
Firely Server

Firely

Jan 02, 2023

FIRELY PRODUCTS

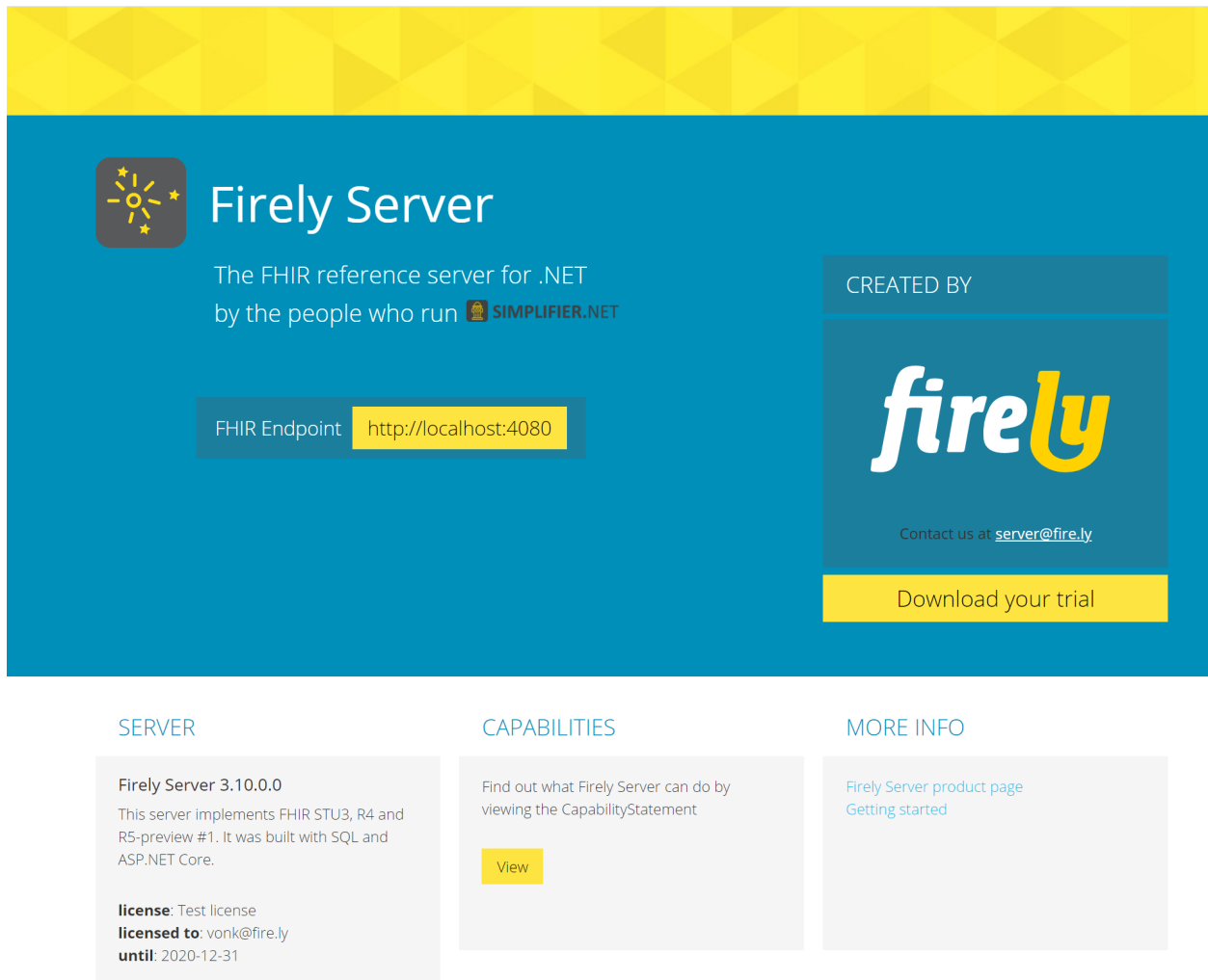
1	Overview of Firely Server, Plugins and Facades	3
1.1	Framework	3
1.2	Complete Server	4
1.3	Plugins	5
1.4	Facades	5
2	Getting Started	7
2.1	Configuration	8
2.2	Running the server	8
3	Release notes Firely Server	9
3.1	Older Vonk Release notes	9
3.2	Security notifications for Firely Server	30
3.3	Upgrading Firely Server	31
3.4	Public Endpoint Announcement 8 July 2021	31
3.5	Release 4.4.0	31
3.6	Release 4.3.0	32
3.7	Release 4.2.1 hotfix	33
3.8	Release 4.2.0	33
3.9	Release 4.1.3 hotfix	34
3.10	Release 4.1.2 hotfix	35
3.11	Release 4.1.1 hotfix	35
3.12	Release 4.1.0	35
3.13	Release 4.0.0	37
3.14	Release 3.9.3 hotfix	39
3.15	Release 3.9.2 hotfix	40
3.16	Release 3.9.1 hotfix	40
3.17	Release 3.9.0	40
3.18	Release 3.8.0	41
3.19	Release 3.7.0	42
3.20	Release 3.6.1	43
3.21	Release 3.6.0	43
3.22	Release 3.5.0	44
3.23	Release 3.4.0	46
3.24	Release 3.3.0	46
3.25	Release 3.2.1	47
3.26	Release 3.2.0	47
3.27	Release 3.1.3 hotfix	49
3.28	Release 3.1.0	49
3.29	Release 3.0.0	50


