BASTA!
.NET, WINDOWS, VISUAL STUDIO

Besseres C#

Workshop

BASTA! Spring 2014

Async und Parallel

Workshop

Rainer Stropek

software architects gmbh

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

time cockpit

Saves the day.

Async Programming

In C# and .NET 4.5

Synchronous

```
private static void DownloadSomeTextSync()
{
  using (var client = new WebClient())
  {
    Console.WriteLine(
      client.DownloadString(new Uri(string.Format(
        "http://{0}",
        (Dns.GetHostAddresses("www.basta.net"))[0]))));
  }
}
```

IAsyncResult Pattern

```
private static void DownloadSomeText()
{
   var finishedEvent = new AutoResetEvent(false);

   // Notice the IAsyncResult-pattern here
   Dns.BeginGetHostAddresses("www.basta.net", GetHostEntryFinished,
     finishedEvent);
   finishedEvent.WaitOne();
}

private static void GetHostEntryFinished(IAsyncResult result)
{
   var hostEntry = Dns.EndGetHostAddresses(result);
   using (var client = new WebClient())
   {
     // Notice the Event-based asynchronous pattern here
     client.DownloadStringCompleted += (s, e) =>
     {
       Console.WriteLine(e.Result);
       ((AutoResetEvent)result.AsyncState).Set();
     };
     client.DownloadStringAsync(new Uri(string.Format(
       "http://{0}",
       hostEntry[0].ToString())));
   }
}
```

IAsyncResult Pattern

With Lambdas

```
private static void DownloadSomeText()
{
  var finishedEvent = new AutoResetEvent(false);

  // Notice the IAsyncResult-pattern here
  Dns.BeginGetHostAddresses(
    "www.basta.net",
    (result) =>
    {
      var hostEntry = Dns.EndGetHostAddresses(result);
      using (var client = new WebClient())
      {
        // Notice the Event-based asynchronous pattern here
        client.DownloadStringCompleted += (s, e) =>
        {
          Console.WriteLine(e.Result);
          ((AutoResetEvent)result.AsyncState).Set();
        };
        client.DownloadStringAsync(new Uri(string.Format(
          "http://{0}",
          hostEntry[0].ToString())));
      }
    },
    finishedEvent);
  finishedEvent.WaitOne();
}
```

TPL

Notice the use of the new Task Async Pattern APIs in .NET 4.5 here

```
private static void DownloadSomeTextUsingTask()
{
  Dns.GetHostAddressesAsync("www.basta.net")
    .ContinueWith(t =>
    {
      using (var client = new WebClient())
      {
        return client.DownloadStringTaskAsync(
          new Uri(string.Format(
            "http://{0}",
            t.Result[0].ToString())));
      }
    })
    .ContinueWith(t2 => Console.WriteLine(t2.Unwrap().Result))
    .Wait();
}
```

Rules For Async Method Signatures

- Method name ends with `Async`
- Return value
  `Task` if sync version has return type `void`
  `Task<T>` if sync version has return type `T`
- Avoid `out` and `ref` parameters
  Use e.g. `Task<Tuple<T1, T2, …>>` instead

