Training Material

P/Invoke

.NET Interop

Rainer Stropek

software architects gmbh

Web http://www.timecockpit.com
Mail rainer@timecockpit.com
Twitter @rstropek

time cockpit
Saves the day.

C++ Basics

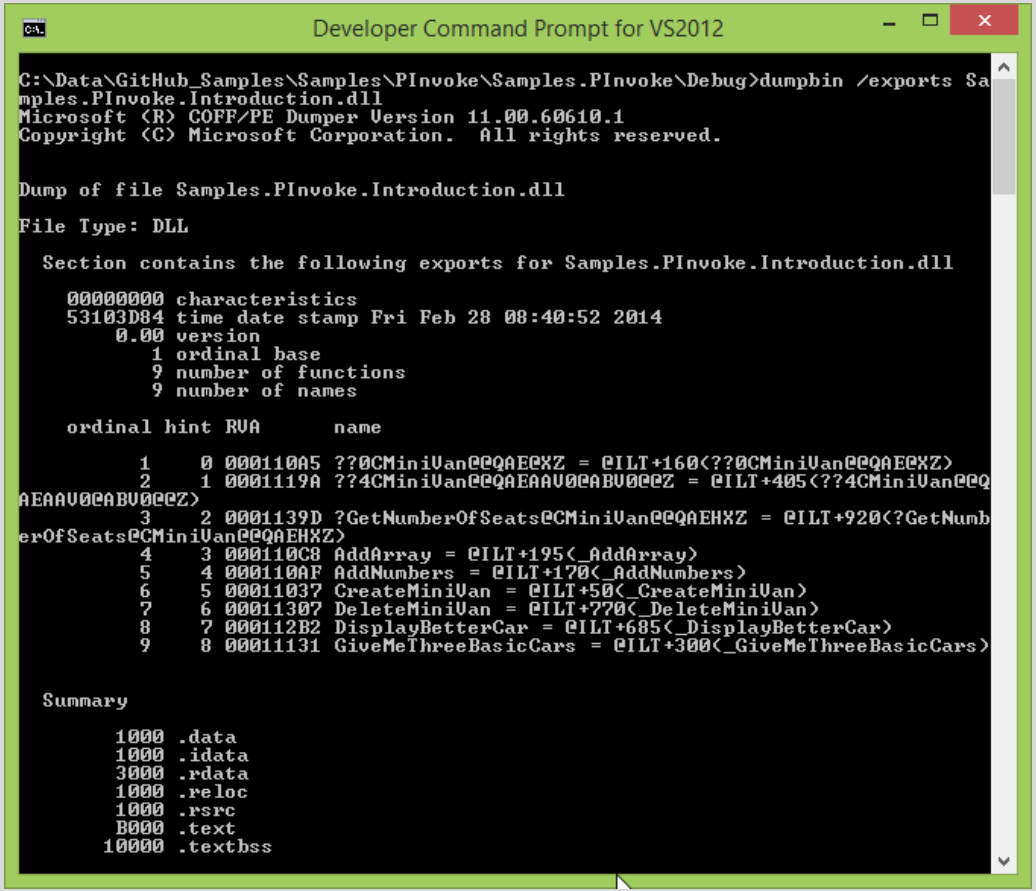
Interop-related topics for C++

```
// The following ifdef block is the standard way of creating macros which make exporting
// from a DLL simpler. All files within this DLL are compiled with the PINVOKE_EXPORTS
// symbol defined on the command line. This symbol should not be defined on any project
// that uses this DLL. This way any other project whose source files include this file see
// PINVOKE_EXPORTS functions as being imported from a DLL, whereas this DLL sees symbols
// defined with this macro as being exported.
#ifdef PINVOKE_EXPORTS
#define PINVOKE_API __declspec(dllexport)
#else
#define PINVOKE_API __declspec(dllimport)
#endif



// Some quite simple functions to begin with
extern "C" PINVOKE_API int AddNumbers(int x, int y);
extern "C" PINVOKE_API int AddArray(int x[], int size);



// A very simple DLL export.
extern "C" PINVOKE_API int AddNumbers(int x, int y)
{
   return x + y;
}

// A method taking an array.
extern "C" PINVOKE_API int AddArray(int x[], int size)
{
   auto ans = 0;
   for (int i = 0; i < size; i++)
   {
      ans += x[i];
   }

   return ans;
}
```

