HL7 FHIR DevDays 2017

Advanced .NET API

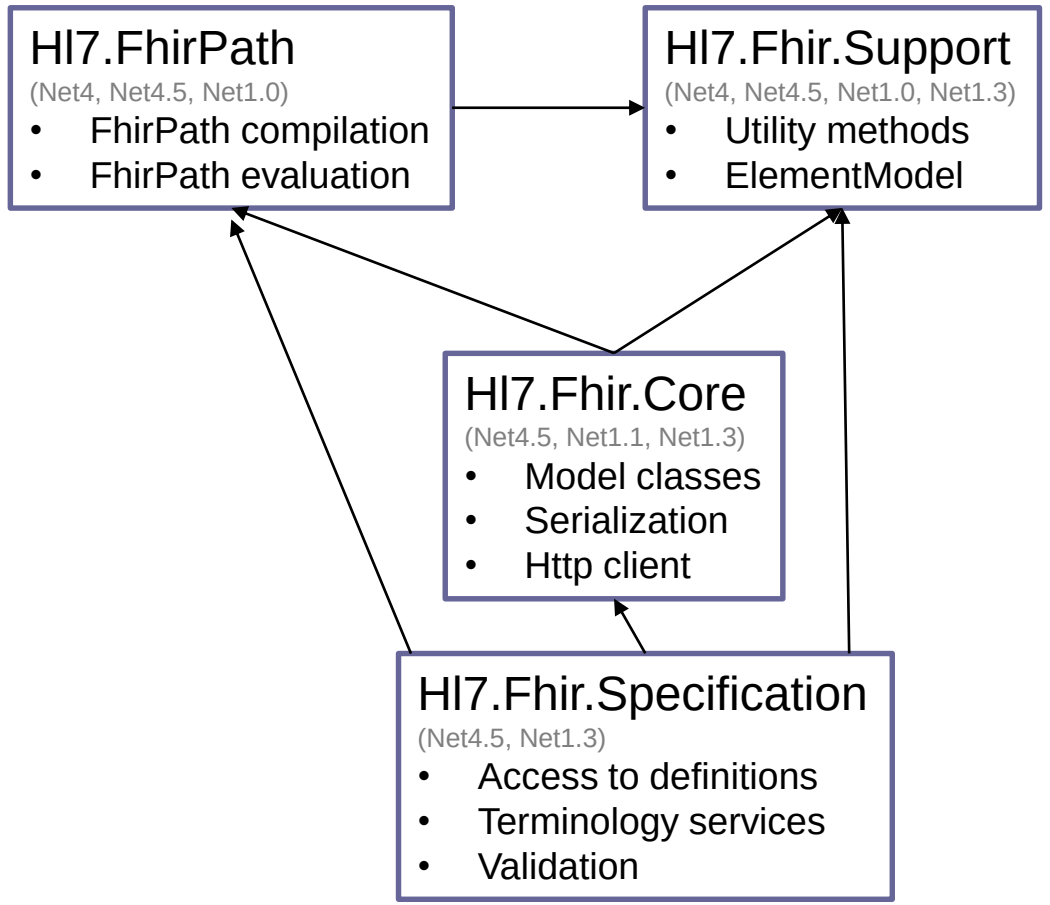
Ewout Kramer, Furore Health Informatics

Amsterdam, 15-17 November | @fhir_furore | #fhirdevdays17 | www.fhirdevdays.com

Who am I?

- **Name:** Ewout Kramer
- **Company:** Furore Health Informatics
- **Background:**
  - Computer Science (operating systems)
  - In Health IT since 1999
  - FHIR Core team
  - Lead dev on the .NET API
- e.kramer@furore.com, @ewoutkramer
- http://thefhirplace.com

