Einfacher Logger mit C#

Wer kennt es nicht, in der Entwicklungsumgebung funktioniert die programmierte Software einwandfrei, auf einem anderen Testrechner aus unbekannten Gründen leider nicht mehr.

Um den Fehler schnell zu finden bieten sich Logfiles an. Gut dass .Net von Haus aus die Klasse Trace im namespace System.Diagnostics eine Lösung bietet.

Eine kleines Beispiel:

1. in der App.config den Tracer hinzufügen:
   [crayon-67679ef003f65090119069/]

2. Neue statische Classe erstellen:
   [crayon-67679ef003f6d050467774/]
   Nun kann man mit z.B: Logger.Info(„Programmstart“,“Main“) im Programmcode Informationen an die application.log Datei anhängen. Das tolle auch das Datum wird nun mit übertragen.

Eigenen Event erstellen

Ein Event oder deutsch Ereignis ist immer an einen Delegatentypen gebunden, welcher wiederum eine bestimmte Signatur von (Rückgabewert + Parameter).

Ungefähr, stellt es dieses Muster da:

Zuerst brauchen wir einen delegaten. Jetzt stellt sich die Frage, die Methoden, auf die der Delegat später verweist, wie soll die aussehen? soll sie einen Rückgabewert haben? Soll

irgend ein Wert an die Methode übergeben werden? Ja? Dann brauchen wir Parameter. Um das noch ein wenig verständlicher zu machen, wenn wir eine for-Schleife haben und wollen, dass das Event nach jedem Schleifendurchgang immer abgefeuert wird, und uns den Wert liefert. Dann würden wir einen Delegaten mit einem Parameter vom Typ int z.B. machen.
[crayon-67679ef004654564882273/]

Dann braucht man das Event an sich. Dieser muss vom Typ des Delegaten sein.
[crayon-67679ef004658268666965/]

Jetzt muss gesagt werden, wann das Event abgefeuert werden muss. Und, wichtig, wir haben dem Delegaten einen Parameter (int a) angegeben. Dieser Parameter muss nun durch das Event gefüllt werden. Das machen wir, indem wir
[crayon-67679ef00465a818105168/]
Jetzt wird bei jedem Schleifendurchgang das Event abgefeuert.

Ok, das war unsere Quell klasse.

Jetzt erstellen wir ein Objekt dieser Klasse und weisen dem Event eine Methode zu, die ausgeführt werden soll, sobald das Event abgefeuert wird. Die Methode muss die selbe Signatur haben wie unser Delegat. Zum Schluss müssen wir die Methode in der Quell klasse ausführen und das Event wird abgefeuert und führt ebenso die Methode mit dem Parameter aus, die wir in der Ziel Methode eingegeben haben.
[crayon-67679ef00465b684627632/]
[crayon-67679ef00465d774196271/]

Download Beispielprojekt Eigener Event

Delegaten in .NET

Das .Net Framework beinhaltet bereits Delegaten, welche bestimmte Signaturen haben.

1. Der EventHandler

stellt einen Delegaten dar, der keinen Rückgabewert hat (void), und 2 Parameter hat. 1. Ist der object sender , welcher das Senderobject implementiert und EventArgs[] welcher evtl. Argumente beinhalten kann. Diesen Delegaten findet man, wenn man zu einem Event von Steuerelementen eine Methode aufruft.