DLL Exports

See also MSDN

dumpbin



Get list of exports of an unmanaged DLL

Parameters

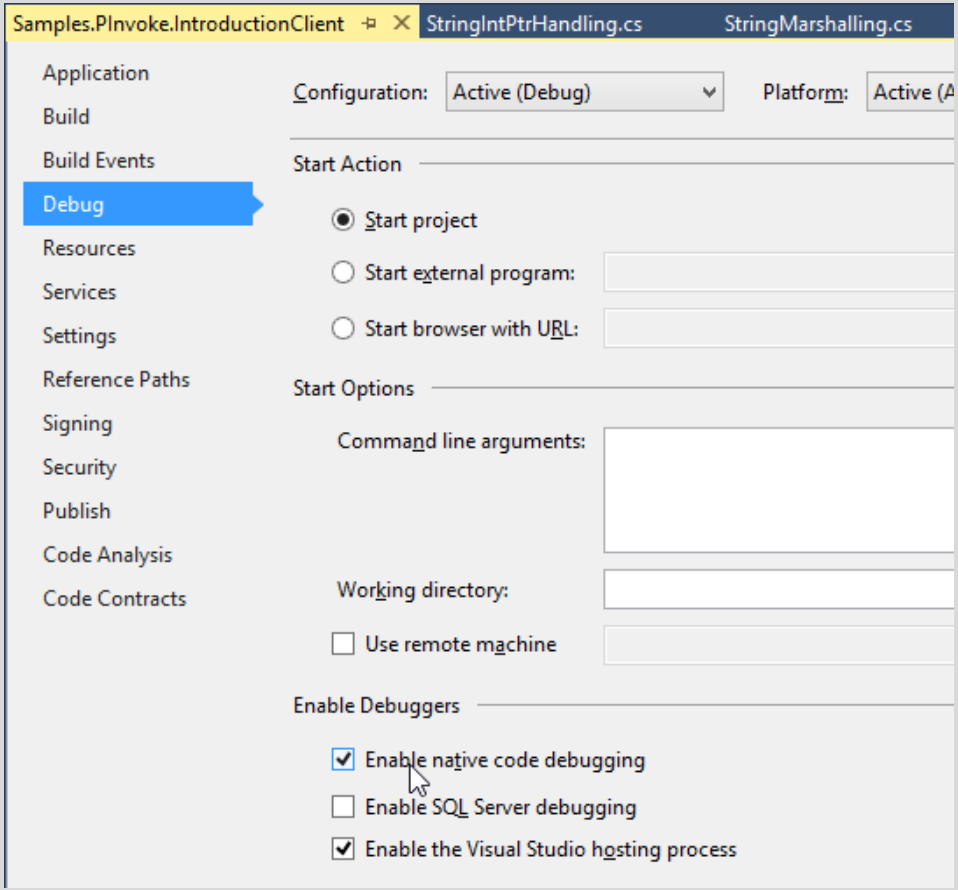
- /imports
- /exports

See MSDN for details

Memory Handling

- Native DLL should avoid allocating memory and expect the caller to free the memory
- Use *CoTaskMemAlloc* instead of *new*
  Freed by Interop Marshaller of .NET automatically using *CoTaskMemFree*
  Can be freed manually using *Marshal.FreeCoTaskMem*
- Native DLL could offer a function to free the memory
  See *CMiniVan* example (*CreateMiniVan* and *DeleteMiniVan*)

Debugging



Enable native code debugging

Step into unmanaged code

Interop Basics

Interop Basics

- *System.Runtime.InteropServices* Namespace
- System.Runtime.InteropServices.Marshal class
  Important helper methods for working with unmanaged code
- System.Runtime.InteropServices.DllImportAttribute
  Import a method from an unmanaged DLL

