

Когда в С# не хватает C++ vol 2

Сергей Балтийский
JetBrains

Why?

∞ Скорость

- ∞ Оптимизация CPU
- ∞ Затраты на переключение контекста

∞ Память

- ∞ Управление памятью
- ∞ GC

∞ Legacy

- ∞ Библиотеки на C/C++

How?


∞ C++/CLI

∞ COM

∞ PInvoke

∞ Native Memory in C#

Unsafe Code

- ∞ Verifiable-код — ценность C#
 - ∞ В C# можно писать C-style код
- 
- ∞ Хочется разделять эти вещи
 - ∞ `unsafe` keyword
 - ∞ `UnverifiableCodeAttribute` custom attribute

Unsafe or not?

```
POINT pt = new POINT();  
User32Dll.GetCursorPos(&pt);
```

```
IntPtr ptr = Marshal.AllocHGlobal(sizeof(RECT));  
Marshal.StructureToPtr(new RECT(0,0,100,100), ptr, false);
```

```
var handle = SetWindowsHookExW(HookType.WH_CALLWNDPROCRET,  
    new HOOKPROC(HookProc), IntPtr.Zero, Kernel32Dll.GetCurrentThreadId());
```

```
[DllImport("user32.dll")]  
static extern bool GetScrollBarInfo(IntPtr hWnd, long idObject, IntPtr psbi);
```

Unmanaged Pointers

- ∞ `IntPtr`

- ∞ `(native int)`

- ∞ `UIntPtr`

- ∞ `(native unsigned int)`

- ∞ `void*`

- ∞ `byte*`

IntPtr vs. T*

∞ Арифметика

- ∞ T* — операции сложения по правилам указателей
- ∞ IntPtr — присутствуют в отдельных новых версиях

∞ Разница в Sign Extension

- ∞ T* никак специально не трогает старшие биты
- ∞ IntPtr при кастах расширяет старшим битом

∞ unsafe keyword

IntPtr vs. T*

∞ Арифметика указателей: типичный антипаттерн

```
IntPtr ptr = data.Scan0;
for (int a = count; a --> 0;)
{
    ProcessByte(Marshal.ReadByte(ptr));
    ptr = (IntPtr)((Int32)ptr + Marshal.SizeOf(typeof(Byte)));
}
```

```
var pb = (byte*)data.Scan0;
for(int a = count; a --> 0;)
{
    ProcessByte(*pb);
    pb++;
}
```


IntPtr vs. T*

∞ Нежелательный sign extension

```
uint theirs = 0xdeadbeef;  
  
var received1 = (IntPtr)(void*)theirs;  
var received2 = (void*)theirs;  
  
ulong ours1 = (ulong)(long)received1;  
ulong ours2 = (ulong)received2;
```

	theirs	(IntPtr) ours1	(void*) ours2
32-bit	DEADBEEF	FFFFFFFFDEADBEEF	DEADBEEF
64-bit	DEADBEEF	DEADBEEF	DEADBEEF

Pinning

- ∞ Задача: получить указатель на value type
- ∞ Value type на стеке:
 - ∞ Не может перемещаться в памяти
 - ∞ Можно непосредственно взять указатель

```
RECT rc = new RECT();  
RECT *pRect = &rc;
```

Pinning

- ∞ Задача: получить указатель на value type
- ∞ Value type внутри reference type object
 - ∞ Адрес в памяти может меняться при GC
 - ∞ Interior pointer
 - ∞ Pinning, чтобы на время запретить перемещать объект

```
class WindowWrapper  
{ public RECT Bounds; }
```

```
WindowWrapper ww = new WindowWrapper();  
fixed(RECT *pRect = &ww.Bounds)  
    Use(pRect);
```

Pinning by C# Compiler

- ∞ fixed()
- ∞ Реализовано как атрибут локальной переменной
 - ∞ Нет императивной команды pin/unpin
 - ∞ Нет ограничения на тип объекта
 - ∞ Но компилятор C# ограничивает до “unmanaged types”
 - ∞ Только пока выполняется функция
 - ∞ Кроме closures & coroutines

Pinning by C# Compiler

∞ Специальная магия для массивов

```
fixed(byte* pBuf1 = buffer)    { }  
  
fixed(byte* pBuf2 = &buffer[0]) { }
```

∞ Специальная магия для строк

```
fixed(char* pch = text)    { }  
  
System.Runtime.CompilerServices.RuntimeHelpers::OffsetToStringData
```

Pinning with GC Handle

- ∞ Создаём GC Handle специального типа

```
GCHandle::Alloc()  
GCHandleType::Pinned  
GCHandle::AddrOfPinnedObject()
```

- ∞ Время жизни не ограничено
 - ∞ Для этого и берут
 - ∞ Из-за длительных пинов GC Heap может держать много «пустой» памяти
- ∞ Только blittable types