2. Action<>

Der Action Delegat besitzt keinen Rückgabewert, wohl aber bis zu 16 Parameterelemente. Dieser Delegat kann z.B. dazu verwendet werden um eine Methode mit der Task Klasse aufzurufen und diese im neuen Thread zu starten.
[crayon-67679ef004813933697491/]
Das geht natürlich auch mit anonymen Methoden
[crayon-67679ef004817665558693/]
3. Func<>

Der Func Delegat ist dem Action<> Delegaten gleich, liefert im Gegensatz zu ihm aber einen Rückgabewert

Stopwatch – Messung von Zeit von bestimmten Algorithmen oder Operationen

Manchmal möchte man wissen, wie lange mein Computer braucht um einen bestimmten Code auszuführen. Hierzu könnte man natürlich DateTime.Now für die Operationen verwenden. Allerdings ist dieser Wert ungenau. Einen Genaueren Wert erhalten wir mit dem Stopwatch. Dieser funktioniert aus recht simpel:
[crayon-67679ef004a63521560172/]
[crayon-67679ef004a6a920804264/]

C# Vergleichsoperatoren ? : und ??

Oft findet man im Code ewig lange if – else Verzweigungen, die zuviel Platz einnehmen und irgendwann die Übersicht rauben.

Die beiden Operatoren ? : und ?? sehen merkwürdig aus, sind aber in der Handhabung sehr praktikabel.

Der ? : prüft ob der Wert true ist. Falls ja, zählt der linke Wert, wenn nicht, zählt der Wert rechts davon.
[crayon-67679ef004c27895198467/]
Der ?? Operator prüft, ob der Wert null ist. Bei nein, zählt der linke Wert, bei ja der rechte.
[crayon-67679ef004c2b871770104/]

Das ganze in lang könnte so aussehen:

[crayon-67679ef004c2d786748161/]
[crayon-67679ef004c2e480067221/]
Ab C# 6 soll noch das Feature hinzufügen, dass auch das „Elternobjekt“ geprüft werden kann, ob dieser nicht null ist.
[crayon-67679ef004c30439901892/]

TSQL Mod – Eine DLL für schnelle SQL Server arbeiten

TSQLmod Download

Klasse db (Connection String)

benötigt in erster Linie die SQL Instanz. Nachdem das Objekt erfolgreich initialisiert wurde, wird auch gleichzeitig die Verbindung aufgemacht und die folgenden Methoden können genutzt werden.

LookUP(…)

gibt aus einem SQL Query den ersten Treffer der angegebenen Spalte als String wieder. Ideal um einen Wert aus der Datenbank auszulesen. Möglich ist es entweder die Spaltennummer oder den Spaltennamen anzugeben.

getRowList(…)

gibt eine List<string> oder generische List<T> von der angegebenen Spalte zurück. Man erhält quasi aus dem Select eine gewünschte Spalte