3.30	Release 3.0.0-beta2	52
3.31	Release 3.0.0-beta1	53
4	How to upgrade Firely Server?	57
4.1	Upgrading Firely Server	57
4.2	Upgrading Plugins	59
4.3	Upgrading Facades	59
5	Firely Server Roadmap	61
5.1	2021	61
6	Frequently asked questions	63
6.1	Conflicting resources upon import	63
6.2	Searchparameter errors for composite parameters	63
6.3	.NET SDK not found	64
7	Configuring Firely Server	65
7.1	Firely Server settings	65
7.2	Firely Server settings with Environment Variables	75
7.3	Configure the Administration API	77
7.4	Conformance Resources	78
7.5	Validating incoming resources	83
7.6	Using the In-Memory storage	85
7.7	Using MongoDB	86
7.8	Using SQL server	88
7.9	Using SQLite	92
7.10	Using Microsoft Azure CosmosDB	94
7.11	Configure http and https	95
7.12	Cross Origin Resource Sharing (CORS)	97
7.13	Log settings	97
8	Firely Server deployment options	103
8.1	Using Firely Server on Docker	103
8.2	Firely Server deployment on Azure Web App Service	110
8.3	Yellow Button - Firely Server for your Simplifier project	114
8.4	Deploy Firely Server on a reverse proxy	119
8.5	Set up an Identity Provider	129
8.6	Access Control Tokens with Postman	131
8.7	Performance of Firely Server	136
9	Security	141
9.1	Access control and SMART	141
10	Features	151
10.1	FHIR RESTful API	151
10.2	Bulk Data Export	155
10.3	Patient \$everything	158
10.4	Custom Operations	159
10.5	Custom Resources	161
10.6	Terminology services	163
10.7	Custom Search Parameters	168
10.8	Errata to the specification	171
10.9	Subscriptions	171
10.10	Auditing	172
10.11	Reset the database	174


