# Preparando Suas Aplicações Visual Basic 6.0 para a Atualização para o Visual Basic.NET

Microsoft Corporation

Outubro de 2000

**Resumo:** Este documento proporciona recomendações para desenvolvedores Microsoft Visual Basic que pretendem atualizar suas aplicações para o Visual Basic.NET. Ele contém informação sobre a Ferramenta de Atualização do Visual Basic.NET e discute orientações arquiteturais que possibilitam uma atualização suave do Visual Basic 6 para o Visual Basic.NET. (26 páginas impressas)

### Conteúdo



# Visão Geral

Este documento proporciona recomendações para desenvolvedores Microsoft Visual Basic que pretendem atualizar suas aplicações para o Microsoft Visual Basic.NET.
O Visual Basic.NET vai abrir e atualizar projetos Visual Basic 6.0 para tecnologias Visual Basic.NET, mas na maioria dos casos você terá que fazer algumas modificações nos seus projetos após trazê-los para o Visual Basic.NET. O propósito deste documento é recomendar como projetar e implementar seus projetos Visual Basic atuais para minimizar o número de alterações que você terá que fazer quando eles forem atualizados para o Visual Basic.NET. Quando apropriado, nós usamos novas construções de linguagem; entretanto, este documento não pretende ser uma referência da linguagem Visual Basic.NET.

**Nota** O Visual Basic.NET ainda está em desenvolvimento; alguns detalhes de compatibilidade podem mudar antes do lançamento do produto. Seguir as orientações presentes neste documento não garante que seu código não vai precisar ser alterado; as orientações pretendem somente reduzir a quantidade de trabalho necessária para a conversão.

# O Que é Visual Basic.NET?

Visual Basic.NET é a próxima versão do Visual Basic. Ao invés de simplesmente incluir alguns recursos novos no Visual Basic 6.0, a Microsoft reprojetou o produto para tornar mais fácil do que nunca escrever aplicações distribuídas tais como sistemas n-tier Web e corporativos. O Visual Basic.NET possui dois novos pacotes de formulários (Windows Forms e Web Forms); uma nova versão do ADO para acessar fontes de dados desconectadas; e linguagem simplificada, removendo palavras chave legadas, melhorando a segurança de tipo e expondo construções de baixo nível que os desenvolvedores avançados requisitam.

Estes novos recursos abrem novas portas para o desenvolvedor Visual Basic: Com Web Forms e ADO.NET, agora você pode desenvolver rapidamente Web sites escaláveis; com herança, a linguagem agora suporta verdadeiramente programação orientada a objeto; Windows Forms suporta nativamente acessibilidade e herança visual; e implantar suas aplicações é agora tão simples quanto copiar seus executáveis e componentes de diretório para diretório.

Agora o Visual Basic.NET está totalmente integrado às outras linguagens Microsoft Visual Studio.NET. Além de você poder desenvolver componentes de aplicação em linguagens de programação diferentes, suas classes agora podem herdar classes escritas em outras linguagens usando herança cross-language. Com o debugger unificado, agora você pode fazer o debug de aplicações de múltiplas linguagens, independente de elas estarem rodando localmente ou em computadores remotos. Finalmente, independente da linguagem utilizada, o Microsoft .NET Framework proporciona um rico conjunto de APIs para o Microsoft Windows® e a Internet.

# Porque o Visual Basic.NET Não é 100% Compatível?

Havia duas opções para o projeto do Visual Basic.NET—fazer o retrofit da base de código existente para que ela rodasse no topo do .NET Framework, ou começar tudo de novo, aproveitando completamente a plataforma. Para entregar os recursos mais requisitados pelos consumidores (ex. herança, threading), proporcionar acesso total e sem limite à plataforma e garantir que o Visual Basic acompanhasse a próxima geração das aplicações Web, a decisão certa foi construir tudo desde o início para a nova plataforma.

Por exemplo, muitos dos novos recursos encontrados no Windows Forms poderiam ter sido incluídos na base de código existente como novos controles ou mais propriedades. Entretanto, isto acarretaria prejuízo para todos os outros grandes recursos inerentes ao Windows Forms, como segurança e herança visual.

Um dos nossos principais objetivos foi garantir que o código Visual Basic pudesse interoperar totalmente com o código escrito em outras linguagens, como o Microsoft Visual C#™ ou Microsoft Visual C++®, e habilitar o desenvolvedor Visual Basic a utilizar unicamente o .NET Framework, sem recorrer a soluções de programação de contorno tradicionalmente necessárias para fazer funcionar as Windows APIs. Agora o Visual Basic possui os mesmos tipos de variáveis, arrays, tipos definidos pelo usuário, classes e interfaces que o Visual C++ e qualquer outra linguagem direcionada para o Common Language Runtime; entretanto, nós tivemos que remover alguns recursos da linguagem, tais como strings de tamanho fixo e arrays não baseados em zero.

Agora o Visual Basic é uma verdadeira linguagem orientada a objeto; alguns recursos não intuitivos e inconsistentes como **GoSub/Return** e **DefInt** foram removidos da linguagem.

O resultado é um Visual Basic com nova energia, que continuará a ser a ferramenta mais produtiva para a criação de aplicações Windows, e que agora está posicionado como a melhor ferramenta para a criação de Web sites da próxima geração.

# Atualizando para o Visual Basic.NET

O Visual Basic.NET possibilita uma mudança fundamental do desenvolvimento Windows tradicional para a construção de aplicações n-tier e Web da próxima geração. Por esta razão, seu código terá que ser atualizado para aproveitar o Visual Basic.NET.

Isto acontece automaticamente quando você abre um projeto Visual Basic 6.0 no Visual Basic.NET: o Assistente de Atualização orienta você pelo processo de atualização e cria um novo projeto Visual Basic.NET (seu projeto existente não é alterado). Este processo é de mão única; o novo projeto Visual Basic.NET não pode ser aberto no Visual Basic 6.0.

Quando seu projeto é atualizado, a linguagem é modificada para quaisquer alterações de sintaxe e seus Formulários Visual Basic 6.0 são convertidos para Windows Forms. Na maioria dos casos, você terá que fazer algumas alterações no seu código após sua atualização. Isto é necessário porque certos objetos e recursos de linguagem ou não possuem equivalente no Visual Basic.NET, ou possuem um equivalente muito desigual para uma atualização automática. Após a atualização, talvez você também queira alterar sua aplicação para aproveitar alguns dos novos recursos do Visual Basic.NET.

