

A decorative graphic on the left side of the slide, consisting of a grid of squares in various shades of gray and blue, arranged in a pattern that tapers off to the right.

# .NET: Assemblys

© J. Heinzelreiter  
Version 5.5

# Was sind Assemblys?

kann man mit Java Archiven vergleichen, Java Archive ist aber einfacher aufgebaut, sowas wie ein zip Archiv, Metadaten sind auch enthalten;  
bei Assemblys genauer definiert

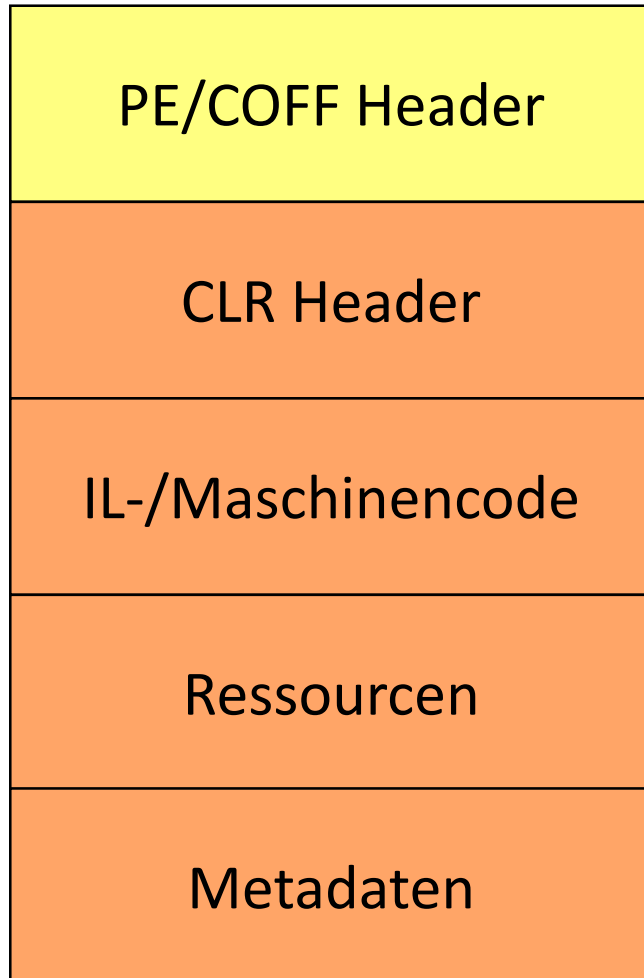
- Eine Assembly fasst folgende Daten zu einer *logischen Einheit* zusammen:
  - Code: Ausführbarer IL-Code.
  - Metadaten: Selbstbeschreibung der Assembly.
  - Ressourcen: Strings, Icons, Bilder, ...
- Ressourcen können in Assembly eingebettet sein oder auf externe Dateien verweisen.

Assembly muss nicht in form von einer einzigen physischen Einheit vorkommen.
- Code und Metadaten können auf mehrere *Module* verteilt sein.

Multimodule assemblies, nicht verbreitet, Visual Studio unterstützt das nicht.
- Assembly enthält ein *Manifest*: „Inhaltsverzeichnis“ der Assembly.

# Aufbau von Modulen

Assembly aus einem oder mehreren Modulen



- PE/COFF :
  - Standard Objekt-Format von Windows.
- CLR-Header:
  - Versionsnummer von benötigter CLR, mit welcher Version wurde übersetzt?
  - Einsprungspunkt, Main(), was muss ich als erstes ausführen?
  - Referenz auf Metadaten.
- Code:
  - üblicherweise CIL-Format,
  - kann aber auch „vorkompiliert“ sein. dann Maschinencode
- Metadaten
  - Beschreibung der definierten und Verweise auf referenzierte Typen.

