TOOL DEVELOPMENT

C# CONTINUED

CONCEPTS / TERMS

JSON

Serialization

NuGet packages

Attributes

Reflection

Assembly

Repository

Embedded resources

JSON

EXPLORE THE FORMAT

json
JAVASCRIPT OBJECT NOTATION

- Similar to XML:
  - initial goal = lightweight data transport
  - does not execute anything, just carries data
  - ➢ plain text with some structure
  - ➢ needs some software to process

- Difference with XML:
  - Json is more lightweight
  - Json is faster! (parsing XML taks longer)
  - Json uses object notation (whereas XML is a markup language)
  - ....

XML vs JSON

- XML:

```
<game id="6a23dabe">
        <name>Ring fit adventure</name>
        <publisher>Nintendo</publisher>
        <release>2019</release>
</game>
```

- JSON:

```
{
        "id":"6a23dabe",
        "publisher":"Nintendo",
        "name":"Ring fit adventure",
        "release":2019
}
```

- no object/class name
- "property":"value"
- { } collect properties of 1 object

XML vs JSON : NESTED

- XML:

```
<game id="6a23dabe">
    <name>Ring fit adventure</name>
    <release_info>
        <publisher>Nintendo</publisher>
        <year>2019</year>
    </release_info>
</game>
```

- JSON:

```
{
    "id":"6a23dabe",
    "name":"Ring fit adventure",
    "releaseInfo":
    {
        "publisher":"Nintendo",
        "year":2019
    }
}
```

XML vs JSON : ARRAYS

XML:

```xml
<games>
    <game>
        <name>Wii Sports</name>
        <publisher>Nintendo</publisher>
        <release>2006</release>
    </game>
    <game>
        <name>Just Dance</name>
        <publisher>Ubisoft</publisher>
        <release>2003</release>
    </game>
    <game>
        <name>Ring fit adventure</name>
        <publisher>Nintendo</publisher>
        <release>2019</release>
    </game>
</games>
```

JSON:

```json
[
    {
        "publisher":"Nintendo",
        "name":"Wii sports",
        "release":2006
    },
    {
        "publisher":"Ubisoft",
        "name":"Just Dance",
        "release":2003
    },
    {
        "publisher":"Nintendo",
        "name":"Ring fit adventure",
        "release":2019
    }
]
```

JSON TO C# CLASS

```
{
        "id":"6a23dabe",
        "name":"Ring fit adventure",
        "releaseInfo":
        {
            "publisher":"Nintendo",
            "year":2019
        }
}
```

```
public class Game
{
    0 references
    public string Code { get; set; }
    2 references
    public string ProductName { get; set; }
    0 references
    public ReleaseData ReleaseInfo { get; set; }

    0 references
    public BitmapImage Image...
}

1 reference
public class ReleaseData
{
    0 references
    public string Publisher { get; set; }
    0 references
    public int Year { get; set; }
}
```

EDIT → PASTE SPECIAL → PASTE JSON AS CLASSES

- Automatic conversion via vs.net:
  - copy json data
  - Paste special in vs.net
  - ➢ classes are auto generated

EDIT → PASTE SPECIAL → PASTE JSON AS CLASSES

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) T03_Reflection

Debug Any CPU Start

Solution Explorer

Search Solution Explorer (Ctrl+;)

Class1.cs*

T03_Reflection

```
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace T03_Reflection.Model
8   {
9
10  }
11
```

PASTE JSON AS CLASSES

- Might be a big help to get a quick o
- but please **do not** use it as it is!!
- Bad naming, bad structure

```
{
        "id":"6a23dabe",
        "name":"Ring fit adventure",
        "originalRelease":
        {
           "publisher":"Nintendo",
           "year":2019
        } ,
        "newestRelease":
        {
           "publisher":"NextNintendo",
           "year":2022
        }
}
```

```
public class Rootobject
{
    0 references
    public string id { get; set; }
    0 references
    public string name { get; set; }
    0 references
    public Originalrelease originalRelease { get; set; }
    0 references
    public Newestrelease newestRelease { get; set; }
}

1 reference
public class Originalrelease
{
    0 references
    public string publisher { get; set; }
    0 references
    public int year { get; set; }
}

=

1 reference
public class Newestrelease
{
    0 references
    public string publisher { get; set; }
    0 references
    public int year { get; set; }
}
```

PASTE JSON AS CLASSES

- Might be a big help to get a quick overview of your classes,
- but please **do not** use it as it is!!
- Bad naming, bad structure

➔ Create your classes **from scratch**!

  ➔ but you can use the above to look at as a help for your structure

SO FAR, SO GOOD...

JSON data

- plain text
- object notation

MODEL(S)

- class structure
- matches JSON data, but:
  - properties might have a different name
  - model might have extra properties (that are not in the JSON data)
  - not all properties in the JSON data must be in your model(s)!