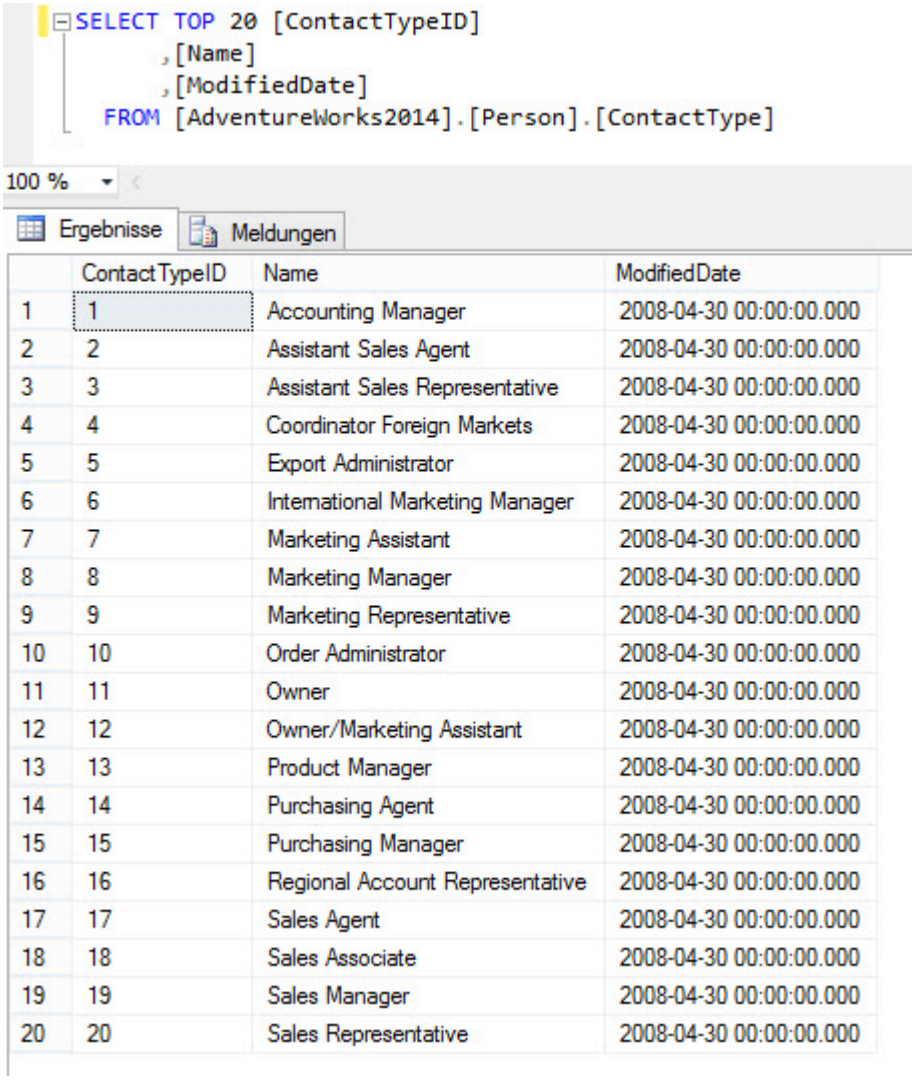
getRowStringBuilder(…)

ähnlich wie die getRowList(…) ist der Rückgabewert aber ein Stringbuilder, in welchen alle Zeilen einer selektierten Spalte enthalten sind.

getDynamicList(…)

erfordert eine Klasse welche dieselben Datentypen und Bezeichnung hat wie das SQL Select. Als Rückgabe erhält man man eine List<meineKlasse>, welche 1:1 so viele Elemente und Spalten hat wie das Sql Query. Das ganze arbeitet nicht mit Reflektionen, sondern nach dem Prinzip von diesem genialen Autor: KLICK

Dazu jeweils ein Beispiel. Ausgehend vom folgenden Select:



```
SELECT TOP 20 [ContactTypeID]
      ,[Name]
      ,[ModifiedDate]
  FROM [AdventureWorks2014].[Person].[ContactType]
```

100 %

Ergebnisse | Meldungen

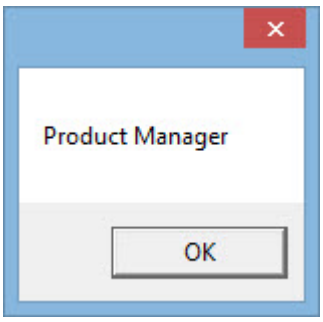
	ContactTypeID	Name	ModifiedDate
1	1	Accounting Manager	2008-04-30 00:00:00.000
2	2	Assistant Sales Agent	2008-04-30 00:00:00.000
3	3	Assistant Sales Representative	2008-04-30 00:00:00.000
4	4	Coordinator Foreign Markets	2008-04-30 00:00:00.000
5	5	Export Administrator	2008-04-30 00:00:00.000
6	6	International Marketing Manager	2008-04-30 00:00:00.000
7	7	Marketing Assistant	2008-04-30 00:00:00.000
8	8	Marketing Manager	2008-04-30 00:00:00.000
9	9	Marketing Representative	2008-04-30 00:00:00.000
10	10	Order Administrator	2008-04-30 00:00:00.000
11	11	Owner	2008-04-30 00:00:00.000
12	12	Owner/Marketing Assistant	2008-04-30 00:00:00.000
13	13	Product Manager	2008-04-30 00:00:00.000
14	14	Purchasing Agent	2008-04-30 00:00:00.000
15	15	Purchasing Manager	2008-04-30 00:00:00.000
16	16	Regional Account Representative	2008-04-30 00:00:00.000
17	17	Sales Agent	2008-04-30 00:00:00.000
18	18	Sales Associate	2008-04-30 00:00:00.000
19	19	Sales Manager	2008-04-30 00:00:00.000
20	20	Sales Representative	2008-04-30 00:00:00.000