Por exemplo, Windows Forms suporta control anchoring, para que você possa remover a maior parte do seu código antigo de dimensionamento de Formulário Visual Basic 6.0:

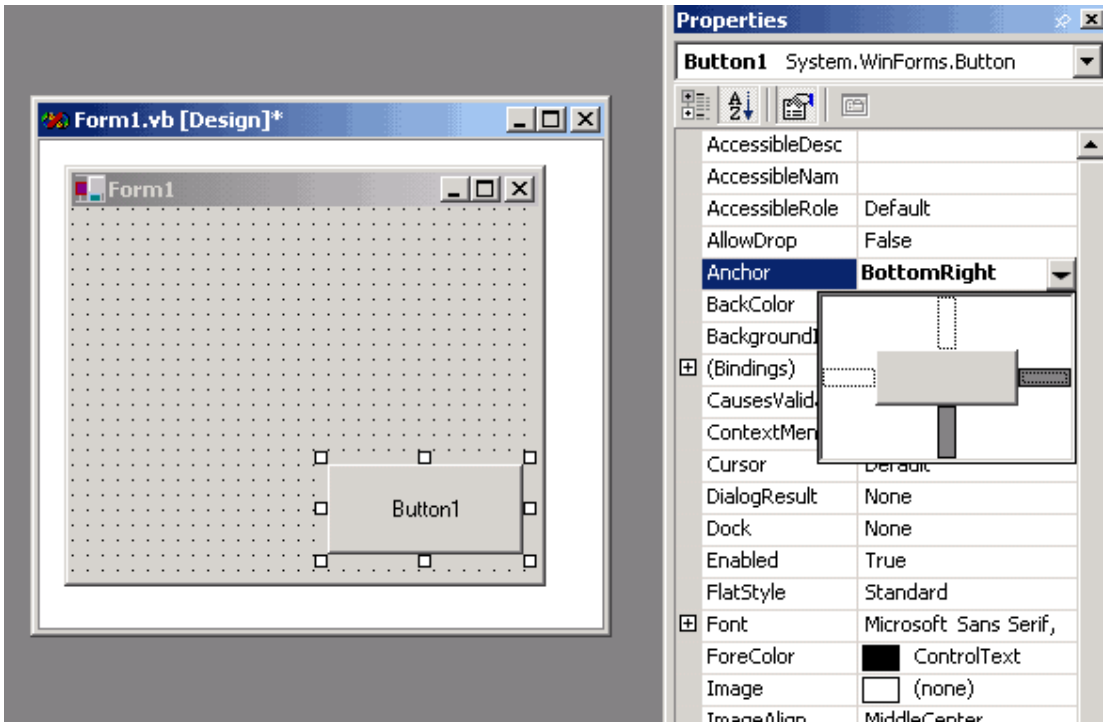


**Figura 1. Suporte VB.NET para control anchoring**

Para ajudá-lo a fazer as alterações, após a atualização do seu projeto, o Visual Basic.NET inclui um 'relatório de atualização' ao seu projeto detalhando quaisquer problemas, e insere comentários no seu código atualizado alertando para comandos que terão que ser alterados. Como estes comentários são apresentados como tarefas 'TO DO' na nova janela Task List, você pode ver facilmente quais alterações são necessárias e navegar para o comando do código simplesmente efetuando o duplo clique na tarefa. Cada tarefa e item no relatório de atualização é associado a um tópico online do Help oferecendo orientação posterior sobre porque o código precisa ser alterado e sobre o que você precisa fazer.

Ao seguir as recomendações neste documento, você pode minimizar e, em alguns casos, eliminar as alterações necessárias após a atualização do seu projeto para o Visual Basic.NET. Na maioria dos casos, as recomendações simplesmente representam boas práticas de programação; entretanto, nós

também identificamos os objetos e métodos que não possuem equivalentes e que deveriam ser usados com parcimônia  se você pretende atualizar seu projeto para o Visual Basic.NET.

# Trabalhando com o Visual Basic 6.0 e com o Visual Basic.NET

O Visual Basic.NET suporta a atualização de projetos Visual Basic 6.0; se você possui um projeto escrito nas versões de 1 a 5 do Visual Basic, nós recomendamos a você carregá-lo no VB6 (escolhendo atualizar controles Microsoft ActiveX®), compilar e salvar o projeto antes de atualizá-lo para o Visual Basic.NET.

O Visual Basic.NET e o Visual Basic 6.0 podem ser instalados no mesmo computador e rodar ao mesmo tempo. Da mesma forma, aplicações escritas em Visual Basic.NET e Visual Basic 6.0 podem ser instaladas e executadas no mesmo computador. Componentes escritos em Visual Basic.NET podem interoperar com componentes COM escritos em versões anteriores do Visual Basic e outras linguagens. Por exemplo, você pode soltar um controle ActiveX escrito em Visual Basic 6.0 em um Visual Basic.NET Windows Form, usar um objeto COM Visual Basic 6.0 de uma biblioteca de classes Visual Basic.NET, ou incluir uma referência para uma biblioteca Visual Basic.NET em um executável Visual Basic 6.0.

Componentes compilados no Visual Basic.NET possuem diferenças de run-time sutis dos componentes compilados no Visual Basic 6.0. Para iniciantes, como objetos Visual Basic.NET são liberados através de garbage collection, onde objetos são destruídos explicitamente, pode haver um atraso antes que eles sejam realmente removidos da memória. Há diferenças adicionais como as alterações variante/objeto descritas mais adiante neste documento. O resultado combinado destas diferenças é que aplicações Visual Basic.NET terão comportamento em tempo de execução similar mas não idêntico ao das aplicações Visual Basic 6.0.

Além disso, o Visual Basic.NET torna desnecessária a compatibilidade binária entre componentes Visual Basic.NET e Visual Basic 6.0. Agora os componentes possuem um sistema de versão e implantação mais robusto do que nunca, arquivos podem ser implantados simplesmente efetuando a cópia para um diretório (não mais RegSvr32), e a atualização para uma nova versão de um componente é tão simples quanto substituir o arquivo antigo por um novo arquivo. Tudo o que você tem a fazer é garantir que classes e métodos são compatíveis com a versão anterior.

# Recomendações Arquiteturais

A plataforma .NET apresenta melhorias sobre as arquiteturas anteriores e inclui maior suporte para escalabilidade e aplicações distribuídas apesar do acesso a dados desconectado, transporte de mensagem baseado em HTTP e implantação baseada em cópia de arquivo (não mais registro de componentes). Para aproveitar melhor estes recursos, você deve projetar suas aplicações com uma arquitetura similar à que você usaria no Visual Basic.NET.

## Aplicações baseadas em Browser

O Visual Basic 6.0 e o Microsoft Visual Studio® 6.0 ofereceram várias tecnologias para a criação de aplicações Internet e intranet baseadas em browser:

- Webclasses
- Projetos DHTML
- Documentos ActiveX
- Active Server Pages (ASP)

O Visual Basic.NET introduz o ASP.NET, uma versão aprimorada do ASP, e melhora a arquitetura com Web Forms, que são páginas HTML com eventos Visual Basic. A arquitetura é baseada em servidor.

Segue abaixo uma lista de recomendações e sugestões arquiteturais para o desenvolvimento de aplicações Visual Basic 6.0 baseadas em browser que serão na sua maioria migradas para projetos Visual Basic.NET:

- Nós recomendamos o uso das orientações arquiteturais multi-camada da Microsoft para criar suas aplicações, criar a interface com ASP e usar objetos COM Visual Basic 6.0 ou Visual C++ 6.0 na sua lógica de negócio. ASP é totalmente suportado no Visual Basic.NET e você pode continuar a estender suas aplicações usando ASP, ASP.NET e Web Forms. Objetos de negócio Visual Basic 6.0 e Visual C++ 6.0 podem ser usados sem modificação ou atualizados para o Visual Studio.NET.

- Aplicações DHTML contêm páginas DHTML e DLLs do lado cliente. Estas aplicações não podem ser atualizadas automaticamente para o Visual Basic.NET. Nós recomendamos a você manter estas aplicações no Visual Basic 6.0.

- Documentos ActiveX não são suportados no Visual Basic.NET, e como projetos DHTML, não podem ser atualizados automaticamente. Nós recomendamos a você manter suas aplicações de documento ActiveX em Visual Basic 6.0 ou, quando possível, substituir documentos ActiveX por controles do usuário.