Table 1-4. Data Type Representation

Unmanaged Type in wtypes.h	Unmanaged C Language Type	Managed Type Representation	Meaning in Life
BOOL	long	System.Int32	32 bits
BYTE	unsigned char	System.Byte	8 bits
CHAR	char	System.Char	ANSI string
DOUBLE	double	System.Double	64 bits
DWORD	unsigned long	System.UInt32	32 bits
FLOAT	float	System.Single	32 bits
HANDLE	void*	System.IntPtr	32 bits
INT	int	System.Int32	32 bits
LONG	long	System.Int32	32 bits
LPCSTR	const char*	System.String or System.StringBuilder	ANSI string
LPCWSTR	const wchar_t*	System.String or System.StringBuilder	Unicode string
LPSTR	char*	System.String or System.StringBuilder	ANSI string
LPWSTR	wchar_t*	System.String or System.StringBuilder	Unicode string
SHORT	short	System.Int16	16 bits
UINT	unsigned int	System.UInt32	32 bits
ULONG	unsigned long	System.UInt32	32 bits
WORD	unsigned short	System.UInt16	16 bits

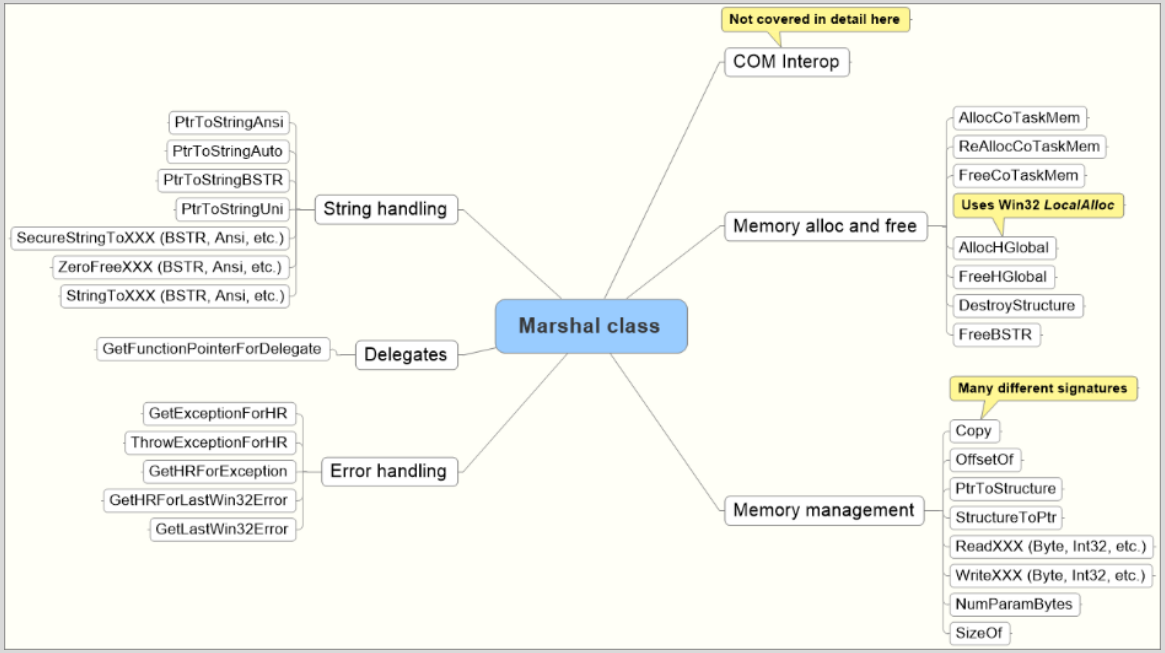
Data Type Mapping

See also MSDN

More detailed list can be found in Nathan, .NET and COM, 779ff

Source: Troelsen, COM and .NET Interoperability

Marshal Class



For details see MSDN

DllImportAttribute

DllImportAttribute

- Calling convention
- Character set
- Entry point
  Can be used to rename functions
- Exact spelling
  Controls whether character set is used to look for entry point name
- Error handling
  SetLastError behavior

Calling Conventions

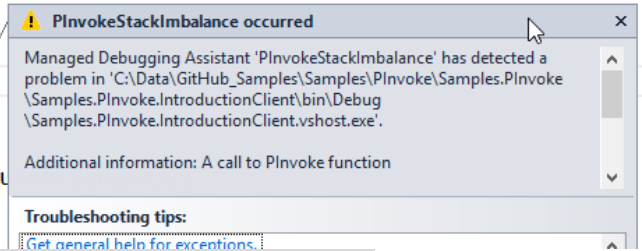
- See also MSDN article about calling conventions
  By default, VC++ uses *_cdecl*
- P/Invoke behavior has changed in .NET 4
  In .NET < 4 you could call a *_cdecl* function with *__stdcall* without any problems
  In .NET >= 4 you will get an exception in the debugger
  See also pInvokeStackImbalance and NetFx40_PInvokeStackResilience