Structure of packages

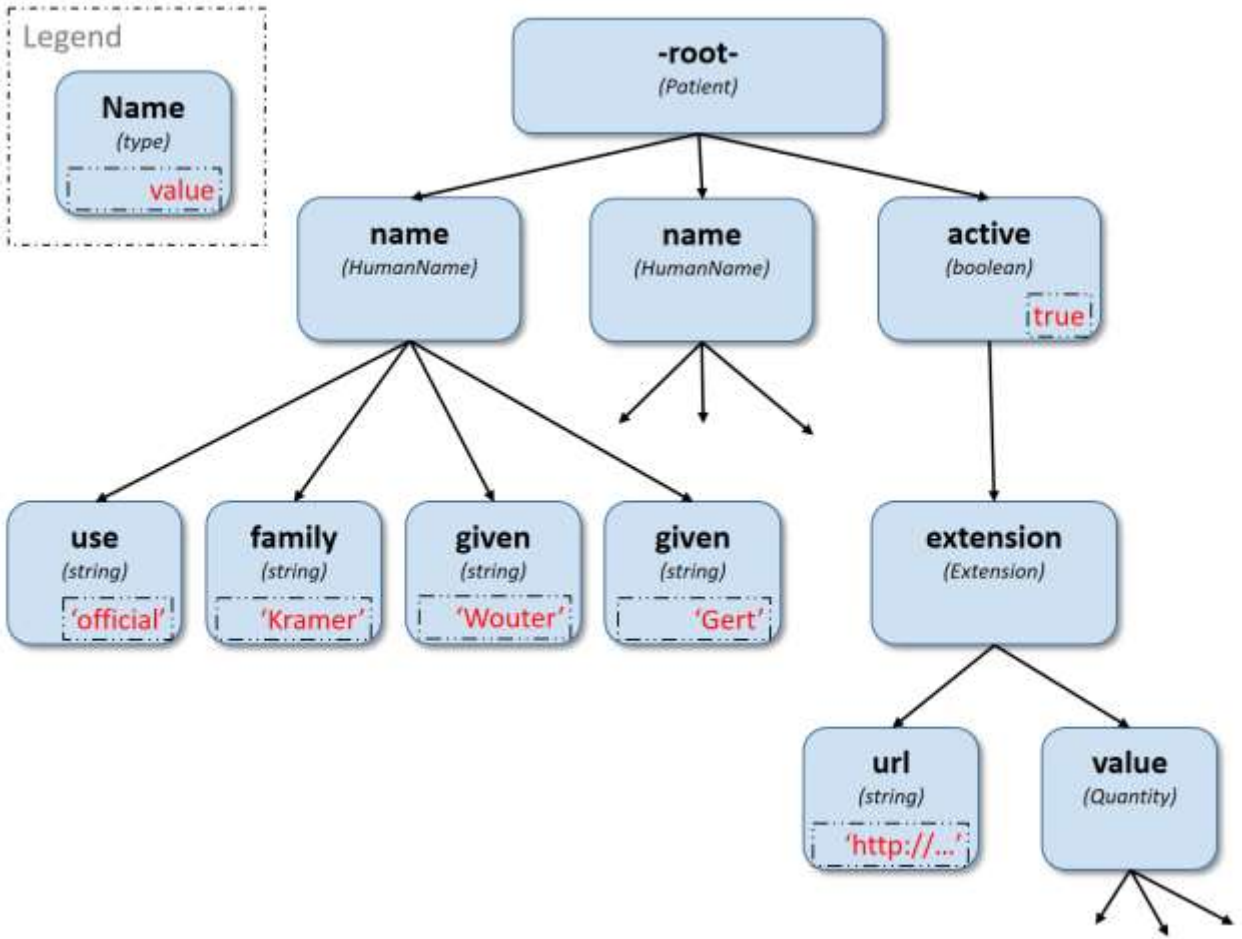


ELEMENTMODEL

Why?

- In Hl7.Fhir.Support
- Lowest-level access to FHIR instance data
- No POCO's involved
- Applications:
  - Access to parts of FHIR data - no need to parse whole POCO's in memory
  - Generic clients/servers not needing pre-compiled model classes
  - Tools that can parse (partially) invalid FHIR data
  - Accessing DSTU2 and STU3 data

“Logical” structure of FHIR data



IElementNavigator

```
namespace Hl7.Fhir.ElementModel
{
    public interface IElementNavigator
    {
        string Name { get; }
        string Type { get; }
        object Value { get; }
        string Location { get; }

        bool MoveToNext(string nameFilter = null);
        bool MoveToFirstChild(string nameFilter = null);

        IElementNavigator Clone();
    }
}
```