- Documentos ActiveX e aplicações DHTML do Visual Basic 6.0 podem interoperar com tecnologias Visual Basic.NET. Por exemplo, você pode navegar de um Visual Basic.NET Web Form para uma página DHTML do Visual Basic 6.0, e vice–versa.

- Webclasses não existem mais no Visual Basic.NET. Aplicações Webclass serão atualizadas para o ASP.NET; entretanto, você terá que fazer algumas modificações após a atualização. Aplicações Webclass existentes podem interoperar com Web Forms e aplicações ASP do Visual Basic.NET, mas para novos projetos nós recomendamos o uso da plataforma Windows DNA do ASP com objetos de negócio do Visual Basic 6.0.

Para maiores informações sobre a construção de aplicações com a arquitetura multi-camada da Microsoft, veja o Web site Microsoft Windows DNA.

## Projetos Cliente/Servidor

O Visual Basic 6.0 ofereceu várias tecnologias para a criação de aplicações cliente/servidor:

- Formulários Visual Basic

- Objetos Microsoft Transaction Server (MTS)/COM+ da camada do meio

- Controles de usuário

No Visual Basic.NET, há um novo pacote de formulário: Windows Forms. Windows Forms possui um modelo de objeto diferente dos Formulários Visual Basic 6.0, mas é amplamente compatível. Quando seu projeto é atualizado, Formulários Visual Basic são convertidos para Windows Forms.

O Visual Basic.NET melhora o suporte ao desenvolvimento de serviços MTS da camada do meio e componentes COM+. Usando o debugger unificado, você pode passar de uma aplicação cliente para um componente MTS/COM+ e voltar para o cliente. Você também pode usar o debugger unificado para caminhar por componentes MTS/COM+ do Visual Basic 6.0 (desde que eles estejam compilados no código nativo, com informação de debug simbólica e sem otimizações).

O Visual Basic.NET também introduz um novo componente da camada do meio, o Web Services. Web Services são hospedados pelo ASP.NET e usam o transporte HTTP permitindo que requisições de

método passem por firewalls. Eles passam e retornam dados usando XML padrão da indústria, permitindo que outras linguagens e outras plataformas acessem sua funcionalidade. Embora eles não suportem transações MTS, você pode querer alterar seus componentes MTS/COM+ para Web Services em situações onde você não precisa de transações distribuídas mas quer interoperar com outras plataformas. Embora não haja nenhum método automático para isto, a tarefa é trivial e pode ser completada em alguns minutos usando uma operação drag-and-drop após a atualização do seu projeto para o Visual Basic.NET.

Quando seu projeto é atualizado, controles de usuário são atualizados para controles Windows; entretanto, configurações de tag de propriedade customizadas e atribuições de chaves aceleradoras não serão atualizadas.

## Aplicações de Uma Camada

O Visual Basic 6.0 suportou a construção de vários tipos de aplicações de uma camada:

- Aplicações de banco de dados de uma camada
- Visual Basic add-ins
- Programas utilitários e jogos

Aplicações de banco de dados de uma camada são caracterizadas por uma aplicação Visual Basic armazenando dados em um banco de dados Microsoft Access. Estas aplicações serão atualizadas para o Visual Basic.NET com algumas limitações (veja a seção Dados mais a frente neste documento).

Agora que o Visual Basic.NET IDE é uma parte totalmente integrada do Visual Studio.NET IDE, o Visual Basic.NET possui um novo modelo de extensibilidade independente de linguagem. Visual Basic.NET add-ins agora são Visual Studio.NET add-ins, e você pode automatizar e incluir recursos a qualquer linguagem no Visual Studio.NET. Por exemplo, você pode escrever um Visual Basic.NET add-in que colore novamente um Visual C# Windows Form ou inclui comentários em uma classe Visual Basic. Para proporcionar esta funcionalidade, o Visual Basic.NET mudou do antigo modelo de extensibilidade, e você terá que alterar os objetos de extensibilidade na sua aplicação para aproveitar os novos recursos.

Muitas aplicações caem na categoria de programas Utilitários. Aplicações utilitárias que manipulam arquivos, parâmetros do registry e semelhantes serão freqüentemente atualizadas sem a necessidade de quaisquer alterações adicionais. Após a atualização, você pode aproveitar muitos recursos novos, como o tratamento de exceção na linguagem para capturar erros do file system, e usar classes do registry do .NET Framework para manipular o registry. É bom estar ciente que aplicações que se apoiam em características de performance específicas do Visual Basic 6.0, como os jogos arcade, provavelmente exigirão algumas modificações pois o Visual Basic.NET possui características de performance diferentes. Para o suporte a jogos no Visual Basic.NET, você pode usar o Microsoft DirectX® 7, ou a nova versão do GDI. O GDI+ introduz muitos novos recursos, incluindo o suporte Alpha blending para todas as primitivas de gráficos 2-D, anti-aliasing e suporte expandido para formatos de arquivo de imagem.

## Dados

O Visual Basic 6.0 propôs vários tipos de acesso a dados:

- ActiveX Data Objects (ADO)
- Remote Data Objects (RDO)
- Data Access Objects (DAO)

O Visual Basic.NET introduz uma versão aprimorada do ADO chamada ADO.NET. ADO.NET tem como alvo dados desconectados e proporciona melhorias de performance com relação ao ADO quando

usado em aplicações distribuídas. ADO.NET oferece data binding de leitura/gravação de controles para Windows Forms e data binding somente de leitura para Web Forms.

DAO, RDO e ADO ainda podem ser usados no código do Visual Basic.NET, com algumas modificações triviais (cobertas na seção linguagem deste documento). Entretanto, o Visual Basic.NET não suporta data binding DAO e RDO de controles, controles de dados ou conexão de Usuário RDO. Nós recomendamos que se suas aplicações contêm data binding DAO ou RDO você as mantenha no Visual Basic 6.0 ou atualize o data binding DAO ou RDO para ADO antes de atualizar seu projeto para o Visual Basic.NET, pois data binding ADO é suportado no Windows Forms. Informação sobre como fazer isso está disponível na Ajuda do Visual Basic 6.0.

Em resumo, nós recomendamos o uso de ADO nos seus projetos Visual Basic 6.0.

# Atualização

Quando seu código é atualizado, o Visual Basic.NET cria um novo projeto atualizado aproveitando ao máximo as alterações necessárias de linguagem e objeto para você. As seções seguintes proporcionam alguns exemplos de como seu código é atualizado.

## Variant para Object

Versões anteriores do Visual Basic suportavam o tipos de dados **Variant**, que podia ser atribuído para qualquer tipo primitivo (exceto strings de tamanho fixo), Empty, Error, Nothing e Null. No Visual Basic.NET, a funcionalidade dos tipos de dados **Variant** e **Object** é combinada em um novo tipo de dados: **Object**. O tipo de dados **Object** pode ser atribuído para tipos de dados primitivos, Empty, Nothing, Null e como um ponteiro para um objeto.

Quando seu projeto é atualizado para o Visual Basic.NET, todas as variáveis declaradas como **Variant** são alteradas para **Object**. Além disso, quando é inserido código no Editor, a palavra chave **Variant** é substituída por **Object**.

