موارد النظام، و ما شابة.
user32: الإجراءات التى تتعامل مع النوافذ، مثل: صنع النوافذ و إظهارها و إخفاءها. الإجراءات التى تتعامل مع "الرسائل" التى ترسلها "ويندوز" للتطبيقات، و الإجراءات الخاصة بالقوائم و أشكال المشيرة .. و غيرها.
gdi32: الإجراءات الخاصة بالرسم و الصور و العرض على الشاشة و الطابعات و إجراءات التعامل مع الخطوط، و غيرها.

تحتوى هذه المكتبات الثلاث على معظم الإجراءات التى يحتاجها المبرمجون عند كتابة برامج لـ "ويندوز". و إضافة إلى هذه المكتبات توجد مكتبات أخرى، و لكنها أقل استخداماً مثل مكتبة lz32 التى تحتوى على مجموعة من الإجراءات الخاصة بضغط الملفات Compression أو مكتبة shell32.dll و عدة مكتبات أخرى. [انتهى الاقتباس]

روقا

الإجراءات ★ فيجوال بيزيك

واجهة برمجة التطبيقات

المؤلف/ محمد فاروق السيد محمد يونس
التاريخ/ الاثنين ٥ مايو سنة ٢٠٠٣ من الميلاد، الموافق ٤ ربيع أول سنة ١٤٢٤ من الهجرة.
الجمهور المستهدف/ أعضاء منتدى مايكروسوفت Visual Basic بموقع منتديات الفريق العربى للبرمجة ArabTeam200
جزء من المواد المرفقة بمشاركة فى موقع المنتديات http://www.arabteam2000.com
عنوان المشاركة : "دروس عن واجهة برمجة التطبيقات API" فى منتدى "مايكروسوفت Visual Basic"

أرجو وجه الله تعالى بهذه المشاركة و أرجو منه سبحانه التوفيق و أدعوه أن يسدد خطانا على سواء الطريق.

تقوم الإجراءات بتبسيط المهام البرمجية عن طريق تقسيم البرنامج لمقاطع أصغر و أكثر تخصصاً و أفضل فى الكتابة و الفهم و أسهل تصحيحاً[1] و أفضل فى استهلاك الذاكرة[2] و استثماراً أفضل للوقت حيث يرسخ فكرة التعليمات القابلة لإعادة الاستخدام Reusable Code حيث يمكنك بتعديل طفيف أو بدون تعديلات على الإطلاق أن تستخدم هذا الإجراء فى أى برنامج آخر. و مع الزمن تتوفر لديك مكتبة من الإجراءات، و عند حاجتك لأحدها فإنك تقوم بسحبه من على الرف و استخدامه و لا تحتاج لإعادة كتابة التعليمات الموجودة به مرة أخرى بنفسك.
مثال: أنت كمبرمج تصمم برامج عربية، و تحتاج بشكل متكرر لأداء عملية معينة، فلنقل مثلاً: إظهار رسائل خطأ باللغة العربية يعتبر عملاً متكرراً قد تحتاج لكتابته فى جزئية معالجة الأخطاء فى كل إجراء تقريباً، فهل تفضل كتابة مجموعة التعليمات الموجودة داخل هذا الإجراء كلما احتجت لإظهار رسالة خطأ باللغة و الشكل العربيين »»

```
Private Sub arErrMsgBox(strMsg As String)
Const strTitle As String= "خطأ"
Const lngButtons As Long = vbOKOnly + vbCritical + vbDefaultButton4_ +
vbMsgBoxRight + vbMsgBoxRtlReading
Const strPrompt As String = "صادف البرنامج الخطأ التالى: " & vbCrLf

MsgBox strPrompt & strMsg, lngButtons, strTitle
End Sub
```

أم تفضل كتابة هذه التعليمات فى إجراء و لمرة واحدة فقط، ثم إستدعاءه بسطر واحد فقط من التعليمات هكذا فى برنامجك ««

```
arErrMsgBox "عبارة تصف الخطأ"
```