Sync vs. Async

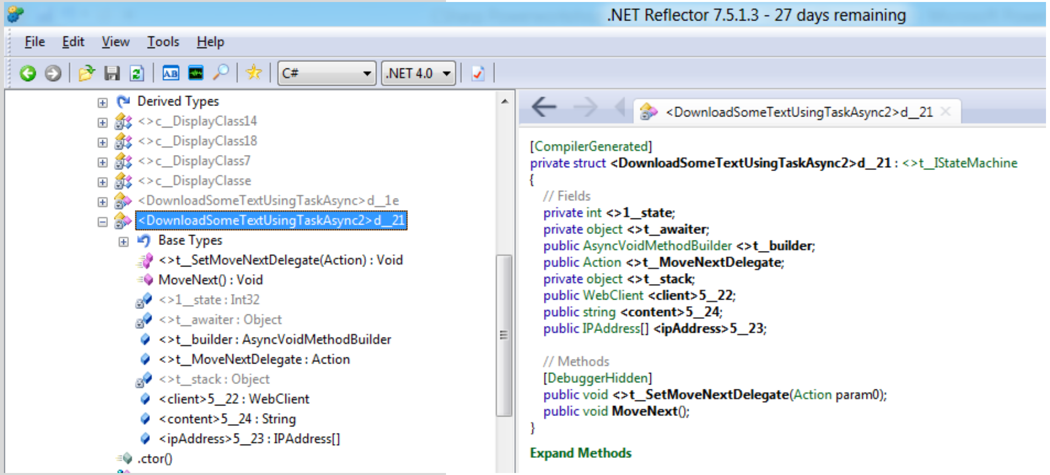
Notice how similar the sync and async versions are!

```
// Synchronous version
private static void DownloadSomeTextSync()
{
  using (var client = new WebClient())
  {
    Console.WriteLine(
      client.DownloadString(new Uri(string.Format(
        "http://{0}",
        (Dns.GetHostAddresses("www.basta.net"))[0]))));
  }
}

// Asynchronous version
private static async void DownloadSomeTextUsingTaskAsync()
{
  using (var client = new WebClient())
  {
    Console.WriteLine(
      await client.DownloadStringTaskAsync(new Uri(string.Format(
        "http://{0}",
        (await Dns.GetHostAddressesAsync("www.basta.net"))[0]))));
  }
}
```

Generated Code

```
private static async void DownloadSomeTextUsingTaskAsync2()
{
  using (var client = new WebClient())
  {
    try
    {
      var ipAddress = await Dns.GetHostAddressesAsync("www.basta.net");
      var content = await client.DownloadStringTaskAsync(
        new Uri(string.Format("htt://{0}", ipAddress[0])));
      Console.WriteLine(content);
    }
    catch (Exception)
    {
      Console.WriteLine("Exception!");
    }
  }
}
```


.NET Reflector 7.5.1.3 - 27 days remaining
File
Edit
View
Tools
Help
C#
.NET 4.0
<DownloadSomeTextUsingTaskAsync2>d__21

```
Derived Types
<>c__DisplayClass14
<>c__DisplayClass18
<>c__DisplayClass7
<>c__DisplayClasse
<DownloadSomeTextUsingTaskAsync>d__1e
<DownloadSomeTextUsingTaskAsync2>d__21
  Base Types
  <>t__SetMoveNextDelegate(Action) : Void
  MoveNext() : Void
  <>1__state : Int32
  <>t__awaiter : Object
  <>t__builder : AsyncVoidMethodBuilder
  <>t__MoveNextDelegate : Action
  <>t__stack : Object
  <client>5__22 : WebClient
  <content>5__24 : String
  <ipAddress>5__23 : IPAddress[]
.ctor()
```

```
[CompilerGenerated]
private struct <DownloadSomeTextUsingTaskAsync2>d__21 : <>t__IStateMachine
{
  // Fields
  private int <>1__state;
  private object <>t__awaiter;
  public AsyncVoidMethodBuilder <>t__builder;
  public Action <>t__MoveNextDelegate;
  private object <>t__stack;
  public WebClient <client>5__22;
  public string <content>5__24;
  public IPAddress[] <ipAddress>5__23;

  // Methods
  [DebuggerHidden]
  public void <>t__SetMoveNextDelegate(Action param0);
  public void MoveNext();
}
```