## Integer para Short

No Visual Basic 6.0, o tipo de dados para números inteiros de 16-bits agora é **Short**, e o tipo de dados para números inteiros de 32-bits agora é **Integer** (**Long** tem agora 64 bits). Quando seu projeto é atualizado, os tipos das variáveis são alterados:

```
Dim x As Integer
dim y as Long
```

é atualizado para:

```
Dim x As Short
dim y as Integer
```

## Sintaxe de Propriedade

O Visual Basic.NET introduz uma sintaxe mais intuitiva para propriedades, que agrupa **Get** e **Set**. Seus comandos de propriedade são atualizados conforme o seguinte exemplo:

```
Property Get MyProperty() As Integer
    m_MyProperty = MyProperty
End Property
Property Let MyProperty(NewValue As Integer)
    m_MyProperty = NewValue
End Property
```

é atualizado para:

```
Property MyProperty() As Short
    Get
        m_MyProperty = MyProperty
    End Get
    Set
        m_MyProperty = Value
    End Set
End Property
```

## Formulários Visual Basic para Windows Forms

O Visual Basic.NET possui um novo pacote de formulários, o Windows Forms, que possui suporte nativo para acessibilidade e possui um editor de menu in-place. Seus Formulários Visual Basic existentes são atualizados para Windows Forms.

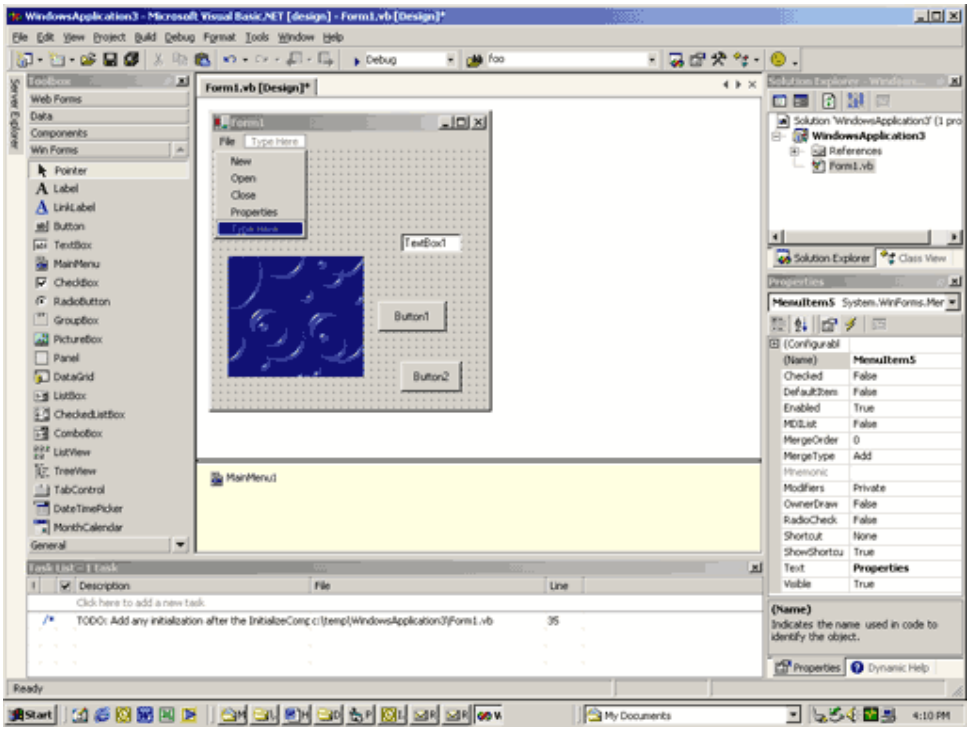

**Figura 2. Editor de menu in-place do Windows Forms. (Clique na figura para ampliá-la).**

## Interfaces

Nas versões anteriores do Visual Basic, interfaces para classes públicas eram sempre escondidas do usuário. No Visual Basic.NET, elas podem ser vistas e editadas no Editor de Código. Quando seu projeto é atualizado, você escolhe ter ou não declarações de interface criadas automaticamente para suas classes públicas.

# Relatório de Atualização e Comentários

Após a atualização do seu projeto, um relatório de atualização é incluído a ele, detalhando quaisquer alterações que você terá que fazer no seu código atualizado. Adicionalmente, são incluídos comentários no seu código para alertá-lo sobre qualquer problema potencial. Estes comentários aparecem automaticamente na Lista de Tarefas do Visual Studio.NET.

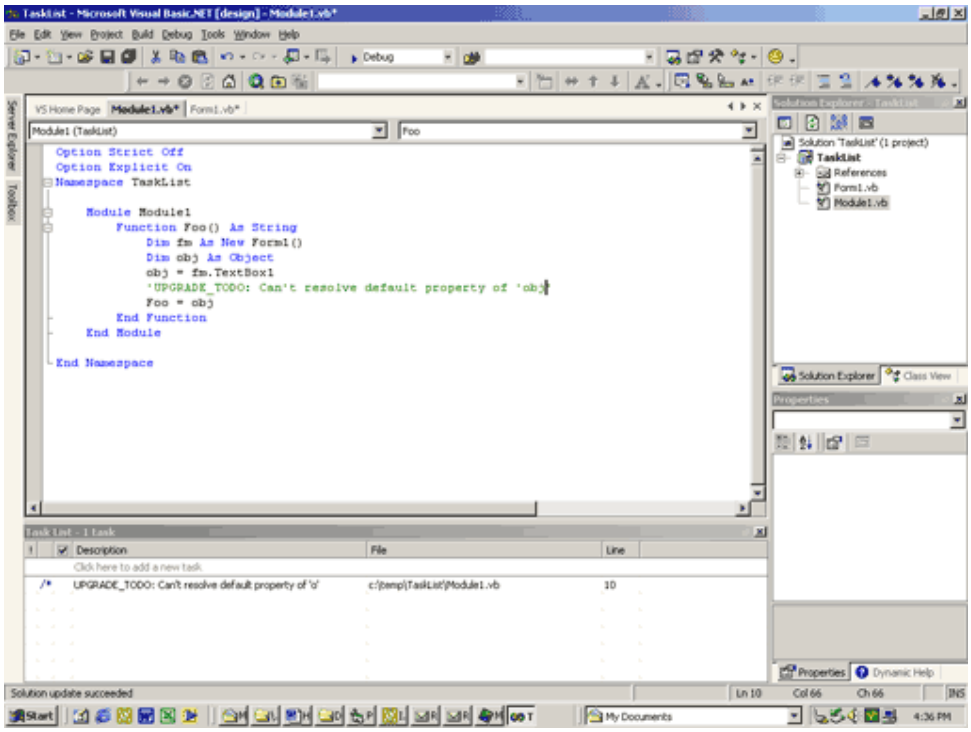


**Figura 3. São incluídos comentários de atualização no código Visual Basic e na Lista de Tarefas. (Clique na figura para ver a imagem ampliada).**

# Recomendações de Programação

Esta seção proporciona recomendações sobre como você deveria escrever código para minimizar as alterações que você terá que fazer após a atualização do seu projeto para o Visual Basic.NET.

## Use Early-Binding

O Visual Basic 6.0 e o Visual Basic.NET suportam objetos late-bound, que é a prática de declarar uma variável como o tipo de dados **Object** e atribuí-la a uma instância de uma classe em tempo de execução. Entretanto, durante o processo de atualização, objetos late-bound podem introduzir problemas na resolução de propriedades padrão, ou quando o modelo básico de objeto foi alterado e propriedades, métodos e eventos têm que ser convertidos. Por exemplo, suponha que você possui um Formulário chamado Form1 com um label nomeado Label1; o seguinte código Visual Basic 6.0 poderia definir o título label como "SomeText":

```
Dim o As Object
Set o = Me.Label1
o.Caption = "SomeText"
```

No Visual Basic.NET Windows Forms, a propriedade **Caption** de um controle de label é agora chamada de **Text**. Quando seu código é atualizado, todas as instâncias da propriedade **Caption** são alteradas para **Text**, mas como um objeto late-bound não possui tipo, o Visual Basic não pode detectar qual é o tipo de objeto, ou se quaisquer propriedades deveriam ser traduzidas. Nestes casos, você mesmo terá que alterar o código após a atualização.

