

Testar envolve a descoberta de problemas em seu código; a depuração consiste no isolamento e na resolução de problemas. Testar e depurar são aspectos inevitáveis do ciclo de desenvolvimento e são melhor incorporados no início do ciclo. Testar e depurar os componentes individuais por completo facilita ainda mais o teste e a depuração dos aplicativos.

Para obter maiores informações sobre como criar um aplicativo, consulte o capítulo 2, [Desenvolvendo um aplicativo](#) e o capítulo 13, [Compilando um aplicativo](#)

Este capítulo aborda tópicos a seguir:

- [Planejando o teste e a depuração](#)
- [Depurando antes de encontrar falhas](#)
- [Isolando problemas](#)
- [Exibindo a saída](#)
- [Registrando alcance de código](#)
- [Gerenciando erros em tempo de execução](#)

## Planejando o teste e a depuração

Geralmente, os desenvolvedores procuram diferentes níveis de resistência ao testarem e depurarem seus aplicativos:

1. Executar sem quebrar ou gerar mensagens de erro.
2. Ação adequada em cenários comuns.
3. Mensagens de erro ou ações razoáveis em um intervalo de cenários.
4. Recuperação hábil a partir de interações inesperadas do usuário.

O Visual FoxPro fornece um conjunto rico de ferramentas que ajudam no isolamento e na identificação de problemas em seu código, de modo que você possa corrigi-los com eficiência. No entanto, uma das melhores maneiras de se criar um aplicativo resistente é procurar problemas potenciais antes que os mesmos ocorram.

## Depurando antes de encontrar falhas

Os estudos comprovam que uma codificação adequada (usando espaço em branco, incluindo comentários, aderindo a convenções de nomenclatura e etc.) tende a reduzir automaticamente a quantidade de falhas no código. Além disso, há algumas etapas que poderão ser seguidas no início do processo de desenvolvimento para facilitar o teste e a depuração posterior, incluindo:

- Criando um ambiente de teste
- Definindo instruções
- Vendo seqüências de evento

## Criando um ambiente de teste

O ambiente do sistema onde o aplicativo deverá ser executado é tão importante quanto o ambiente de dados configurado para o próprio aplicativo. Para assegurar a portabilidade e criar um contexto apropriado para testar e depurar, você deve considerar os elementos a seguir:

- Hardware e software
- Caminhos do sistema e propriedades de arquivos
- Estrutura de diretórios e localizações de arquivos

## Hardware e software

Para obter portabilidade máxima, você deve desenvolver aplicativos na plataforma mínima comum onde eles devem ser executados. Para estabelecer uma plataforma básica:

- Desenvolva os aplicativos usando o modo de vídeo comum mais básico.
- Determine os requisitos básicos para a memória RAM e o espaço de armazenamento de mídia, incluindo todos os drivers necessários ou software em execução simultânea.
- Considere cenários especiais de bloqueio de registros, arquivos e memória para utilização em rede em oposição a versões independentes de aplicativos.

## Caminhos do sistema e propriedades de arquivos

Para garantir que todos os arquivos de programa necessários estejam prontamente acessíveis em cada máquina que irá executar o aplicativo, você também precisará de uma configuração de arquivos básica. Para ajudá-lo a definir uma configuração básica, responda às perguntas a seguir:

- O aplicativo requer caminhos de sistema comuns?
- As propriedades adequadas de acesso aos arquivos foram definidas?
- As permissões de rede estão definidas corretamente para cada usuário?

## Estrutura de diretórios e localizações de arquivos

Se o código fonte fizer referência a nomes de arquivo ou caminhos absolutos, esses arquivos e caminhos exatos deverão existir quando o aplicativo for instalado em qualquer outro computador. Para evitar esse cenário, pode-se:

- Usar os arquivos de configuração do Visual FoxPro. Para obter maiores informações sobre como usar arquivos de configuração, consulte o capítulo 3, [Configurando o Visual FoxPro](#), no *Guia de Instalação e Índice Principal*.
- Criar um diretório separado ou uma estrutura de diretórios para manter os arquivos fonte separados dos arquivos do aplicativo criado. Dessa maneira, é possível testar as referências do aplicativo concluído e saber exatamente quais arquivos serão distribuídos.
- Usar caminhos relativos.

## Definindo instruções

É possível incluir instruções no código para verificar pressuposições feitas sobre o ambiente em tempo de execução do código.

### ► Para definir uma instrução

- Utilize o comando [ASSERT](#) para identificar pressuposições no programa.  
Quando a condição estipulada no comando ASSERT for avaliada como falsa (.F.), uma caixa de mensagem de instrução será exibida e repetida na janela **Depurar saída**.

Por exemplo, você poderia estar escrevendo uma função que espere um valor de parâmetro diferente de zero. A linha de código a seguir na função irá alertá-lo se o valor do parâmetro for 0:

```
ASSERT nParm != 0 MESSAGE "Recebido um parâmetro de 0"
```

É possível especificar se as mensagens de instrução são exibidas com o comando [SET ASSERTS](#). Como padrão, as mensagens de instrução não são exibidas.

