Se o Visual FoxPro ainda não incluir os recursos exigidos para o aplicativo, você poderá estender suas capacidades criando um controle ActiveX (arquivo .OCX) ou uma biblioteca (arquivo .FLL) específica ao Visual FoxPro utilizando um compilador de 32 bites, como o Microsoft Visual C ou C++® versão 4.0. As informações deste capítulo se referem aos dois tipos de programas.

Observação Se você estiver utilizando o Visual C ou C++ versão 2.x para desenvolver um controle ActiveX, será necessário o Control Development Kit. Os procedimentos deste capítulo supõem o Visual C/C++ versão 4.0.

Para obter informações sobre a utilização de controles ActiveX ou FLLs, consulte o capítulo 27, Estendendo o Visual FoxPro com bibliotecas externas".

Este capítulo aborda os tópicos a seguir:

Criando bibliotecas ou controles ActiveX

É possível estender as capacidades do Visual FoxPro por meio da criação de programas em C ou C++ que realizam tarefas exigidas pelo aplicativo. Por exemplo, se o seu aplicativo exigir acesso direto aos recursos do Windows, você poderá gravar um programa C ou C++ que realiza chamadas à API do Windows e, em seguida, retorna informações ao Visual FoxPro.

Você pode criar dois tipos de programas para acessar a API do Visual FoxPro:

- Um controle ActiveX (arquivo .OCX).
- Uma DLL específica ao Visual FoxPro. Como a DLL somente poderá ser chamada a partir do Visual FoxPro, é comum nomeá-la com a extensão .FLL.

Cada tipo de programa apresenta vantagens. Um controle ActiveX:

- Pode ser acessado por técnicas orientadas a objetos padrão, como definir suas propriedades e chamar seus métodos.
- Pode ser dividido em subclasses, e seus métodos podem ser anulados.
- É encapsulado e pode ser chamado (instanciado) várias vezes sem gerenciamento de ambiente complexo para preservar estados de usuário.
- Apresenta passagem de parâmetros mais simples.
- Também pode ser chamado a partir de outros programas do Windows se for programado com essa intenção.

Por outro lado, uma biblioteca .FLL:

- Geralmente, é menor do que um controle ActiveX, exigindo menos espaço em disco e memória.
- Pode ser gravada em C. (os controles ActiveX devem ser gravados em C++.)
- Pode parecer familiar se você utilizou versões anteriores do Visual FoxPro.

Observação Caso queira utilizar uma biblioteca .FLL a partir de uma versão anterior do Visual FoxPro, a biblioteca deverá ser recompilada para funcionar com o Visual FoxPro versão 5.0.

Criando um controle ActiveX básico

Os controles ActiveX específicos ao Visual FoxPro são criados da mesma forma que qualquer controle semelhante. A maioria dos compiladores C++ permitem que você crie estruturas do controle.

Para criar um projeto para o controle ActiveX

1. Inicie o Microsoft Visual C++.
2. No menu **File**, escolha **New**.
3. Na caixa de diálogo **New**, escolha **Project Workspace**.
4. Na caixa de diálogo **New Project Workspace**, especifique um nome de projeto.
5. Na lista **Type**, escolha **OLE ControlWizard**.
6. Escolha **Create** e, em seguida, siga as etapas do assistente.

Quando o assistente for finalizado, você poderá construir um controle ActiveX imediatamente. No entanto, também será necessário definir propriedades e métodos para o controle.

Para adicionar propriedade e métodos ao controle ActiveX

1. No menu **View**, escolha **ClassWizard**.
2. Selecione a guia **OLEAutomation**.
3. Selecione **Add Method** ou **Add Property**.
4. Preencha o nome, parâmetro e outras informações exigidas pelo elemento que está sendo criado e, em seguida, escolha **OK**.
5. Selecione **Edit Code** para exibir o editor e, em seguida, digite o código que define a propriedade ou método que está sendo criado.

Por exemplo, para criar uma propriedade Version que retorne a versão do arquivo .OCX como um inteiro (como 101), deve-se criar a propriedade com um tipo de retorno `long` e adicionar código similar ao seguinte:

```
#define VERSION 101

long CPyCtrl::GetVersion()
{
   // define o número da versão aqui
   return VERSION;
}
```

Como o número da versão é normalmente somente para leitura, você não criaria uma função SetVersion( ).

Criando uma biblioteca FLL básica

Como uma biblioteca FLL é essencialmente uma DLL com chamadas à API do Visual FoxPro, ela pode ser criada seguindo as etapas em seu ambiente de desenvolvimento para a criação de uma DLL.

Para criar um projeto para a biblioteca FLL

1. Inicie o Microsoft Visual C/C++.
2. No menu **File**, selecione **New**.
3. Na caixa de diálogo **New**, selecione **Project Workspace**.
4. Na caixa de diálogo **New Project Workspace**, especifique um nome de projeto.
5. Na lista **Type**, escolha **Dynamic-link library**.

Depois de criar uma estrutura DLL básica, adicione as funções que você deseja chamar a partir do Visual FoxPro. As seções a seguir fornecem estruturas para a criação de funções em C e C++.

Definindo um modelo de biblioteca

Cada biblioteca de função criada tem a mesma estrutura básica. Utilizando um modelo para a

estrutura, tudo o que você deverá fazer é preencher as lacunas que se aplicam à sua rotina de biblioteca específica.