See also http://blogs.msdn.com/b/oldnewthing/archive/2004/01/08/48616.aspx

Calling Conventions

```
// This works
extern "C" PINVOKE_API int __stdcall AddNumbers(int x, int y);

[DllImport("PInvokeIntroduction.dll")]
public static extern int AddNumbers(int x, int y);


// This works, too
extern "C" PINVOKE_API int AddNumbers(int x, int y);

[DllImport("PInvokeIntroduction.dll",
  CallingConvention = CallingConvention.Cdecl)]
public static extern int AddNumbers(int x, int y);


// Anything else will throw an exception in the debugger
```

```
static void Main(string[] args)
{
    Console.WriteLine(PInvokeWrapper.AddNumbers(1, 2));

    var source = new[] { 1, 2, 3 };
    Console.WriteLine(PInvokeWrapper.AddArray(source, sou
}
```



Marshal Strings

- Character sets
  - Ansi
  - Unicode
  - Auto (depends on operating system type)
  - None (obsolete)
- Specify character set
  - In *DllImportAttribute*
  - In *MarshalAsAttribute*

```
using System.Runtime.InteropServices;

namespace Samples.PInvoke.IntroductionClient
{
  [StructLayout(LayoutKind.Sequential)]
  public class Car
  {
    [MarshalAs(UnmanagedType.LPWStr)]
    public string Make;

    [MarshalAs(UnmanagedType.LPWStr)]
    public string Color;
  }
  …
}


[DllImport("PInvokeIntroduction.dll",
  CallingConvention = CallingConvention.Cdecl,
  CharSet = CharSet.Unicode)]
public static extern void DisplayBetterCar(Car2 c);
```

Character Sets

See also MSDN

```
// Rename the MessageBoxW() function to 'DisplayMessage'.
[DllImport("user32.dll",
  ExactSpelling = true,
  CharSet = CharSet.Unicode,
  EntryPoint = "MessageBoxW")]
public static extern int DisplayMessage(int hwnd, string text,
  string caption, int type);
```

Renaming Method

```
// Note that we pass an invalid window handle
// to MessageBox → error
PInvokeWrapper.DisplayMessage(999, "Hello World!", "Greeting",
0);

Console.WriteLine("Last Win32 Error: {0}",
  Marshal.GetLastWin32Error());
Console.WriteLine(new Win32Exception(
  Marshal.GetLastWin32Error()).Message);
```

Win32 Errors

LayoutKind

- Necessary when marshalling structure
  Note that you can marshal structures as C# structs *or* classes
- *LayoutKind.Sequential*
  Recommendation for structure marshalling
  Order of the fields is preserved
- *LayoutKind.Explicit*
  Manually calculate the physical position of fields
  Combined with *FieldOffsetAttribute*
  Use *FieldOffsetAttribute* for implementing *unions*, too
- *LayoutKind.Auto*
  Not used for P/Invoke as C# compiler could change the order of the fields

```
using System.Runtime.InteropServices;

namespace Samples.PInvoke.IntroductionClient
{
  [StructLayout(LayoutKind.Sequential)]
  public class Car
  {
    public string Make;
    public string Color;
  }

  // A structure containing another structure.
  [StructLayout(LayoutKind.Sequential)]
  public class Car2
  {
    public Car Car = new Car();
    public string PetName;
  }
}
```