## Vendo seqüências de evento

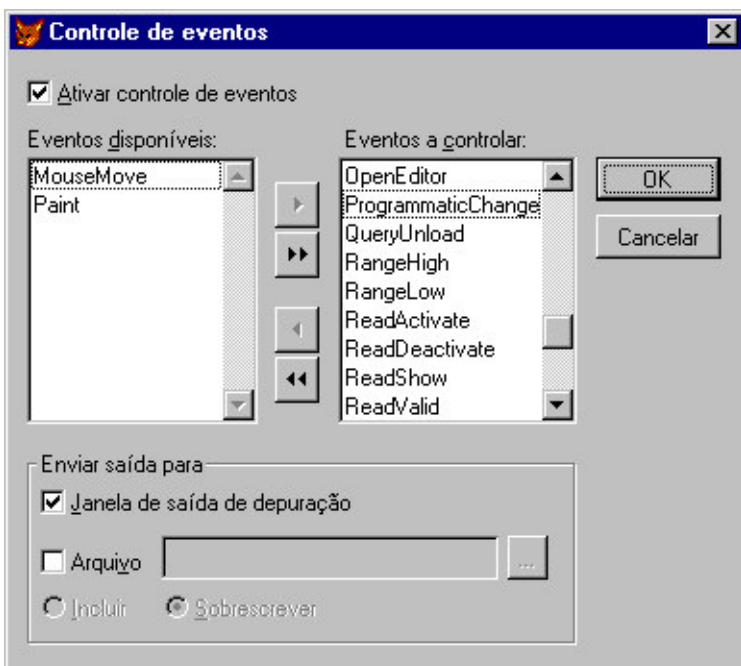
Quando você vir a ocorrência de eventos em relação a outros eventos, poderá determinar o local mais eficaz para incluir o código.

### ► Para controlar eventos

- No menu **Ferramentas** na janela [Depurador](#), escolha **Controle de eventos**.  
– Ou –
- Utilize o comando [SET EVENTTRACKING](#).

A caixa de diálogo **Controle de eventos** permite que você selecione os eventos que deseja ver.

### Caixa de diálogo Controle de eventos



**Observação** Neste exemplo, os eventos **MouseMove** e **Paint** foram removidos de **Eventos** para lista de controle pois eles ocorrem tão freqüentemente que dificultam a visualização das seqüências de outros eventos.

Quando o controle de eventos for ativado, sempre que ocorrer um evento do sistema em **Eventos** para lista de controle, o nome do evento será exibido na janela **Depurar saída** ou escrito em um arquivo. Se você optar por exibir os eventos na janela **Depurar saída**, ainda assim poderá salvá-los em um arquivo conforme descrito em [Exibindo a saída](#) posteriormente neste capítulo.

**Observação** Se a janela **Depurar saída** não estiver aberta, os eventos não serão listados, mesmo se a caixa **Janela de saída de depuração** estiver definida.

## Isolando problemas

Após o teste ter identificado problemas, você poderá usar o ambiente de depuração do Visual FoxPro para isolá-los das formas a seguir:

- Iniciando uma sessão de depuração
- Rastreamento o código
- Suspendendo a execução do programa
- Vendo os valores armazenados
- Exibindo a saída

## Iniciando uma sessão de depuração

Você inicia uma sessão de depuração abrindo o ambiente de depuração.

### ► Para abrir o depurador

- No menu **Ferramentas**, escolha **Depurador**.

**Observação** Se você estiver depurando no ambiente do Visual FoxPro, escolha a ferramenta de depuração que deseja abrir no menu **Ferramentas**.

Você também pode abrir o depurador com qualquer um dos comandos a seguir:

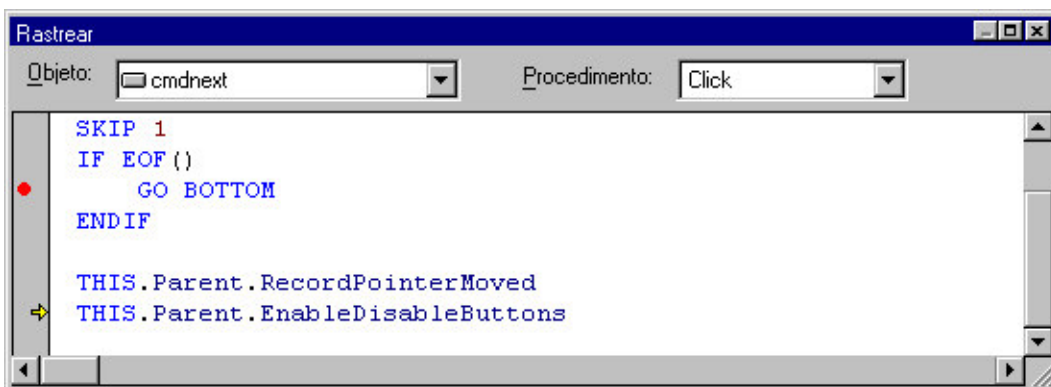
```
DEBUG
SET STEP ON
SET ECHO ON
```

O depurador é aberto automaticamente sempre que uma condição de ponto de interrupção for encontrada.

## Rastreando o código

Uma das estratégias de depuração mais úteis à sua disposição é a capacidade de rastrear o código, ver cada linha de código durante a sua execução e verificar os valores de todas as variáveis, propriedades e definições do ambiente.

### Código na janela Rastrear



#### ► Para rastrear o código

- 1 Inicie uma sessão de depuração.
- 2 Se nenhum programa estiver aberto na janela **Rastrear**, escolha **Executar** no menu **Depurar**.
- 3 Escolha **Depuração total** no menu **Depurar** ou clique sobre o botão da **Barra de ferramentas depuração total**.

Uma seta na área cinza à esquerda do código indica a próxima linha a ser executada.

**Dicas** As dicas a seguir devem ser analisadas:

- Defina os pontos de interrupção para restringir o intervalo de código que você precisa executar.
- Você pode ignorar uma linha de código que gerará um erro, colocando o cursor na linha de código após a linha com problema e escolhendo **Definir próxima instrução** no menu **Depurar**.
- Se a maior parte do código for associada aos eventos Timer, você poderá evitar o rastreamento deste código, limpando **Exibir eventos Timer** na guia **Depurar** da caixa de diálogo **Opções**.

Se um problema for isolado quando estiver depurando um código de objeto ou programa, poderá imediatamente repará-lo.