Há cinco elementos em um modelo de biblioteca do Visual FoxPro:

1. Instrução `#include`
2. Definição da função
3. Código da função
4. Estrutura FoxInfo
5. Estrutura FoxTable

Um modelo C de exemplo

É possível utilizar o modelo a seguir para criar bibliotecas gravadas em C.

```
#include <pro_ext.h>

void Internal_Name(ParamBlk *parm)
{
// código da função aqui.
}

FoxInfo myFoxInfo[] = {
   {"FUNC_NAME", (FPFI) Internal_Name, 0, ""},
};

FoxTable _FoxTable = {
   (FoxTable *)0, sizeof(myFoxInfo)/sizeof(FoxInfo), myFoxInfo
};
```

Um modelo C++ de exemplo

Para rotinas C++ você pode utilizar o modelo a seguir. Esse modelo difere do modelo C pois declara a estrutura `FoxTable` como externa.

```
#include <pro_ext.h>

void Internal_Name(ParamBlk  *parm)
{
// código da função aqui.
}

   FoxInfo myFoxInfo[] = {
      {"FUNC_NAME", (FPFI) Internal_Name, 0, ""},
   };

extern "C" {
   FoxTable _FoxTable = {
      (FoxTable *)0, sizeof(myFoxInfo)/sizeof(FoxInfo), myFoxInfo
   };
}
```

Utilizando o modelo

Para utilizar o arquivo principal e criar uma biblioteca compilada, é necessário:

- O arquivo principal PRO_EXT.H. Você pode imprimir esse arquivo para visualizar as declarações de funções, definições de tipos e estruturas utilizadas na API do Visual FoxPro.
- O arquivo WINAPIMS.LIB.

Ambos os arquivos são instalados no subdiretório API durante a instalação do Visual FoxPro.

A definição de função retorna `void` e espera o parâmetro a seguir: `ParamBlk *parm`. A estrutura

`ParamBlk` é abordada em Passando e recebendo parâmetros, posteriormente neste capítulo.

Além dos arquivos listados acima, os únicos outros elementos exigidos de uma biblioteca do Visual FoxPro são as estruturas FoxInfo e FoxTable.

Utilizando estruturas FoxInfo e FoxTable

As funções de biblioteca comunicam-se com o Visual FoxPro por meio da estrutura FoxInfo. A partir dessa estrutura, o Visual FoxPro determina o nome da função e o número e tipo dos parâmetros. A estrutura FoxTable é uma lista vinculada que controla as estruturas FoxInfo. Consulte PRO_EXT.H no diretório da API do Visual FoxPro para obter definições das estruturas FoxInfo e FoxTable.

Estrutura FoxInfo

A estrutura FoxInfo é o veículo utilizado para comunicar nomes de funções e descrições de parâmetros entre o Visual FoxPro e sua biblioteca. Uma estrutura FoxInfo genérica é mais ou menos assim:

```
FoxInfo nomematriz[ ] = {
      {funcNome1, FPFI função1, parmCont1, parmTipos1}
      {funcNome2, FPFI função2, parmCont2, parmTipos2}
         . . .
      {funcNomeN, FPFI funçãoN, parmContN, parmTiposN}
   };
```

Os marcadores de lugar são definidos como segue:

nomematriz Uma variável de tipo FoxInfo. Observe que é possível incluir várias linhas de estrutura FoxInfo nessa matriz.

funcNome Contém o nome (em maiúsculas e com até 10 caracteres) que o usuário do Visual FoxPro chama para recorrer à função.

função O endereço da rotina de linguagem C. Esse é o nome exato (considera maiúsculas/minúsculas) utilizado para definir a função.

parmCont Especifica o número de parâmetros descritos na seqüência *parmTypes* ou um dos valores de sinalização a seguir.

Valor	Descrição
INTERNAL	Especifica que a função não pode ser chamada diretamente do Visual FoxPro.
CALLONLOAD	Especifica que a rotina deve ser chamada quando a biblioteca for carregada. CALLONLOAD não pode chamar qualquer rotina que retorne resultados ao Visual FoxPro.
CALLONUNLOAD	Especifica que a rotina deve ser chamada quando a biblioteca for descarregada ou quando o comando **QUIT** do Visual FoxPro for emitido. CALLONUNLOAD não pode chamar qualquer rotina que retorne resultados ao Visual FoxPro.

parmTipos Descreve o tipo de dados de cada parâmetro. A tabela a seguir lista os valores válidos para *parmTipos.*

Valor	Descrição
`""`	Sem parâmetro
`"?"`	Qualquer tipo pode ser passado. No corpo da função, será necessário verificar o tipo do parâmetro passado.
`"C"`	Parâmetro tipo Caractere
`"D"`	Parâmetro tipo Data

"I"	Parâmetro tipo Inteiro
"L"	Parâmetro tipo Lógico
"N"	Parâmetro tipo Numérico
"R"	Referência
"T"	Parâmetro tipo DataHora
"Y"	Parâmetro tipo Moeda
"O"	Parâmetro tipo Objeto

Inclui um valor de tipo para cada parâmetro passado para a biblioteca. Por exemplo, se você criar uma função que aceita um parâmetro caractere e um numérico, substitua *parmType* por "CN".

Observação Para indicar que um parâmetro é opcional, preceda-o com um ponto. Apenas parâmetros à direita podem ser omitidos.

A estrutura `FoxInfo` a seguir define uma biblioteca com uma função —internamente denominada `dates` e externamente acessada como `DATES` — que aceita um parâmetro tipo Caractere:

```
FoxInfo myFoxInfo[] = {
   { "DATES", (FPFI) dates, 1, "C" }
};
```

Após compilar a biblioteca com essa estrutura FoxInfo e carregá-la no Visual FoxPro com o comando SET LIBRARY TO, será possível chamar essa função no Visual FoxPro com a linha de código a seguir:

```
=DATES("01/01/95")
```

Estrutura FoxTable

A estrutura FoxTable é uma lista vinculada que controla todas as estruturas FoxInfo de uma determinada biblioteca:

FoxTable _FoxTable = { *proxBiblio*, *infoCont*,*infoPtr*};
onde os marcadores de lugar são definidos como segue:

proxBiblio Um ponteiro utilizado internamente pelo Visual FoxPro; deve ser inicializado a 0.

infoCount O número de rotinas externas do Visual FoxPro definido nessa biblioteca.

infoPtr O endereço do primeiro elemento de uma matriz de estruturas FoxInfo. Esse nome deve corresponder ao nome da matriz listado na instrução FoxInfo.