einem erstellten Objekt der Klasse db:
[crayon-67679ef004e01889138730/]
und ein string mit folgendem select:
[crayon-67679ef004e04296029776/]

LookUp(…)

[crayon-67679ef004e06191181894/]

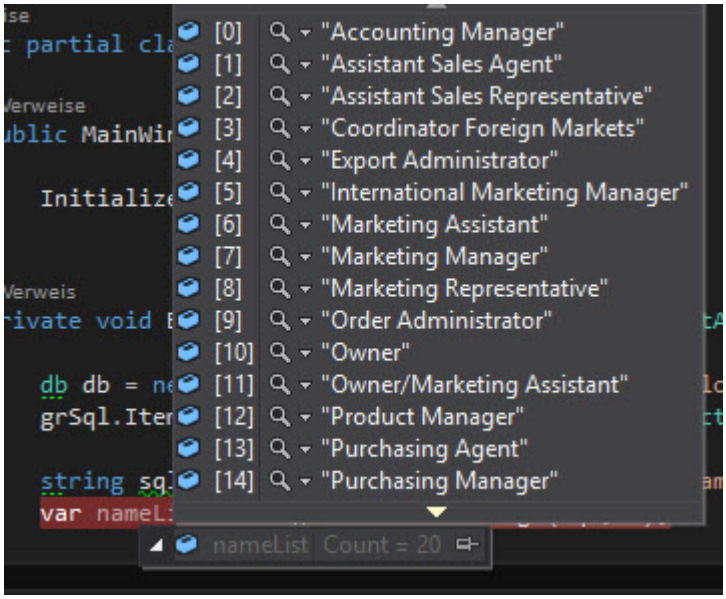
Antwort:



getRowList(…)

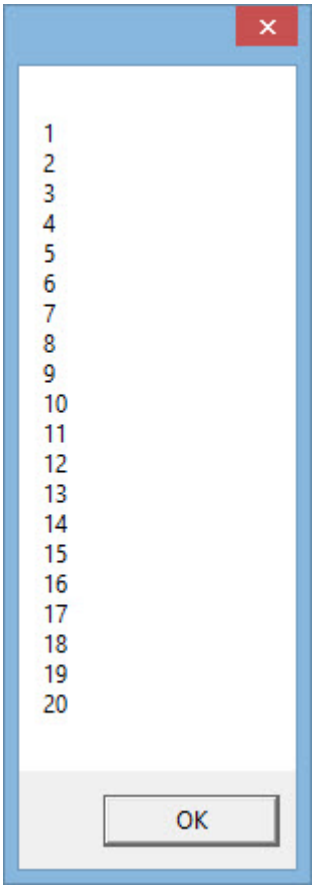
[crayon-67679ef004e07151479385/]

Antwort:



getRowStringBuilder(…)