#### ► Para reparar os problemas encontrados durante o rastreamento do código

- No menu **Depurar**, escolha **Ajustar**.

Ao escolher **Ajustar** no menu **Depurar**, a execução do programa será cancelada e o editor de código será aberto no local onde o cursor está posicionado na janela **Rastrear**.

## Suspendendo a execução do programa

Você pode suspender a execução do programa definindo pontos de interrupção. Uma vez suspensa a execução do programa, você poderá verificar os valores de variáveis e propriedades, ver as definições do ambiente e analisar as seções de código, linha a linha, sem ter que executar todo o

código.

**Dica** Você também pode suspender a execução de um programa na janela **Rastrear** pressionando ESC.


## Suspendendo a execução em uma linha de código

Você pode suspender a execução do programa definindo pontos de interrupção de diversas maneiras. Se souber o local exato onde deseja suspender a execução do programa, poderá definir um ponto de interrupção diretamente nessa linha de código.

### ► Para definir um ponto de interrupção em uma determinada linha de código

Na janela **Rastrear**, localize a linha de código na qual deseja definir o ponto de interrupção e utilize um dos procedimentos a seguir:

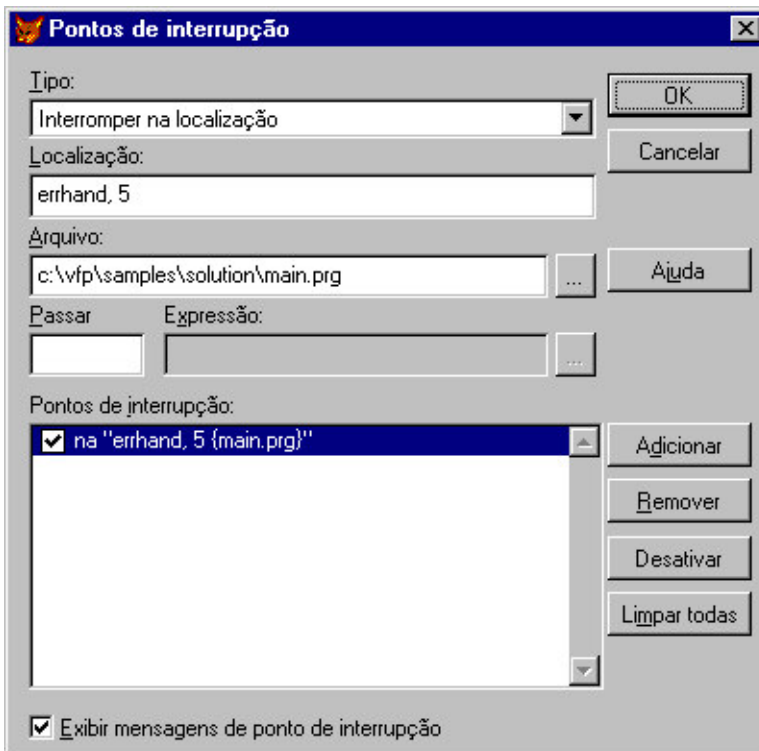
- 1 Posicione o cursor na linha de código.
- 2 Pressione F9 ou clique sobre o botão **Alternar ponto de interrupção** na **Barra de ferramentas depurador**.
  - Ou –
  - Clique duas vezes sobre a área cinza à esquerda da linha de código.

Um ponto sólido  será exibido na área cinza à esquerda da linha de código para indicar que um ponto de interrupção foi definido nesta linha.

**Dica** Se você estiver depurando objetos, poderá localizar determinadas linhas de código na janela **Rastrear**, escolhendo o objeto na lista Objeto e o método ou evento na lista Procedimento.

Além disso, você pode definir pontos de interrupção especificando localizações e arquivos na caixa de diálogo **Pontos de interrupção**.

### Interrompendo em uma localização



### Exemplos de localizações e arquivos para pontos de interrupção

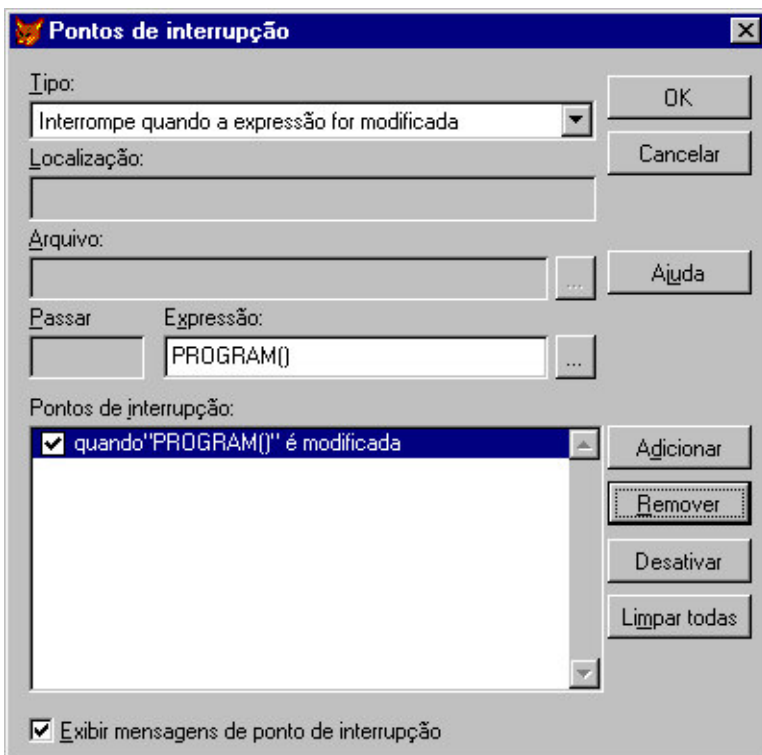
Localização	Arquivo	Onde a execução será
-------------	---------	----------------------