A seguir, é apresentado um exemplo de uma instrução FoxTable. Se o nome da matriz FoxInfo for `myFoxInfo`, nunca será necessário alterar essa instrução.

```
FoxTable _FoxTable = {
   (FoxTable  *) 0,
   sizeof( myFoxInfo) / sizeof( FoxInfo ),
   myFoxInfo
};
```

Adicionando chamadas de API do Visual FoxPro

Para integrar seu programa ao Visual FoxPro, é possível chamar rotinas do Visual FoxPro API. Essas rotinas de API são funções que podem ser chamadas de qualquer programa C ou C++, inclusive um arquivo .OCX ou .FLL, que fornece acesso a variáveis, gerencia operações de banco de dados e realiza muitas outras tarefas específicas do Visual FoxPro.

A tabela a seguir lista as categorias gerais de chamadas de API disponíveis no Visual FoxPro. Para detalhes sobre funções de API individuais, consulte Rotinas externas A-Z ou Rotinas externas por categoria.

Categorias de rotinas de API do Visual FoxPro

Debugging	Memo Field Input/Output	Selection e Clipboard
Dialog	Memory e String Handling	Statement e Expression

Editor Environment	Memory Management	Streaming Output
Error Handling	Variable e Array	Table Input/Output
File Input/Output	Menu	Undo e Redo
Manipulation	Navigation	User Interface
File e Window Management	Returning Results to Visual FoxPro	Windowing Output

Para utilizar as rotinas de API do Visual FoxPro, é necessário incluir o arquivo PRO_EXT.H, disponível no diretório API do Visual FoxPro API. Esse arquivo inclui os protótipos das funções e estruturas que permitem o compartilhamento de informações com o Visual FoxPro.

Se estiver gravando um controle ActiveX, você também deve adicionar chamadas para inicializar e limpar a API.

Para adicionar rotinas de API do Visual FoxPro ao controle ActiveX

1. Utilize `#INCLUDE` para incluir o arquivo PRO_EXT.H com quaisquer outros arquivos principais exigidos.
2. No **Construtor do controle** (o método Init), chame _OCXAPI( ) para inicializar a interface com o Visual FoxPro utilizando este código:
   ```
   _OCXAPI(AfxGetInstanceHandle(),DLL_PROCESS_ATTACH);
   ```
3. Inclua chamadas à API do Visual FoxPro como exigido no controle.
4. No **Destrutor do controle** (método Destroy), chame _OCXAPI( ) novamente para liberar o processo criado no Construtor utilizando este código:
   ```
   _OCXAPI(AfxGetInstanceHandle(),DLL_PROCESS_DETACH);
   ```

Para um arquivo de exemplo .OCX que inclua chamadas à API do Visual FoxPro, consulte FOXTLIB.OCX em VFP\API\SAMPLES. Para um exemplo de uma biblioteca .FLL que inclua chamadas à API do Visual FoxPro, consulte os programas de exemplo em VFP\API\SAMPLES que têm a extensão C: EVENT.C, HELLO.C, e assim por diante.

Caso utilize chamadas de API do Visual FoxPro no controle ActiveX ou biblioteca .FLL, o código que contém as chamadas será incompatível com outros aplicativos. Em seguida, talvez você queira criar um ou mais testes no programa para determinar se o controle está sendo chamado do Visual FoxPro.

Por exemplo, se você estiver criando um controle ActiveX, poderá alterar o código do **Construtor do controle** para incluir um teste e, em seguida, avisar ao usuário se o controle foi chamado de um programa diferente do Visual FoxPro:

```
CFoxtlibCtrl::CFoxtlibCtrl()
{
   InitializeIIDs(&IID_DFoxtlib, &IID_DFoxtlibEvents);
   if (!_OCXAPI(AfxGetInstanceHandle(),DLL_PROCESS_ATTACH))
   {
      ::MessageBox(0,"Esse controle somente pode ser utilizado no Visual FoxPro","",0);
   }
}
```

Nesse exemplo, o controle não sai e continuará executando depois que o usuário tiver reconhecido a mensagem. A estratégia escolhida depende de como o controle a ser utilizado é antecipado. Por exemplo, se você detectar que o controle está sendo utilizado fora do Visual FoxPro, poderá definir um sinalizador que é testado em cada ponto no controle onde a API do Visual FoxPro for chamada. Se o sinalizador indicar que o controle está fora do Visual FoxPro, será possível desviar pela chamada da API para um meio alternativo de realizar a mesma tarefa.

Passando e recebendo parâmetros

Quando o programa é chamado a partir do Visual FoxPro, ele pode receber parâmetros. Por

exemplo, um controle ActiveX pode receber parâmetros quando um de seus métodos for chamado. De maneira semelhante, um programa do Visual FoxPro pode chamar uma função na biblioteca .FLL e passar parâmetros para ela.

O Visual FoxPro pode passar parâmetros ao seu programa por valor ou por referência. Como padrão, os parâmetros respeitam a definição feita com SET UDFPARMS. Outras variáveis (como matrizes ou campos) e expressões são passadas por valor.

Para forçar um parâmetro a ser passado por referência, anteceda a referência da variável com o operador @. Para forçar um parâmetro a ser passado por valor, coloque-o entre parênteses.

Observação No Visual FoxPro, os elementos individuais de matriz são sempre passados por valor. Quando SET UDFPARMS é definido como VALUE e nenhum elemento de matriz é especificado, o nome da matriz se refere ao primeiro elemento da matriz (a menos que seja precedido por @).

Como os controles ActiveX são programas padrão do Windows, nenhum mecanismo especial é exigido para passar parâmetros do Visual FoxPro e do seu programa. É possível gravar o programa como se ele estivesse recebendo parâmetros de qualquer programa C ou C++.