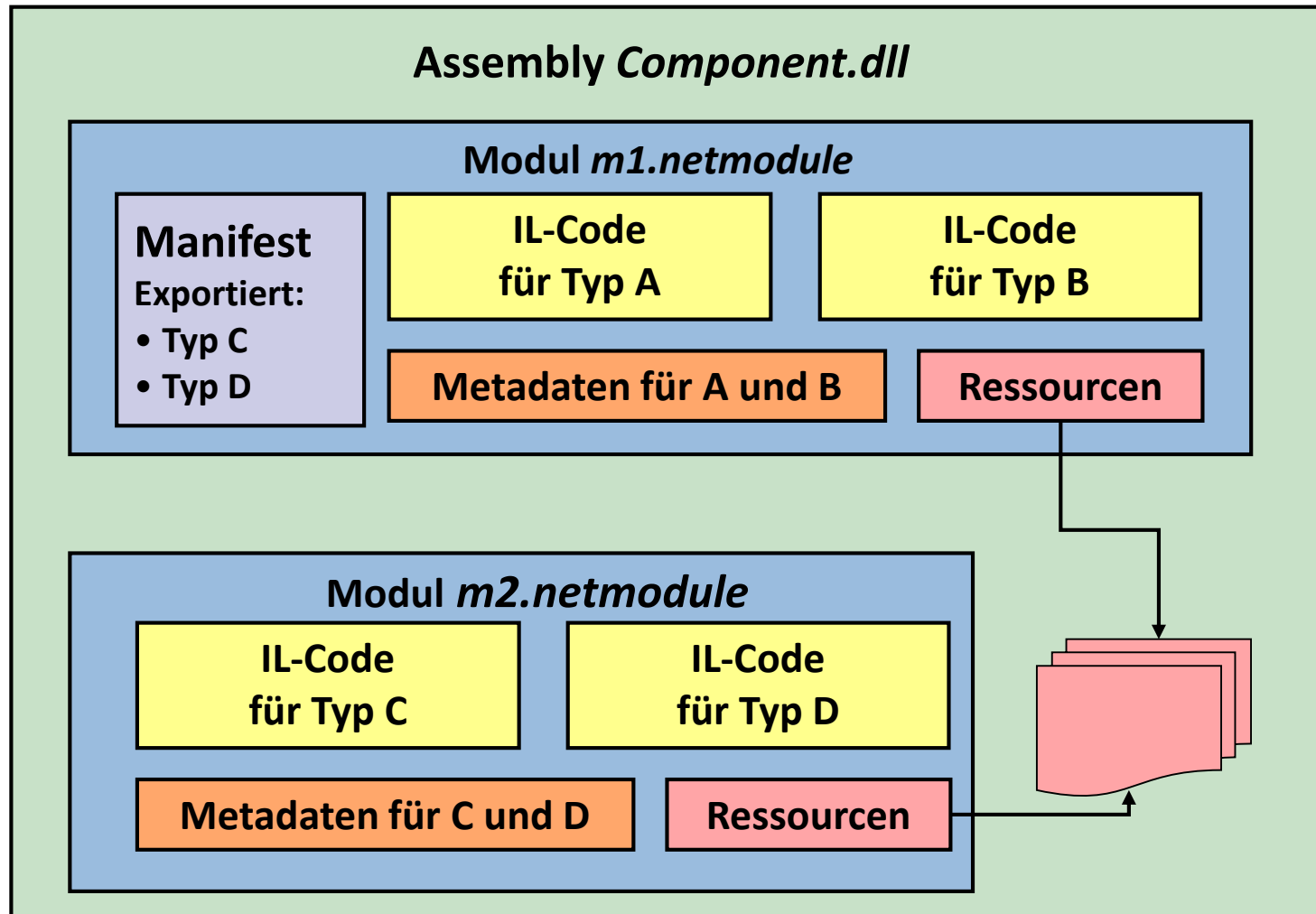
# Metadaten

- Struktur der Metadaten
  - Definitionstabellen (pro Modul)
    - **TypeDef**: definierte Typen.
    - **Field**: Typ und Attribute (Zugriffsrechte, ...) der Datenkomponenten.
    - **Method**: Signatur, Attribute, Parameterliste der Methoden, Verweis auf IL-Code.
  - Referenztabellen (pro Modul)
    - **AssemblyRef**: Verweis auf referenzierte Assemblys.
    - **ModuleRef** : Verweise auf „Nebenmodule“.
    - **TypeRef**: referenzierte Typen in „Nebenmodulen“ und anderen Assemblys.
  - Manifesttabellen (nur in Hauptmodul)

# Aufbau von Assemblys

Theoretisch könnten die Module in verschiedenen Sprachen entwickelt werden und zu einem Assembly kombiniert werden

oder manche Typen aus dem 2. Assembly werden nur sehr selten benötigt -> nur nachladen wenn man es braucht



selber Ordner in dem das Hauptprogramm ist, dort findet man auch die Private Assemblies -> Assemblies die nur von diesem Programm verwendet werden.