➢ Next step: **convert** JSON data to object(s) of these types

LIBRARIES / NUGET PACKAGES

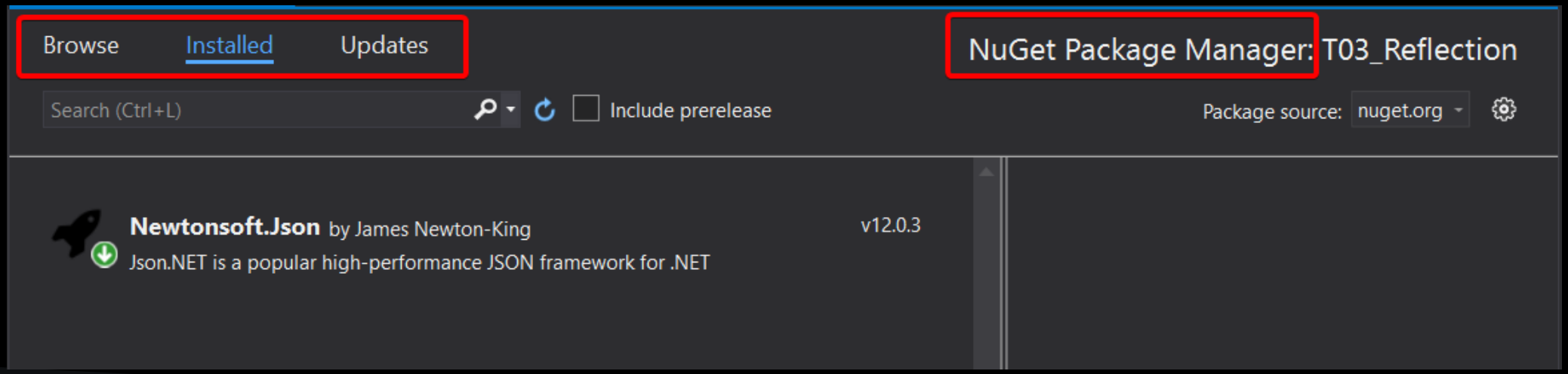
USING LIBRARIES IN C#

https://docs.microsoft.com/en-us/nuget/create-packages/overview-and-workflow

LIBRARIES & NUGET PACKAGES

- **Library:** compiled package of classes
  - Add as project reference to make available
  - Accessible in code files using the namespace

- **NuGet:**
  - NuGet package contains one or more (versions of) libraries
  - Central access through NuGet package manager
  - Also easy to manage updates of packages in your project



using libraries in c#

NEWTONSOFT NUGET PACKAGE (JSON)

- Right click **project** → Manage NuGet packages… → tab Browse

Browse | Installed | Updates 1

NuGet Package Manager: DemoProject

newtonsoft | Include prerelease | Package source: nuget.org

Newtonsoft.Json by James Newton-King, **858M** downloads — v12.0.3
Json.NET is a popular high-performance JSON framework for .NET

Newtonsoft.Json.Bson by James Newton-King, **80.2M** downloads — v1.0.2
Json.NET BSON adds support for reading and writing BSON

Microsoft.AspNetCore.Mvc.NewtonsoftJson by Microsoft, **44M** downloads — v5.0.2
ASP.NET Core MVC features that use Newtonsoft.Json. Includes input and output formatters for JSON and JSON PATCH.

Newtonsoft.Json.Schema by Newtonsoft, **9.33M** downloads — v3.0.13
Json.NET Schema is a complete and easy-to-use JSON Schema framework for .NET

Swashbuckle.AspNetCore.Newtonsoft by Swashbuckle.AspNetCore.Newtonsoft, **7.53M** — v6.0.2
Swagger Generator opt-in component to support Newtonsoft.Json serializer behaviors

Newtonsoft.Json — nuget.org

Version: Latest stable 12.0.3 | Install

Options

Description
Json.NET is a popular high-performance JSON framework for .NET

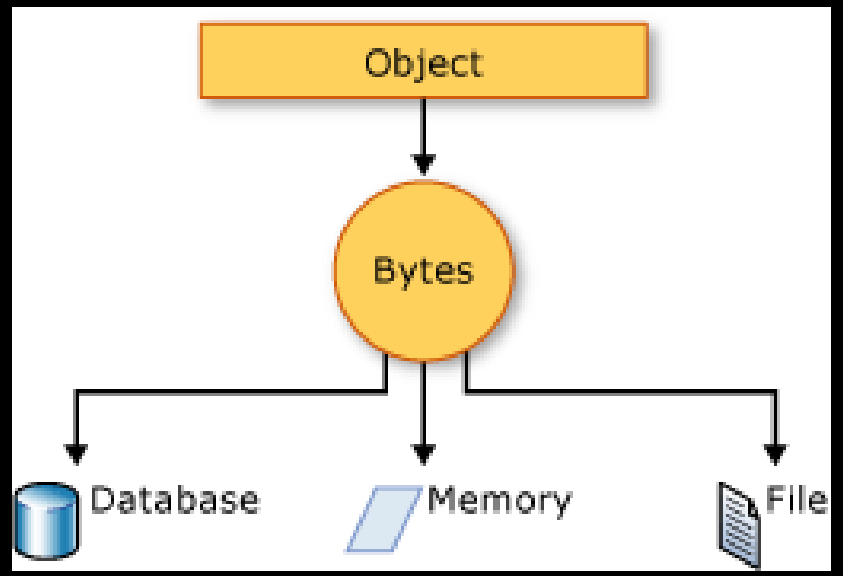
Version:	12.0.3
Author(s):	James Newton-King
License:	MIT
Date published:	Saturday, November 9, 2019 (11/9/2019)
Project URL:	https://www.newtonsoft.com/json
Report Abuse:	https://www.nuget.org/packages/Newtonsoft.Json/12.0.3/ReportAbuse
Tags:	json

SERIALISATION & ATTRIBUTES

FROM JSON TO C# (AND VICE VERSA)

SERIALIZATION – WHAT & WHY?