<b>suspensa</b>		
ErrorHandler	C:\Myapp\Main.prg	A primeira linha executável em um procedimento chamada <code>ErrorHandler</code> em <code>MAIN.PRG</code> .
Main, 10	C:\Myapp\Main.prg	A décima linha no programa chamada <code>Main</code> .
Click	C:\Myapp\Form.scx	A primeira linha executável de qualquer procedimento, função, método ou evento chamada <code>Click</code> em <code>FORM.SCX</code> .
cmdNext.Click	C:\Myapp\Form.scx	A primeira linha executável associada ao evento <code>Click</code> de <code>cmdNext</code> em <code>Form.scx</code> .
cmdNext::Click		A primeira linha executável no evento <code>Click</code> de qualquer controle cuja ClassePai é <code>cmdNext</code> em qualquer arquivo.

## Suspendendo a execução quando os valores forem alterados

Se você deseja saber quando o valor de uma variável ou propriedade será alterado ou quando uma condição em tempo de execução será alterada, poderá definir um ponto de interrupção em uma expressão.

### Interrompendo quando uma expressão for alterada



- Para suspender a execução do programa quando o valor de uma expressão for alterado

- 1 No menu **Ferramentas** na janela **Depurador**, escolha **Pontos de interrupção** para abrir a caixa de diálogo **Pontos de interrupção**.
- 2 Na lista **Tipo**, escolha **Interrompe quando a expressão for modificada**.
- 3 Insira a expressão na caixa **Expressão**.

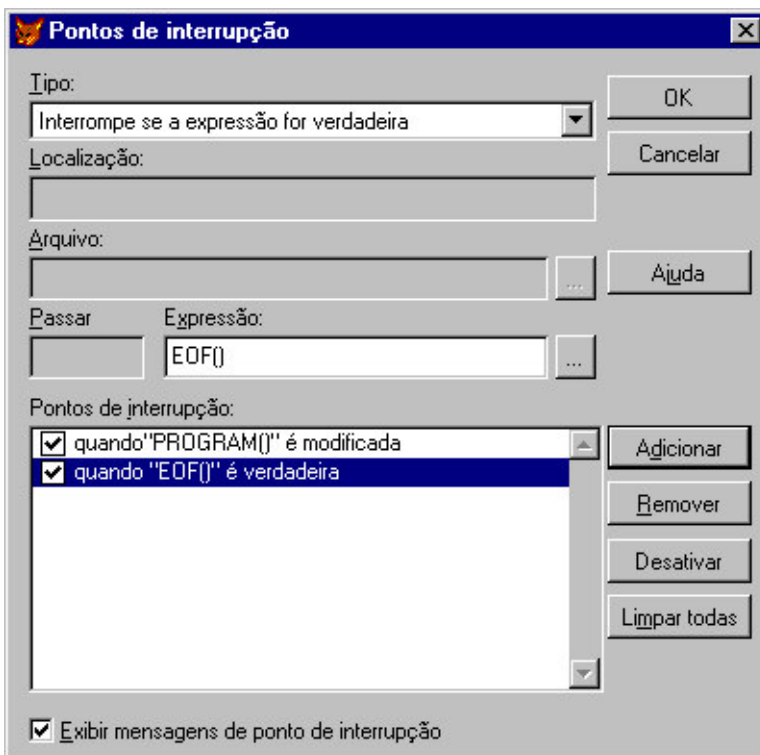
#### Exemplos de expressões de ponto de interrupção

Expressão	Utilize
RECNO ( )	Suspende a execução quando o ponteiro do registro se mover na tabela.
PROGRAM ( )	Suspende a execução na primeira linha de qualquer evento, método, procedimento ou programa novo.
myform.Text1.Value	Suspende a execução sempre que o valor desta propriedade for alterado de forma interativa ou por programação.

### Suspendendo a execução em modo condicional

Algumas vezes, você pode querer suspender a execução do programa, não em uma linha em especial, mas quando determinada condição for verdadeira.

#### Interrompendo em uma expressão



- Para suspender a execução do programa quando uma expressão for avaliada como verdadeira

- 1 No menu **Ferramentas** na janela **Depurador**, escolha **Pontos de interrupção** para abrir a caixa de diálogo **Pontos de interrupção**.
- 2 Na lista **Tipo**, escolha **Interrompe se a expressão for verdadeira**.

3 Insira a expressão na caixa **Expressão**.

4 Escolha **Adicionar** para adicionar o ponto de interrupção à lista **Pontos de interrupção**.

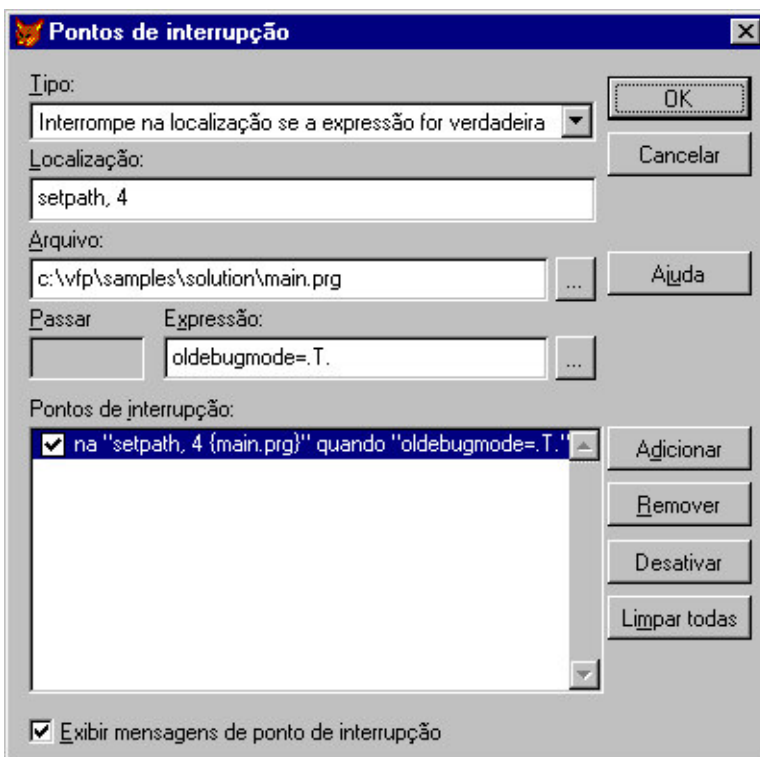
#### Exemplos de expressões de ponto de interrupção