كما فى المثال التالى
و الآن إذا لم يوجد قرص بمحرك الأقراص [A:] فسيوف تظهر للمستخدم رسالة خطأ بالغة العربية.

```
Private Sub Form_Load()
On Error GoTo ErrHandler

Open "a:\file.ext" For Input As #1
Close

Exit Sub
ErrHandler:
    arErrMsgBox Err.Description
End Sub
```

و عند قيامك بتصميم برنامج ما مستقبلاً فيمكنك إدراج هذا الإجراء فيه دون أى تعديل ليؤدى نفس المهمة.
و فائدة أخرى هنا، أنك لو أحببت أن تدخل تعديل ما على الطريقة التى تظهر بها الرسالة، فستقوم بهذا التعديل مرة واحدة فقط داخل هذا الإجراء و لن تضطر لتعديله فى باقى أجزاء البرنامج. و لن تحتاج لتغيير كلمة واحدة إضافية فى أى جزء آخر من البرنامج.

---

[1] التصحيح Debugging هى عملية اكتشاف الأخطاء المنطقية و أخطاء وقت التشغيل .... و محاولة تلافيها.
[2] حيث يتم كتابة مجموعة التعليمات مرة واحدة فقط و استدعائها أى عدد من المرات، بدون تكرار كتابة التعليمات نفسها.

و نقطة أخرى قد لا تتضح بصورة كبيرة من هذا المثال فهو صغير جداً، و لكن تخيل أن مجموعة العمليات التى تريد تنفيذها كبيرة و معقدة و تستهلك مساحة كبيرة من الذاكرة، أليس من الأفضل كتابتها مرة واحدة فقط مما يقلل من إستهلاك موارد النظام؟

أنواع الإجراءات فى فيجوال بيزيك

يوجد فى فيجوال بيزيك ثلاثة أنواع من الإجراءات و هى حصراً:
الإجراء الفرعى Sub: و هو إجراء لا يعود بقيمة.
الوظيفة Function : إجراء يعود بقيمة لها نوع بيانات محدد (إذا لم تحدده بنفسك، يحدد فيجوال بيزيك النوع Variant)
الخاصية Property : إجراء يمكن أن يخصص قيم و/أو يعود بقيم.

الإجراء الفرعى Sub:

الصيغة العامة للإجراء الفرعى Sub هى:

```
[Private|Public][Static]Sub اسم الإجراء (المعاملات: إختيارية)
'
'"مجموعة التعليمات "الأكواد
'
End Sub
```

و لاحظ أن الكلمات الموجودة بين معقوفتين [ ] مثل Private و Public و Static هى كلمات إختيارية، أى أنك لست ملزماً بكتابتها و هذا تقليد عام فى الكتابة، فلن أشرحه مرة أخرى عندما يتكرر أثناء متابعتنا للدروس.
و لاحظ أن كلمتى Private و Public موجودين بين علامة [ ] واحدة يفصل بينهما علامة البايب | و معنى ذلك أنه يمكنك إختيار استخدام أياً منهما إذا أحببت و لكن ليس كلاهما، أى لا يمكن الإعلان عن الإجراء على أنه خاص Private و عام Public فى نفس الوقت و هذا منطقى أليس كذلك؟
و لكن كلمة Static موجودة بين علامة [ ] منفصلة فيمكن الإعلان مثلاً عن الإجراء باستخدام أياً من الطرق التالية و يكون الإعلان صحيحاً:

-١
```
Private Sub MySub()

End Sub
```

-٢
```
Public Sub MySub()

End Sub
```

-٣
```
Private Static Sub MySub()

End Sub
```

-٤
```
Public Static Sub MySub()

End Sub
```

-٥
```
Private Sub MySub(MyVar as DataType)

End Sub
```

-٦
```
Public Sub MySub(MyVar1 as نوع بيانات,Optional MyVar2 as نوع بيانات=قيمة افتراضية)

End Sub
```

-٧
```
Public Sub MySub(ByVal MyVar1)'Variant هنا لم يتم تحديد نوع بيانات و بالتالى سيعتبر
       'لاحظ أن الكلمة ByVal تعنى تمرير المعامل للإجراء عن طريق القيمة و ليس المرجعية
End Sub
```