- Serialization: object → byte stream → database/file/memory
- Reverse process is deserialization



- Why?
  - Send/receive data through web service
  - *(Also other things like communication between domains, passing through firewall, or maintain user/security information across application, ...)*
- How?
  - up to the programmer to decide ☺
  - format (xml, json,..)
  - structure/names using attributes

ATTRIBUTES IN C#

- Powerful system to add extra information to properties, methods,....
- Eg.:

```
[Serializable]
0 references
public class GameRepository
{

    [Obsolete ("GetGames is obsolete, please consider using GetGamesAsync", false)]
    0 references
    public static List<Game> GetGames()
    {
        [redacted]
    }

    0 references
    public static async Task<List<Game>> GetGamesAsync()
    {
        [redacted]
    }
```

ATTRIBUTES & NEWTONSOFT (JSON)

- [JsonProperty] , [JsonIgnore]
-

```
public class Game
{
    [JsonProperty(PropertyName="id")]
    0 references
    public string Code { get; set; }

    [JsonProperty(PropertyName = "name")]
    2 references
    public string ProductName { get; set; }

    [JsonProperty(PropertyName = "releaseInfo")]
    0 references
    public ReleaseData ReleaseInfo { get; set; }

    [JsonIgnore]
    0 references
    public BitmapImage Image...
}
```

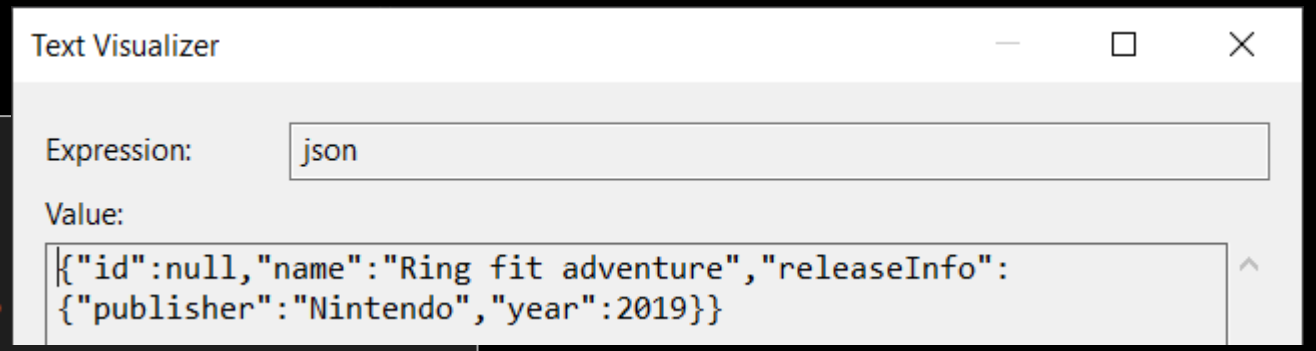
- id: name in json
- Code: name in C#
- not serialized to json

SERIALIZE JSON GAME OBJECTS WITH NEWTONSOFT

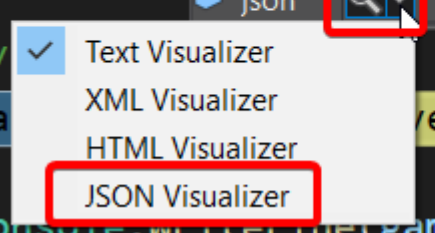
- Use static JsonConvert class

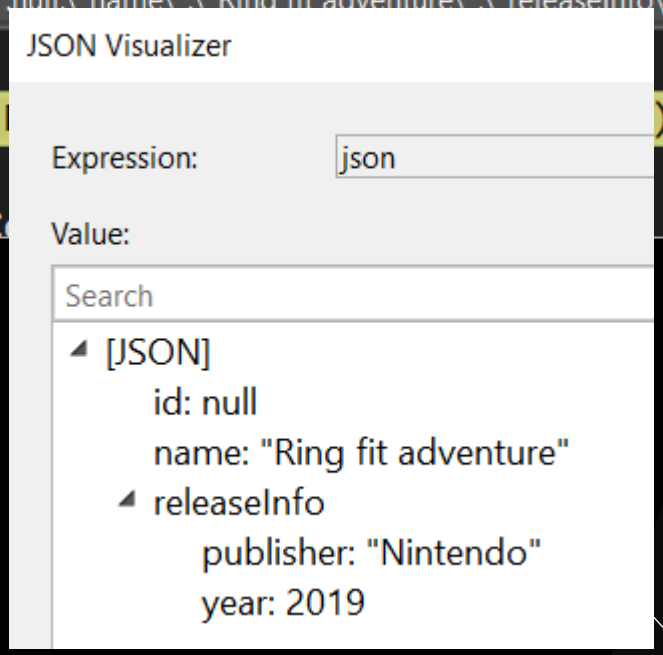
```
Game ringFit = new Game()
{
    ProductName = "Ring fit adventure",
    ReleaseInfo = new ReleaseData
    {
        Publisher = "Nintendo",
        Year = 2019
    }
};

//serialize
string json = JsonConvert.SerializeObject(ringFit);
```




```
//serialize
string json = JsonConvert.SerializeObject(ringFit);
```





- Ready to send through web service,....