# Private Assemblies

- Assemblies werden durch Kopieren installiert (keine Einträge in Registry).
- *Private Assemblies* werden von **einer Anwendung** benutzt.  
Versionierung spielt keine Rolle bei Private Assemblies!!!
- Installationsort von *privaten Assemblies*:
  - Selbes Verzeichnis wie Anwendung.
  - Unterverzeichnis mit Namen der Assembly. sinnvoll wenn Assembly mehrere physische Einheiten hat
  - Unterverzeichnis des Anwendungsverzeichnisses: Konfigurations-Datei *Anwendung.exe.config* enthält Suchpfad. kann man Einträge für die Runtime machen  
zb. : schau auch unter bin;obj

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="bin;obj"/>
    </assemblyBinding>
  </runtime>
</configuration>
```

# Öffentliche (Shared) Assemblies

- Öffentlichen Assemblys wird ein *Strong Name* zugewiesen.
- Ein *Strong Name* besteht aus folgenden Teilen:
  - *Name der Komponente*
  - *öffentlicher Schlüssel*: Identifiziert Komponenten einer Firma.
  - *Culture*: Sprache/Land, z.B. neutral, „en-US“, „de-AT“.
  - *Version*: *<Hauptversion> . <Nebenversion> . <Buildnr.> . <Ild. Nr>*
  - Beispiel: `"System, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"`
- Öffentliche Assemblys werden im *Global Assembly Cache* (GAC) installiert (nur im Full Framework).
- Jede Anwendung ist fest an bestimmte Assembly-Versionen gebunden.
- *Side-by-Side Execution*: CLR kann mehrere Versionen eines Assemblys verwalten und bei Bedarf auch gleichzeitig laden.

Versionierung sehr wichtig!!!

Zur Verwendung in mehreren sln.

Shared Assemblys werden im Global Assembly Cache gespeichert. Dort werden verschiedene Versionen des Assemblys gespeichert.

!!!!!!!

Es gibt zwei Versionen, welche wird verwendet? Nicht automatisch die höchste sondern die, gegen die kompiliert wurde! Es kann aber mehrere Versionen geben die installiert sind.

# Auflösen von Assembly-Referenzen

Assembly redirect -> Version 2 statt 1  
soll nun verwendet werden -> die beim  
Kompilieren festgelegte wird  
überschrieben

- Im Manifest ist festgelegt, an welche Assembly-Versionen eine Anwendung gebunden ist.
- In der Konfigurations-Datei *Anwendung.exe.config* können bestehende Referenzen auf neue Version umgeleitet werden.

```
<configuration>
  <runtime>
    <assemblyBinding xmlns=...>
      <dependentAssembly>
        <assemblyIdentity name="AdvCalc"
                           publicKeyToken="ca940d6011fb6820"
                           culture="neutral" />
        <bindingRedirect oldVersion="1.0.0.0-1.9.9.9"
                           newVersion="2.0.0.0"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