10.12	Preloading resources	174
10.13	Multiple versions of FHIR	175
10.14	HIPAA compliance	178
11	Firely Server Administration API	181
11.1	Functions	181
11.2	Configuration	181
11.3	Database	181
12	Firely Server Plugins	183
12.1	Configure the pipeline	183
12.2	Configuration classes	185
12.3	Detailed logging of loading plugins	186
12.4	The order of plugins	188
12.5	Important classes and interfaces	191
12.6	Template for a plugin	191
12.7	Returning non-FHIR content from a plugin	192
12.8	Firely Server Plugin example - Create a new landing page	194
12.9	Firely Server Plugin example - \$document operation	201
12.10	BinaryWrapper plugin	201
12.11	Convert plugin	203
13	Firely Server Facade	205
13.1	Facade setup configuration	205
13.2	Exercise: Build your first Facade	206
13.3	Prerequisites and Preparations	206
13.4	Starting your project	207
13.5	Mapping the database	207
13.6	Enable Search	210
13.7	2. Get the data and map to FHIR	212
13.8	Finalizing search	214
13.9	Debugging the Facade	216
13.10	Finalizing your project	217
13.11	Enable changes to the repository	220
14	Firely Server Reference Documentation	225
14.1	Plugins available for Firely Server	225
14.2	Important classes and interfaces	247
14.3	Architecture	269
14.4	Dependencies of Firely Server and their licenses	270
15	Contact us	273
	Index	275

Firely Server is Firely's FHIR server. It was formerly named 'Vonk' and is the successor to our Spark server.

Firely Server is the answer to the growing need for a stable server that can be used in a variety of production environments. A public sandbox is available at <https://server.fire.ly>, which is free to use and intended for testing and educational purposes only.



 **Firely Server**

The FHIR reference server for .NET
by the people who run  SIMPLIFIER.NET

FHIR Endpoint `http://localhost:4080`

CREATED BY

firely

Contact us at server@fire.ly

[Download your trial](#)

SERVER

Firely Server 3.10.0.0
This server implements FHIR STU3, R4 and R5-preview #1. It was built with SQL and ASP.NET Core.

license: Test license
licensed to: vonk@fire.ly
until: 2020-12-31

CAPABILITIES

Find out what Firely Server can do by viewing the CapabilityStatement

[View](#)

MORE INFO

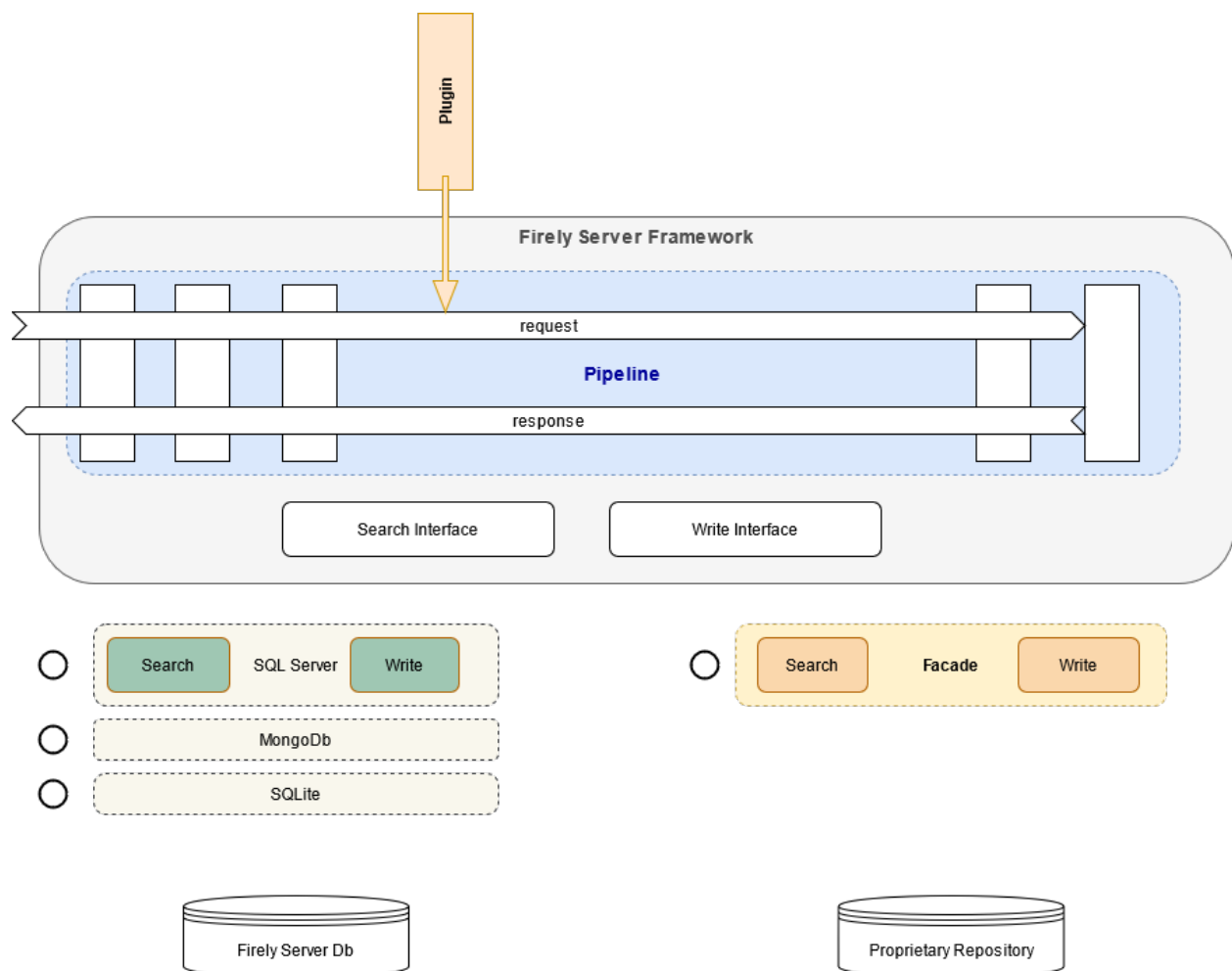
[Firely Server product page](#)
[Getting started](#)