Se você reescrever o código usando objetos early-bound, ele será atualizado automaticamente:

```
Dim o As Label
Set o = Me.Label1
o.Caption = "SomeText"
```

Sempre que possível, você deve declarar variáveis com o tipo de objeto apropriado ao invés de simplesmente declará-las como o tipo de dados **Object**.

Nos casos onde você usa variáveis Object e Variant no seu código Visual Basic 6.0, nós recomendamos a você usar conversões explícitas ao assinalar variáveis, efetuar operações nas variáveis, ou passar as variáveis para uma função. Por exemplo, a intenção da operação '+' no código seguinte não está clara:

```
Dim Var1 As Variant
Dim Var2 As Variant
Dim Var3 As Variant
Var1 = "3"
Var2 = 4
Var3 = Var1 + Var2              'RUIM: Qual é a intenção?
```

Var1 e Var2 devem ser incluídas como strings ou integers?

O exemplo acima pode resultar em um erro de run-time no Visual Basic.NET. A rescrita da linha final para usar conversões explícitas garante que o código irá funcionar:

```
Var3 = CInt(Var1) + CInt(Var2)   'BOM: conversão explícita
```

O Visual Basic.NET suporta funções de overloading com base no tipo do parâmetro. Por exemplo, a função **Environ** possui agora duas formas:

```
Environ( Expression As Integer) As String
Environ( Expression As String ) As String
```

O Visual Basic.NET determina qual função chamar com base no tipo do parâmetro. Se você passar um inteiro para **Environ()**, a versão integer será chamada; se você passar um string, então a versão string será chamada. Código que passa um tipo de dados **Variant** ou **Object** para uma função overloaded pode causar um erro de compilação ou runtime. O uso de uma conversão explícita, como no exemplo seguinte, significa que o seu código irá funcionar como desejado após sua atualização para o Visual Basic.NET:

```
Dim a As String
Dim v As Variant
v = "Path"
a = Environ(CStr(v))   'BOM: conversão explícita
```

O uso de conversões explícitas de objetos late bound é uma boa prática de codificação. Ela facilita a determinação da intenção do código e torna mais fácil para você mover seu projeto para o Visual Basic.NET.

## Use Date para Armazenar Datas

Versões anteriores do Visual Basic suportavam o uso do tipo de dados **Double** para armazenar e manipular datas. Você não deve fazer isto no Visual Basic.NET, pois datas não são armazenadas internamente como doubles. Por exemplo, o código seguinte é válido no Visual Basic 6.0, mas pode causar um erro de compilação no Visual Basic.NET:

```
Dim dbl As Double
Dim dat As Date
dat = Now
dbl = dat          'RUIM: Double não pode ser atribuído para uma data
dbl = DateAdd("d", 1, dbl)   'RUIM: Double não pode ser usado em
funçõe s de data
dat = CDate(dbl)   'RUIM: CDate não pode converter um double para uma
data
```

O .NET framework proporciona as funções **ToOADate** e **FromOADate** para a conversão entre doubles e datas. Entretanto, quando seu projeto é atualizado para o Visual Basic.NET, é difícil determinar a intenção do código que usa doubles para armazenar datas. Para evitar modificações desnecessárias no seu código em Visual Basic.NET, use sempre o tipo de dados **Date** para armazenar datas.

## Resolva Propriedades Padrão Sem Parâmetro

No Visual Basic 6.0, muitos objetos expõem propriedades padrão, que podem ser omitidas como um atalho de programação. Por exemplo, **TextBox** possui uma propriedade padrão **Text**, então ao invés de escrever:

```
MsgBox Form1.Text1.Text
```

você usa o atalho:

```
MsgBox Form1.Text1
```

A propriedade padrão é resolvida quando o código é compilado. Além disso, você também deve usar propriedades padrão com objetos late-bound, como no exemplo abaixo:

```
Dim obj As Object
Set obj = Form1.Text1
MsgBox obj
```

No exemplo late-bound, a propriedade padrão é resolvida em tempo de execução, e o **MsgBox** mostra o valor da propriedade padrão do **TextBox** como **Text1**.

O Visual Basic.NET não suporta propriedades padrão sem parâmetro, e consequentemente não permite este atalho de programação. Quando seu projeto é atualizado, o Visual Basic.NET resolve as propriedades padrão sem parâmetro, mas usos de late-bound que se apoiam em resolução em tempo de execução não podem ser resolvidos automaticamente. Nestes casos, você mesmo terá que alterar o código. Uma complicação adicional é que muitas bibliotecas implementam propriedades padrão usando uma propriedade chamada **_Default**. **_Default** age como um proxy, passando chamadas para a propriedade padrão real. Dessa forma, quando seu projeto é atualizado, algumas propriedades padrão serão resolvidas para **_Default**. O código ainda funcionará como sempre, mas será mais difícil de entender do que o código escrito explicitamente usando-se a propriedade real. Por estes motivos, tente evitar o uso de propriedades padrão sem parâmetro no seu código Visual Basic 6.0. Ao invés de escrever:

```
Dim obj As Object
Set obj = Me.Text1
MsgBox obj        'RUIM: Apoio em propriedade padrão
MsgBox Me.Text1   'RUIM: Apoio em propriedade padrão
```

use:

```
Dim obj As Object
Set obj = Me.Text1
MsgBox obj.Text        'BOM: Propriedade padrão está resolvida
MsgBox Me.Text1.Text   'BOM: Propriedade padrão está resolvida
```

Embora propriedades padrão *sem parâmetro* não sejam suportadas no Visual Basic.NET, propriedades padrão *com parâmetro* são suportadas. Para entender a diferença entre os dois tipos, considere que propriedades padrão com parâmetro sempre possuem um índice. Um exemplo é a propriedade padrão do ADO **recordset**: a coleção **Fields**. O código:

```
Dim rs As ADODB.Recordset
rs("CompanyName") = "SomeCompany"
rs!CompanyName = "SomeCompany"
```

é na verdade um atalho para:

```
Dim rs As ADODB.Recordset
rs.Fields("CompanyName").Value = "SomeCompany"
rs.Fields!CompanyName.Value = "SomeCompany"
```

Neste caso, a propriedade **Fields** possui parâmetro, e portanto seu uso é válido no Visual Basic.NET; entretanto, a propriedade padrão da propriedade **Fields**, **Value,** não possui parâmetro, então o uso correto no Visual Basic.NET é:

```
Dim rs As ADODB.Recordset
rs("CompanyName").Value = "SomeCompany"
rs!CompanyName.Value = "SomeCompany"
```

Este exemplo e a maior parte das outras propriedades padrão são resolvidos para você quando o projeto é atualizado, portanto resolvê-los no Visual Basic 6.0 é simplesmente uma boa prática de programação. Entretanto, você deve evitar usar propriedades padrão com os tipos de dados **Object** e **Variant**, pois eles não podem ser resolvidos e você mesmo terá que corrigir o código no projeto atualizado.