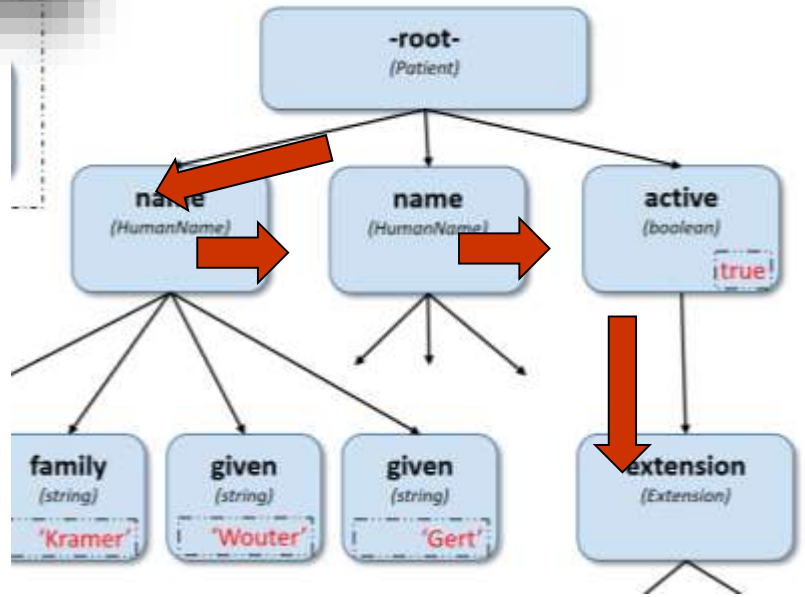
- Represents a position in a tree of FHIR data
- Has all the aspects of a node (from last slide)
- Location:

```
Patient.name[1]
        .family[0]
```

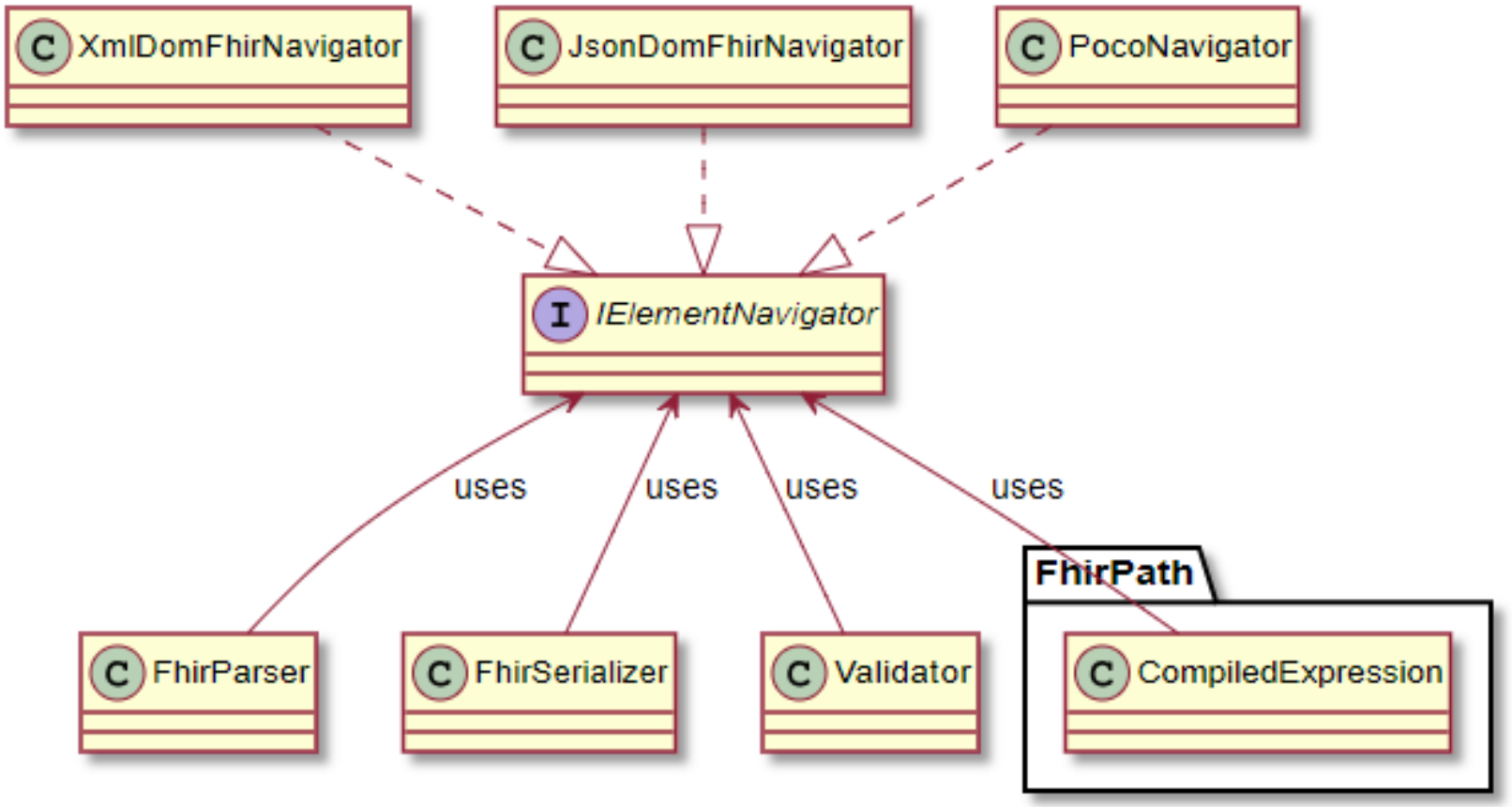
IElementNavigator extension methods

```
var nav = XmlDomFhirNavigator.Create("some xml here");
var given = nav.Children("name")
               .Children("given")
               .Select(n => n.Value as string);
```