Expressão	Utilize
EOF ( )	Suspende a execução quando o ponteiro do registro mover depois do último registro em uma tabela.
'CLICK' \$PROGRAM ( )	Suspende a execução na primeira linha de código associada a um evento Click ou DblClick.
nReturnValue = 6	Se o valor de retorno de uma caixa de mensagem estiver armazenado em nReturnValue, suspende a execução quando um usuário escolher <b>Sim</b> na caixa de mensagem.

### Suspendendo a execução em modo condicional em uma linha de código

Você só poderá especificar que a execução do programa será suspensa em uma determinada linha quando uma condição for verdadeira.

#### Interrompendo quando uma expressão for verdadeira



#### ► Para suspender a execução do programa em uma determinada linha quando uma expressão for avaliada como verdadeira

- 1 No menu **Ferramentas** na janela **Depurador**, escolha **Pontos de interrupção** para abrir a caixa de diálogo **Pontos de interrupção**.
- 2 Na lista **Tipo**, escolha **Interrompe na localização se a expressão for verdadeira**.
- 3 Insira a localização na caixa **Localização**.



- 4 Insira a expressão na caixa **Expressão**.
- 5 Escolha **Adicionar** para adicionar o ponto de interrupção à lista **Pontos de interrupção**.
- 6 Escolha **OK**.

**Dica** Às vezes, é mais fácil localizar a linha de código na janela **Rastrear**, definir um ponto de interrupção e, em seguida, editar esse ponto de interrupção na caixa de diálogo **Pontos de interrupção**. Para tal, altere o **Tipo** de **Interrompe na localização** para **Interrompe na localização se a expressão for verdadeira** e depois adicione a expressão.

## Removendo pontos de interrupção

É possível desativar pontos de interrupção sem removê-los na caixa de diálogo **Pontos de interrupção**. Você pode excluir pontos de interrupção do tipo “interrompe na localização” na janela **Rastrear**.

### ► Para remover um ponto de interrupção de uma linha de código

Na janela **Rastrear**, localize o ponto de interrupção e utilize um dos procedimentos a seguir:

- Posicione o cursor na linha de código e, em seguida, escolha **Alternar ponto de interrupção** na **Barra de ferramentas depurador**.
  - Ou –
- Clique duas vezes sobre a área cinza à esquerda da linha de código.

## Vendo os valores armazenados

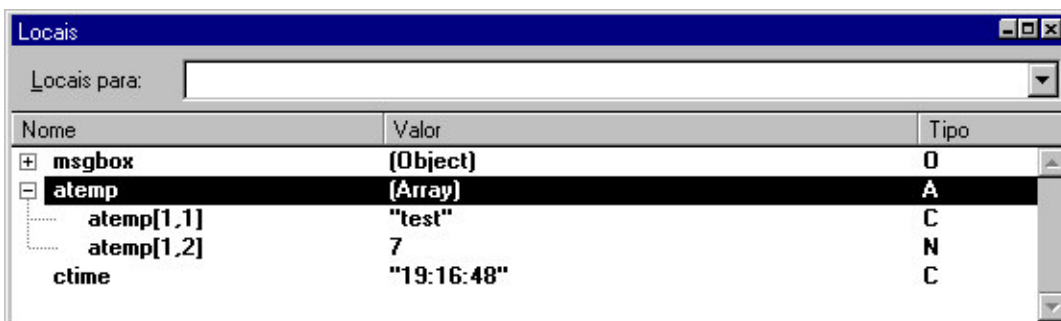
Na janela **Depurador**, você pode ver com facilidade os valores em tempo de execução de variáveis, elementos de matriz, propriedades e expressões nas janelas a seguir:

- Janela **Locais**
- Janela **Observar**
- Janela **Rastrear**

## Vendo os valores armazenados na janela Locais

A janela **Locais** exibe todas as variáveis, as matrizes, os objetos e os membros do objeto visíveis em qualquer programa, procedimento ou método na pilha de chamada. Como padrão, os valores do programa em execução no momento são exibidos na janela **Locais**. Você pode ver esses valores para outros programas ou procedimentos na pilha de chamada, escolhendo os programas ou procedimentos na lista **Locais para**.

### Janela Locais



Você pode analisar matrizes ou objetos clicando sobre o sinal de adição (+) localizado ao lado do nome do objeto ou da matriz nas janelas **Locais** e **Observar**. Durante a análise, você poderá ver os valores de todos os elementos nas matrizes e todas as definições de propriedade nos objetos.