-٨
```
Public Sub MySub(ByRef MyVar1)
       'لاحظ أن الكلمة ByRef تعنى تمرير المعامل للإجراء عن طريق المرجعية و ليس القيمة
End Sub
```

-٩
```
Public Sub MySub(ParamArray intNums())
       'هنا تمرير عدد غير محدود من المتغيرات للإجراء
End Sub
```

```
Sub MySub()
        إذا لم يحدد صراحة نوع الإجراء خاص Private أو عام Public فإن فيجوال بيزيك يعتبره إجراء عام Public تلقائياً'
End Sub
```

و الآن لنتناول بعض النقاط التى تحتاج للمزيد من الشرح فى الأمثلة السابقة:

الكلمة Private:
و تقوم بتعريف الإجراء على أنه إجراء خاص، أى أن الوصول له (استخدامه) مقصور على الإجراءات الموجودة فى نفس الموديول Module المعلن به.

الكلمة Public:
إذا تم الإعلان عنه فى موديول قياسى Standard Module:
و تقوم بتعريف الإجراء على أنه إجراء عام، أى أن الوصول له (استخدامه) متاح لكافة الإجراءات الموجودة فى كل الوحدات Modules. و لو تم استخدام هذه الكلمة فى الإعلان عن الإجراء فى وحدة تحتوى على الجملة Option Private فإن هذا الإجراء لن يكون متاحاً للإجراءات الموجودة خارج المشروع.
و لكن فى حال الإعلان عنه فى موديول فورم Form Module فيجب أولاً أن تذكر اسم الفورم سابقاً لاسم الإجراء، مثال:

```
FormName.FunctionName
```

هكذا:

```
FrmTaxes.CalcTaxes(19771026) 'لحساب الضرائب لقيمة محددة من المال
```

الكلمة Static:
توضح أن كافة المتغيرات المحلية المعلنة بالإجراء ستحتفظ بقيمتها بين الاستدعاءات، و لا توثر الخاصية Static على المتغيرات المعلنة خارج الإجراء حتى و لو استخدمت فى الإجراء.

و هناك كلمة أخرى هى Friend و لكنى أغفلتها عمداً لأنها تستخدم فقط فى وحدات الصفوف Class Modules.

الكلمة Optional:
يمكنك أن تحدد أن واحداً أو أكثر من المتغيرات الممررة للإجراء إختيارى "يمكن تمريره أو عدم تمريره" عن طريق وضع الكلمة المفتاحية Optional و لكن متى حددت أن أحد المعاملات إختيارى فيجب أن تعلن كافة المعاملات التالية على أنها إختيارية أيضاً، و بالتالى تكتب كافة المعاملات الإختيارية معاً فى آخر قائمة المعاملات المكتوبة فى الاجراء.
مثال: الإعلان التالى للإجراء يعتبر صحيحاً:

```
Private Sub SubName(Var1, Var2, Optional Var3, Optional Var4, Optional Var5)

End Sub
```

و لكن الإعلان التالى لا يعد صحيحاً:

```
Private Sub SubName(Var1, Var2, Optional Var3, Var4, Optional Var5)

End Sub
```

فالمتغير Var4 كان يجب أن يحدد أنه Optional لأن أحد المتغيرات قبله تم تحديده على أنه إختيارى. و يظهر مترجم لغة فيجوال بيزيك Compiler رسالة مثل هذه

و يمكنك تحديد قيمة إفتراضية للمعامل الإختيارى، فإذا تم تجاهل تمرير قيمة لهذا المعامل عند استدعاء الإجراء فإن الإجراء يستخدم القيمة الإفتراضية.
مثال:

```
Private Sub Form_Load()
    ShowCubeRoot
End Sub

Private Sub ShowCubeRoot(Optional lngNum As Long = 64)
    MsgBox lngNum ^ (1 / 3)
End Sub
```

هنا تم استدعاء الوظيفة ShowCubeRoot و لكن لم يتم تمرير قيمة للمعامل lngNum و لكن الإجراء استخدم القيمة الإفتراضية المحددة للمعامل "٦٤" و تظهر الرسالة و بها الرقم "٤" الجذر التكعيبى للرقم "٦٤"
تمرير المعاملات للإجراء عن طريق القيمة ByRef
إن تمرير المعاملات عن طريق المرجعية يتيح للإجراء الوصول لمحتويات المتغير الفعلية فى موقعها بالذاكرة، و كنتيجة لذلك يمكن أن تتغير قيمة المتغير بصورة نهائية بواسطة الإجراء الذى مُرِّر له. و يعد التمرير بالمرجعية هو الوضع الإفتراضى فى فيجوال بيزيك.