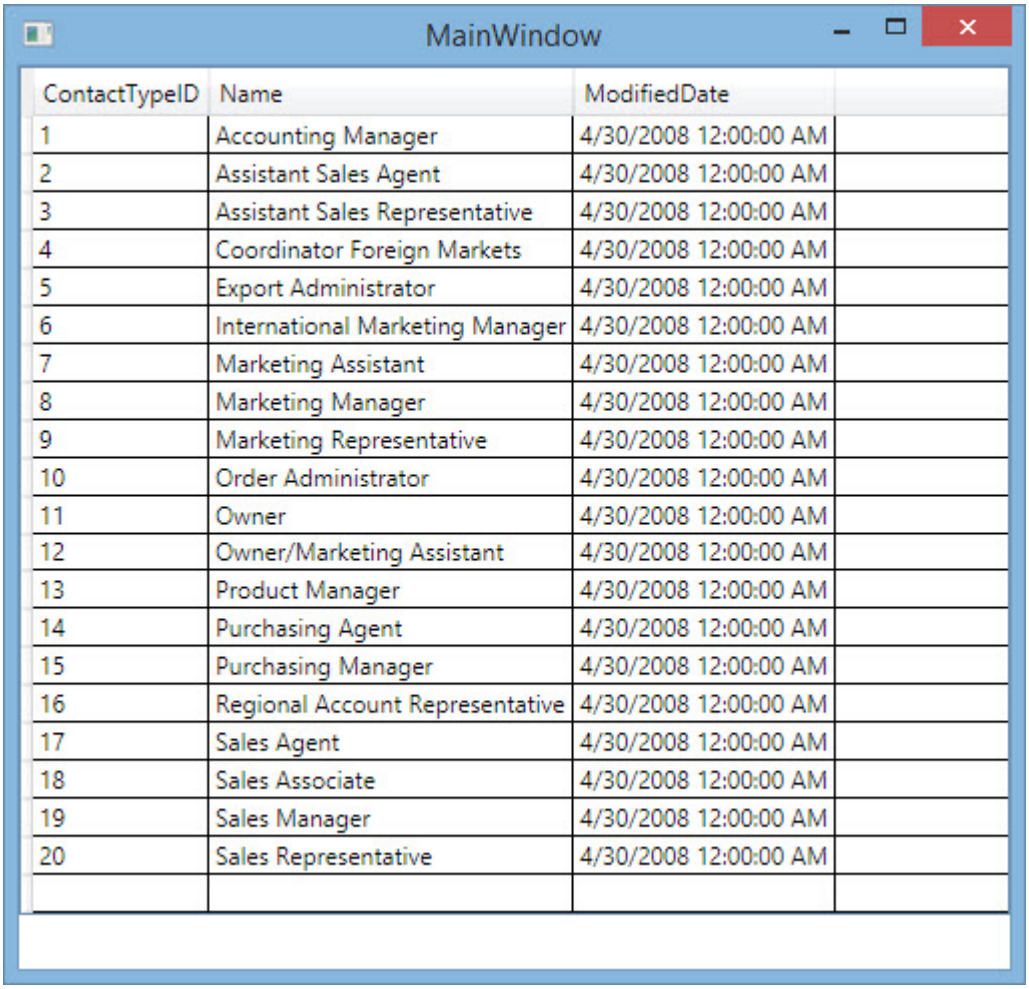
[crayon-67679ef004e08546884695/]

Antwort: 

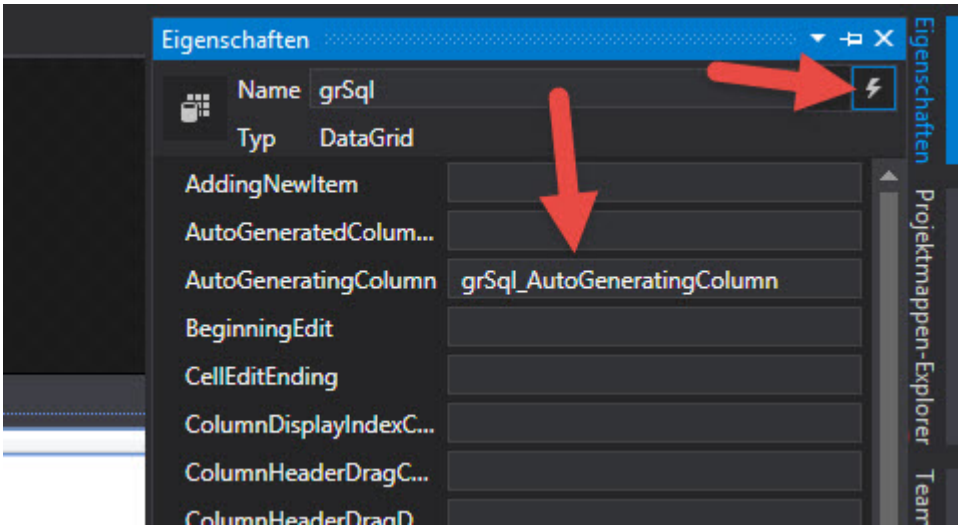
getDynamicList(…)