## Use Comparações Booleanas com AND/OR/NOT

As palavras chave **And** e **Or** funcionam de forma diferente no Visual Basic.NET e no Visual Basic 6.0. No Visual Basic 6.0, a palavra chave **And** efetuava um **AND** lógico assim como um Bitwise **AND** dependendo dos tipos dos operandos (devido ao fato de **True** ter o valor de –1). No Visual Basic.NET, **AND** efetua somente um **AND** lógico. No Visual Basic.NET, um novo conjunto de operadores foi incluído na linguagem para efetuar operações Bitwise: **BitAnd**, **BitOr**, **BitNot** e **BitXor**.

O exemplo seguinte demonstra o efeito desta diferença:

```
Dim a As Integer
Dim b As Integer
Dim c As Boolean
a = 1
b = 2
c = a And b
MsgBox ("A resposta é " & c)
```

Quando este código é executado no Visual Basic 6.0, a resposta é **False** (Bitwise **AND**); entretanto, no Visual Basic.NET, a resposta é **True** (**AND** lógico). Para garantir que seu código se comporte da mesma maneira após a atualização, o Visual Basic.NET inclui as funções de compatibilidade **VB6.And**, **VB6.Or** e **VB6.Not**, que avaliam **AND/OR/NOT** da mesma forma que o Visual Basic 6.0 (escolhendo lógica ou Bitwise dependendo dos operandos). Quando o código acima é atualizado, o resultado será parecido com o seguinte:

```
Dim a As Short
Dim b As Short
Dim c As Boolean
a = 1
b = 2
c = VB6.And(a, b)
MsgBox ("A resposta é " & c)
```

O código atualizado irá produzir a resposta **False**, da mesma forma que o código original no Visual Basic 6.0.

Para evitar que seu código seja atualizado para as funções de compatibilidade, tente garantir que seus comandos **AND/OR/NOT** utilizam comparações Booleanas. Por exemplo, se o exemplo acima é modificado para:

```
Dim a As Integer
Dim b As Integer
Dim c As Boolean
a = 1
b = 2
c = a <> 0 And b <> 0
MsgBox ("A resposta é " & c)
```

então após a atualização do projeto, o código resultante será mais familiar:

```
Dim a As Short
Dim b As Short
Dim c As Boolean
a = 1
b = 2
c = a <> 0 And b <> 0
MsgBox ("A resposta é " & c)
```

A diferença é que cada operador sendo comparado é uma expressão Booleana, e portanto usa o **AND** lógico no Visual Basic 6.0. O **AND** lógico produz o mesmo resultado tanto no Visual Basic 6.0 quanto no Visual Basic.NET e portanto o código não é alterado. Isto significa que você pode cortar e colar código entre o Visual Basic.NET e o Visual Basic 6.0, e seu código será executado mais rapidamente no Visual Basic.NET pois ele está usando o operador **AND** nativo ao invés de uma função de compatibilidade.

O Visual Basic.NET trata funções em operações **AND/OR/NOT** de forma diferente que o Visual Basic 6.0. Considere o seguinte exemplo:

```
Dim b As Boolean
b = Function1() And Function2()
```

No Visual Basic 6.0, **Function1** e **Function2** são avaliados. No Visual Basic.NET, **Function2** só é avaliada se **Function1** retorna **True**. Isto é conhecido como *curto-circuito* dos operadores lógicos. Na maioria dos casos a única diferença em tempo de execução é que a versão curto-circuitada executa mais rapidamente; entretanto, se **Function2** possui efeitos colaterais, como a manipulação de um banco de dados ou uma variável global, então o comando terá um comportamento em tempo de execução diferente do que no Visual Basic 6.0. Para evitar este problema, se seus comandos **AND/OR/NOT** contêm funções, métodos ou propriedades então o comando é atualizado para uma versão de compatibilidade que avalia as funções. O exemplo acima seria atualizado para o seguinte:

```
Dim b As Boolean
b = VB6.AND(Function1(), Function2())
```

Para evitar que seu código seja atualizado para a versão de compatibilidade, faça as seguintes modificações:

```
Dim b As Boolean
Dim c As Boolean
Dim d As Boolean
c = Function1()
d = Function2()
b = c And d
```

Também é importante notar que no Visual Basic.NET, o valor base de **True** foi alterado de –1 para 1. Esta alteração foi feita para ajudar aplicações Visual Basic a interoperar com outras linguagens .NET, e resolve finalmente uma diferença importante com o Visual C++. Devido a esta alteração, nas suas aplicações Visual Basic 6.0, você sempre deve usar a constante **True** ao invés de -1, e tipos Boolean ao invés de integers para tipos Boolean. Para ilustrar a importância disto, considere o exemplo abaixo, que produz o resultado **True** no Visual Basic 6.0, e **False** no Visual Basic.NET:

```
Dim i As Integer
```

```
i = True
If i = -1 Then
    MsgBox ("True")
Else
    MsgBox ("False")
End If
```

Entretanto, alterar este código para usar Booleanos gera o resultado **True** tanto no Visual Basic 6.0 quanto no Visual Basic.NET, e também gera um código mais legível:

```
Dim i As Boolean
i = True
If i = True Then
    MsgBox ("True")
Else
    MsgBox ("False")
End If
```

As coisas mais importantes para lembrar e implementar provenientes deste exemplo são:

- Sempre use os nomes constantes **True** e **False** ao invés de seus valores base 0 e –1.
- Use o tipo de dados Boolean para armazenar valores Boolean.

Se você não fizer estas duas coisas, talvez você tenha que fazer alterações no seu projeto após ele ter sido atualizado para o Visual Basic.NET.

## Evite Propagação de Null

Versões anteriores do Visual Basic suportavam propagação de Null. Propagação de Null suporta a premissa de que quando o null é usado em uma expressão, o resultado da expressão será ele próprio Null. Em cada caso do exemplo seguinte, o resultado de V é sempre Null.

```
Dim V
V = 1 + Null
V = Null + Right$("SomeText", 1)
V = Right("SomeText", 0)
```

Propagação de Null não é suportada no Visual Basic.NET. O comando **1+Null** irá gerar uma má combinação de tipos no Visual Basic.NET. Além disso, onde o Visual Basic 6.0 tinha duas versões da função **Left**—**Left$** retornando um string, **Left** retornando uma variant que poderia ser Null—o Visual Basic.NET possui somente uma versão, **Left**, que sempre retorna um string.

Para estar compatível com o Visual Basic 6.0 e com o Visual Basic.NET você deve sempre escrever código que testa o Null ao invés de apoiar-se na propagação de Null. Além disso, no Visual Basic.NET, as seguintes funções não mais retornarão Null:

| | |
|---|---|
| **Chr** | **Mid** |
| **Command** | **Oct** |
| **CurDir** | **Right** |
| **Date** | **RTrim** |
| **Environ** | **Space** |
| **Error** | **Str** |
| **Hex** | **Time** |
| **Lcase** | **Trim** |
| **Ltrim** | **UCase** |

Propagação de Null normalmente é usada em aplicações de banco de dados, onde você precisa verificar se um campo de banco de dados contém Null. Nestes casos você deve verificar resultados usando a função **IsNull()** e efetuar a ação apropriada.