StructLayout

```
// A Method returning an array of structs.
extern "C" PINVOKE_API void GiveMeThreeBasicCars(CAR** theCars)
{
    auto numbOfCars = 3;

    // Use CoTaskMemAlloc instead of new as .NET's P/Invoke uses
    // CoTaskMemFree. See also http://blogs.msdn.com/b/dsvc/archive/2009/06/22/troubleshooting-pinvoke-related-issues.aspx
    // and http://stackoverflow.com/questions/3614367/c-sharp-free-memory-allocated-by-operator-new-from-p-invoke-dll
    // for details.
    *theCars = (CAR*)CoTaskMemAlloc(numbOfCars * sizeof(CAR));

    LPSTR carMakes[3] = { "BMW", "Ford", "Viper" };
    LPSTR carColors[3] = { "Green", "Pink", "Red" };

    auto pCurCar = *theCars;
    for (int i = 0; i < numbOfCars; i++, pCurCar++)
    {
        pCurCar->color = carColors[i];
        pCurCar->make = carMakes[i];
    }
}
```

```
[DllImport("PInvokeIntroduction.dll", CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi)]
public static extern void GiveMeThreeBasicCars(out IntPtr theCars);

public static IEnumerable<Car> GiveMeThreeBasicCarsHelper() {
    const int size = 3;
    var result = new List<Car>(size);

    // Pass in an IntPtr as an output parameter.
    IntPtr outArray;
    PInvokeWrapper.GiveMeThreeBasicCars(out outArray);
    try {
        // Helper for iterating over array elements
        IntPtr current = outArray;
        for (int i = 0; i < size; i++) {
            // Get next car using Marshal.PtrToStructure()
            var car = Marshal.PtrToStructure<Car>(current);
            result.Add(car);

            // Calculate location of next structure using Marshal.SizeOf().
            current = (IntPtr)((int)current + Marshal.SizeOf<Car>());
        }
    }
    finally {
        // Free memory for the allocated array.
        Marshal.FreeCoTaskMem(outArray);
    }

    return result;
}
```

Marshal an Array

Marshalling Classes

```
#include "stdafx.h"

// A class to be exported.
class PINVOKE_API CMiniVan
{
private:
  int m_numbSeats;

public:
  CMiniVan()
  {
    m_numbSeats = 9;
  }

  int GetNumberOfSeats()
  {
    return m_numbSeats;
  }
};

// Functions for class marshaling.
extern "C" PINVOKE_API CMiniVan* CreateMiniVan();
extern "C" PINVOKE_API void DeleteMiniVan(CMiniVan* obj);
extern "C" PINVOKE_API int GetNumberOfSeats(CMiniVan* obj);
```

Marshalling Classes

```
// extern "C" PINVOKE_API CMiniVan* CreateMiniVan();
[DllImport("PInvokeIntroduction.dll",
  CallingConvention = CallingConvention.Cdecl)]
public static extern IntPtr CreateMiniVan();

// extern "C" PINVOKE_API void DeleteMiniVan(CMiniVan* obj);
[DllImport("PInvokeIntroduction.dll",
  CallingConvention = CallingConvention.Cdecl)]
public static extern void DeleteMiniVan(IntPtr miniVan);

// extern "C" PINVOKE_API int GetNumberOfSeats(CMiniVan* obj);
[DllImport("PInvokeIntroduction.dll",
  CallingConvention = CallingConvention.Cdecl)]
public static extern int GetNumberOfSeats(IntPtr miniVan);


var miniVan = PInvokeWrapper.CreateMiniVan();
try
{
  Console.WriteLine(PInvokeWrapper.GetNumberOfSeats(miniVan));
}
finally
{
  PInvokeWrapper.DeleteMiniVan(miniVan);
}
```

```
typedef void (CALLBACK *SAYHELLOCALLBACK)();
extern "C" PINVOKE_API void CallMeBackToSayHello(
  SAYHELLOCALLBACK callback);

[DllImport("PInvokeIntroduction.dll",
  CallingConvention = CallingConvention.Cdecl)]
public static extern void CallMeBackToSayHello(
  Action callback);
```

Marshalling Callbacks

Marshalling Callbacks

```
typedef struct
{
  double a;
  double b;
  double c;
} TRIANGLE;
typedef void (CALLBACK *PYTHAGORASCALLBACK)(TRIANGLE result);
extern "C" PINVOKE_API void ReportPythagorasBack(
  double a, double b, PYTHAGORASCALLBACK callback);
```