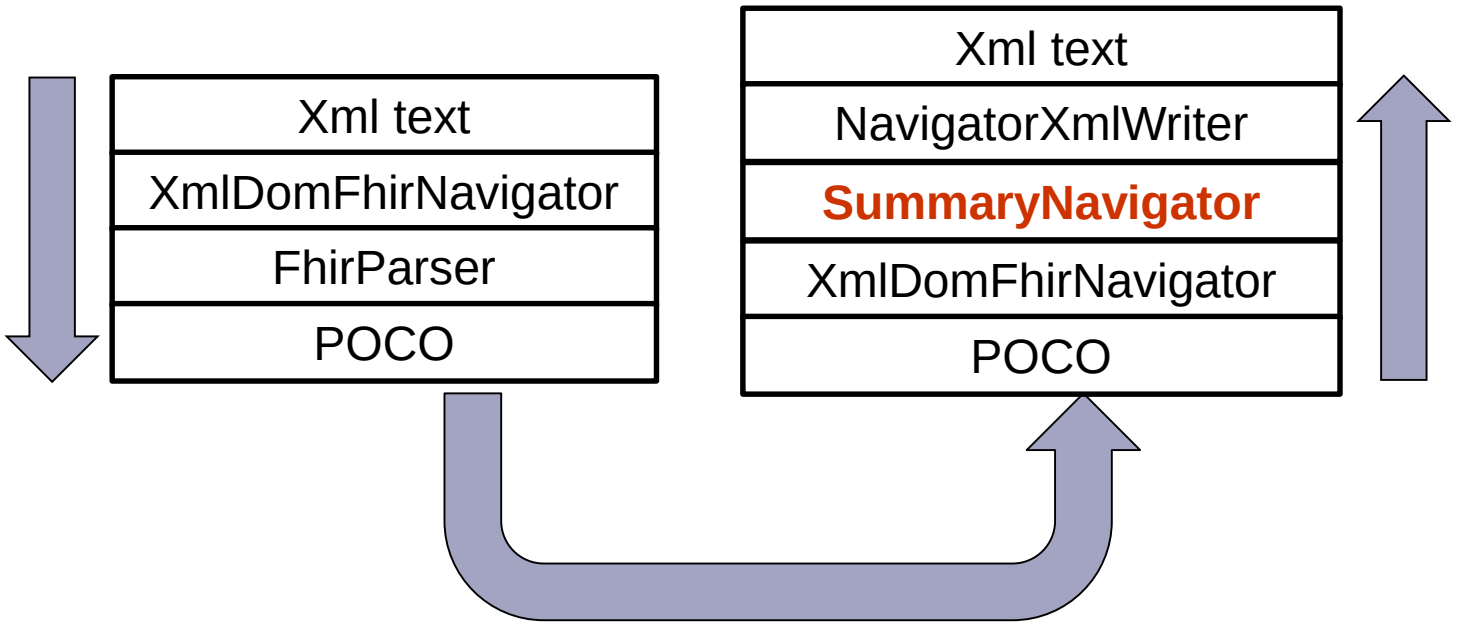
LINQ to ElementModel extension methods go beyond the basic MoveToNext() and MoveToFirstChild() methods