Expand Methods

Guidelines for `async/await`

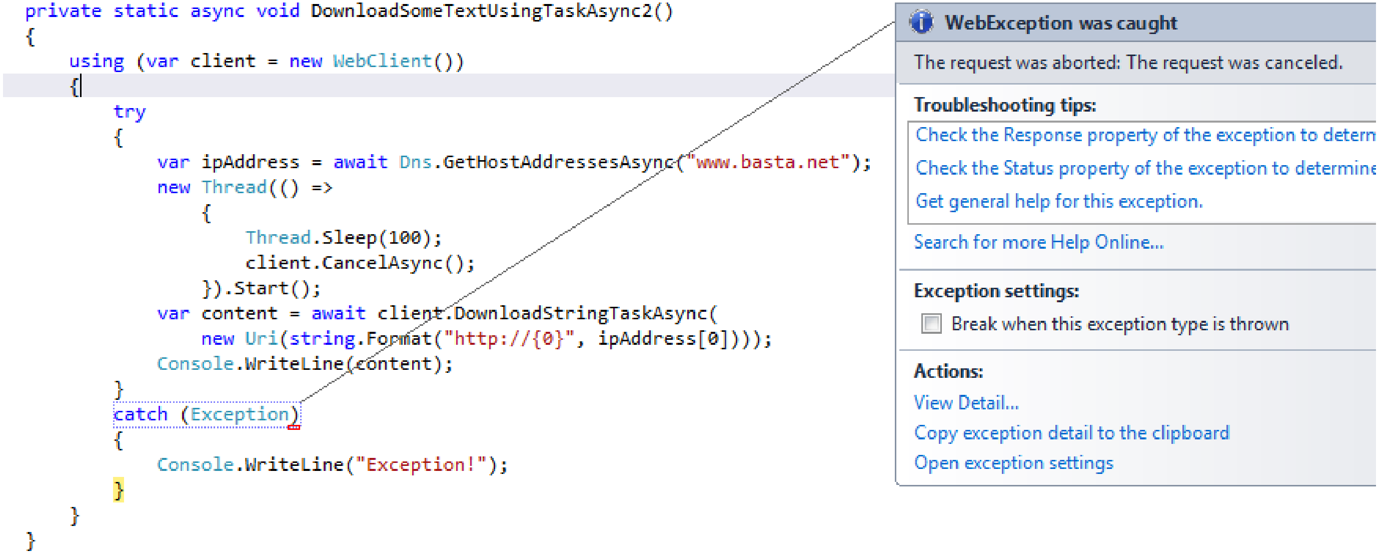
- If `Task` ended in `Canceled` state, `OperationCanceledException` will be thrown

TPL

TaskCompletionSource<T>

```
private async static void CancelTask()
{
   try
   {
      var cancelSource = new CancellationTokenSource();
      var result = await DoSomethingCancelledAsync(cancelSource.Token);
      Console.WriteLine(result);
   }
   catch (OperationCanceledException)
   {
      Console.WriteLine("Cancelled!");
   }
}

private static Task<int> DoSomethingCancelledAsync(CancellationToken token)
{
   // For demo purposes we ignore token and always return a cancelled task
   var result = new TaskCompletionSource<int>();
   result.SetCanceled();
   return result.Task;
}
```



```
private static async void DownloadSomeTextUsingTaskAsync2()
{
    using (var client = new WebClient())
    {
        try
        {
            var ipAddress = await Dns.GetHostAddressesAsync("www.basta.net");
            new Thread(() =>
                {
                    Thread.Sleep(100);
                    client.CancelAsync();
                }).Start();
            var content = await client.DownloadStringTaskAsync(
                new Uri(string.Format("http://{0}", ipAddress[0])));
            Console.WriteLine(content);
        }
        catch (Exception)
        {
            Console.WriteLine("Exception!");
        }
    }
}
```

WebException was caught

The request was aborted: The request was canceled.

Troubleshooting tips:

Check the Response property of the exception to determ

Check the Status property of the exception to determine

Get general help for this exception.

Search for more Help Online...

Exception settings:

☐ Break when this exception type is thrown

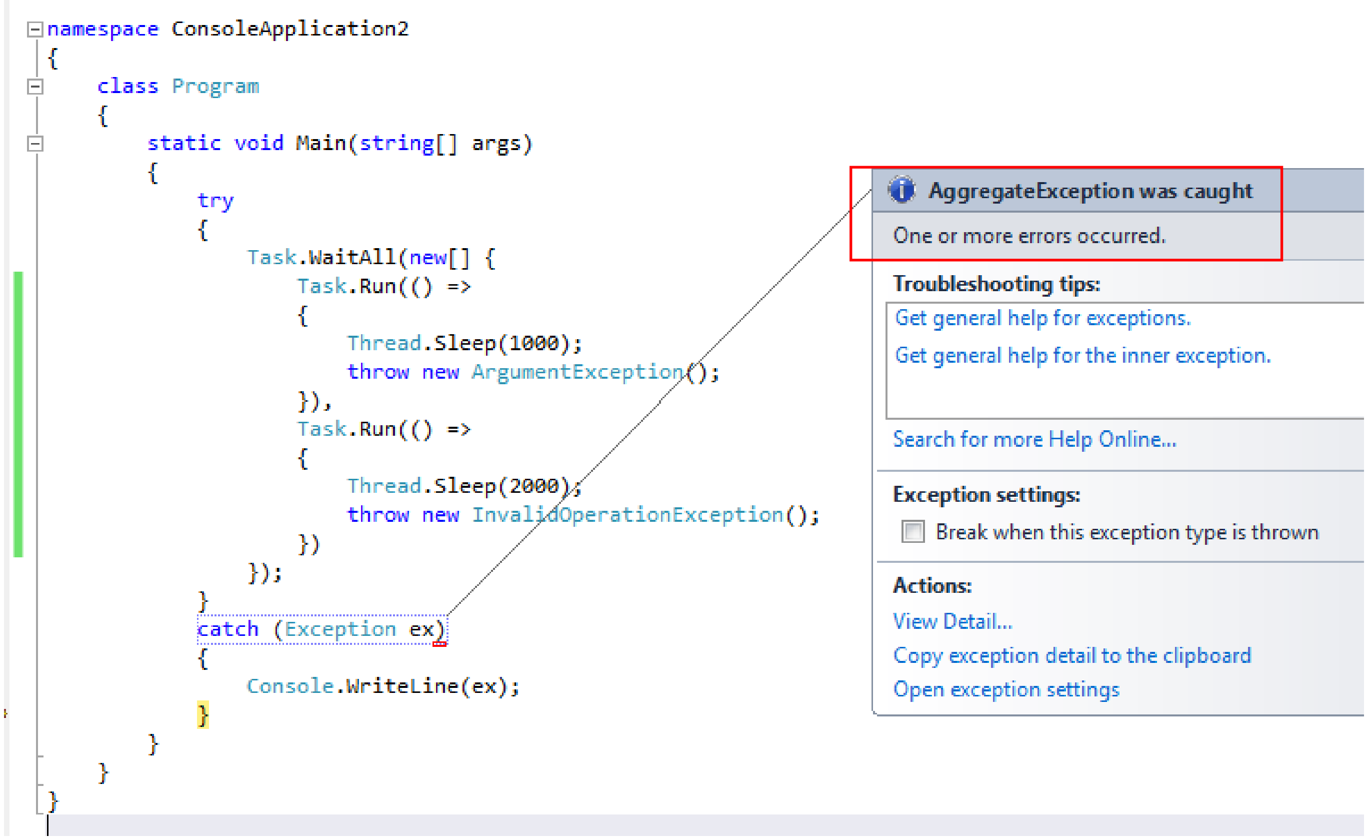
Actions:

View Detail...

Copy exception detail to the clipboard

Open exception settings

Note that async API of `WebClient` uses existing cancellation logic instead of `CancellationTokenSource`



```
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Task.WaitAll(new[] {
                    Task.Run(() =>
                    {
                        Thread.Sleep(1000);
                        throw new ArgumentException();
                    }),
                    Task.Run(() =>
                    {
                        Thread.Sleep(2000);
                        throw new InvalidOperationException();
                    })
                });
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
}
```

AggregateException was caught

One or more errors occurred.

Troubleshooting tips:

Get general help for exceptions.

Get general help for the inner exception.

Search for more Help Online...

Exception settings:

☐ Break when this exception type is thrown

Actions:

View Detail...

Copy exception detail to the clipboard

Open exception settings

Guidelines for `async/await`

- Caller runs in parallel to awaited methods
- Async methods sometimes do not run async (e.g. if task is already completed when **`async`** is reached)

Guidelines for `async/await` (UI Layer)

- `async/await` use `SynchronizationContext` to execute the awaiting method → UI thread in case of UI layer
- Use `Task.ConfigureAwait` to disable this behavior
  E.g. inside library to enhance performance