DESERIALIZE JSON GAME OBJECTS WITH NEWTONSOFT

- Use static JsonConvert class

```
//serialize
string json = JsonConvert.SerializeObject(ringFit);

//deserialize
Game game = JsonConvert.DeserializeObject<Game>(json);
```

- Ready to use in code

Name	Value	Type
game	{T03_Reflection.Model.Game}	T03_Reflection.Model...
Code	null	string
Image	'game.Image' threw an exception of type 'System.UriFormatException'	System.Windows.Med...
ProductName	"Ring fit adventure"	string
ReleaseInfo	{T03_Reflection.Model.ReleaseData}	T03_Reflection.Model...
Publisher	"Nintendo"	string
Year	2019	int

DESERIALIZE A GIVEN LIST OF DATA

- So what if we want to 'import' a whole list of data?
  - ✓ from a web service
    - Very common
    - Coming up the next weeks
  - ✓ from a local resource file
    - good as a step in between!
    - Coming now ☺

READING A LOCAL JSON FILE

```
using System.IO;
```

```
using(var reader = new StreamReader("Resources/files/myData.json"))
{
    string json
    Game game =
}
```

Exception User-Unhandled

System.IO.DirectoryNotFoundException: 'Could not find a part of the path 'C:\Users\LPK\source\repos\T03_Reflection\bin\Debug\Resources\files\myData.json'.'

View Details | Copy Details | Start Live Share session...

Exception Settings

- Somehow we must find the resource at runtime

REFLECTION

GET (CODE) INFORMATION AT RUNTIME

REFLECTION: WHAT & WHY?

```
using System.Reflection;
```

Reflection (C#)

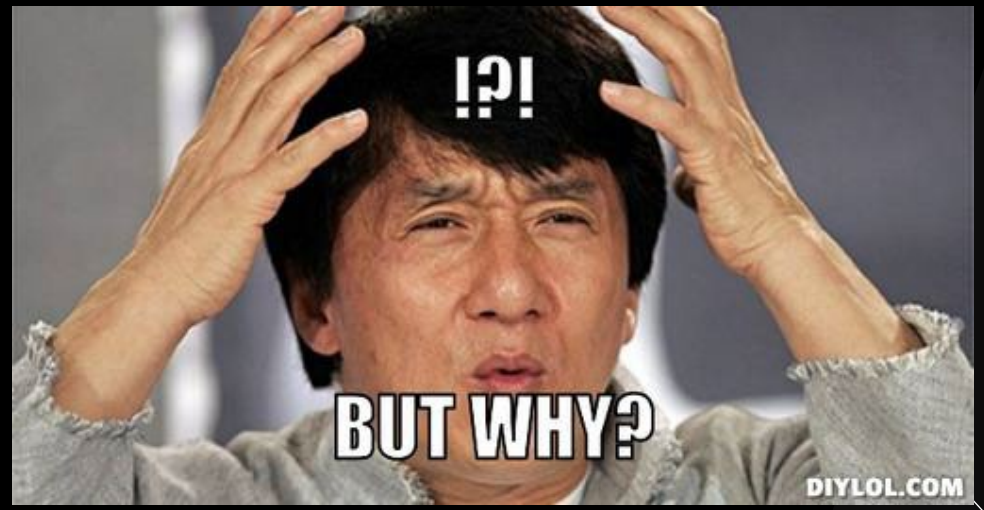
07/20/2015 • 2 minutes to read • +9

Reflection provides objects (of type Type) that describe assemblies, modules, and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them. For more information, see Attributes.

- Dynamically create an instance of a class (= at runtime)

```
//dynamic creation of an instance (at runtime):
Game myGame = Activator.CreateInstance<Game>();
```

- Eg.: create instances based on type in file
  - *example follows in lab*



REFLECTION: WHAT & WHY?

```
using System.Reflection;
```

Reflection (C#)

07/20/2015 • 2 minutes to read • +9

Reflection provides objects (of type Type) that describe assemblies, modules, and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them. For more information, see Attributes.

- Get the type of an existing object:
  - Invoke (execute) methods at runtime
  - Access its fields & properties at runtime

```
Type type = lstMethods.SelectedItem.GetType(); //get type of object
MethodInfo[] info = type.GetMethods(); //get methods of this object

//execute first method that needs a string as a parameter
info[0].Invoke(lstMethods.SelectedItem, new object[] { "filter" });

//ask for the value of a Name property (crashes if non existing in type)
string value = (string) type.GetField("Name").GetValue(lstMethods.SelectedItem);
```

REFLECTION: WHAT & WHY?

```
using System.Reflection;
```

Reflection (C#)

07/20/2015 • 2 minutes to read • +9

Reflection provides objects (of type Type) that describe assemblies, modules, and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them. For more information, see Attributes.