Em contraste, as funções em uma biblioteca FLL utilizam a estrutura FoxInfo para receber dados do Visual FoxPro. A estrutura FoxInfo lista as funções de biblioteca, bem como o número e tipo de parâmetros esperados por elas. Por exemplo, a estrutura `FoxInfo` a seguir pertence a uma biblioteca com uma única função, internamente denominada dates, que aceita um parâmetro do tipo Caractere:

```
FoxInfo myFoxInfo[] = {
   { "DATES", (FPFI) dates, 1, "C" }
};
```

As funções definidas nas suas bibliotecas na verdade recebem apenas um parâmetro, um ponteiro para o bloco de parâmetros. Esse bloco de parâmetros, definido na estrutura `ParamBlk`, armazena todas as informações sobre os parâmetros que foram passados a partir da chamada de função do Visual FoxPro. Sua declaração de função segue este formato:

void *nome_função*(ParamBlk **parm*)
Por exemplo, a definição de função para dates é:

```
void dates(ParamBlk *parm)
```

A estrutura `ParamBlk` consiste em um inteiro que representa o número de parâmetros, seguido imediatamente de uma matriz de uniões de parâmetros. A definição da estrutura está incluída em PRO_EXT.H:

```
/* Uma lista de parâmetros para uma função de biblioteca.       */
typedef struct {
   short int pCount;     /* número de parâmetros passados */
   Parameter p[1];          /* parâmetros pCount */
} ParamBlk;
```

A definição de tipo `Parameter` incluída na estrutura `ParamBlk` é uma união das estruturas Value e Locator. A chamada por valor é gerenciada por uma estrutura Value; a chamada por referência é gerenciada por uma estrutura Locator. Você utiliza essas estruturas para acessar os parâmetros passados para sua função quando ela for chamada no Visual FoxPro.

As informações a seguir são extraídas do arquivo `PRO_EXT.H` e mostram a definição do tipo `Parameter`.

```
/* Um parâmetro para uma função de biblioteca.            */
typedef union {
   Value val;
   Locator loc;
} Parameter;
```

Definição da estrutura Value

Caso um parâmetro seja passado para uma função por valor, utilize a estrutura Value para

acessá-lo. A definição da estrutura `Value` a seguir é extraída do arquivo PRO_EXT.H.

```
// Valor de uma expressão.
Typedef struct {
   char         ev_type;
   char         ev_padding;
   short        ev_width;
   unsigned     ev_length;
   long         ev_long;
   double          ev_real;
   CCY          ev_currency;
   MHANDLE      ev_handle;
   ULONG        ev_object;
} Value;
```

Campos da estrutura Value

A tabela a seguir é um guia para os valores que você pode passar e receber na estrutura Value para diferentes tipos de dados. Apenas os campos de estrutura listados para um tipo de dados são utilizados para aquele tipo de dados.

Conteúdo da estrutura Value para diferentes tipos de dados

Tipo de dados	Campo da estrutura	Valor
Caractere	`ev_type`	'C'
	`ev_length`	comprimento da seqüência
	`ev_handle`	MHANDLE para a seqüência
Numérico	`ev_type`	'N'
	`ev_width`	Largura de exibição
	`ev_length`	Casas decimais
	`ev_real`	Precisão dupla
Inteiro	`ev_type`	'I'
	`ev_width`	Largura de exibição
	`ev_long`	Inteiro longo
Data	`ev_type`	'D'
	`ev_real`	Data1
Data Hora	`ev_type`	'T'
	`ev_real`	Data + (segundos/86400.0)
Moeda	`ev_type`	'Y'
	`ev_width`	Largura de exibição
	`ev_currency`	Moeda valor2
Lógico	`ev_type`	'L'
	`ev_length`	0 ou 1
Memo	`ev_type`	'M'
	`ev_width`	FCHAN
	`ev_long`	Comprimento do campo Memo
	`ev_real`	Deslocamento do campo Memo
Geral	`ev_type`	'G'
	`ev_width`	FCHAN

	ev_long	Comprimento do campo Geral
	ev_real	Deslocamento do campo Geral
Objeto	ev_type	'O'
	ev_object	Identificador do objeto
Nulo	ev_type	'0' (zero)
	ev_long	Tipo de dados

1 A data é representada como um número de dia juliano de ponto flutuante de dupla precisão calculado utilizando-se o Algoritmo 199 de Collected Algorithms do ACM.

2 O valor de moeda é um inteiro longo, com um ponto decimal indicado na frente dos últimos quatro dígitos.

Observação ev_length é o único indicador verdadeiro do comprimento de uma seqüência. A seqüência não pode apresentar um finalizador nulo, pois pode conter caracteres nulos incorporados.

Definição da estrutura Locator

Utilize a estrutura Locator para manipular os parâmetros passados por referência. A definição da estrutura Locator a seguir é extraída do arquivo PRO_EXT.H.

```
typedef struct {
  char  l_type;
  short l_where, /* Número do banco de dados ou -1 para memória */
      l_NTI,     /* Deslocamento da tabela de nomes de variáveis */
      l_offset,  /* Índice no banco de dados*/
      l_subs,    /* n°. de subscritos especificados: 0 <= x <= 2 */
      l_sub1, l_sub2; /* valores integrais do subscrito */
} Locator;
```

Campos da estrutura Locator

A tabela a seguir é um guia para os campos da estrutura Locator.

Campos da estrutura Locator

Campo de Locator	Utilização do campo
l_type	'R'
l_where	O número da tabela que contém esse campo ou -1 para uma variável.
l_NTI	Índice de tabela de nomes. Utilização interna do Visual FoxPro.
l_offset	Número do campo dentro da tabela. Utilização interna do Visual FoxPro.
l_subs	Apenas para variáveis, o número de subscritos (0–2).
l_sub1	Apenas para variáveis, o primeiro subscrito se l_subs não for 0.
l_sub2	Apenas para variáveis, o segundo subscrito se l_subs for 2.

Observação É recomendável verificar o tipo de parâmetro em ev_type para ajudar a determinar quais campos deverão ser acessados a partir da estrutura Value.

Um exemplo de parâmetros de acesso em uma biblioteca FLL

O exemplo a seguir utiliza _StrCpy( ) para retornar um tipo Caractere ao Visual FoxPro que é a concatenação dos seus dois parâmetros de Caractere. Observe que, embora o identificador da estrutura Value de cada parâmetro seja utilizado como memória de trabalho para executar a concatenação, as alterações nessa alocação de memória não afetam o argumento do Visual FoxPro que foi passado por valor.