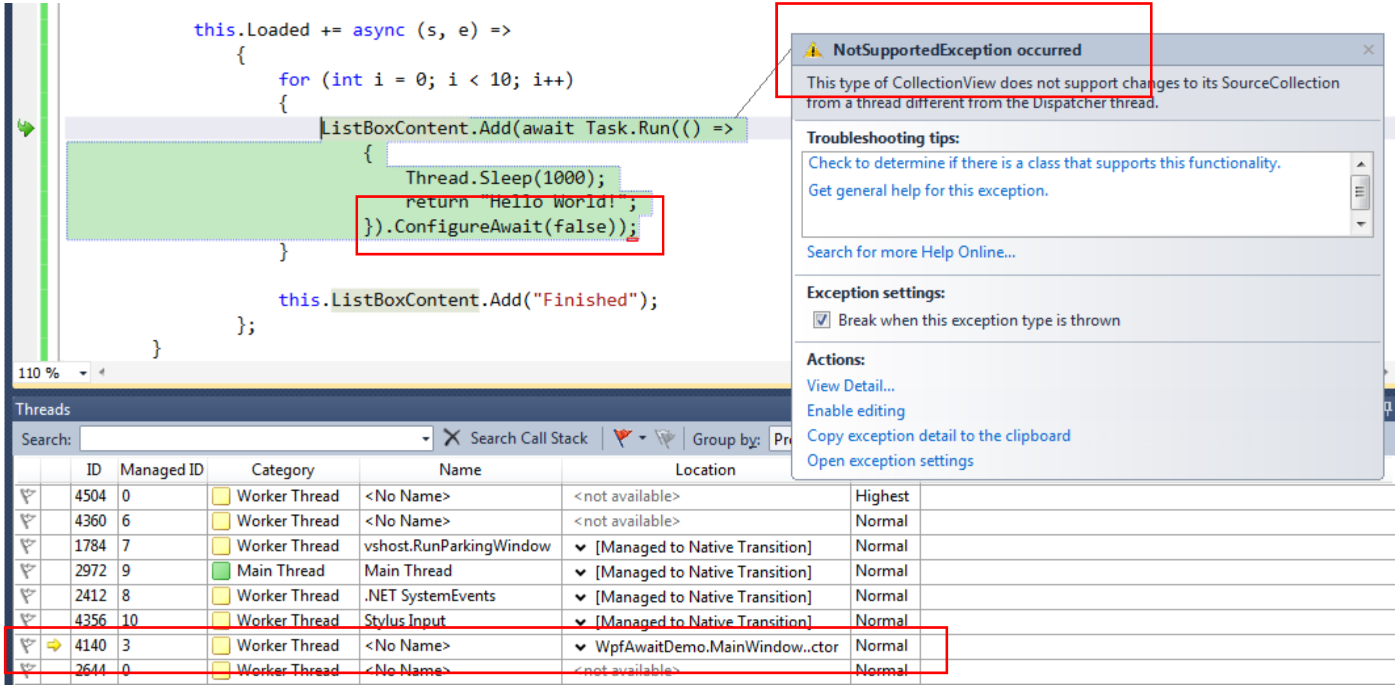
Async/await im UI

```
public partial class MainWindow : Window
{
public MainWindow()
{
  this.DataContext = this;
  this.ListBoxContent = new ObservableCollection<string>();
  this.InitializeComponent();
  this.ListBoxContent.Add("Started");

  this.Loaded += async (s, e) =>
     {
       for (int i = 0; i < 10; i++)
       {
          ListBoxContent.Add(await Task.Run(() =>
            {
               Thread.Sleep(1000);
               return "Hello World!";
            }));
       }

       this.ListBoxContent.Add("Finished");
     };
}

public ObservableCollection<string> ListBoxContent { get; private set; }
```



```
this.Loaded += async (s, e) =>
{
    for (int i = 0; i < 10; i++)
    {
        ListBoxContent.Add(await Task.Run(() =>
        {
            Thread.Sleep(1000);
            return "Hello World!";
        }).ConfigureAwait(false));
    }

    this.ListBoxContent.Add("Finished");
};
}
```

NotSupportedException occurred

This type of CollectionView does not support changes to its SourceCollection from a thread different from the Dispatcher thread.

Troubleshooting tips:

Check to determine if there is a class that supports this functionality.

Get general help for this exception.

Search for more Help Online...

Exception settings:

☑ Break when this exception type is thrown

Actions:

View Detail...