Central connecting concept



From xml to POCO and back



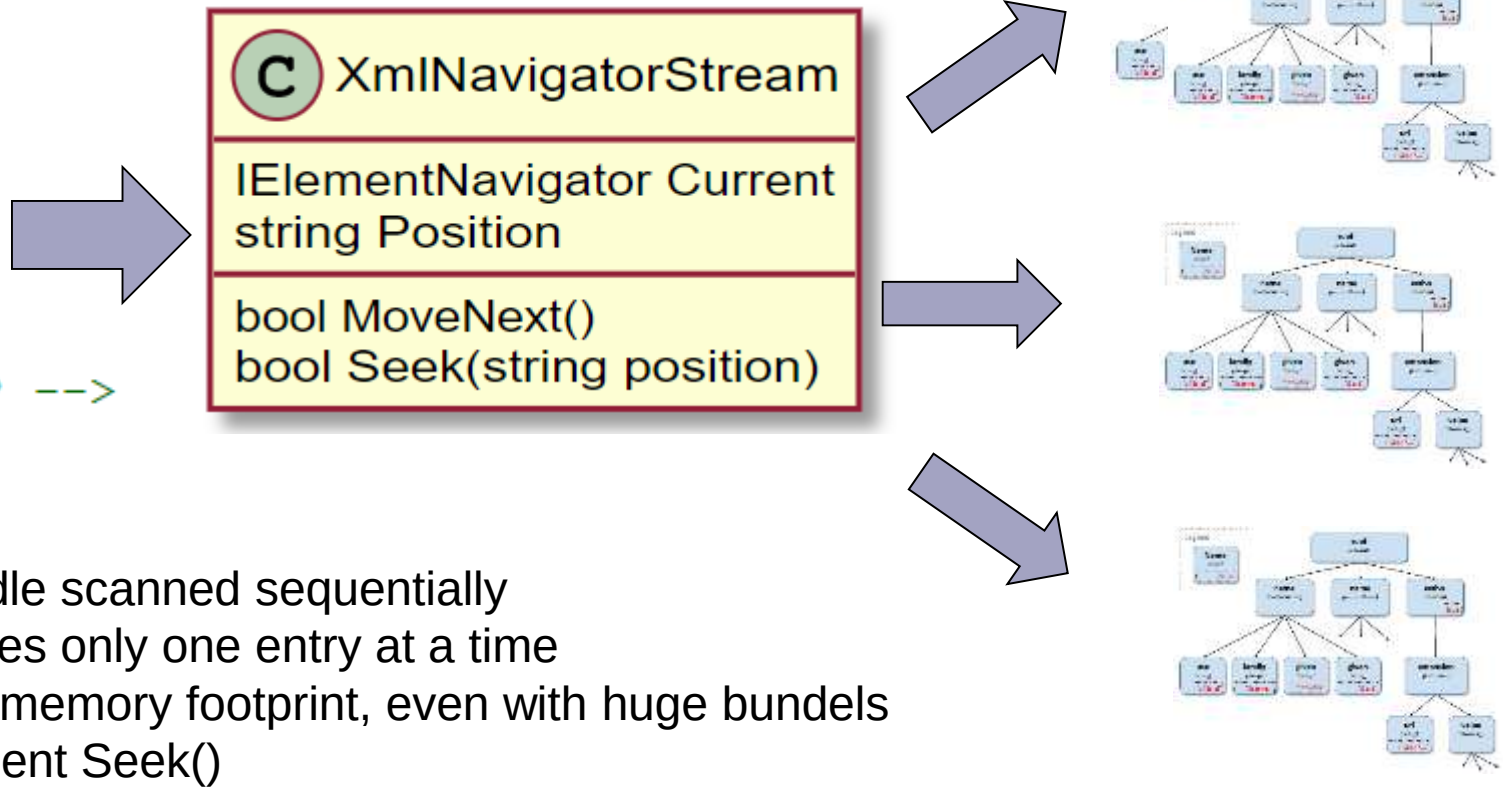
Parsing huge bundles

```
<Bundle>
    <entry>

    </entry>
    <entry>

    </entry>
    <!-- thousands more? -->
    <entry>

    </entry>
</Bundle>
```