## Use Arrays Delimitados por Zero

O Visual Basic 6.0 permitiu a você definir arrays com limites inferiores e superiores de qualquer número inteiro. Você também pode usar **ReDim** para reinstalar uma variant como um array. Para habilitar interoperabilidade com outras linguagens, arrays no Visual Basic.NET devem ter um limite inferior como zero, e o **ReDim** não pode ser usado a menos que a variável tenha sido declarada anteriormente com Dim As Array. Embora isto restrinja a forma como arrays podem ser definidos, isto realmente permite a você passar arrays entre o Visual Basic.NET e qualquer outra linguagem .NET. O exemplo seguinte mostra a restrição:

```
Dim a(1 To 10) As Integer   'RUIM: LBound deve ser 0
Dim v
ReDim v(10)                 'RUIM: Não pode usar ReDim sem Dim
Dim b(10) As Integer        'OK: Cria um array de 10 inteiros
ReDim b(5) As Integer       'OK: Pode fazer ReDim de var
anteriormente Dimed
```

Além disso, no Visual Basic 6.0, Dim (0 to 10) As Integer criava um array de 11 inteiros, indexado de 0 to 10. O mesmo comando no Visual Basic.NET cria um array de 10 inteiros, de 0 a 9.

Um efeito colateral é que Option Base 0|1 foi removido da linguagem.

Quando seu projeto é atualizado para o Visual Basic.NET, quaisquer comandos baseados em opção são removidos do seu código. Se o array é limitado por zero, ele não é alterado. Entretanto, se um array não é limitado por zero, então ele é atualizado para uma classe wrapper array, como no exemplo seguinte:

```
Dim a(1 To 10) As Integer
```

é alterado para:

```
Dim a As Object = New VB6.Array(GetType(Short), 1,10)
```

A classe wrapper array é bem mais lenta que o array nativo, e há limitações ao uso de dois tipos de array na mesma aplicação. Por exemplo, você não pode passar um wrapper array para algumas funções com parâmetros do tipo Array, e talvez você não consiga passar um wrapper array para uma classe Visual C# ou Visual C++.

Por esta razão, você deve usar arrays delimitados por zero no seu código Visual Basic 6.0, evitar usar **ReDim** como uma declaração de array e evitar usar Option Base 1.

## Use Constantes Ao Invés de Valores Básicos

Ao escrever código, tente usar constantes ao invés de seus valores básicos. Por exemplo, se você está maximizando um formulário em tempo de execução, use:

```
Me.WindowState = vbMaximized   'BOM: Nome constante
```

ao invés de:

```
Me.WindowStyle = 2   'Ruim: Valor básico
Me.WindowStyle = X   'Ruim: Variável
```

Da mesma forma, use **True** e **False** ao invés de -1 e 0.

No Visual Basic.NET, os valores e em alguns casos os nomes de algumas propriedades e constantes foram alterados; por exemplo, o valor de **True** muda de -1 para 1. Quando seu projeto é atualizado para

o Visual Basic.NET, a maior parte das constantes são alteradas automaticamente para você; entretanto, se você está usando valores ou variáveis básicas ao invés dos nomes das constantes, muitos casos não podem ser atualizados automaticamente. O uso de nomes de constante minimiza o número de modificações que você tem que fazer.

## Arrays e Strings de Tamanho Fixo em Tipos Definidos pelo Usuário

Devido às alterações feitas que permitem que arrays e estruturas do Visual Basic.NET sejam totalmente compatíveis com outras linguagens Visual Studio.NET, strings de tamanho fixo não são mais suportados na linguagem. Na maioria dos casos isto não é um problema, pois há uma classe de compatibilidade que proporciona comportamento de string de tamanho fixo, e portanto o código:

```
Dim MyFixedLengthString As String * 100
```

é atualizado para:

```
Dim MyFixedLengthString As New VB6.FixedLengthString(100)
```

Entretanto, strings de tamanho fixo realmente causam um problema quando usados em estruturas (também conhecidas como tipos definidos pelo usuário). O problema surge pois a classe de string de tamanho fixo não é criada automaticamente quando o tipo definido pelo usuário é criado. Um problema adicional é que arrays de tamanho fixo também não são criados quando o tipo definido pelo usuário é criado.

Quando seu código é atualizado, tipos definidos pelo usuário com strings ou arrays de tamanho fixo serão marcados com um comentário avisando-o para iniciar o string ou array de tamanho fixo antes de usar o tipo definido pelo usuário. Entretanto, você pode evitar esta modificação alterando seus tipos Visual Basic 6.0 definidos pelo usuário para usar strings ao invés de strings de tamanho fixo, e arrays não inicializados ao invés de arrays de tamanho fixo. Por exemplo:

```
Private Type MyType
    MyArray(5) As Integer
    MyFixedString As String * 100
End Type
Sub Bar()
    Dim MyVariable As MyType
End Sub
```

pode ser alterado para:

```
Private Type MyType
    MyArray() As Integer
    MyFixedString As String
End Type
Sub Bar()
    Dim MyVariable As MyType
    ReDim MyVariable.MyArray(5) As Integer
    MyVariable.MyFixedString = String$(100, " ")
End Sub
```

## Evite Recursos Legados

Por terem sido removidas da linguagem, você deve evitar usar as seguintes palavras chave:

- **Def<type>**
- **Computed GoTo/GoSub**
- **GoSub/Return**

- **Option Base 0|1**
- **VarPtr, ObjPtr, StrPtr**
- **LSet**

Elas são explicadas em maiores detalhes abaixo.

### Def<type>

Nas versões anteriores do Visual Basic, DefBool, DefByte, DefInt, DefLng, DefCur, DefSng, DefDbl, DefDec, DefDate, DefStr, DefObj e DefVar eram usadas na seção de declarações de um módulo para definir um faixa de variáveis como de um certo tipo. Por exemplo:

```
DefInt A-C
```

definia todas as variáveis começando com as letras A, B, ou C como um inteiro. Ao invés de usar comandos **Def<type>**, você deve declarar variáveis explicitamente.

### Computed GoTo/GoSub

Comandos **Computed GoTo/GoSub** têm esta forma:

```
On x GoTo 100, 200, 300
```

Eles não são suportados no Visual Basic.NET. No lugar deles, você deve usar comandos **If**, e construções Select Case.

### GoSub/Return

Comandos **GoSub** e **Return** não são suportados no Visual Basic.NET. Na maioria dos casos você pode substituí-los por estas funções e procedimentos.

### Option Base 0|1

**Option Base 0|1** era usado para especificar o limite inferior padrão de um array. Como mencionado anteriormente, este comando foi removido da linguagem pois o Visual Basic.NET somente suporta nativamente arrays com um limite inferior zero. Arrays com limite inferior não zero são suportados através de uma classe wrapper.

### VarPtr, ObjPtr, StrPtr

**VarPtr**, **VarPrtArray**, **VarPtrStringArray**, **ObjPtr** e **StrPtr** eram funções não documentadas usadas para obter o endereço básico de memória das variáveis. Estas funções não são suportadas no Visual Basic.NET.

### LSet

No Visual Basic 6.0, o comando **LSet** podia ser usado para atribuir uma variável de um tipo definido pelo usuário para outra variável de um tipo diferente definido pelo usuário. Esta funcionalidade não é suportada no Visual Basic.NET.