- Get all methods/properties/.. with a certain [attribute]
  - For example list all obsolete methods,
  - or in Xamarin: get OS specific behavior (IOS, Android,..)

REFLECTION: WHAT & WHY?

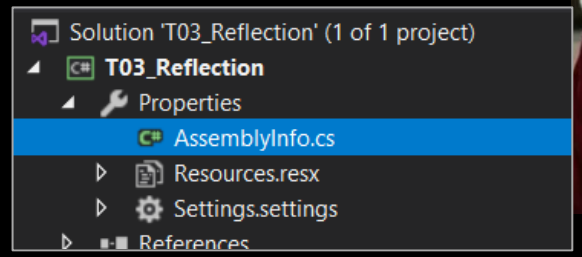
```
using System.Reflection;
```

Reflection (C#)

07/20/2015 • 2 minutes to read • +9

Reflection provides objects (of type Type) that describe assemblies, modules, and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them. For more information, see Attributes.

- Get runtime information about your assembly
  - logical unit of code,
  - being an exe or dll
  - contains at least 1 module:
    - a manifest file
      (contains metadata and reference & version information)
    - metadata and Intermediate Language code
    - resources



ASSEMBLY - INTERMEDIATE LANGUAGE CODE [VS.NET]

- Class Definition

Program.cs | Console [from metadata]

System.Console | BackgroundColor

```
Assembly mscorlib.dll, v4.0.0.0

using System.IO;
using System.Runtime.ConstrainedExecution;
using System.Security;
using System.Text;

namespace System
{
    ...public static class Console
    {
        // Summary:
        //     Gets or sets the background color of the console.
        //
        // Returns:
        //     A System.ConsoleColor that specifies the background color o
        //     that is, the color that appears behind each character. The
        //
        // Exceptions:
        //   System.ArgumentException:
        //     The color specified in a set operation is not a valid membe
        //
        //   System.Security.SecurityException:
        //     The user does not have permission to perform this action.
        //
        //   System.IO.IOException:
        //     An I/O error occurred.
        public static ConsoleColor BackgroundColor { get; set; }
        ...public static int BufferHeight { get; set; }
        ...public static int BufferWidth { get; set; }
```

- dynamically generates the information if the source code is not available

- Object Browser

Object Browser | Program.cs | Console [from metadata]