Классификация объектов

POD

POJO

Blittable

Unmanaged

Value Type

Reference Type

Managed

Blittable Objects

C# declaration
layout

Marshalled
memory layout

.NET Runtime
memory layout



Blittable Objects

- ∞ Гарантированный memory layout
 - ∞ Идентичный результат через `Marshal` и через `T*`
 - ∞ Почти аналогичен C++ POD
 - ∞ Trivial Classes
 - ∞ Standard Layout Classes
 - ∞ Не забыть про `StructLayoutAttribute::Pack`

Blittable Objects

- ∞ Нет compile-time индикации, что объект blittable
 - ∞ `fixed()` всё равно компилируется
 - ∞ `MethodTable::IsBlittable` в CLR
 - ∞ Косвенные измерения, например, попытка создать Pinned GC Handle

Suddenly, non-blittable

- ∞ T* memory layout отличается от ожидаемого
- ∞ Interop с C/C++/WinAPI ломается
- ∞ Бинарный формат меняется
 - ∞ Может быть несущественно в пределах одного CLR
 - ∞ Бинарная несовместимость между разными CLR
 - ∞ Например, CLR4 и Mono

Blittable Types

- ∞ Signed/unsigned integers
- ∞ Signed/unsigned native integers
- ∞ Single, Double
- ∞ Value types:
 - ∞ C LayoutKind Sequential или Explicit,
 - ∞ И с blittable types внутри
- ∞ Одномерные массивы из blittable types

Why non-blittable?

- ∞ LayoutKind Auto

- ∞ CLR может переставлять данные в памяти для оптимальной упаковки

Why non-blittable?

∞ Boolean type

- ∞ Размер зависит от контекста маршallingа
 - ∞ WinAPI BOOL → 32-bit integer
 - ∞ А в массиве может занимать 1 байт
- ∞ False → 0, а True в общем случае всё остальное
 - ∞ WinAPI предпочитает 1, Visual Basic — -1

```
public static void CheckBools(bool x, bool y)
{
    if(!x)    return;
    if(!y)    return;
    if(x!=y) throw new InvalidOperationException("x!=y");
}
```

Why non-blittable?

- ∞ Char type

- ∞ Поддержка ANSI encodings в CLR

 - ∞ Windows 98 (!)

 - ∞ ANSI-варианты WinAPI на Windows NT

 - ∞ Маршаллер умеет конвертировать UTF-16LE <-> ANSI

 - ∞ Это меняет размер и layout структуры

Char Mitigation

∞ Использовать **Int16** вместо **Char**

∞ Везде выставить

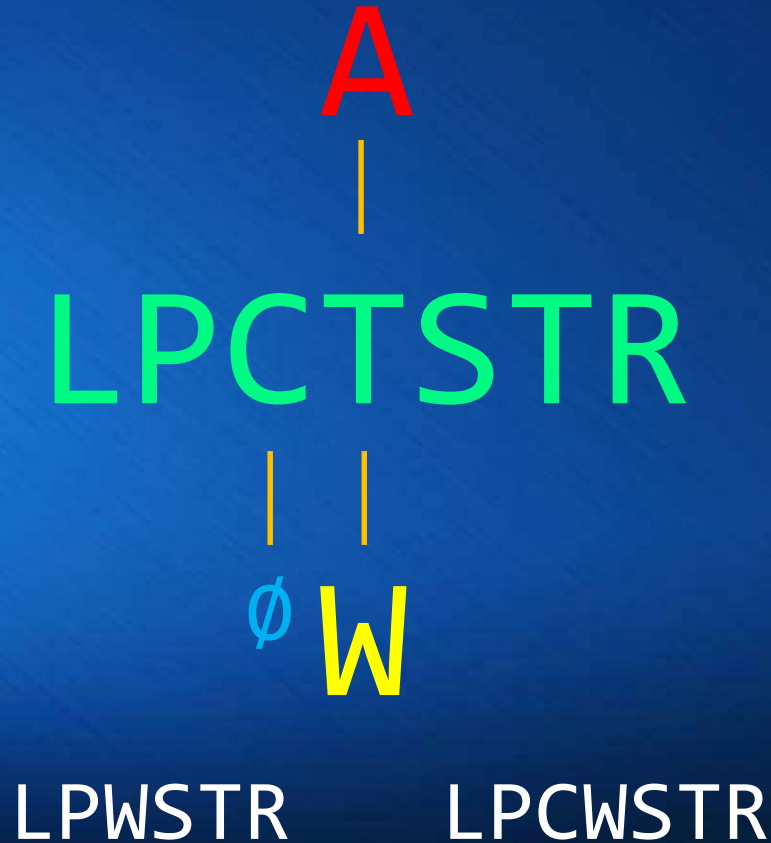
```
StructLayoutAttribute::CharSet ← CharSet.Unicode
```