On these pages we provide you with the documentation you need to get up and running with your own standard Firely Server installation, as well as information on how to contact us when you have additional needs, such as a custom implementation or support contract.

OVERVIEW OF FIRELY SERVER, PLUGINS AND FACADES

1.1 Framework

Firely Server is not just a FHIR Server, it is a processing pipeline for handling standard and custom FHIR requests. *Firely Server* consists of this pipeline filled with processors to handle the interactions defined in the FHIR RESTful API. With *Plugins* you can add your own processors to the framework to perform custom operations, or fill in cross-cutting concerns for your business. A *Facade* is a type of plugin that provides a data access layer for an existing data repository (e.g. a proprietary system). This image sums it all up:



Firely Server comes in several editions:

- Sandbox: try it right now on <https://server.fire.ly>
- Evaluation: get an evaluation license from [Simplifier.net](https://simplifier.net), allowing you to explore all functionality for free during a week (renewable)
- Community: use Firely Server for free, but only on SQLite
- Commercial use with professional support in different scales: Startup, Growth, Scale and Enterprise

For more information and pricing visit the [product site](#).

1.2 Complete Server

Firely Server is a FHIR Server out of the box. It is built with Microsoft .NET Core and runs on any of the platforms for which a [.NET Core Runtime](#) is available. Linux, Windows, MacOS, Docker etcetera. Installation can be done in minutes. After that you can configure main features and further details:

- Choose your *database*: [SQLite](#) is configured by default, but for serious use you'd want to configure [MongoDB](#) or [SQL Server](#).
- Configure the level of *validation*: Firely Server can be very loose or very strict on the validity of the resources that you send to it.
- Configure *endpoints* for FHIR versions that you want to support (since Firely Server (Vonk) 3.0.0: FHIR STU3 and FHIR R4)
- Fill in your *licensefile*.
- Adjust the *processing* pipeline by trimming it down (excluding certain plugins) or extending it with extra plugins.