مثال:

```
Private Sub Form_Load()
Dim lngVar As Long
Const strMsg = "نعم. تغيرت قيمة المتغير عندما مُرر كمعامل للإجراء" & vbCrLf & _
"و أصبح: " & vbCrLf & "ByRef عن طريق المرجعية"

lngVar = 100&
ChangeVariablesData lngVar

MsgBox strMsg & lngVar, vbMsgBoxRight + vbMsgBoxRtlReading, "هل حدث تغيير؟"
End Sub

Private Sub ChangeVariablesData(lngAnyVar As Long)
'نقوم بتغيير قيمة المتغير الممرر لهذا الإجراء عن طريق المرجعية
'لاحظ التغيير فى قيمة المتغير فى الإجراء المعلن فيه ، بعد استدعاء هذا الإجراء
lngAnyVar = lngAnyVar * 2
End Sub
```

أرجو أن أكون قد وفقت فى كتابة مثال يوضح المقصود، هل تفحته جيداً؟ قمت بالإعلان عن متغير فى إجراء معين و قمت بتخصيص قيمة محددة له عن قصد حتى نختبر مدى التغير فيها، و مررنا عنوان هذا المتغير فى الذاكرة "التمرير بالمرجعية" لإجراء آخر يقوم هذا الإجراء الأخير ChangeVariablesData بتعديل القيمة التى يحتوى عليها المتغير، فعندما خصصنا القيمة "١٠٠" للمتغير قبل استدعاء الإجراء ChangeVariablesData و من ثم قمنا باستدعاء هذا الإجراء و بعد ذلك إختبرنا قيمة المتغير مرة أخرى، عن طريق عرض محتواه فى مربع رسائل باستخدام الدالة MsgBox وجدنا أن محتواه قد تبدل و أصبح "٢٠٠" نتيجة للعمليات التى أجريت له فى الإجراء المُمرر له ChangeVariablesData.

و بالتالى يجب التنبه لهذه النقطة، فإن أردت أن تمنع الإجراءات من تعديل قيم المتغيرات التى تمررها لها، فعليك بإتباع الطريقة التالية، و هى التمرير بالقيمة ByVal كما هو موضح.

التمرير بالقيمة ByVal:
عند تمرير متغير كمعامل لأحد الإجراءات فإن ما يُرسَل حقيقةً هو نسخة من هذا المتغير "نسخة من بياناته"، و لو قام الإجراء بتغيير قيمة المتغير الممرر له، فهو حقيقةً سيغير قيمة النسخة و ليس المتغير الأصلى.
استخدم الكلمة ByVal قبل اسم المعامل المعلن عنه فى الإجراء، لتشير لتمرير المتغيرات له عن طريق القيمة.

مثال، راجع نفس المثال السابق بعد إجراء التعديلات المطلوبة لتمرير المعاملات عن طريق القيمة.

```
Private Sub Form_Load()
Dim lngVar As Long
Const strMsg = "لا. لم تتغير قيمة المتغير عندما مُرر كمعامل للإجراء" & vbCrLf & _
"و بقى: " & vbCrLf & "ByVal عن طريق القيمة"

lngVar = 100&
ChangeVariablesData lngVar

MsgBox strMsg & lngVar, vbMsgBoxRight + vbMsgBoxRtlReading, "هل حدث تغيير؟"
End Sub

Private Sub ChangeVariablesData(ByVal lngAnyVar As Long)
'نختبر حدوث تغيير فى قيمة المتغير الممرر لهذا الإجراء عن طريق القيمة
'لاحظ عدم حدوث تغيير فى قيمة المتغير فى الإجراء المعلن فيه ، بعد استدعاء هذا الإجراء
lngAnyVar = lngAnyVar * 2
End Sub
```