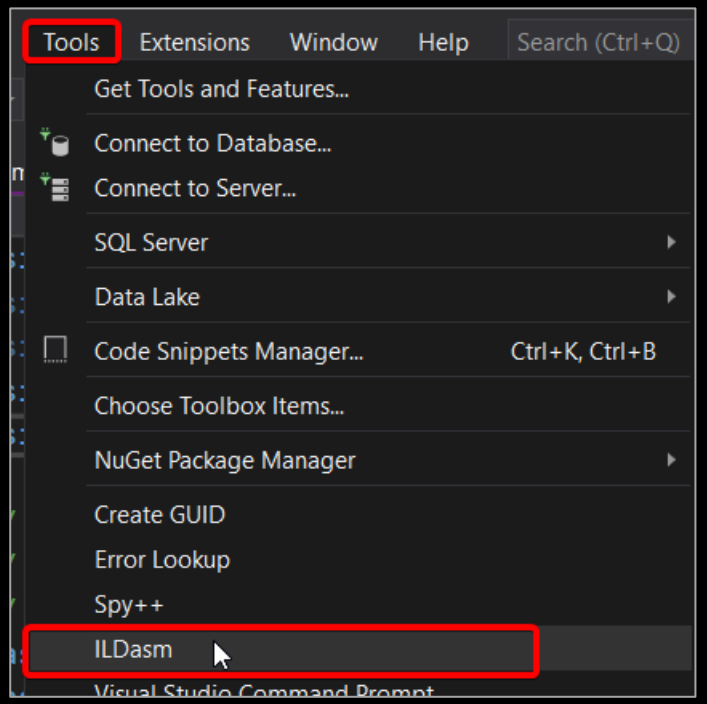
Browse: My Solution

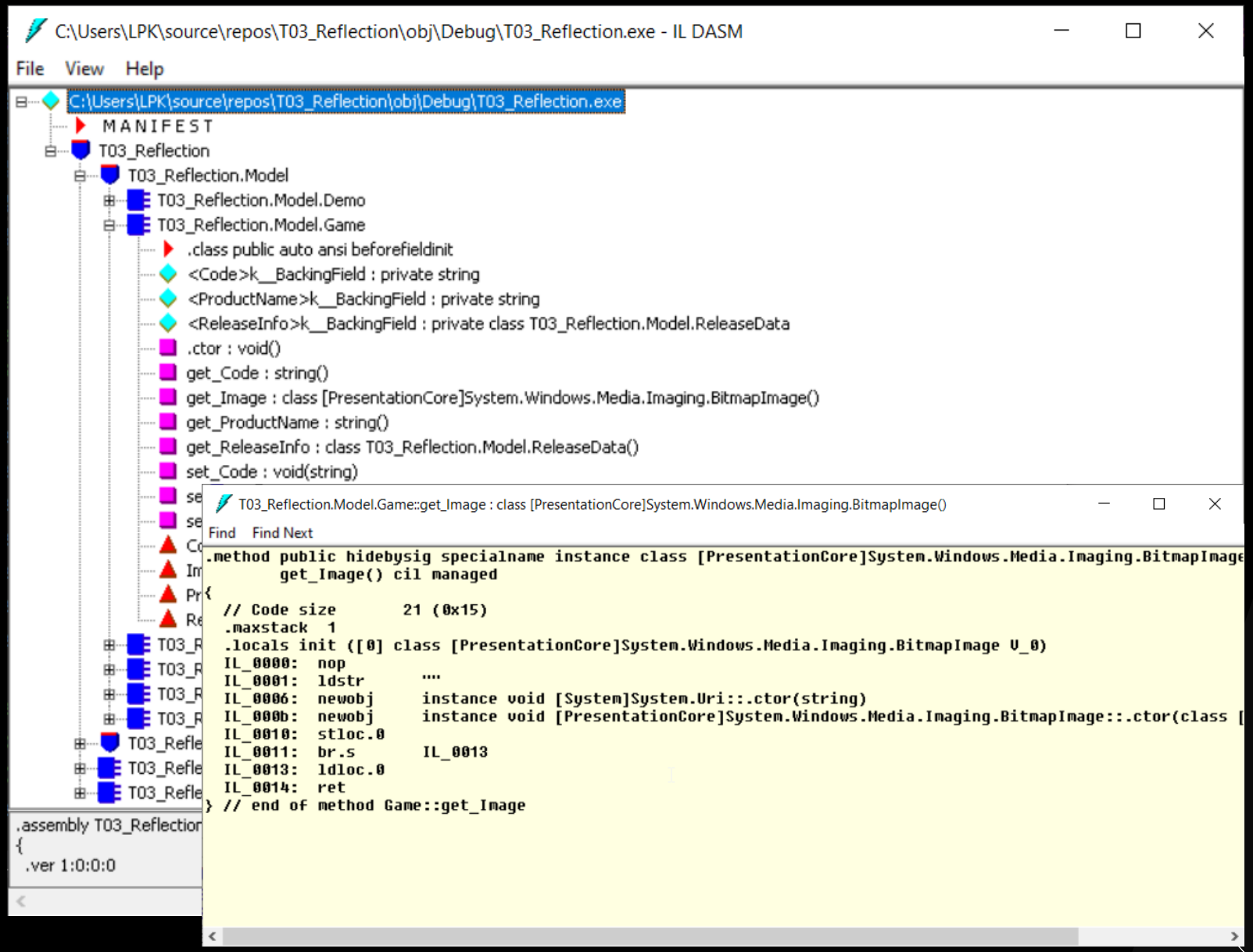
<Search>

BadImageFormatExceptio	Beep(int, int)
Base64FormattingOptions	Beep()
BitConverter	Clear()
Boolean	MoveBufferArea(int, int, int, int, int, int, char, System.(
Buffer	MoveBufferArea(int, int, int, int, int, int)
Byte	OpenStandardError(int)
CannotUnloadAppDomai	OpenStandardError()
Char	OpenStandardInput(int)
CharEnumerator	OpenStandardInput()
CLSCompliantAttribute	OpenStandardOutput(int)
Comparison<in T>	OpenStandardOutput()
Console	Read()
ConsoleCancelEventArgs	ReadKey(bool)
ConsoleCancelEventHanc	ReadKey()
ConsoleColor	ReadLine()
ConsoleKey	ResetColor()
ConsoleKeyInfo	SetBufferSize(int, int)
ConsoleModifiers	SetCursorPosition(int, int)
ConsoleSpecialKey	SetError(System.IO.TextWriter)
ContextBoundObject	SetIn(System.IO.TextReader)
ContextMarshalException	SetOut(System.IO.TextWriter)
ContextStaticAttribute	SetWindowPosition(int, int)
Convert	SetWindowSize(int, int)
Converter<in TInput,out T	Write(string)
CrossAppDomainDelegat	Write(object)

- displays the public members

ASSEMBLY - INTERMEDIATE LANGUAGE CODE [ILDASM]





- Auto installed with vsnet
- Error on ILDasm?
  - Tools → External tools
  - Check ILDasm path
    (search ildasm.exe in program files x86)

EXERCISE: REFLECTION

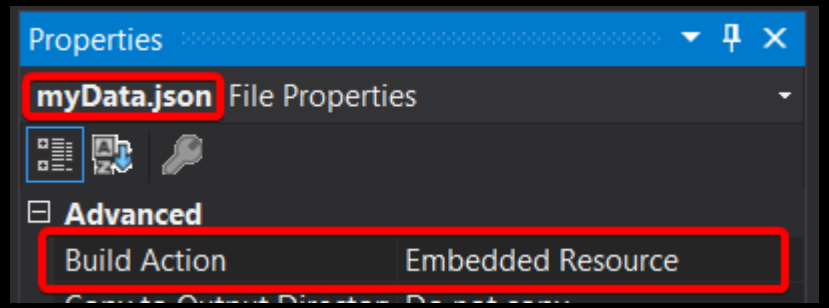
- See Leho assignment **T03-Reflection files**
  - **Plugin system**