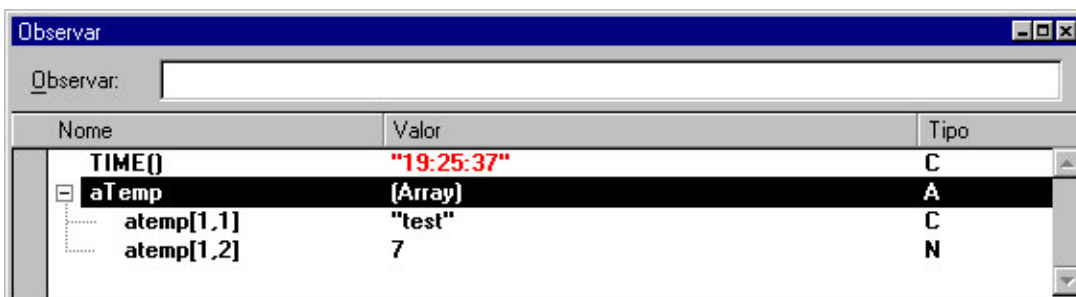
Você pode até alterar os valores em variáveis, elementos de matriz e propriedades nas janelas

**Locais e Observar**, selecionando a variável, o elemento de matriz ou a propriedade, clicando sobre a coluna Valor e digitando um novo valor.

## Vendo os valores armazenados na janela Observar

Na caixa **Observar** da janela **Observar**, digite qualquer expressão válida do Visual FoxPro e pressione ENTER. O valor e o tipo da expressão será exibido na lista da janela **Observar**.

### Janela Observar



Nome	Valor	Tipo
TIME()	"19:25:37"	C
aTemp	(Array)	A
atemp[1,1]	"test"	C
atemp[1,2]	7	N

**Observação** Você não pode inserir expressões na janela **Observar** que criem objetos.

Você também pode selecionar variáveis ou expressões na janela **Rastrear** ou em outras janelas de depuração e arrastá-las para a janela **Observar**.

O valores que foram alterados são exibidos em vermelho na janela **Observar**.

### ► Para remover um item da lista da janela Observar

Selecione o item e utilize um dos procedimentos a seguir:

- Pressione DEL.
  - Ou –
- No menu de atalho, escolha **Excluir observação**.

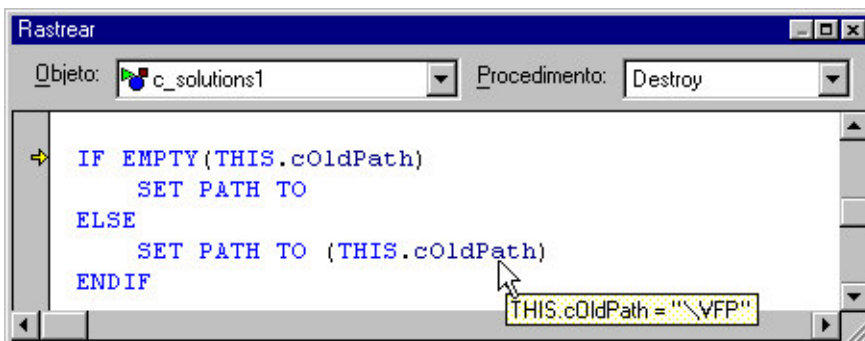
### ► Para editar uma observação

- Clique duas vezes sobre a observação na janela **Observar** e edite no local.

## Vendo os valores armazenados na janela Rastrear

Posicione o cursor sobre qualquer variável, elemento de matriz ou propriedade na janela **Rastrear** para exibir o seu valor atual em uma dica de valor.

### Uma dica de valor na janela Rastrear



## Exibindo a saída

O comando **DEBUGOUT** permite que você escreva valores na janela **Depurar saída** para um

registro de arquivo de texto. De forma alternativa, você pode usar o comando `SET DEBUGOUT TO` ou a guia **Depurar** da caixa de diálogo **Opções**.

Se você não estiver escrevendo comandos DEBUGOUT em um arquivo de texto, a janela **Depurar saída** deve ficar aberta para que os valores DEBUGOUT sejam escritos. A linha de código a seguir é impressa na janela **Depurar saída** ao mesmo tempo em que a linha de código é executada:

```
DEBUGOUT DATETIME ( )
```

Além disso, você pode ativar o controle de eventos, descrito anteriormente neste capítulo, e optar pela exibição do nome e dos parâmetros de cada evento ocorrido na janela **Depurar saída**.

## Registrando alcance de código

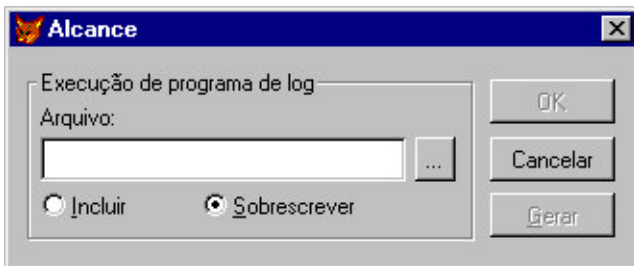
Posteriormente, no processo de desenvolvimento, é possível que você queira aprimorar o código para o desempenho e assegurar que o teste do código tenha sido executado devidamente, através do registro das informações de alcance de código.

O alcance de código fornece informações sobre as linhas de código que foram executadas e o tempo necessário para executá-las. Essas informações podem ajudá-lo a identificar áreas de código que não estão sendo executadas e, portanto, não estão sendo testadas, assim como áreas de código para as quais você talvez quera um ajuste perfeito para o desempenho.



É possível alternar o alcance de código entre ativo e desativado, clicando sobre o botão da **Barra de ferramentas alcance de código** na janela **Depurador**. Se você ativar o alcance de código, a caixa de diálogo **Alcance** abrirá para que você possa especificar um arquivo no qual salvar as informações de alcance.