Para obter um exemplo que utiliza a estrutura Locator para gerenciar um parâmetro passado por referência, consulte Retornando um valor de uma biblioteca FLL, posteriormente neste capítulo.

```
#include <pro_ext.h>

Example(ParamBlk *parm)
{
// facilita o gerenciamento da estrutura
// paramBlk utilizando atalhos #define
#define p0 (parm->p[0].val)
#define p1 (parm->p[1].val)

// assegura que há memória suficiente
if (!_SetHandSize(p0.ev_handle, p0.ev_length + p1.ev_length))
   _Error(182); // "Memória insuficiente"

// bloqueia os identificadores
_HLock(p0.ev_handle);
_HLock(p1.ev_handle);

// converte identificadores em ponteiros e assegura que
// as seqüências são finalizadas em nulos
((char *)_HandToPtr(p0.ev_handle))[p0.ev_length] = '\0';
((char *)_HandToPtr(p1.ev_handle))[p1.ev_length] = '\0';

// concatena as seqüências utilizando a função _StrCpy da API
_StrCpy((char *)_HandToPtr(p0.ev_handle) + p0.ev_length,
_HandToPtr(p1.ev_handle));

// retorna a seqüência concatenada para o Visual FoxPro
_RetChar(_HandToPtr(p0.ev_handle));

// desbloqueia os identificadores
_HUnLock(p0.ev_handle);
_HUnLock(p1.ev_handle);
}

FoxInfo myFoxInfo[] = {
   {"STRCAT", Example, 2, "CC"},
};

FoxTable _FoxTable = {
   (FoxTable *) 0, sizeof(myFoxInfo)/sizeof(FoxInfo), myFoxInfo
};
```

Retornando valores ao Visual FoxPro

O método utilizado para retornar um valor do seu programa ao Visual FoxPro depende de se está sendo criado um controle ActiveX ou uma biblioteca FLL.

Retornando um valor de um controle ActiveX

Para retornar um valor do controle ActiveX ao Visual FoxPro, utilize a instrução RETURN no controle passando um único valor, como no exemplo a seguir:

```
#define VERSION 101

// outros códigos aqui

long CPyCtrl::GetVersion()
{
   // define o número da versão na variável fVersion
```

```
    return VERSION;
}
```

Retornando um valor de uma biblioteca FLL

Para retornar valores de uma biblioteca FLL, utilize funções da API e não comandos nativos C ou C++. As funções a seguir permitem que você retorne valores ao Visual FoxPro.

Observação Não utilize a função da API a seguir para retornar um valor de um arquivo OCX; utilize a instrução RETURN. As funções de retorno da API somente devem ser utilizadas em bibliotecas FLL.

Função	Descrição
_RetChar(char *_string_))	Define o valor de retorno da função como uma seqüência finalizada com valor nulo.
_RetCurrency(CCY _cval_, int _width_)	Define o valor de retorno da função como um valor de moeda.
_RetDateStr(char *_string_)	Define o valor de retorno da função como uma data. A data é especificada no formato dd/mm/aa[aa].
_RetDateTimeStr(char *_string_))	Define o valor de retorno da função como uma data e hora especificadas no formato dd/mm/aa[aa] hh:mm:ss.
_RetFloat(double _flt_, int _width_, int _dec_)	Define o valor de retorno da função como um valor flutuante.
_RetInt(long _ival_, int _width_)	Define o valor de retorno da função como um valor numérico.
_RetLogical(int _flag_)	Define o valor de retorno da função como um valor lógico. Zero é considerado FALSE. Qualquer valor diferente de zero é considerado TRUE.
_RetVal(Value *val)	Passa uma estrutura Value completa do Visual FoxPro; qualquer tipo de dado do Visual FoxPro, exceto memo, pode ser retornado. Você deve chamar _RetVal( ) para retornar uma seqüência que contenha caracteres nulos incorporados ou para retornar um valor .NULL.

Observação Para retornar o valor de um tipo de dados de objeto, utilize a função _RetVal(), preenchendo o campo `ev_object` na estrutura Value.

No exemplo a seguir, `Sum` aceita uma referência a um campo numérico em uma tabela e utiliza `_RetFloat` para retornar a soma dos valores no campo.

```
#include <pro_ext.h>

Sum(ParamBlk *parm)
{
// declara variáveis
double tot = 0, rec_cnt;
int i = 0, workarea = -1; // -1 é a área de trabalho atual
Value val;
```

```
// GO TOP
_DBRewind(workarea);

// Obtém RECCOUNT( )
rec_cnt = _DBRecCount(workarea);

// Efetua um loop na tabela
for(i = 0; i < rec_cnt; i++)
{
   //Coloca o valor do campo na estrutura Value
   _Load(&parm->p[0].loc, &val);

   // adiciona o valor ao total cumulativo
   tot += val.ev_real;
   // SKIP 1 na área de trabalho
   _DBSkip(workarea, 1);
}

// Retorna o valor da soma ao Visual FoxPro
_RetFloat(tot, 10, 4);
}
// A função Sum recebe um parâmetro de Referência
FoxInfo myFoxInfo[] = {
   {"SUM", Sum, 1,"R"}
};
FoxTable _FoxTable = {
   (FoxTable *) 0, sizeof(myFoxInfo)/sizeof(FoxInfo), myFoxInfo
};
```

Supondo que exista um campo numérico denominado `amount` na tabela atualmente aberta, a linha de código a seguir em um programa do Visual FoxPro chama a função:

```
? SUM(@amount)
```

Passando parâmetros para as funções da API do Visual FoxPro

Freqüentemente, as rotinas da API do Visual FoxPro exigirão parâmetros de uma estrutura de dados do Visual FoxPro em particular. As seções a seguir fornecem uma lista de tipos de dados do Visual FoxPro e estruturas de dados adicionais. Para as definições de estrutura e as definições de dados atuais, consulte o arquivo PRO_EXT.H.

Tipos de dados da API do Visual FoxPro

Os tipos de dados a seguir são utilizados em rotinas da API do Visual FoxPro.