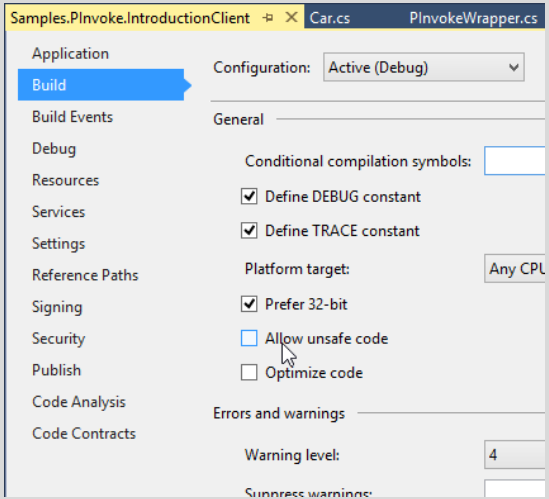
```
[StructLayout(LayoutKind.Sequential)]
public struct Triangle
{
public double a;
public double b;
public double c;
}

public delegate void TriangleCallback(Triangle t);

[DllImport("PInvokeIntroduction.dll",
  CallingConvention = CallingConvention.Cdecl)]
public static extern void ReportPythagorasBack(
  double a, double b, TriangleCallback callback);
```

Unsafe C#

Unsafe C#

- Use *unsafe* keyword
  - Used with type or member
  - Unsafe code block
- Adds pointer arithmetic to C#
  - Similar to C#
- *Unsafe* does not mean less safe than normal C#
  - If you use *IntPtr*, you are not safer
  - In both scenarios, C# does *not do any checks for buffer overruns*
- You can program P/Invoke without unsafe
  - Use *IntPtr* and methods of the *Marshal* class instead

```
type *identifier;
void *identifier;// Ptr to unknown type


int* pNumber;
int** ppNumber;  // Ptr to a ptr to int
int*[] pArray;   // Array of int*


int* pNumber
Console.WriteLine(*pNumber);
                 // Get value that pNumber points to
```



Pointers in C#

Supported types

- sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double, decimal, or bool.
- Enums
- Pointers
- User-defined structs with unmanaged types

```
namespace FixedSizeBuffers
{
    internal unsafe struct MyBuffer
    {
        public fixed char fixedBuffer[128];
    }

    internal unsafe class MyClass
    {
        public MyBuffer myBuffer = default(MyBuffer);
    }

    internal class Program
    {
        static void Main()
        {
            MyClass myC = new MyClass();

            unsafe
            {
                // Pin the buffer to a fixed location in memory.
                fixed (char* charPtr = myC.myBuffer.fixedBuffer)
                {
                    *charPtr = 'A';
                }
            }
        }
    }
}
```

Fixed size buffers

Buffer with fixed size

Typically used inside a struct

Supported data types

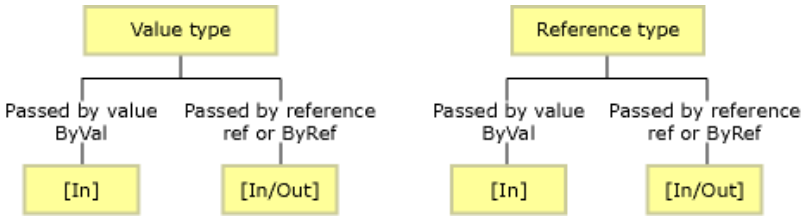
bool, byte, char, short, int, long, sbyte, ushort, uint, ulong, float, or double

Marshalling Details

Tipps & Tricks

- *BOOL* can be marshalled to *System.Int32* or *bool*
  Marshalling to bool is a little bit slower but convenient
  See MSDN for details

- *InAttribute* and *OutAttribute*
  By default, direction attribute for non-blittable types is *In* (performance reasons)
  Best practice: Specify *[In, Out]* for input and output parameters (even for blittable types)
  Exception: *StringBuilder* (*In, Out* by default)
  See also sample in MSDN



Memory Management

- Marshaler always attempts to free memory allocated by unmanaged code
  Use *IntPtr* to prevent this
- Memory is always freed using *CoTaskMemFree*
  Library has to provide a dedicated free method if allocated differently
- See MSDN for details about copying/pinning

String Marshalling

- *System.String* for constant strings
  LPCSTR, *LPCWSTR*, etc.
- *System.Text.StringBuilder* for string buffers that can change
  LPSTR, *LPWSTR*, etc.
  Always initialize size of the *StringBuilder* (constructor, *Capacity* property)
  StringBuilder are guaranteed to have a terminating null character
- Use *IntPtr* and *Marshal.PtrToStringXXX* if caller should not free the memory