Wie oben bereits erwähnt, ist hierfür eine Klasse mit Propertys notwendig. Diese kann man ganz einfach auch mit den Methoden aus CreateClass – Klasse erstellen. Dazu weiter unten.
[crayon-67679ef004e09524947188/]
dann kann man so eine dynamische Liste ganz einfach erstellen:
[crayon-67679ef004e0b084412000/]

diese Liste kann man nun z.B. einem Datagrid aus WPF zuordnen:
[crayon-67679ef004e0d809392797/]
[crayon-67679ef004e0e890591789/]
Das ganze sieht dann so aus:



wem das Datumsformat stört, der kann dem Ereignis AutogeneratingColumn aus dem Datagrid eine Änderung des Datumsformates durchführen:



folgendes soll nun passieren, wenn das Ereignis eintrifft:

[crayon-67679ef004e0f168087215/]
Nun sieht das ganze so aus:

ContactTypeID	Name	ModifiedDate
1	Accounting Manager	30.04.2008
2	Assistant Sales Agent	30.04.2008
3	Assistant Sales Representative	30.04.2008
4	Coordinator Foreign Markets	30.04.2008
5	Export Administrator	30.04.2008
6	International Marketing Manager	30.04.2008
7	Marketing Assistant	30.04.2008
8	Marketing Manager	30.04.2008
9	Marketing Representative	30.04.2008
10	Order Administrator	30.04.2008
11	Owner	30.04.2008
12	Owner/Marketing Assistant	30.04.2008
13	Product Manager	30.04.2008
14	Purchasing Agent	30.04.2008
15	Purchasing Manager	30.04.2008
16	Regional Account Representative	30.04.2008
17	Sales Agent	30.04.2008
18	Sales Associate	30.04.2008
19	Sales Manager	30.04.2008
20	Sales Representative	30.04.2008

XAML Assembly Version an window Title binden

Um die Assembly Version irgendwo im Programm optisch darzustellen, kann man diese z.B. im Titelbereich des windows anzeigen lassen.

1. in app.xaml.cs folgende Eigenschaft hinzufügen:
[crayon-67679ef00506a855580629/]
2. Der XAML Code zum binden sieht dann so aus

[crayon-67679ef00506e097772062/]

Methode mit unendlich vielen Parameter mit params

Hat man den Fall, dass man an eine Methode unendlich viele Parameter übergeben möchte oder die Anzahl vielleicht unbekannt ist, so bietet C# die Möglichkeit über params und ein Array unendlich viele Parameter zu hinterlegen. Ein Beispiel:
[crayon-67679ef0051fc390851720/]
Zu beachten ist, dass der Typ unbedingt ein array [] sein muss.

HttpListenerContext decode/encode Umlaute

Liest man die Url aus dem HttpListenerContext, die Umlaute wie äöü enthält, so sieht das ungefähr so aus:

aus süß wird s%FC%df.