Enable editing

Copy exception detail to the clipboard

Open exception settings

	ID	Managed ID	Category	Name	Location	
	4504	0	Worker Thread	<No Name>	<not available>	Highest
	4360	6	Worker Thread	<No Name>	<not available>	Normal
	1784	7	Worker Thread	vshost.RunParkingWindow	[Managed to Native Transition]	Normal
	2972	9	Main Thread	Main Thread	[Managed to Native Transition]	Normal
	2412	8	Worker Thread	.NET SystemEvents	[Managed to Native Transition]	Normal
	4356	10	Worker Thread	Stylus Input	[Managed to Native Transition]	Normal
➡	4140	3	Worker Thread	<No Name>	WpfAwaitDemo.MainWindow..ctor	Normal
	2644	0	Worker Thread	<No Name>	<not available>	Normal

Guidelines For Implementing Methods Ready For `async/await`

- Return `Task/Task<T>`
- Use postfix `Async`
- If method support cancelling, add parameter of type `System.Threading.CancellationToken`
- If method support progress reporting, add `IProgress<T>` parameter
- Only perform very limited work before returning to the caller (e.g. check arguments)
- Directly throw exception only in case of *usage* errors

Progress Reporting

```
public class Program : IProgress<int>
{
  static void Main(string[] args)
  {
    var finished = new AutoResetEvent(false);
    PerformCalculation(finished);
    finished.WaitOne();
  }

  private static async void PerformCalculation(AutoResetEvent finished)
  {
    Console.WriteLine(await CalculateValueAsync(
      42,
      CancellationToken.None,
      new Program()));
    finished.Set();
  }

  public void Report(int value)
  {
    Console.WriteLine("Progress: {0}", value);
  }
```

Cancellation

```
    private static Task<int> CalculateValueAsync(
       int startingValue,
       CancellationToken cancellationToken,
       IProgress<int> progress)
    {
       if (startingValue < 0)
       {
          // Usage error
          throw new ArgumentOutOfRangeException("startingValue");
       }

       return Task.Run(() =>
          {
             int result = startingValue;
             for (int outer = 0; outer < 10; outer++)
             {
                cancellationToken.ThrowIfCancellationRequested();

                // Do some calculation
                Thread.Sleep(500);
                result += 42;

                progress.Report(outer + 1);
             }

             return result;
          });
    }
 }
```

Cancellation

```
private static async void PerformCalculation(AutoResetEvent
finished)
{
  try
  {
    var cts = new CancellationTokenSource();
    Task.Run(() =>
      {
        Thread.Sleep(3000);
        cts.Cancel();
      });
    var result = await CalculateValueAsync(
      42,
      cts.Token,
      new Program());
  }
  catch (OperationCanceledException)
  {
    Console.WriteLine("Cancelled!");
  }

  finished.Set();
}
```

Task.FromResult

```
private static Task<int> CalculateValueAsync(
  int startingValue,
  CancellationToken cancellationToken,
  IProgress<int> progress)
{
  if (startingValue < 0)
  {
    // By definition the result has to be 0 if startingValue < 0
    return Task.FromResult(0);
  }

  return Task.Run(() =>
    {
      […]
    });
}
```

Note how *Task.FromResult* is used to return a pseudo-task

Note that you could use *TaskCompletionSource* instead

Async Web API

```
namespace MvcApplication2.Controllers
{
    public class BlogController : ApiController
    {
        // GET api/values/5
        public async Task<BlogItem> Get(int id)
        {
            // Open context to underlying SQL database
            using (var context = new BlogContext())
            {
                // Make sure that it contains database
                await context.GenerateDemoDataAsync();

                // Build the query
                var blogs = context
                    .BlogItems
                    .Where(b => b.BlogId == id);

                // Execute query
                return await blogs.FirstOrDefaultAsync();
            }
        }
    }
}
```

Async Unit Test

```
namespace MvcApplication2.Tests.Controllers
{
    [TestClass]
    public class BlogControllerTest
    {
        [TestMethod]
        public async Task GetById()
        {
            BlogController controller = new BlogController();

            var result = await controller.Get(1);
            Assert.IsNotNull(result);

            result = await controller.Get(99);
            Assert.IsNull(result);
        }
    }
}
```

BASTA! Spring 2014

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

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.