Besides configuration of the settings, Firely Server features an [Administration API](#) that allows you to configure the so-called [Conformance Resources](#) that drive parsing, serialization, validation and terminology. The Administration API is pre-filled with Conformance Resources such as the StructureDefinitions, Searchparameters, CodeSystems and ValueSets that come with the FHIR Specification. Beyond that you can use the Administration API to make Firely Server aware of:

- Custom profiles, e.g. national or institutional restrictions on the standard FHIR resources.
- *Custom resources*: you can even define resources beyond those in FHIR and they are treated as if they were standard FHIR resources.
- CodeSystem and ValueSet resources for *terminology*.
- *Custom Searchparameters*: have Firely Server index and search resources on properties that are not searchable with the searchparameters from the FHIR Specification itself.

Read more on Firely Server:

- [Getting Started](#)
- [Features](#)
- [Firely Server deployment options](#)
- [Configuring Firely Server](#)
- [Release notes Firely Server](#)
- [Firely Server Administration API](#)

1.3 Plugins

A plugin is a library of code that you can buy, clone or create yourself that implements additional or replacement functionality in Firely Server. Examples are:

- Implementation of a custom operation. E.g. \$document (generate a document Bundle based on a Composition resource), which is available on GitHub. Or \$transform (execute a FHIR Mapping on a source structure to produce a target structure), which is developed by Healex and can be bought separately.
- Implementation of a cross-cutting concern. Imagine that in your organization every resource that is created or updated must be logged to a very specific location. You may create a plugin that does exactly that.
- Special handling of specific requests. E.g. requests for a Binary resource where you need to merge in binary data from one of your systems.
- Provide custom authentication and authorization methods for compliancy with business or governmental rules.

In all cases, a Plugin is technically a .NET Core assembly (or a set of them) containing well-defined configuration methods that allow Firely Server to:

- add services
- add a processor to the request processing pipeline

Most plugins do both, registering (testable) services that do the actual work and a thin layer around it that adds it as a processor to the pipeline.

Read more on *Firely Server Plugins*.

View the [session on Plugins](#) from DevDays 2018.

1.4 Facades

A Facade is a Firely Server processing pipeline working on an existing data repository. That repository could be the database of proprietary system, some API of an existing system or a whole Clinical Data Repository specifically created to open up data through a FHIR API.

The implementation of a Facade is a special type of plugin that registers services to access the existing data repository. By building the data access layer you leverage all of the FHIR processing in Firely Server, connected to your repository - thus creating a FHIR RESTful API for that repository with the least amount of work.

So a Facade is still a Plugin, and therefore technically a .NET Core assembly (or a set of them) having the same well-defined configuration methods. In the case of a Facade it usually only registers services (and no processor), specifically implementing the interfaces that define the data access layer in Firely Server:

- ISearchRepository, for reading and searching
- IResourceChangeRepository: for creating, updating, and deleting

Read more on *Firely Server Facade*.

View the [session on Facade](#) from DevDays 2018.

GETTING STARTED

If you want to start using the standard Firely Server in your own Windows environment, follow the steps on this page to install and run the server. For non Windows systems, or if you want to use Docker for Windows, please look at the *Using Firely Server on Docker* section.