نجد أن قيمة المتغير لم تتغير بعد تمريره للإجراء، لأن الإجراء قام بالعمليات على النسخة.
و يجب أن تختار النوع المناسب "التمرير بالمرجعية أو بالقيمة" كما تُملى عليك المهمة البرمجية التى تريد تنفيذها.
فإن كنت قد صممت الإجراء فى الأساس ليقوم بمجموعة عمليات على متغير معين بحيث تنعكس نتيجة هذه العمليات على هذا المتغير فعليك بتمرير المتغير عن طريق المرجعية، و إلا فيفضل أن تمرر المعامل بالقيمة، حتى لا يقوم الإجراء بتغيير غير مرغوب فيه، و الأهم أنه غير موضوع فى الحسبان، و بالتالى يعطى الإجراء نتائج غير متوقعة و غير مرغوبة.

تمرير عدد غير محدود من المعاملات للإجراء عن طريق مصفوفة المعاملات ParamArray:
عندما استدعاء إجراء ما، فإنك عادة ما تمرر له عدداً من المعاملات يماثل عدد المعاملات المعلن فيه، أو على الأقل يماثل عدد المعاملات الغير إختيارية Optional كما أسلفت فى الشرح. و لكن لنفرض أنك تريد تمرير معاملات غير معروفة العدد مسبقاً

لإجراء، فما الحل؟ يتم ذلك كما هو موضح فى عنوان هذه الفقرة بكتابة الكلمة ParamArray، و لكن لاحظ أنك وقتها لن تتمكن من تحديد نوع بيانات المعاملات الغير محددة العدد. و ستكون مصفوفة من المتغيرات من نوع Variant.
كما لا يمكنك إعلان معاملات أخرى بعد المعامل المحدد بـ ParamArray، و لكن يمكنك الإعلان عن المعاملات المطلوبة قبله.
لاحظوا المثال التالى

```
Private Sub Form_Click()
      DelOrders "Roka", 26, 10, 1977, 14, 11, 1397
End Sub

Private Sub DelOrders(strUser As String, ParamArray Orders())
Dim varOrder As Variant

'قم بتغيير الخاصية RightToLeft للفورم إلى True لأفضل النتائج
'و على كل فهذا مثال لتوضيح نقطة فنية و ليست تطبيق واقعى من الحياة العملية
'فنحن لن نقوم فى الإجراء الفعلى بطباعة أرقام الطلبيات و لكن سنحذفها فعلاً
'من قاعدة البيانات، و لكن لأن التعامل مع قواعد البيانات خارج نطاق شرح هذا المثال
'و لأننا نشرح نقطة فنية فقط، فتم كتابة المثال ليركز على الجزئية الفنية فقط

Print "سيتم حذف الطلبات التالية من جدول طلبيات العميل: "; strUser
For Each varOrder In Orders
    Print varOrder
Next
End Sub
```

كتابة جمل أبسط و أسهل عن طريق المُعامِلات المُسماة Named Arguments:
يتيح فيجوال بيزيك تمرير المعاملات عن طريق اسمها و ليس فقط عن طريق ترتيبها فى الإعلان فى جسم الإجراء بالتوالى، و هذا فى كثير من الإجراءات المبنية داخله، فمثال: فى الدالة الشهيرة MsgBox ، حاول أن تكتب كلمة MsgBox و بعدها مسافة، إذا لم تكن قد عطلت تقنية IntelliSense®[2] فسيظهر لك شكل مشابة للتالى:

```
Private Sub MySub()
        msgbox
End Sub
```

MsgBox(**Prompt**, [*Buttons As* VbMsgBoxStyle = vbOKOnly], [*Title*], [*HelpFile*], [*Context*])
As VbMsgBoxResult

المهم، تجد أن فيجوال بيزيك قد عرض لك قائمة **بأسماء** المعاملات التى تمررها للدالة MsgBox و وضع بعضها بين علامة [ ]، أتذكرون ما قلته سابقاً؟ نعم معنى ذلك أنها معاملات إختيارية، و لكن بيت القصيد هو أننا إذا أردنا أن نظهر رسالة عن طريق تمرير النص المطلوب إظهاره فى مربع الرسائل و تحديد عنوان مربع الرسائل، بدون استخدام المعاملات المسماة فإننا نكتب:

```
MsgBox "My prompt here", ,"My title"
```

لاحظ أننا إذا ام نود أن نمرر معامل إختيارى فإننا نهمله و نترك مكانه خالياً مع مراعاة كتابة الفاصلة ",".