String Marshalling

- Use *[MarshalAs(UnmanagedType.ByValXXX, SizeConst=XXX)]* for fixed size char buffers
- See MSDN for details

Strings and *IntPtr*

```
public static class StringIntPtrHandling
{
    // Note that GetPrivateProfileSectionNames returns a string with embedded NULL characters.
    // See http://msdn.microsoft.com/en-us/library/windows/desktop/ms724352(v=vs.85).aspx
    // for details.

    [DllImport("kernel32.dll")]
    static extern int GetPrivateProfileSectionNames(IntPtr lpszReturnBuffer, int nSize, string lpFileName);

    public static void ExecuteSample()
    {
        IntPtr ptr = IntPtr.Zero;
        string s = string.Empty;

        try
        {
            // Allocate a buffer in unmanaged memory
            ptr = Marshal.AllocHGlobal(1024);

            // Call Kernel API
            var numChars = GetPrivateProfileSectionNames(
                ptr,
                1024,
                Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location), "Sample.ini"));

            // Copy the buffer into a managed string
            s = Marshal.PtrToStringAnsi(ptr, numChars - 1);
        }
        finally
        {
            // Free the unmanaged buffer
            if (ptr != IntPtr.Zero)
            {
                Marshal.FreeHGlobal(ptr);
            }
        }

        // Display the sections by splitting the string based on NULL characters
        foreach (var section in s.Split('\0'))
        {
            Console.WriteLine(section);
        }
    }
}
```

```
// Display the sections by splitting the string based on NULL characters
foreach (var section in s.Split('\0'))
{
    Console.WriteLine(section);
}
```

s "DbServers\0FileServers"

Marshalling Arrays

```
typedef struct
{
   // Note that this structure contains an array of characters
   char make[256];
   char color[256];
} CARFIXED;

extern "C" PINVOKE_API void FillThreeBasicCars(CARFIXED* theCars);
```

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
public struct CarStruct
{
   [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
   public string Make;
   [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
   public string Color;
}

[DllImport("PInvokeIntroduction.dll",
   CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi)]
public static extern void FillThreeBasicCars(
   [In, Out, MarshalAs(UnmanagedType.LPArray)] CarStruct[] theCars);
```

```
extern "C" PINVOKE_API void GiveMeMakes(BSTR** makes, int *length);
```

```
[DllImport("PInvokeIntroduction.dll",
   CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Unicode)]
public static extern void GiveMeMakes(
   [Out, MarshalAs(UnmanagedType.LPArray,
      ArraySubType = UnmanagedType.BStr,
      SizeParamIndex = 1)] out string[] makes,
   [Out] out int length);
```

Marshalling Arrays

Training Material

Q&A

Thank your for coming!

Rainer Stropek

software architects gmbh

Mail rainer@timecockpit.com
Web http://www.timecockpit.com
Twitter @rstropek

time cockpit
Saves the day.

time cockpit is the leading time tracking solution for knowledge workers. Graphical time tracking calendar, automatic tracking of your work using signal trackers, high level of extensibility and customizability, full support to work offline, and SaaS deployment model make it the optimal choice especially in the IT consulting business.

Try time cockpit for free and without any risk. You can get your trial account at http://www.timecockpit.com. After the trial period you can use time cockpit for only 0,20€ per user and day without a minimal subscription time and without a minimal number of users.

time cockpit

time cockpit ist die führende Projektzeiterfassung für Knowledge Worker. Grafischer Zeitbuchungskalender, automatische Tätigkeitsaufzeichnung über Signal Tracker, umfassende Erweiterbarkeit und Anpassbarkeit, volle Offlinefähigkeit und einfachste Verwendung durch SaaS machen es zur Optimalen Lösung auch speziell im IT-Umfeld.

Probieren Sie time cockpit kostenlos und ohne Risiko einfach aus. Einen Testzugang erhalten Sie unter http://www.timecockpit.com. Danach nutzen Sie time cockpit um nur 0,20€ pro Benutzer und Tag ohne Mindestdauer und ohne Mindestbenutzeranzahl.