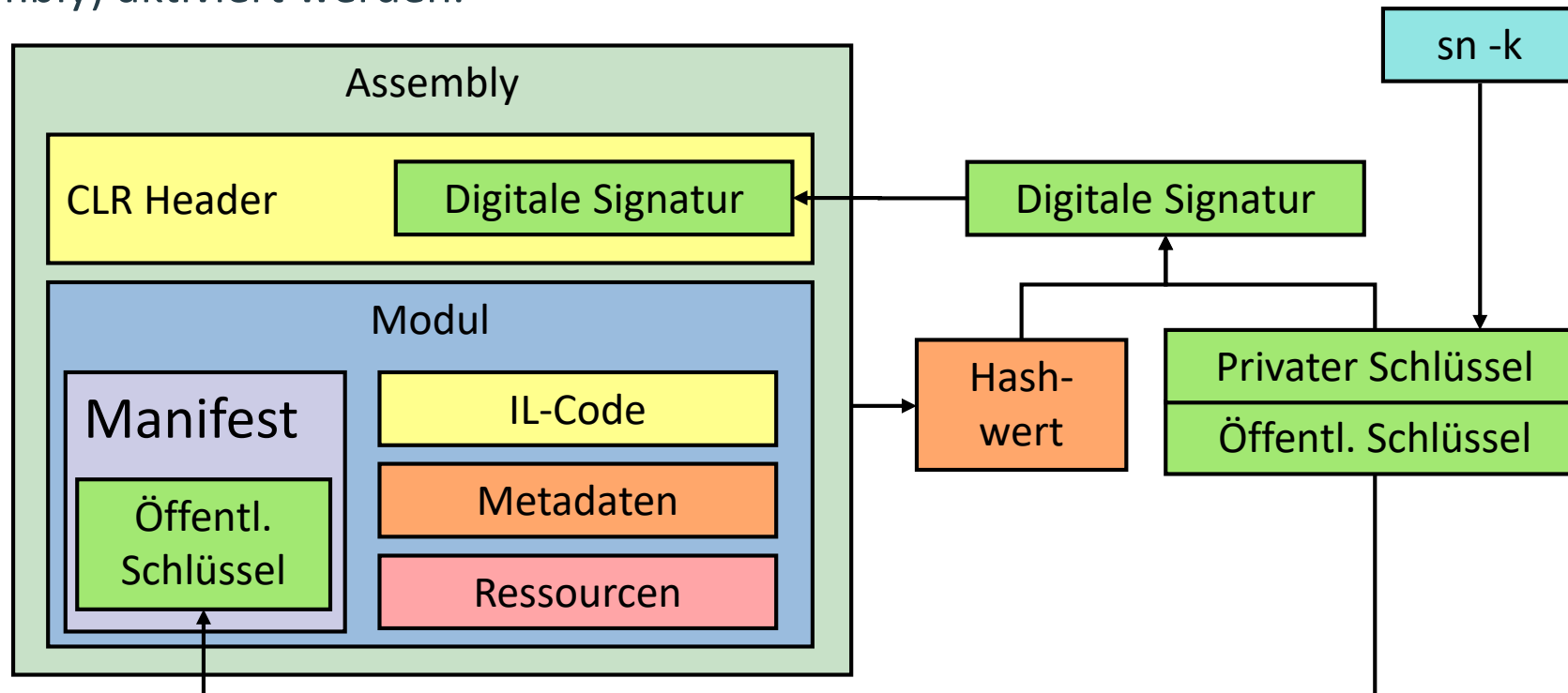


# Digitales Signieren von Assemblys

- Shared Assemblys werden digital signiert.
- Ermöglicht Überprüfung, ob ein Assembly verändert wurde.
- Seit .NET 4.0 deaktiviert. Kann über Registry (maschinenweit) bzw. Konfigurationsdatei (für Assembly) aktiviert werden.