RESOURCE MANAGEMENT IN WPF

EMBEDDED RESOURCES (ASSEMBLY)

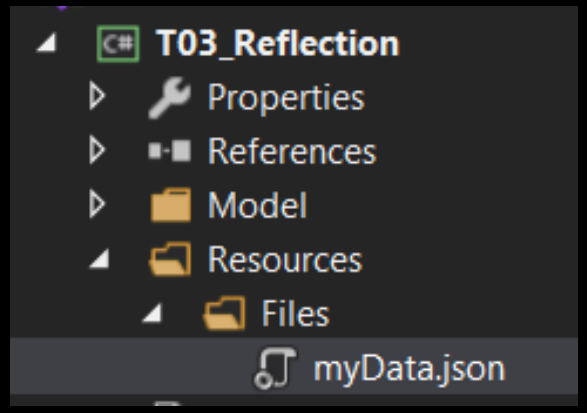
EMBEDDED RESOURCES IN WPF

- Placed directly into executable as manifest resources
- Set file as embedded resource in properties!



- Gets a manifest resource id:
  - `project_namespace.subfolder.filename.ext`
  - Eg.:

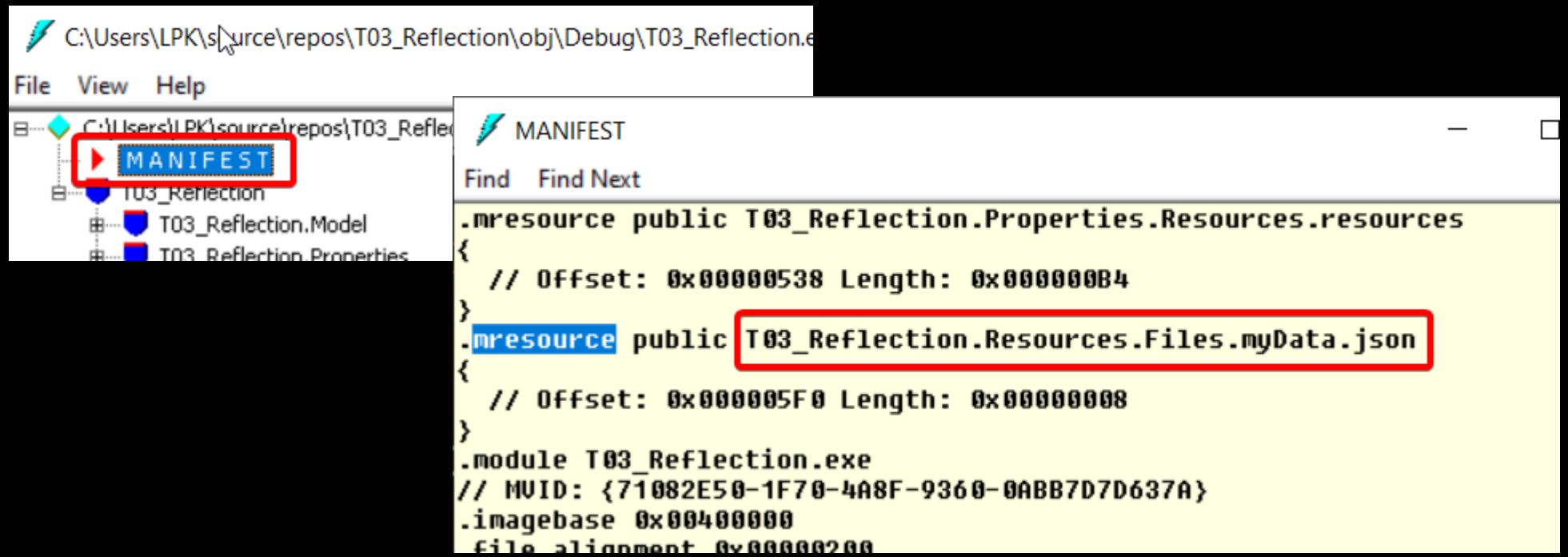
```
T03_Reflection.Resources.Files.myData.json
```



EMBEDDED RESOURCES IN WPF: INSPECT [ILDASM]

- Embedded in executable,
- so cannot be found in bin/debug

- Solution: ILDasm!
  - every embedded resource gets a .mresource record



EMBEDDED RESOURCES IN WPF: INSPECT [CODE]

- Embedded in executable,
- so cannot be found in bin/debug

- Solution: loop in code:

```
//executing assembly
var assembly = System.Reflection.Assembly.GetExecutingAssembly();
//get all embedded resources
string[] resourceNames = assembly.GetManifestResourceNames();

//get specific embedded resource
var resourceName = "T03_Reflection.Resources.Files.myData.json";
Stream stream = assembly.GetManifestResourceStream(resourceName);
```

BACK TO READING A LOCAL FILE

SO WHERE WERE WE....