∞ Структуры вместо fixed arrays

∞ Собственно, компилятор так и делает

```
public fixed Char cFileName[260];  
  
public CharSet cFileName;  
  
[StructLayout(LayoutKind.Explicit, Size = 260*2)]  
public struct CharSet { }
```


Marshalling Strings



String Representation

LPCWSTR

WCHAR*

_wchar_t*

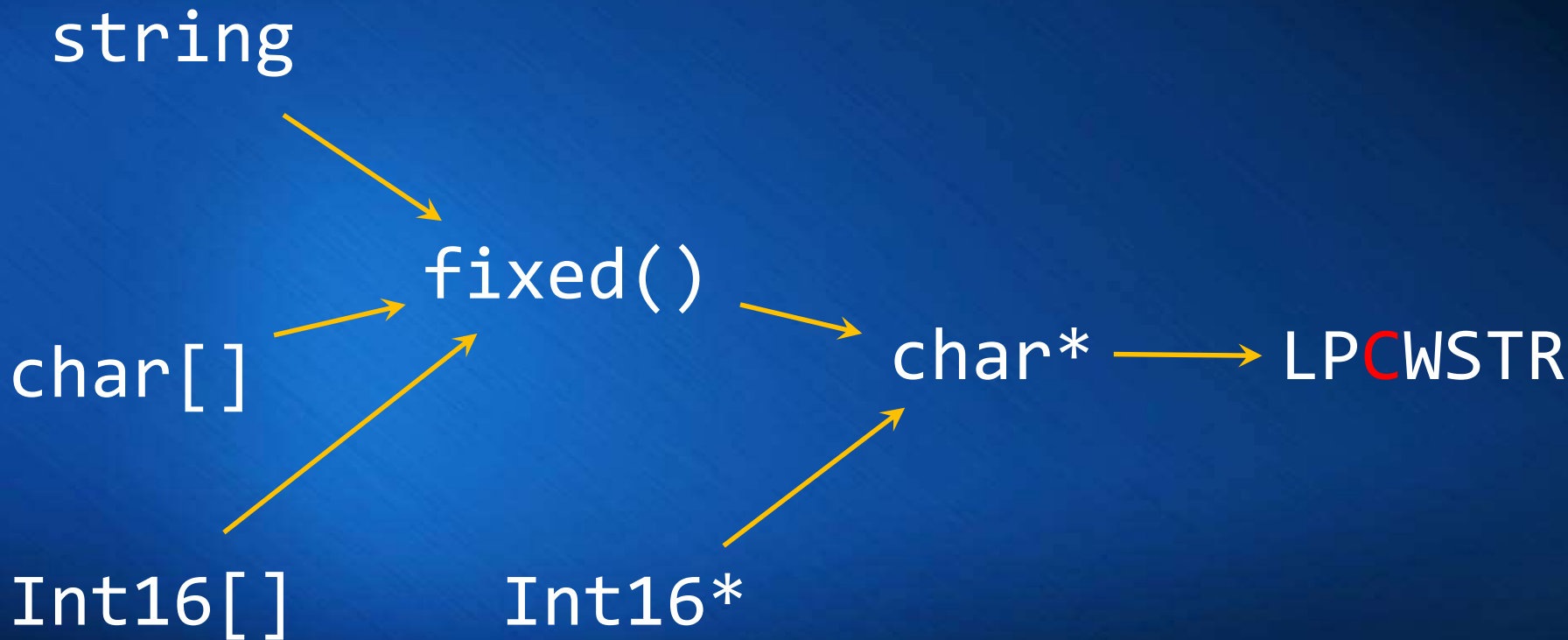
System.Char*

System.String

UTF16-LE

ASCIIIZ

Strings: CLR → Native



Внимание на NULL-terminated

Strings: Native → CLR

- ∞ Строка в статичной native памяти
 - ∞ Просто `new string()`
 - ∞ Внимание на длину и terminating null
- ∞ Native выделил память специально для нас
 - ∞ `new string()` и освободить память за собой
 - ∞ Правильной функцией
- ∞ Мы сами выделяем буфер

Strings: Native → CLR

- ∞ Мы сами выделяем буфер
 - ∞ Возможно, придётся договариваться о размере
 - ∞ Вариант `StringBuilder`
 - ∞ Вариант `stackalloc`
 - ∞ Вариант `pooled byte[]`
 - ∞ Вариант выделения `native` памяти

Strings: Native → CLR

- ∞ Нужен ли нам `string` object?
 - ∞ `new string()` это нагрузка на GC
 - ∞ Interning, кэширование?
 - ∞ Достаточно hash code, equals, compare?
 - ∞ Можно реализовать прямо на `char*`
 - ∞ Частные хитрости
 - ∞ Потребовать уникальность хеша
 - ∞ Использовать metadata token вместо type full name

THE END