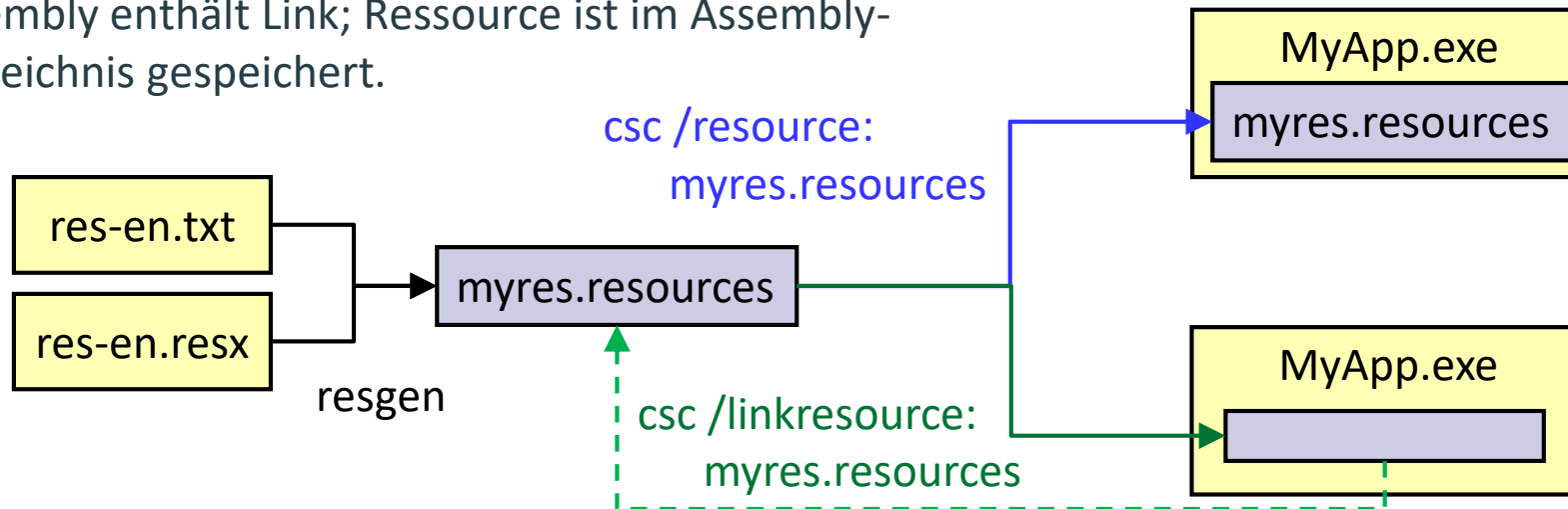
Mit private public key system

Hashwert über assembly. mit privaten Schlüssel signiert. Mit öffentlichen Schlüssel kann Signatur überprüft werden. aus performnace gründen standardmäßig deaktiviert



# Ressourcen

- Ressourcen können in einer Text- (Zeichenketten) oder einer XML-Datei (Zeichenketten, Bilder, ...) definiert werden.
- Ressourcen müssen in Binärform übersetzt werden hineinpacken oder referenzieren
  - `resgen x.txt/x.resx → x.resources`
- Speicherort von Ressourcen:
  - In Assembly eingebettet.
  - Assembly enthält Link; Ressource ist im Assembly-Verzeichnis gespeichert.