### Caixa de diálogo Alcance



Você também pode alternar o registro de alcance entre ativo e desativado por programação, usando o comando `SET COVERAGE TO`. Por exemplo, você poderia incluir o comando a seguir em seu aplicativo antes de uma parte do código que você deseja investigar:

```
SET COVERAGE TO mylog.log
```

Após a seção de código para a qual você deseja registrar o alcance, poderia incluir o comando a seguir para desativar o alcance de código:

```
SET COVERAGE TO
```

Após ter especificado um arquivo para as informações de alcance, alterne para a janela principal do Visual FoxPro e execute o programa, formulário ou aplicativo. Para cada linha de código executada, serão escritas as informações a seguir no arquivo de registro:

- Quantos segundos a linha demora para ser executada.
- A classe à qual o código pertence (se houver).
- O método ou procedimento onde a linha de código está localizada.
- O número da linha de código.
- O arquivo onde o código está localizado.

A forma mais fácil de extrair informações do arquivo de registro é convertê-lo em uma tabela para que você possa definir filtros, executar consultas e relatórios, executar comandos e manipular a tabela de outras formas.

O programa a seguir converte o arquivo de texto criado pelo registro de alcance em uma tabela:

```
cFileName = GETFILE('DBF')
IF EMPTY(cFileName)
    RETURN
ENDIF

CREATE TABLE (cFileName) ;
    (duration n(7,3), ;
    class c(30), ;
    procedure c(60), ;
    line i, ;
    file c(100))

APPEND FROM GETFILE('log') TYPE DELIMITED
```

## Gerenciando erros em tempo de execução

Os erros em tempo de execução ocorrem após o início da execução do aplicativo. As ações que gerariam erros em tempo de execução incluem: escrever em um arquivo que não existe; tentar abrir uma tabela que já está aberta, tentar selecionar uma tabela que foi fechada; encontrar um conflito de dados, dividir um valor por zero e etc.

Os comandos e funções a seguir são úteis ao prever e gerenciar erros em tempo de execução.

Para	Utilize
Preencher uma matriz com informações de erro	<a href="#">AERROR( )</a>
Abrir a janela <b>Depurador</b> ou <b>Rastrear</b>	<a href="#">DEBUG</a> ou <a href="#">SET STEP ON</a>
Gerar um erro específico para testar o gerenciamento de erros	<a href="#">ERROR</a>
Retornar um número de erro	<a href="#">ERROR( )</a>
Retornar uma linha de programa em execução	<a href="#">LINENO( )</a>
Retornar uma sequência de mensagem de erro	<a href="#">MESSAGE( )</a>
Executar um comando quando um erro ocorrer	<a href="#">ON ERROR</a>
Retornar comandos atribuídos a comandos de gerenciamento de erros	<a href="#">ON( )</a>
Retornar o nome do programa em execução no momento	<a href="#">PROGRAM( )</a> OU <a href="#">SYS(16)</a>
Executar novamente o comando anterior	<a href="#">RETRY</a>
Retornar qualquer parâmetro de mensagem de erro atual	<a href="#">SYS(2018)</a>

## Prevendo erros

A primeira linha de defesa contra erros em tempo de execução é prever o local onde eles podem ocorrer e codificá-los. Por exemplo, a linha de código a seguir move o ponteiro do registro para o próximo registro na tabela.

```
SKIP
```

Este código funciona a menos que o ponteiro do registro já tenha passado pelo último registro na

tabela, no momento em que um erro ocorrer.

As linhas de código a seguir prevêm este erro e o evitam:

```
IF !EOF()
    SKIP
    IF EOF()
        GO BOTTOM
    ENDIF
ENDIF
```

Como outro exemplo, a linha de código a seguir exibe a caixa de diálogo **Abrir** para permitir que um usuário abra uma tabela em uma área de trabalho nova:

```
USE GETFILE('DBF') IN 0
```

O problema é que o usuário poderia escolher **Cancelar** na caixa de diálogo **Abrir** ou digitar o nome de um arquivo que não existe. O código a seguir prevê isto, certificando-se da existência do arquivo antes que o usuário tente usá-lo:

```
cNewTable = GETFILE('DBF')
IF FILE(cNewTable)
    USE (cNewTable) IN 0
ENDIF
```

É possível também que o usuário digite o nome de um arquivo que não esteja em uma tabela do Visual FoxPro. Para contornar este problema, você pode abrir o arquivo com as funções de E/S do arquivo de nível mínimo, analisar o cabeçalho binário e certificar-se de que o arquivo seja de fato uma tabela válida. No entanto, a execução deste procedimento seria muito trabalhosa e tornaria nitidamente o seu aplicativo mais lento. Você poderia lidar melhor com a situação em tempo de execução, exibindo uma mensagem como “Abra outro arquivo. Este arquivo não é uma tabela quando o erro 15, “Não é uma tabela” ocorrer.

Você não pode, e talvez não queira prever todos os erros possíveis. Sendo assim, você precisará localizar alguns erros escrevendo o código a ser executado no evento de um erro em tempo de execução.

## Gerenciando erros de procedimento

Quando um erro ocorrer no código de procedimento, o Visual FoxPro verificará o código de gerenciamento de erros associado a uma rotina [ON ERROR](#). Se nenhuma rotina ON ERROR existir, o Visual FoxPro exibirá a mensagem de erro padrão do Visual FoxPro. Para obter uma lista completa das mensagens de erro e dos números de erro do Visual FoxPro, consulte a **Ajuda**.

### Criando uma rotina ON ERROR

Você pode incluir qualquer expressão ou comando válido do FoxPro após ON ERROR, mas normalmente você chama um programa ou procedimento de gerenciamento de erros.