## Windows APIs

Muitas APIs podem ser usadas exatamente como no Visual Basic 6.0, lembrando que você tem que ajustar seus tipos de dados de acordo. O tipo de dados **Long** do Visual Basic 6.0 é agora o tipo de dados **Integer** do Visual Basic.NET, e o tipo de dados **Integer** do Visual Basic 6.0 é agora o tipo de

dados **Short** no Visual Basic.NET. Durante a atualização, estas alterações são feitas para você, e APIs simples funcionam exatamente como no Visual Basic 6.0. Por exemplo:

```
Private Declare Function GetVersion Lib "kernel32" () As Long
Function GetVer()
    Dim Ver As Long
    Ver = GetVersion()
    MsgBox ("System Version is " & Ver)
End Function
```

é alterado para:

```
Private Declare Function GetVersion Lib "kernel32" () As Integer
Function GetVer()
    Dim Ver As Integer
    Ver = GetVersion()
    MsgBox("System Version is " & Ver)
End Function
```

Além das atualizações dos tipos de dados numéricos, o Visual Basic 6.0 tinha um tipo de dados de string de tamanho fixo que não é suportado no Visual Basic.NET, e que é atualizado para uma classe wrapper string de tamanho fixo. Em muitos casos no Visual Basic 6.0 você pode efetuar a mesma ação usando um string normal. Por exemplo:

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Long) As
Long
Function GetUser()
    Dim Ret As Long
    Dim UserName As String
    Dim Buffer As String * 25
    Ret = GetUserName(Buffer, 25)
    UserName = Left$(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox (UserName)
End Function
```

pode ser melhor escrito usando-se um string normal definido explicitamente para o tamanho 25 ao invés de um string de tamanho fixo:

```
    Dim Buffer As String
    Buffer = String$(25, " ")
```

Este código é atualizado para o Visual Basic.NET da seguinte forma:

```
Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Integer) As
Integer
Function GetUser()
    Dim Ret As Integer
    Dim UserName As String
    Dim Buffer As String
    Buffer = New String(CChar(" "), 25)
    Ret = GetUserName(Buffer, 25)
    UserName = Left(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox(UserName)
End Function
```

Em alguns casos, o Visual Basic.NET trata melhor a passagem de strings para APIs, pois você pode declarar opcionalmente como você quer que os strings sejam passados usando as palavras chave **ANSI** e **UNICODE**.

Há três casos onde você precisa fazer algumas alterações. O primeiro é a passagem de tipos definidos pelo usuário que contêm strings de tamanho fixo ou byte arrays para APIs. No Visual Basic.NET talvez você tenha que alterar seu código, incluindo o atributo **MarshallAs** (a partir do **System.Runtime.InteropServices**) para cada string de tamanho fixo ou byte array no tipo definido pelo usuário. O segundo caso é o uso do tipo de variável **As Any** em um comando **Declare**. Isto não é suportado no Visual Basic.NET. Variáveis do tipo **As Any** sempre foram usadas para passar uma variável do tipo string ou Null; você pode substituir este uso no Visual Basic 6.0 declarando duas formas da API, uma com longs, uma com strings. Por exemplo, a API GetPrivateProfileString API tem um parâmetro *lpKeyName* do tipo **As Any**:

```
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias
   "GetPrivateProfileStringA" (ByVal lpApplicationName As String,
ByVal
      lpKeyName As Any, ByVal lpDefault As String, ByVal
         lpReturnedString As String, ByVal nSize As Long, ByVal
            lpFileName As String) As Long
```

Você pode remover o "As Any" substituindo o **Declare** por duas versões; uma que aceita um long, e uma que aceita um string:

```
Private Declare Function GetPrivateProfileStringKey Lib "kernel32"
Alias
   "GetPrivateProfileStringA" (ByVal lpApplicationName As String,
ByVal
      lpKeyName As String, ByVal lpDefault As String, ByVal
         lpReturnedString As String, ByVal nSize As Long, ByVal
            lpFileName As String) As Long
Private Declare Function GetPrivateProfileStringNullKey Lib
"kernel32"
   Alias "GetPrivateProfileStringA" (ByVal lpApplicationName As
String,
      ByVal lpKeyName As Long, ByVal lpDefault As String, ByVal
         lpReturnedString As String, ByVal nSize As Long, ByVal
            lpFileName As String) As Long
```

Quando você quer passar o valor Null para a API, você usa a versão **GetPrivateProfileStringNullKey**. Desta forma a função pode ser atualizada para o Visual Basic.NET.

A última área onde você pode ter que fazer algumas alterações é se você estiver usando APIs que efetuam criação de thread, Windows subclassing, message queue hooking, e assim por diante. Algumas destas funções causarão um erro de run-time no Visual Basic.NET. Muitas destas APIs possuem equivalentes no Visual Basic.NET ou no .NET Framework. Você terá que corrigir estas situações caso a caso.

## Considerações para Formulários e Controles

O Visual Basic.NET possui um novo pacote de formulários, o Windows Forms. Windows Forms é amplamente compatível com o pacote de formulários encontrado no Visual Basic 6; entretanto, há algumas diferenças importantes descritas abaixo:

- Windows Forms não suporta o controle de OLE container; você deve evitar o uso deste controle nas suas aplicações Visual Basic 6.0.

- Não há controle de figuras no Windows Forms. Figuras quadradas e retangulares serão atualizadas para labels, e figuras ovais e círculos não podem ser atualizados. Você deve evitar o uso destas figuras nas suas aplicações.

- Não há controle de linha no Windows Forms. Linhas horizontais e verticais são atualizadas para labels. Linhas diagonais não são atualizadas, e você deve evitar seu uso.

- Windows Forms possui um novo conjunto de comandos gráficos que substituem os métodos Form **Circle**, **CLS**, **PSet**, **Line** e **Point**. Como o novo modelo de objeto é muito diferente do anterior no Visual Basic 6.0, estes métodos não podem ser atualizados.

- Para o controle de Timer, a definição da propriedade **Interval** como 0 não desabilita o timer; ao invés disto o intervalo é redefinido como 1. Nos seus projetos Visual Basic 6.0, você deve configurar **Enabled** para **False** ao invés de definir o **Interval** para 0.

- Windows Forms possui dois controles de menu, MainMenu e ContextMenu, sendo que o Visual Basic 6.0 possui apenas um controle de menu, Menu, que pode ser aberto como um MainMenu ou um ContextMenu. Controles de menu são atualizados para controles MainMenu, mas você não poderá usá-los como ContextMenus; você terá que recriar seus ContextMenus.

- Windows Forms não possui suporte para Dynamic Data Exchange (DDE).

- Windows Forms não suporta o método **Form.PrintForm**.

- Embora Windows Forms tenha suporte para a funcionalidade drag-and-drop, o modelo de objeto é muito diferente do usado no Visual Basic 6.0. Dessa forma, as propriedades e métodos drag-and-drop do Visual Basic 6.0 não podem ser atualizados.

- O framework .NET possui um objeto Clipboard aprimorado (**System.WinForms.Clipboard**) que oferece mais funcionalidade e suporta mais formatos clipboard do que o objeto Clipboard do Visual Basic 6.0. Entretanto, devido às diferenças nos modelos de objeto, comandos clipboard não podem ser atualizados automaticamente.

- Windows Forms não suporta a propriedade **Name** para formulários e controles em tempo de execução; dessa forma você não deve escrever código que repete a coleção **Controls** procurando um controle com um certo nome (esta funcionalidade agora está disponível usando-se as classes .NET **System.Reflection**.)

- Para garantir que seus formulários sejam atualizados para o tamanho certo, você deve sempre usar o ScaleMode padrão de twips nas suas aplicações. Durante a atualização, o Visual Basic.NET transforma suas coordenadas de formulários de twips para pixels.

---

© 2000 Microsoft Corporation. Todos os direitos reservados. Termos de uso.