Tipo de dados	Descrição
`EDLINE`	O número de uma linha em um arquivo aberto em uma janela de edição. A primeira linha é 1.
`EDPOS`	A posição de deslocamento de um caractere em um arquivo aberto em uma janela de edição. A posição de deslocamento do primeiro caractere no arquivo ou campo Memo é 0.
`FCHAN`	Canal do arquivo. Todos os arquivos abertos pelo Visual FoxPro ou por meio da API utilizando _FCreate( ) e _FOpen( ), aos quais é atribuído um FCHAN.
`FPFI`	Um ponteiro de 32 bites para uma função que retorna um inteiro.
`ITEMID`	Um identificador exclusivo atribuído a um único

	comando em um menu.
MENUID	Um identificador exclusivo atribuído a um menu.
MHANDLE	Um identificador exclusivo dado a cada bloco de memória alocado pelo Visual FoxPro ou por meio da API utilizando _AllocHand( ). A referência a seu ponteiro pode ser removida utilizando _HandToPtr( ).
NTI	Índice da tabela de nome. Todas os nomes de campos de tabela e variáveis têm uma entrada nesta tabela.
WHANDLE	Identificador de janela. Um identificador exclusivo atribuído a todas as janelas abertas pelo Visual FoxPro ou pela API utilizando _WOpen( ).

Observação Como os ponteiros FAR não são apropriados para compiladores de 32 bites, instruções `#define` em PRO_EXT.H redefinem `FAR`, `_far` e `__far` como valores nulos.

Estruturas de dados da API do Visual FoxPro

As estruturas de dados primários utilizadas na biblioteca da API do Visual FoxPro estão listadas na tabela a seguir.

Estrutura	Descrição
EventRec	Uma estrutura utilizada para descrever o que o sistema está fazendo em um determinado momento.
FoxInfo	Utilizada em bibliotecas FLL para comunicação entre o Visual FoxPro e o seu programa; não utilizada em arquivos .OCX. Abordada em Utilizando estruturas FoxInfo e FoxTable anteriormente neste capítulo.
FoxTable	Utilizada em bibliotecas FLL para comunicação entre o Visual FoxPro e o seu programa; não utilizada em arquivos .OCX. Abordada em Utilizando estruturas FoxInfo e FoxTable, anteriormente neste capítulo.
Locator	Uma estrutura utilizada para acessar valores de parâmetros (FLL) ou variáveis ou campos do Visual FoxPro (FLL e OCX).
ParamBlk	Utilizada em bibliotecas FLL para comunicação entre o Visual FoxPro e o seu programa; não utilizada em arquivos .OCX. Abordada em Utilizando estruturas FoxInfo e FoxTable, anteriormente neste capítulo.
Parameter	Utilizada em bibliotecas FLL para comunicação entre o Visual FoxPro e o seu programa; não utilizada em arquivos .OCX. Abordada em Utilizando estruturas FoxInfo e FoxTable, anteriormente neste capítulo.
Point	Uma estrutura que define as coordenadas horizontais e verticais de um ponto único na tela. As coordenadas são especificadas em linhas e colunas.
Rect	Uma estrutura que define as coordenadas de um retângulo na tela. O canto superior esquerdo do retângulo é definido por `(`*superior*`,` *esquerdo*`)` e o canto inferior direito é definido por `(`*inferior*`-1,` *direito*`-1)`. As coordenadas são especificadas em linhas e colunas.
Value	Uma estrutura utilizada para acessar valores de parâmetros (FLL) ou variáveis ou campos do Visual FoxPro (FLL e OCX).

Acessando variáveis e campos do Visual FoxPro

É possível acessar as variáveis ou os campos do Visual FoxPro em seu controle ActiveX ou função FLL para que sejam lidos ou definidos. Além disso, é possível criar novas variáveis que podem ser acessadas a partir do Visual FoxPro.

Variáveis e campos são disponibilizadas no Visual FoxPro em uma tabela de nomes, que é uma matriz contendo os nomes de todos os campos e as variáveis atualmente definidas. Você pode acessar um elemento individual em uma matriz utilizando um índice de tabela de nomes (NTI, *Name Table Index*). Uma função especial da API, _NameTableIndex( ), retorna o índice de uma variável ou campo existente baseado em um nome por você fornecido. Após ter determinado o NTI para uma dada variável, é possível ler essa variável utilizando a função _Load( ) da API ou defini-la utilizando a função _Store( ) da API. Para criar uma nova variável, você pode chamar a função _NewVar( ) da API.

Para acessar variáveis ou campos do Visual FoxPro, utilize as estruturas Value e Locator definidas em PRO_EXT.H. Se estiver criando uma biblioteca FLL, poderá utilizar a mesma técnica utilizada para acessar parâmetros passados às suas funções. Para obter detalhes sobre as estruturas Value e Locator, consulte Passando e recebendo parâmetros, anteriormente neste capítulo.

O exemplo a seguir, tirado do programa FOXTLIBCTL.CPP no diretório API\SAMPLES\FOXTLIB do Visual FoxPro, ilustra como você pode utilizar as estruturas Value e Locator em um controle ActiveX para acessar as variáveis do Visual FoxPro:

```
long CFoxtlibCtrl::TLGetTypeAttr(long pTypeInfo, LPCTSTR szArrName)
{
  int nResult = 1;
  TYPEATTR *lpTypeAttr;
  Locator loc;
  Value val;
  OLECHAR szGuid[128];
  char *szBuff;
__try {
   if (_FindVar(_NameTableIndex(( char *)szArrName),-1,&loc)) {
      ((ITypeInfo *)pTypeInfo)->GetTypeAttr(&lpTypeAttr);
      if (_ALen(loc.l_NTI, AL_ELEMENTS) < 16) {
         _Error(631); //Argumento da matriz de tamanho inadequdo.
      }

      //1 = Guid
      StringFromGUID2(lpTypeAttr->guid, (LPOLESTR )&szGuid,sizeof(szGuid));
      OLEOleToAnsiString(szGuid,&szBuff);
      val.ev_type = 'C';
      val.ev_length = strlen(szBuff);
      val.ev_handle = _AllocHand(val.ev_length);
      _HLock(val.ev_handle);
      _MemMove((char *) _HandToPtr( val.ev_handle ), szBuff, val.ev_length);
      OLEFreeString((void **)&szBuff);
      _HUnLock(val.ev_handle);
      loc.l_sub1 = 1;
      _Store(&loc,&val);
      _FreeHand(val.ev_handle);

      //2 = LCID
      loc.l_sub1 = 2;
      val.ev_type = 'I';
      val.ev_long = lpTypeAttr->lcid;
      _Store(&loc,&val);

      // code for values 3 - 16 here
      ((ITypeInfo *)pTypeInfo) -> ReleaseTypeAttr(lpTypeAttr);
```

```
      }
   } __except  (EXCEPTION_EXECUTE_HANDLER) {
      nResult = 0;
   }
return nResult;
```