- Bundle scanned sequentially
- Parses only one entry at a time
- Low memory footprint, even with huge bundels
- Efficient Seek()

GETTING TO METADATA

Specification data

- All the definitions of the core resources, types, search parameters, operations, etc. are available through the `Hl7.Specification.[STU3/DSTU2]` package on NuGet
- The package contains a zip (specification.zip) with meta data produced by the FHIR publication process
  - profiles-resources.xml, profiles-types.xml, extension-definitions.xml
  - search-parameters.xml
  - v2-tables.xml, v3-codesystems.xml, valuesets.xml
  - xsd schemas, schematrons, others....

Conformance Resources

- Provide “metadata” about
  - Model Definitions (StructureDefinition)
  - Operations (OperationDefinition)
  - Search parameters (SearchParameter)
  - List of codes (ValueSet)
- All are identified and referenced by a “canonical url” that SHOULD resolve, e.g. `“http://hl7.org/fhir/StructureDefinition/Patient”`

Resolution

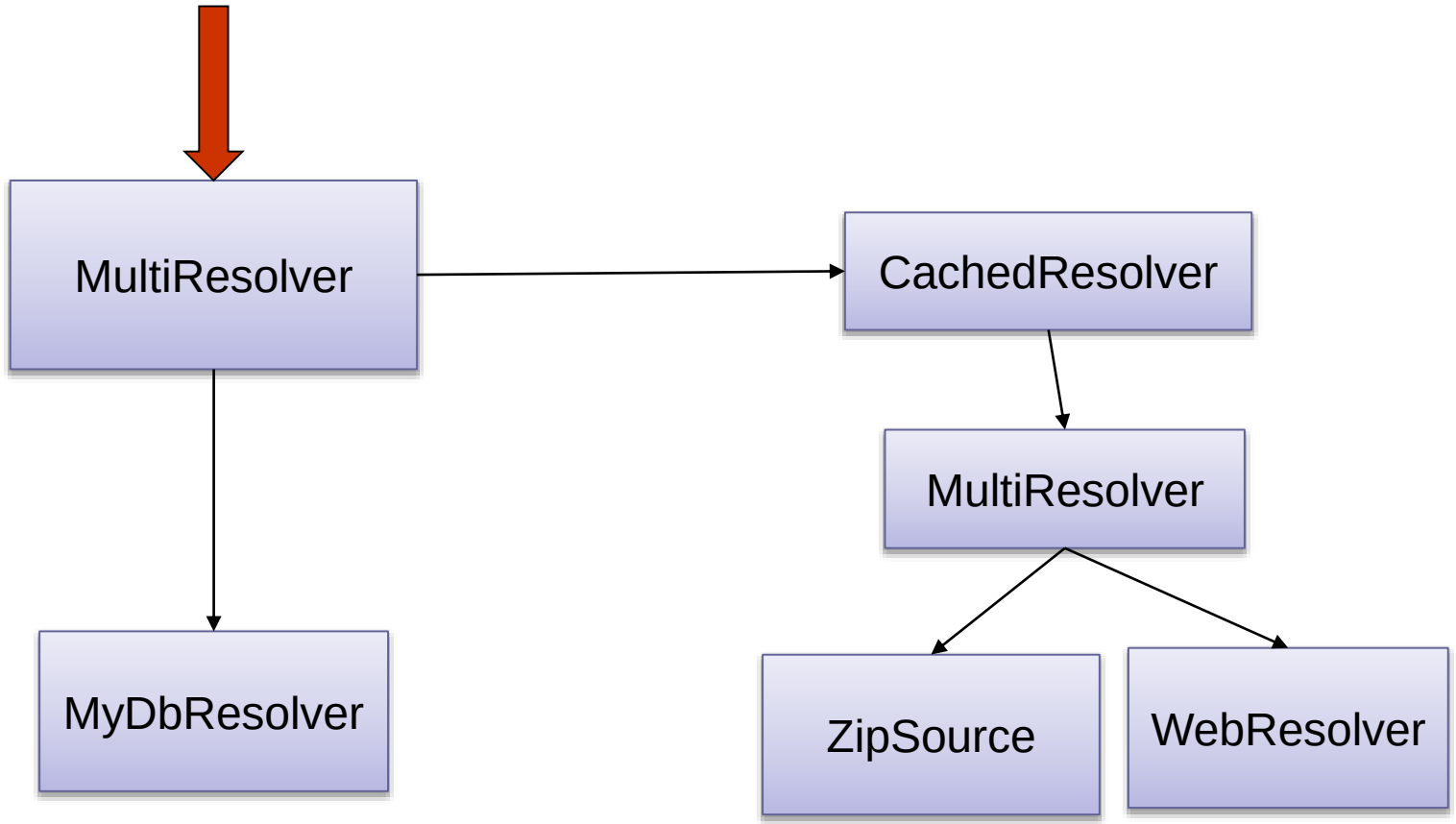
- Directly using a FHIR REST call to the url
- More likely:
  - As packaged in specification.zip
  - As part of a “snapshot”/zip of files delivered with your app
  - Compiled into your app
  - In a database