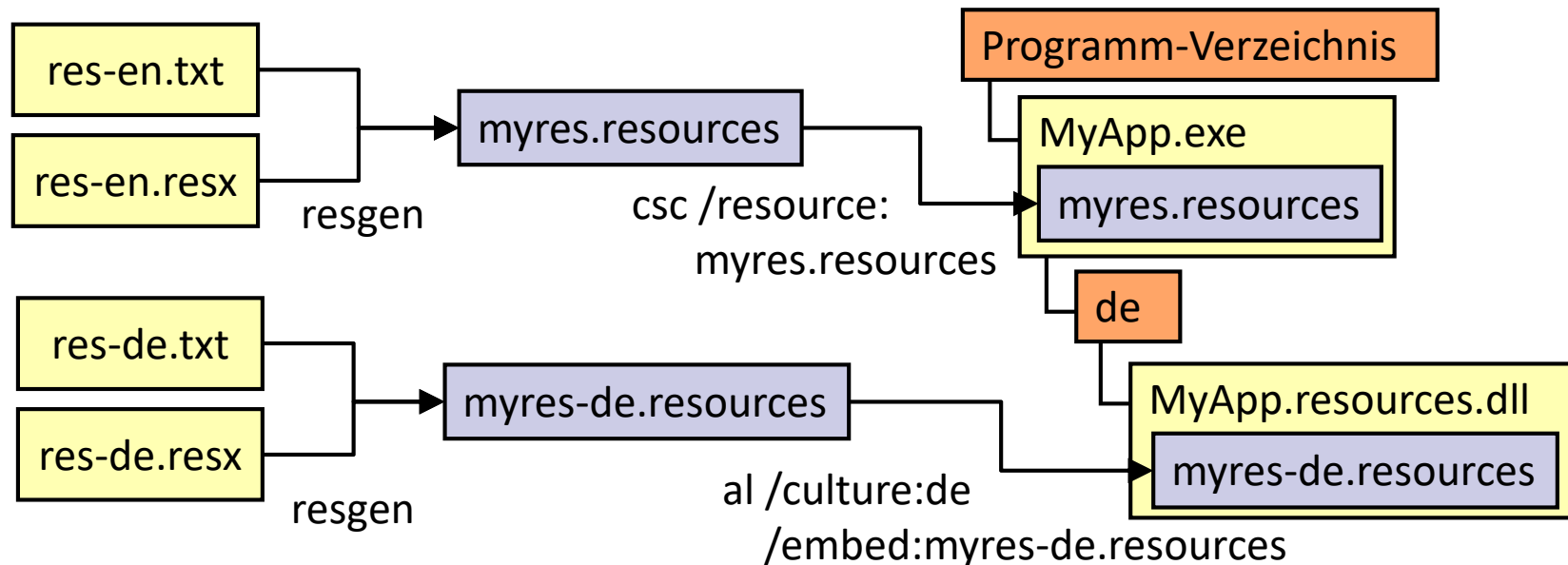


# Satelliten-Assemblys

Ressourcen will ich sprachabhängig machen,  
defaultversion gebe ich in hauptassembly

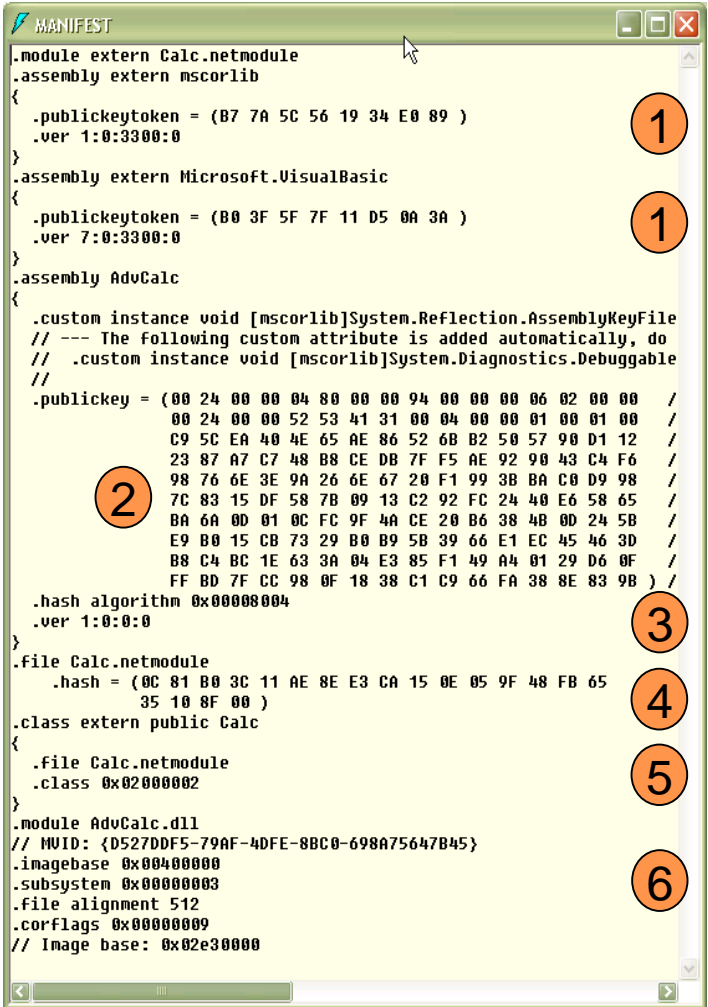
assembly dass nur ressourcen enthält kann ich dann  
zum Default assembly hinzufügen, gibt es nichts  
wird default verwendet

- Satelliten-Assemblys enthalten sprachspezifische Ressourcen.
- Die Standardwerte der Ressourcen werden in der Haupt-Assembly gespeichert.
- Welche Satelliten-Assembly geladen wird, hängt von der Kultureinstellung (*UICulture*) ab.



# Das Assembly Manifest

- Das Manifest enthält
  - Referenzierte Assemblys (1)
  - Assembly-Identität:
    - Öffentlicher Schlüssel (2)
    - Versionsnummer (3)
  - Liste der Module, aus denen das Assembly besteht (4)
  - Exportierte Typen (5)
  - Assembly-Art (subsystem) (6)
    - Exe,
    - Windows-Exe,
    - Library.