Gerenciando a memória

A API do Visual FoxPro fornece acesso direto ao **Gerenciador de memória dinâmica** do Visual FoxPro. Para rotinas da API que exigem alocações de memória, é retornado um identificador de memória. A arquitetura de carregamento de segmentos do Visual FoxPro utiliza identificadores em vez de ponteiros para que possa gerenciar a memória de forma mais eficiente.

Observação As técnicas descritas nesta seção para o gerenciamento de memória utilizando a API do Visual FoxPro aplicam-se a controles ActiveX e bibliotecas FLL.

Utilizando identificadores

Um *identificador* se refere a um identificador de memória, que é basicamente um índice em uma matriz de ponteiros. Os ponteiros apontam para blocos de memória reconhecidos pelo Visual FoxPro. Quase todas as referências à memória na API são feitas por meio de identificadores em vez dos ponteiros C, mais tradicionais.

Para alocar e utilizar memória em sua biblioteca

1. Aloque um identificador com AllocHand( ).
2. Bloqueie o identificador com HLock( ).
3. Converta o identificador em um ponteiro com _HandToPtr( ).
4. Faça referência à memória utilizando o ponteiro.
5. Desbloqueie o identificador com HUnLock( ).

Observação Para evitar danos a arquivos de memo, não grave em um arquivo de memo antes de chamar AllocMemo( ).

Para endereçar a memória alocada, as rotinas da API devem converter o identificador em um ponteiro chamando a rotina _HandToPtr( ). Mesmo que o **Gerenciador de memória** do Visual FoxPro precise reorganizar a memória a fim de obter mais memória contígua para exigências de memória subseqüentes, o identificador permanecerá o mesmo. Também são fornecidas as rotinas que aumentam, diminuem, liberam e bloqueiam alocações de memória.

Ao criar rotinas externas, tente minimizar a utilização da memória. Se você criar uma rotina externa que aloque a memória de forma dinâmica, tente utilizar a menor quantidade de memória possível. Tenha cuidado especial com o bloqueio de grandes alocações de memória por longos períodos de tempo. Lembre-se de desbloquear identificadores de memória com _HUnLock( ) quando não precisarem mais ser bloqueados, pois o desempenho do Visual FoxPro pode ser afetado adversamente por identificadores de memória bloqueados.

Importante A utilização excessiva de memória dinâmica priva o Visual FoxPro de memória para buffers, janelas, menus e outros, além de diminuir seu desempenho, pois a memória utilizada para preencher exigências de rotinas da API é gerenciada pelo **Gerenciador de memória** do Visual FoxPro. Alocar e manter identificadores grandes pode fazer com que o Visual FoxPro fique sem memória e seja finalizado de forma anormal.

O ambiente do Visual FoxPro não tem proteção de memória. A rotina da API externa não pode fornecer toda a validação inerente a um programa padrão do Visual FoxPro. Se a memória for danificada, você receberá mensagens como “Identificador desconhecido”, “Erro de consistência interna” e “Nó desconhecido durante compactação”.

A função a seguir de uma biblioteca FLL ilustra a alocação de memória. O exemplo utiliza RetDateStr( ) para retornar um tipo Data do Visual FoxPro (considerando que o parâmetro Caractere é uma data apropriada).

```
#include <pro_ext.h>

void dates(ParamBlk  *parm)
{
   MHANDLE mh;
   char *instring;

   if ((mh = _AllocHand(parm->p[0].val.ev_length + 1)) == 0) {
      _Error(182); // "Memória insuficiente"
   }
   _HLock(parm->p[0].val.ev_handle);
   instring = _HandToPtr(parm->p[0].val.ev_handle);
   instring[parm->p[0].val.ev_length] = '\0';
   _RetDateStr(instring);
   _HUnLock(parm->p[0].val.ev_handle);
}
FoxInfo myFoxInfo[] = {
   {"DATES", (FPFI) dates, 1, "C"}
};
FoxTable _FoxTable = {
   (FoxTable *) 0, sizeof(myFoxInfo)/sizeof(FoxInfo), myFoxInfo
};
```

Sobre as pilhas

O controle ou biblioteca que você cria não apresenta uma pilha própria. Em vez disso, utiliza a pilha do programa de chamada, nesse caso a pilha do Visual FoxPro. Não é possível controlar o tamanho da pilha do Visual FoxPro nem afetar a quantidade de espaço de pilha disponível para um controle ActiveX ou um arquivo .FLL.

Em circunstâncias normais, essa distinção não é importante. A pilha do Visual FoxPro geralmente é grande o suficiente para sustentar as variáveis automáticas que você possa precisar alocar em um controle ou biblioteca. Caso fique sem espaço de pilha, sempre será possível alocar memória adicional na pilha de forma dinâmica.

Seguindo regras de identificadores

As regras a seguir se aplicam à utilização de identificadores e à responsabilidade de liberá-los:

- Os usuários devem liberar todos os identificadores que alocarem, incluindo os identificadores alocados por funções, como Load().
- Load() somente criará um identificador quando a variável que estiver sendo carregada for uma seqüência de caracteres (ou seja, `ev_type = 'C'`). Todos os outros tipos de dados armazenam os valores na estrutura Value propriamente dita, ao passo que carregar uma seqüência de caracteres coloca um MHANDLE no `ev_handle` da estrutura Value.
- Em uma biblioteca FLL, o Visual FoxPro é responsável por liberar todos os identificadores retornados com RetVal( ). Os usuários não devem liberar esses identificadores, mesmo que os tenham alocado.
- Os usuários não devem liberar identificadores passados para eles por seu `ParamBlk`.