Para ver como ON ERROR funciona, você pode digitar um comando não reconhecido na janela **Comando**, como:

```
qxy
```

Você receberá uma caixa de diálogo de mensagem de erro padrão do Visual FoxPro “Verbo de comando não reconhecido “. Mas se você executar as linhas de código a seguir, verá o número de erro, 16, impresso na janela de saída ativa em vez da mensagem de erro padrão exibida em uma caixa de diálogo:

```
ON ERROR ?ERROR()
```

```
qxy
```

Emitir ON ERROR sozinho redefinirá a mensagem de erro interna do Visual FoxPro:

```
ON ERROR
```

Organizado em forma estrutural, o código a seguir ilustra um manipulador de erros ON ERROR:

```

LOCAL lcOldOnError

* Salva o manipulador de erros original
lcOldOnError = ON("ERROR")

* Emite ON ERROR com o nome de um procedimento
ON ERROR DO errhandler WITH ERROR(), MESSAGE()

* código ao qual a rotina de gerenciamento de erros se aplica

* Redefine o manipulador de erros original
ON ERROR &lcOldOnError

PROCEDURE errhandler
LOCAL aErrInfo[1]
AERROR(aErrInfo)
DO CASE
CASE aErrInfo[1] = 1 && O Arquivo não existe
* exibe uma mensagem adequada
* e executa uma ação para reparar o problema.
OTHERWISE
* exibe uma mensagem genérica, talvez
* envia mensagem de alta prioridade para um administrador
ENDPROC

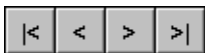
```

## Gerenciando erros em classes e objetos

Quando ocorre um erro no código de método, o Visual FoxPro verifica o código de gerenciamento de erros associado ao evento Error do objeto. Se nenhum código foi escrito no nível de objeto do evento Error, o código de evento Error herdado da classe pai, ou de outra classe acima da hierarquia de classes, será executado. Se nenhum código foi escrito para o evento Error em qualquer local da hierarquia de classes, o Visual FoxPro verifica uma rotina ON ERROR. Se nenhuma rotina ON ERROR existir, o Visual FoxPro exibirá a mensagem de erro padrão do Visual FoxPro.

O interessante das classes é que você pode encapsular tudo que um controle precisa, incluindo o gerenciamento de erros, tendo a possibilidade de usar o controle em uma variedade de ambientes. Posteriormente, se você descobrir outro erro encontrado pelo controle, pode adicionar o gerenciamento desse erro à classe, e todos os objetos com base em sua classe herdarão automaticamente o novo gerenciamento de erros.

Por exemplo, a classe `vcr` na biblioteca de classes `BUTTONS.VCX`, localizada em `VFP\SAMPLES\CLASSES\`, baseia-se na classe recipiente do Visual FoxPro.



Quatro botões de comando no recipiente gerenciam a navegação pela tabela, movendo o ponteiro do registro em uma tabela com os comandos a seguir:

```

GO TOP
SKIP - 1
SKIP 1
GO BOTTOM.

```

Pode ocorrer um erro se o usuário escolher um dos botões e nenhuma tabela estiver aberta. O Visual FoxPro tenta escrever valores de buffer em uma tabela quando o ponteiro do registro for movido. Portanto, também pode ocorrer um erro se o buffer de linha otimista estiver ativado e outro usuário tiver alterado um valor no registro de buffer.

Esses erros podem ocorrer se o usuário escolher qualquer um dos botões; conseqüentemente, não faz sentido ter quatro métodos de gerenciamento de erros separados. O código a seguir associado ao evento Error de cada um dos botões de comando passa informações de erro para uma única rotina de gerenciamento de erros da classe:

```
LPARAMETERS nError, cMethod, nLine
THIS.Parent.Error(nError, cMethod, nLine)
```

O código a seguir está associado ao evento Error da classe vcr. O código efetivo é diferente devido às exigências de codificação para localização.

```
Parameters nError, cMethod, nLine
DO CASE
CASE nError = 13 && Alias não encontrada
    cNewTable = GETFILE('DBF')
    IF FILE(cNewTable)
        SELECT 0
        USE (cNewTable)
        This.SkipTable = ALIAS()
    ELSE
        This.SkipTable = ""
    ENDIF
CASE nError = 1585 && Conflito de Dados
* Atualiza o conflito manipulado por uma classe de verificação de dados
    nConflictStatus = ;
    THIS.DataChecker1.CheckConflicts()
    IF nConflictStatus = 2
        MESSAGEBOX "Impossível resolver um conflito de dados."
    ENDIF
OTHERWISE
* Exibe informações sobre outros erros.

    cMsg="Error:" + ALLTRIM(STR(nError)) + CHR(13) ;
    + MESSAGE()+CHR(13)+"Program:"+PROGRAM()

    nAnswer = MESSAGEBOX(cMsg, 2+48+512, "Error")
DO CASE
    CASE nAnswer = 3 &&Aborta
        CANCEL
    CASE nAnswer = 4 &&Tenta novamente
        RETRY
    OTHERWISE && Ignora
        RETURN
ENDCASE
ENDCASE
```

Você deseja certificar-se de que fornece informações para um erro ainda não gerenciado. Caso contrário, o código de evento de erro será executado mas não realizará nenhuma ação e a mensagem de erro padrão do Visual FoxPro não será mais exibida. Você, assim como o usuário, não saberá o que aconteceu.

Dependendo dos usuários-alvo, talvez você queira fornecer maiores informações no caso de um erro ainda não gerenciado, como o nome e o telefone de uma pessoa para ajudá-lo.

## A partir do código de gerenciamento de erros

Depois da execução do código de gerenciamento de erros, a linha de código após a linha que causou o erro será executada. Se você desejar executar novamente a linha de código que causou o erro após ter alterado a situação que causou o erro, utilize o comando **RETRY**.

**Observação** O evento Error pode ser chamado quando o erro encontrado não estiver associado a uma linha do seu código. Por exemplo, se você chamar um método CloseTables do ambiente de dados no código quando AutoCloseTables for definido como verdadeiro (.T.) e, em seguida, liberar o formulário, um erro interno será gerado quando o Visual FoxPro tentar fechar as tabelas novamente. É possível localizar este erro, mas não há linha de código para RETRY.