IResourceResolver

```
public interface IResourceResolver
{
    Resource ResolveByUri(string uri);
    Resource ResolveByCanonicalUri(string uri);
}
```

- Concrete implementations in API:
  - DirectorySource, ZipSource
  - WebSource
  - CachedResolver (wraps another resolver)
  - MultiResolver (tries a list of resolvers)

Combine at will



Practicalities

- Packaged in Hl7.Fhir.Specification assembly
- Namespace Hl7.Fhir.Specification.Source
- Includes useful extension methods like:
  - FindStructureDefinition(this IResourceResolver resolver, string uri, bool requireSnapshot = false)
  - FindStructureDefinitionForCoreType(this IResourceResolver resolver, FHIRDefinedType type)
  - FindExtensionDefinition(this IResourceResolver resolver, string uri, bool requireSnapshot = false)
- Data for base spec is in “specification.zip”, easy to get to using ZipSource.CreateValidationSource()

TERMINOLOGY

FHIR Resources

- CodingSystem resource (STU3)
  - A dictionary of concepts (possibly huge!)
- ValueSet (DSTU2, STU3)
  - A (use-case specific) selection of concepts from 1..* CodingSystems
  - May be directly enumerated ('extensional')
  - May be composed using filters ('intensional')
    - "all the LOINC codes in `LOINC Part Concept Cholesterol | Bld-Ser-Plas (LP43571-6)`, except for `5932-9 Cholesterol [Presence] in Blood by Test strip`"
  - May be composed from other ValueSets

Main functionality

- Expand an *intensional* ValueSet
- Determine whether some code is member of a ValueSet
- Do a code-lookup
  - Find details like alternative designation

- Could be done “in-memory” (i.e. using an expanded ValueSet)
- By calling the FHIR terminology operations

In-memory

- Expand using the `ValueSetExpander` class in `Hl7.Fhir.Specification.Terminology`
- Set limits on expansion size
- Uses `IResourceResolver` to locate ValueSets
- Current limitations:
  - No filters
  - No imports of whole CodeSystems

Working with the expansion

- Useful extension methods on ValueSet
  - HasExpansion()
  - ExpansionSize()
  - FindInExpansion(String code, string system)
  - bool CodeInExpansion(String code, string system)

- Note: no automatic expansion, use ValueSetExpander

Terminology operations

- https://www.hl7.org/FHIR/valueset-operations.html
- $expand, $lookup, $validate-code

- class FhirClient
  - ExpandValueSet()
  - ConceptLookup()
  - ValidateCode()

ITerminologyService


```
namespace Hl7.Fhir.Specification.Terminology
{
    public interface ITerminologyService
    {
        OperationOutcome ValidateCode(string canonical = null, string context = null, ValueSet valueSet = null,
                string code = null, string system = null, string version = null, string display = null,
                Coding coding = null, CodeableConcept codeableConcept = null, FhirDateTime date = null,
                bool? @abstract = null, string displayLanguage = null);

    }
}
```

ITerminologyService (2)

- Current implementations:
  - `LocalTerminologyService`
    Does an in-memory expand & lookup
  - `ExternalTerminologyService`
    Uses the FhirClient calls to validate a code (possibly even sending your valueset across!)
  - `FallbackTerminologyService`
    First tries LocalTerminologyService, if that fails (too complex!), invoke an external service