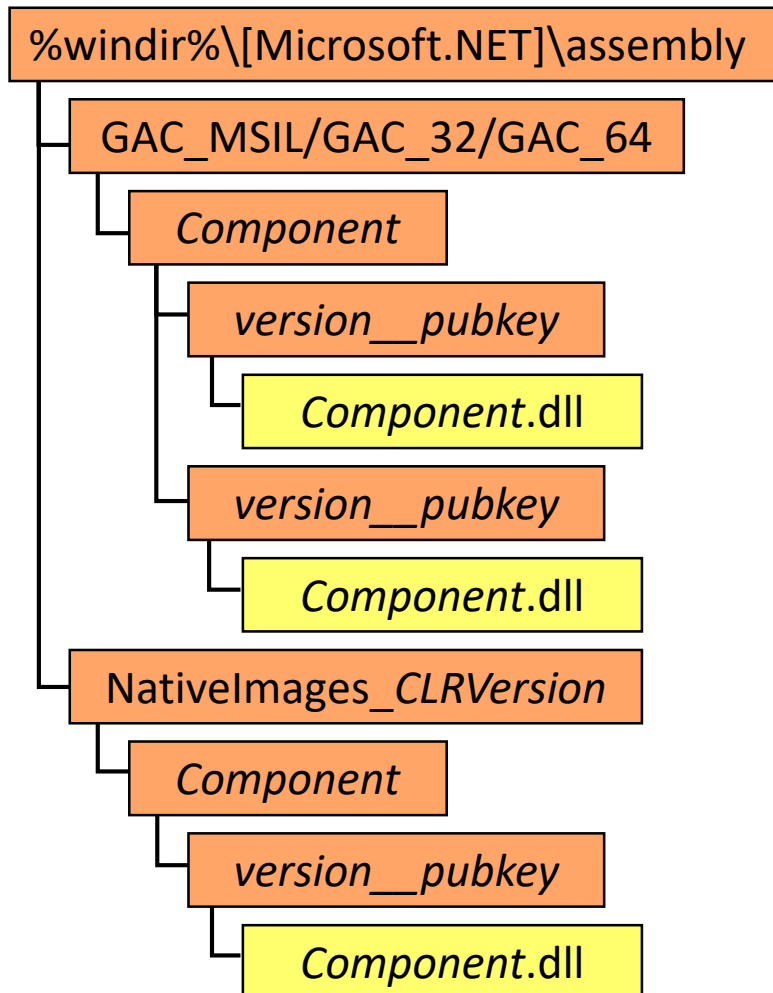


```
MANIFEST
.module extern Calc.netmodule
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 1:0:3300:0
}
.assembly extern Microsoft.VisualBasic
{
  .publickeytoken = (B0 3F 5F 7F 11 D5 0A 3A )
  .ver 7:0:3300:0
}
.assembly AdvCalc
{
  .custom instance void [mscorlib]System.Reflection.AssemblyKeyFile
  // --- The following custom attribute is added automatically, do
  // .custom instance void [mscorlib]System.Diagnostics.Debuggable
  //
  .publickey = (00 24 00 00 04 80 00 00 94 00 00 00 06 02 00 00 /
               00 24 00 00 52 53 41 31 00 04 00 00 01 00 01 00 /
               C9 5C EA 40 4E 65 AE 86 52 6B B2 50 57 90 D1 12 /
               23 87 A7 C7 48 B8 CE DB 7F F5 AE 92 90 43 C4 F6 /
               98 76 6E 3E 9A 26 6E 67 20 F1 99 3B BA C0 D9 98 /
               7C 83 15 DF 58 7B 09 13 C2 92 FC 24 40 E6 58 65 /
               BA 6A 0D 01 0C FC 9F 4A CE 20 B6 38 4B 0D 24 5B /
               E9 B0 15 CB 73 29 B0 B9 5B 39 66 E1 EC 45 46 3D /
               B8 C4 BC 1E 63 3A 04 E3 85 F1 49 A4 01 29 D6 0F /
               FF BD 7F CC 98 0F 18 38 C1 C9 66 FA 38 8E 83 9B ) /
  .hash algorithm 0x00000004
  .ver 1:0:0:0
}
.file Calc.netmodule
  .hash = (0C 81 B0 3C 11 AE 8E E3 CA 15 0E 05 9F 48 FB 65
          35 10 8F 00 )
.class extern public Calc
{
  .file Calc.netmodule
  .class 0x02000002
}
.module AdvCalc.dll
// MVID: {D527DDF5-79AF-4DFE-8BC0-698A75647B45}
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 512
.corflags 0x00000009
// Image base: 0x02e30000
```