READING A LOCAL JSON FILE USING REFLECTION

```
using System.IO;
```

```
using(var reader = new StreamReader("Resources/files/myData.json"))
{
    string json = reader.ReadToEnd();
    Game game = JsonConvert.DeserializeObject<Game>(json);
}
```

- Somehow we must find the resource at runtime
  - Reflection

READING A LOCAL JSON FILE USING REFLECTION

```
using System.IO;
using System.Reflection;
```

```
//executing assembly
var assembly = System.Reflection.Assembly.GetExecutingAssembly();

//generated embedded resource name: namespace.subfolder.filename
var resourceName = "T03_Reflection.Resources.Files.myData.json";

using (Stream stream = assembly.GetManifestResourceStream(resourceName))
{
    using (var reader = new StreamReader(stream))
    {
        string json = reader.ReadToEnd();
        Game game = JsonConvert.DeserializeObject<Game>(json);
    }
}
```

from json to c#

DESERIALIZE JSON FILE

```
[
  {
    "id": null,
    "name": "Ring fit adventure",
    "releaseInfo": {
      "publisher": "Nintendo",
      "year": 2019
    }
  },
  {
    "id": null,
    "name": "Wii Sports",
    "releaseInfo": {
      "publisher": "Nintendo",
      "year": 2006
    }
  },
  {
    "id": null,
    "name": "Just Dance",
    "releaseInfo": {
      "publisher": "Ubisoft",
      "year": 0
    }
  }
]
```

```
Game game = JsonConvert.DeserializeObject<Game>(json);
```

Exception User-Unhandled

Newtonsoft.Json.JsonSerializationException: 'Cannot deserialize the current JSON array (e.g. [1,2,3]) into type 'T03_Reflection.Model.Game' because the type requires a JSON object (e.g. {"name":"value"}) to deserialize correctly.
To fix this error either change the JSON to a JSON object (e.g.

from json to c#

DESERIALIZE JSON FILE

```
[
  {
    "id": null,
    "name": "Ring fit adventure",
    "releaseInfo": {
      "publisher": "Nintendo",
      "year": 2019
    }
  },
  {
    "id": null,
    "name": "Wii Sports",
    "releaseInfo": {
      "publisher": "Nintendo",
      "year": 2006
    }
  },
  {
    "id": null,
    "name": "Just Dance",
    "releaseInfo": {
      "publisher": "Ubisoft",
      "year": 0
    }
  }
]
```

```
List<Game> gameList = JsonConvert.DeserializeObject<List<Game>>(json);
```

Name	Value
gameList	Count = 3
[0]	{T03_Reflection.Model.Game}
Code	null
Image	'(new System.Collections.Generic.Mscorlib_Collect
ProductName	"Ring fit adventure"
ReleaseInfo	{T03_Reflection.Model.ReleaseData}
[1]	{T03_Reflection.Model.Game}
Code	null
Image	'(new System.Collections.Generic.Mscorlib_Collect
ProductName	"Wii Sports"
ReleaseInfo	{T03_Reflection.Model.ReleaseData}
[2]	{T03_Reflection.Model.Game}
Code	null
Image	'(new System.Collections.Generic.Mscorlib_Collect
ProductName	"Just Dance"
ReleaseInfo	{T03_Reflection.Model.ReleaseData}

REPOSITORY

EXTRA LAYER

REPOSITORY

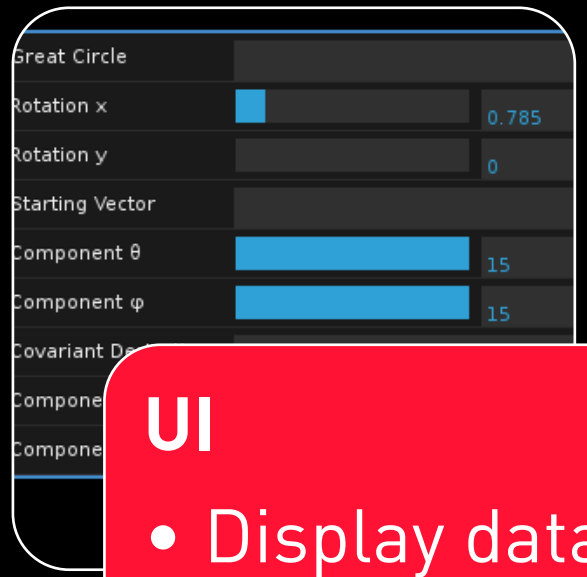
- Class that adds an extra layer between DATA and UI

DATA

- JSON
- Database
- Local
- Webservice

REPOSITORY

- Fetch data **from** source
- Write data **to** source
- (de)serialize



UI

- Display data
- Handle input

REPOSITORY EXAMPLE

```
public class GameRepository
{

    0 references
    public static List<Game> GetGames()
    {
        [redacted]
    }

    0 references
    public static async Task<List<Game>> GetGamesAsync()
    {
        [redacted]
    }
```

CONCEPTS / TERMS

JSON

Serialization

NuGet packages

Attributes

Reflection

Assembly

Repository

Embedded resources