- Note: returns `OperationOutcome`

FHIRPATH SUPPORT

FhirPath in .NET

- Available as a .NET Core library Hl7.FhirPath on NuGet
- Is built on top of `IElementNavigator` – so it could work on *any* object model
- Advantage: Compile once, run many (fast)
  - `class FhirPathCompiler`
  - `public CompiledExpression Compile(string expression)`
  - `CompiledExpression` is a native Lambda that will run the statement against an `IElementNavigator`
- You can also compile to an Expression tree (for debug/display purposes)

Using CompiledExpression

- First, compile the statement
  - `var ce = compiler.Compile("Patient.name");`
- Then:
  - `var result = ce(…)`
  - `result` is a set of IElementNavigators
- Or:
  - `object s = ce.Scalar(…);`
  - `bool p = ce.Predicate(…);`
  - `bool b = ce.IsBoolean(true, …);`

Convenience methods

- Comparable methods exist on `IElementNavigator` so you can directly query a source of data
- Example:

```
IElementNavigator nav =
     new PocoNavigator(myPatient);
object cnt =
      nav.Scalar("Patient.name.count()");
```

- Compiles & caches last 500 expressions for you

On POCO's directly…

- The Hl7.Fhir.Core assembly has extension methods in Hl7.Fhir.FluentPath for working directly on FHIR POCO's

```
Patient p = …;
object cnt = p.Scalar("Patient.name.count()");
```

HIDDEN GEMS

ResourceIdentity

- class ResoureIdentity : Uri
- Builds FHIR RESTful Uri's
- Factory methods Build():
  - .Build("Patient", "4E75", vid: "4")
    - `"/Patient/4E75/_history/4"`
- StructureDefinition URLs for core types
  - .Core(FHIRDefinedType.HumanName)
    - `"http://hl7.org/fhir/StructureDefinition/HumanName"`

ResourceIdentity (2)

- Even more useful: parsing RESTful Urls
- var ep = “http://server.org/fhir/Patient/4”
  var u = new ResourceIdentity(ep);

- u.ResourceType
  - “Patient”
- u.Id
  - “4”

Extension Manipulations

- IEnumerable<Extension> AllExtensions()
  - returns extensions + modifier extensions
- Extension GetExtension(string url)
  - searches both extensions + modifier extensions
- bool GetBoolExtension(string url), string GetStringExtension(...), integer GetIntegerExtension(...)
- Also setters, adders, removers, ...

Annotations

- POCO's and IElementNavigator also implement *IAnnotated and IAnnotatable*
- You can set transient annotations on such instances. Useful to keep in-memory state on FHIR Poco objects
  - e.g. url where you got the POCO from, resolved references, etc.
  - these are not serialized nor parsed!
  - Create a class for each type of annotation
  - Set or retrieve them using SetAnnotation<A>(A yourAnnotation)
  - Get them using A GetAnnotation<A>()

Annotations (2)

```
internal class ResourceOriginData
{
    public Uri Origin;
    public DateTimeOffset RetrievedWhen;
}


internal static ResourceOriginData GetResourceOrigin(this Resource subject)
        => subject.Annotation<ResourceBaseData>();

internal static void SetResourceOrigin(this Resource subject, ResourceOriginData data)
        => subject.SetAnnotation(data);
```

TransactionBuilder


```
var p = new Patient();
var b = new TransactionBuilder("http://myserver.org/fhir")
            .Create(p)
            .ResourceHistory("Patient","7")
            .Delete("Patient","8")
            .Read("Patient","9", versionId: "bla")
            .ToBundle();
```

QUESTIONS?

Suggestions for a pleasurable afternoon

HANDS-ON TRACK

IResourceResolver

- Try to use the ZipSource with the specification.zip to get some core resource definitions

- Now create an implementation of IResourceResolver that contains a List<ValueSet>, so it will resolve a list of in-memory ValueSets.

ITerminologyService

- Use the new IResourceResolver you wrote in the previous exercise to resolve a ValueSet. Then use the ValuesetExpander to expand it.
- Use ValueSet's FindInExpansion() to verify whether you succeeded.
- Look at the C# implementation of LocalTerminologyService. Could you make an implementation of ITerminologyService that calls the FhirClient.ValidateCode() instead, so it will use an external terminology service?