Abhilfe schaft da die Klasse HttpUtility:
[crayon-67679ef005378130678147/]
möchte man zurück encodieren macht man einfach :
[crayon-67679ef00537d537478780/]

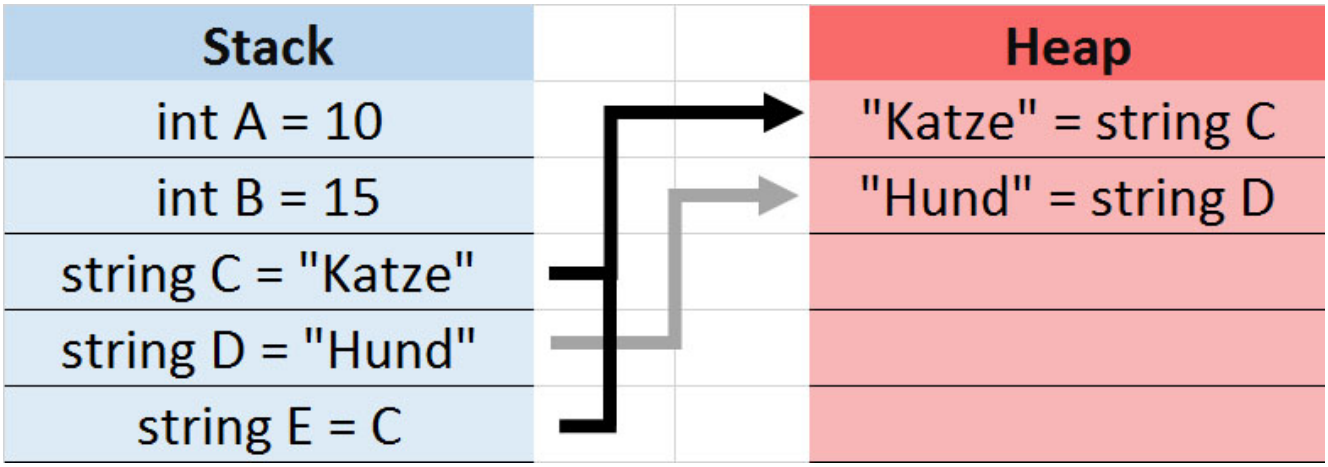
Referenztypen, Wertetypen

Stack und Heap sind Teil des Arbeitsspeichers, die zu mindestens unter C# vom Kompiler automatisch verwaltet wird.

Beim **Stack** (dt. Stapel) werden die Daten quasi aufeinander gelegt. Wenn Speicher freigegeben werden kann, so wird dies auch von oben heraus getan (LIFO Prinzip). Durch dieses Prinzip ist der Stack sehr schnell in seiner Arbeitsweise. Verlässt der Programmcode die geschwungene Klammer { }, werden sämtliche innerhalb angelegte Variablen aus dem Stack entfernt. Obwohl der Speicher sehr gering ist, wächst und schrumpft er während des Programmablaufs.

Der **Heap** ist nicht so strukturiert wie der Stack, dort liegen die Daten quasi durcheinander und werden vom Stack aus per Zeiger gefunden. Deswegen ist der Heap auch deutlich langsamer in seiner Arbeitsweise.

Was in den Stack und was in den Heap gelangt, bestimmen unter anderem die Datentypen. Weiter beinhaltet der Stack auch die Zeiger der Variablen auf den Heap.



Stack – Wertetypen:

• Alle numerischen Datentypen
• Boolean, Char und Date
• Alle Strukturen, auch wenn ihre Member Verweistypen sind
• Enumerationen, da der zugrunde liegende Typ immer SByte, Short, Integer, Long, Byte, UShort, UInteger oder ULong ist

Heap – Referenztypen, Verweistypen:

• String
• Alle Arrays, auch wenn ihre Elemente Wertetypen sind
• Klassentypen, z.B. Form
• Delegaten
• reine Objekte

Das Verschieben der Daten vom Stack (Wertetyp) zu Heap (Referenztyp) bezeichnet man als **boxing**
[crayon-67679ef005519681168714/]
und vom Heap zu Stack als **unboxing**
[crayon-67679ef00551d623365399/]

Auch beim **casten** können die Daten vom Heap zum Stack geschoben werden:
[crayon-67679ef00551e226411622/]

Quelle:

Understanding Boxing and Unboxing