و لكن مع استخدام المعاملات المسماة، فيمكن كتابة نفس الأمر بالصورة التالية:

```
MsgBox Title:="My title", prompt:="My prompt here"
```

فقد كتبنا المعاملات التى نريد و بالترتيب الذى نريده، و بالإضافة لذلك فالتعليمات بهذه الصورة مفهومة أكثر، أعتقد توافقونى.
و لكن لا داعى للقول أنه لا يمكنك تجاهل كتابة المعاملات الإجبارية "الغير إختيارية".

---

[3] تقنية الحس الذكى هى اختصار Intelligent Sense و تم إعتماد هذه التقنية فى بيئة تطوير فيجوال ستوديو ٦، فى فيجوال بيزيك و فيجوال سي ... و من ضمن ما تقوم به هذه التقنية هو تكمله الجمل بعرض كافة الوظائف و الخصائص المتاحة للكائنات فى قائمة و عرض تعليقات التعليمات و معلومات الإعلان عن الإجراءات و المتغيرات، كما يقوم الإختيار Complete Word (من قائمة Edit أو من قائمة الزر اليمين) بتكملة الوظائف و المتغيرات أو عرض قائمة بالاحتمالات المرشحة لإكمالها.

```
Private Sub Form_Load()
'Complete Word شرع

End
```

Abs
ActiveControl
AlignConstants
AlignmentConstants
App
AppActivate
Appearance

```
Private Sub Form_Load()
    me.
End
```

Appearance
AutoRedraw
BackColor
BorderStyle
Caption
Circle
ClipControls

لاحظ الشكل التالى

و الآن أعتقد أننا فرغنا من هذه الجزئية، كتمهيد لشرح "إجراءات" واجهة برمجة التطبيقات، و سأتطرق لها فور توافر الوقت بالشكل التالى:
نظرة عامة و تمهيد لاستدعاء إجراءات واجهة برمجة التطبيقات من تطبيقات فيجوال بيزيك – ثم كيفية الإعلان عنها- ثم تناول لبعض أنواع البيانات المخصصة الأكثر شهرة و المستخدمة مع إجراءات واجهة البرمجة – ثم كيفية استخدام عارض الإجراءات API Viewer – ثم كيفية تمرير المعاملات "نصوص/مصفوفات/أنواع مخصصة ..." – ثم جزء مفصل عن تمرير مؤشرات "عناوين" الوظائف لإجراءات واجهة البرمجة – ثم تحويل إعلانات لغة C -المستخدمة غالباً فى كتابة مكتبات الربط الديناميكى و الإجراءات الموجودة بها- لإعلانات لغة فيجوال بيزيك. و كذا مسائل متنوعة مثل التمرير بالقيمة و المرجعية لإجراءات واجهة البرمجة، و استدعاء إجراءات واجهة البرمجة ذات الأسماء الغير قياسية، و كذا بعض الإعتبارات الواجب مراعاتها نظراً لإختلاف تناول واجهة برمجة التطبيقات من لغة فيجوال بيزيك عنه من أى لغة أخرى.
و فى الدوس القادمة سنعرض بإذن الله لبعض هذه الإجراءت تفصيلاً، ففكروا معى من الآن، هل تريدون عرض الإجراءات مصنفة تبعاً للمهام التى تقوم بها، أو مصنفة أبجدياً، أو حسب التداول و كثرة استخدام الإجراءات "الإجراءات الشهيرة أولاً، ثم الأقل إستخداماً" ، أو وفقاً لتصنيف آخر تقترحونه.

أنتظر ردكم قبل المتابعة و شكراً.

---

* المُراجِع: كتاب Programmer's Guide الفصل الخامس Programming Fundamentals و الناشر Microsoft Press.