Importante Ao gravar uma rotina externa que chame funções, certifique-se de seguir todas as regras e verificar os resultados retornados. Uma referência a um ponteiro ou identificador perdido poderia danificar as estruturas de dados internas do Visual FoxPro, provocando uma finalização anormal imediata ou problemas posteriores, que podem resultar em perdas de dados.

Construindo e depurando bibliotecas e controles ActiveX

Após criar um projeto, você estará pronto para construí-lo e depurá-lo.

Construindo o projeto

Antes de construir, é necessário estabelecer as definições do projeto. Algumas definições feitas dependem de se você deseja criar uma versão de depuração ou liberação do controle ou biblioteca. Como uma regra, crie versões de depuração do programa até estar satisfeito de que está trabalhando corretamente e, em seguida, crie uma versão de liberação.

Para especificar uma versão de depuração ou liberação

1. No menu **Build**, selecione **Set Default Configuration**.
2. Escolha se está criando uma versão de depuração ou liberação do controle.
3. Selecione **OK**.

Para estabelecer definições de projeto

1. No menu **Build**, selecione **Settings**.
2. Em **Settings For**, escolha se está criando uma versão de depuração ou de liberação do programa.
3. Clique na guia **C/C++** e, em seguida, faça estas definições:
   - Na lista **Category**, selecione **Code generation**.
   - Na lista **Calling Convention**, selecione **_fastcall**.
   - Na lista **Use run-time library**, selecione **Multithreaded DLL**.
4. Selecione a guia **Link** e, em seguida, na caixa de texto **Object/library modules**, adicione uma das bibliotecas a seguir:
   - Se estiver construindo uma .OCX, adicione `OCXAPI.LIB` do diretório da API do Visual FoxPro.
   - Se estiver construindo uma .FLL, adicione `WINAPIMS.LIB` do diretório da API do Visual FoxPro.
5. Defina **Ignore all default libraries**.
6. Selecione **OK**.

Para certificar-se de que o compilador pode localizar os arquivos necessários

1. No menu **Tools**, selecione **Options**.
2. Clique na guia **Directories**.
3. Na lista **Show directories for**, selecione **Include files**.
4. Na **Barra de ferramentas directories**, clique no botão **Add**.
5. Adicione o diretório com PRO_EXT.H.
6. Na lista **Show directories for**, selecione **Library files**.
7. Na **Barra de ferramentas directories**, clique no botão **Add**.
8. Adicione o diretório com OCXAPI.LIB do diretório da API do Visual FoxPro (ao criar um controle) ou adicione o WINAPIMS.LIB do diretório da API do Visual FoxPro (ao criar uma FLL).
9. Na caixa de diálogo **Options**, selecione **OK**.

Após ter especificado as definições, você poderá compilar e vincular o seu programa.

Para compilar e vincular um arquivo .OCX

- No menu **Build**, selecione **Build** *nomeproj***.OCX**.

Ao compilar e vincular o arquivo .OCX, o Visual C++ registra automaticamente o controle no computador no qual foi construído. Se por alguma razão você tiver que registrar o controle manualmente, poderá fazê-lo utilizando o procedimento a seguir.

Para registrar o controle ActiveX

- No menu **Tools** no Visual C++ Developer Studio, selecione **Register Control**.

  –Ou –

- Declare e chame DLLRegisterServer( ) do programa.

Depurando um controle ActiveX ou uma biblioteca FLL

Depurar o controle ou a biblioteca no contexto de um aplicativo completo do Visual FoxPro é mais difícil do que depurá-los separadamente do aplicativo. É aconselhável criar um programa de teste simples para testar a operação do seu controle ou biblioteca.

Depurando com o Microsoft Visual C++

O Microsoft Visual C++ versão 4.0 e posterior oferece um ambiente de depuração integrado que facilita a definição de pontos de interrupção e a execução do código. É possível até mesmo executar o Visual FoxPro a partir do Visual C++.

Para iniciar a depuração com o Microsoft Visual C++

1. A partir do menu **Build**, selecione **Settings**.
2. Na caixa de diálogo **Project Settings**, clique na guia **Debug**.
3. Na caixa de texto **Executable for debug session**, digite o caminho seguido de VFP.EXE.
   Por exemplo, digite **C:\VFP\VFP.EXE**.
4. Selecione **OK**.
5. Defina um ponto de interrupção na biblioteca.
6. A partir do menu **Build**, selecione **Debug**. Em seguida, a partir do submenu, selecione **Go**.
7. Quando o Developer Studio exibir uma mensagem que diz “VFP.EXE não contém informações de depuração”, selecione **Yes** para continuar.

Para obter maiores informações sobre a depuração no Visual C++, consulte a documentação do Visual C++.

Depurando com outros depuradores

Você pode depurar um controle ou biblioteca com qualquer depurador que identifique corretamente um INT 3 (_BreakPoint( )) incorporado no seu programa. É possível utilizar qualquer depurador para uma depuração simbólica se ele puder realizar todas as tarefas a seguir:

- Fazer uma tabela de símbolos a partir de um arquivo de mapa.
- Carregar a tabela de símbolos independente do programa.
- Transferir os símbolos para um novo endereço.

Para depurar uma biblioteca

1. Adicione uma chamada BreakPoint( ) à rotina no ponto onde a depuração começará.
2. Construa o controle ou biblioteca.
3. Chame o depurador.
4. Se o depurador suportar símbolos, carregue a tabela de símbolos para a sua biblioteca.
5. Inicie o Visual FoxPro.
6. Chame a rotina da sua biblioteca a partir do Visual FoxPro.
7. Quando o ponto de interrupção for alcançado, faça ajustes à base de símbolos para alinhar os símbolos com o local atual onde a biblioteca foi carregada.
8. Aumente o registro do ponteiro de instrução (IP, *Instruction Pointer*) em 1 para pular a instrução INT 3.
9. Continue depurando da maneira que faria com um programa normal.

Observação Sempre remova qualquer ponto de interrupção especificado no depurador antes de liberar o produto.