The screenshot shows a text editor window titled 'MANIFEST' containing the assembly manifest code. Numbered annotations are placed on the right side of the code:

- 1: Points to the `.assembly extern mscorlib` and `.assembly extern Microsoft.VisualBasic` lines.
- 2: Points to the `.publickey =` block.
- 3: Points to the `.hash algorithm` and `.ver` lines.
- 4: Points to the `.file Calc.netmodule` line.
- 5: Points to the `.class extern public Calc` line.
- 6: Points to the `.module AdvCalc.dll` line.

# Der Global Assembly Cache (GAC) – Full Framework

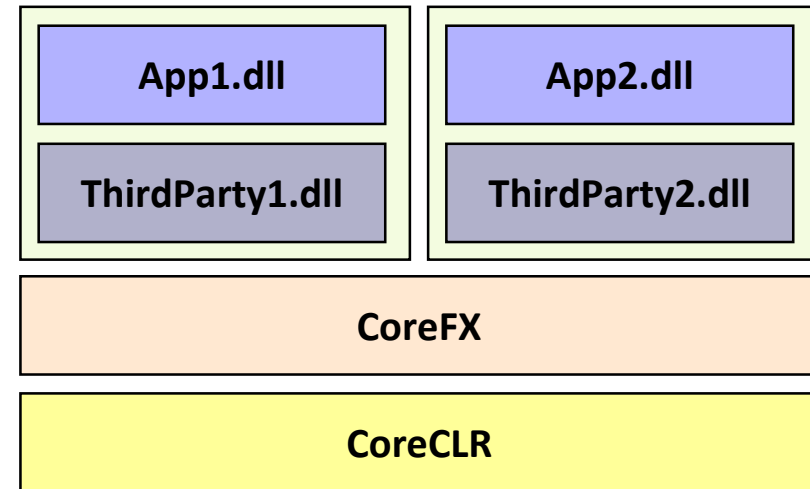


- GAC ist der zentrale Speicherort für gemeinsam genutzte Assemblys.
- Im GAC können mehrere Versionen einer Komponente gespeichert werden.
- Im GAC werden IL-basierte und vorübersetzte Assemblys gespeichert.
- `%windir%\\assembly` → CLR 2.0, .NET Framework 2.0 – 3.5
- `%windir%\\Microsoft.NET\\assembly` → CLR 4, .NET Framework 4.x
- `GAC_MSIL`: Architektur-unabhängige Assemblys.
- `GAC32/GAC64`: Assemblys für entsprechende Betriebssystem-architektur.

# .NET-Core: Framework-dependent Deployment

- Es gibt eine geteilte (systemweite) Installation von .NET-Core
  - Deployment enthält Code der Komponente und Komponenten von Drittherstellern.
  - Windows:  
C:\Program Files\dotnet\shared
  - Linux: /usr/share/dotnet/shared

mit dieser Installation  
muss ich auskommen



wird eine  
gemeinsame  
Basis  
verwendet

- Vorteile
  - Code läuft auf verschiedenen .NET-Installationen und Plattformen
  - Effiziente Ausnutzung des (Festplatten-)Speicherplatzes
- Nachteile
  - Version, gegen die kompiliert wurde (oder höhere), muss auf Zielsystem installiert sein.
  - Verhalten einer CoreFX-Komponente könnte sich ändern.

# .NET-Core: Self-contained Deployment

- Anwendung wird mit allen zum Betrieb notwendigen Komponenten ausgeliefert:

- Komponente
- Komponenten von Drittherstellern
- Bibliotheken von .NET Core
- Laufzeitumgebung

Treeshaking - was wird wirklich verwendet? nur der Teil kommt ins Deployment paket

kann mehrere nebeneinander laufen lassen, nur blöd wenn CoreFX (1) und 2 zufällig das selbe wären

- Vorteile

- Volle Kontrolle über verwendete Komponenten
- Mehrere Laufzeitumgebungen können nebeneinander existieren

- Nachteile

- Verschwendung von (Festplatten-)Speicherplatz