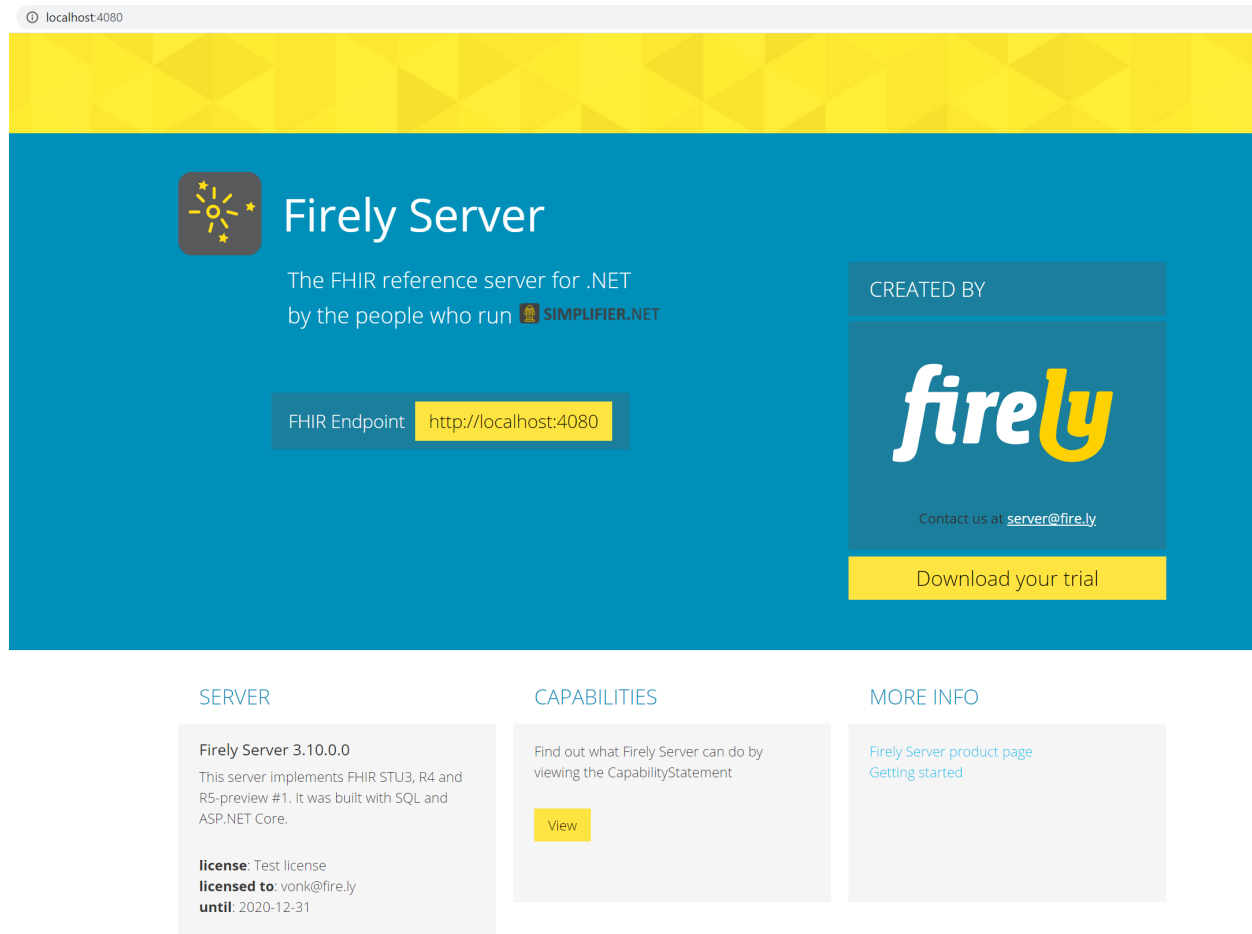
1. Download the Firely Server binaries and the license file from [Simplifier.net](https://simplifier.net).
2. Extract the downloaded files to a location on your system, for example: C:\FirelyServer. We will call this the working directory.
3. Put the license file in the working directory.
4. In the working directory create a new JSON file and name it `appsettings.json`. You will use this file for settings that you want to differ from the defaults in `appsettings.default.json`. For more background on how the settings are processed, see *Firely Server settings*
5. Open `appsettings.json`, copy the `LicenseFile` setting from `appsettings.default.json` to it and change this property to the name of your license file. For example

```
"License": {  
  "LicenseFile": "firelyserver-trial-license.json"  
}
```

Important: The next step assumes you have a .Net Core environment installed. If not, please [download and install ASP.NET Core Runtime 3.1.x](#) before you continue. Choose the latest security patch to mitigate security issues in previous versions.


6. Open a command prompt or Powershell, navigate to the working directory and run: `> dotnet .Firely.Server.dll`

Firely Server will then run on port 4080 of the system.
7. If you want to check if Firely Server is running correctly, open a browser and go to `localhost:4080`. You should see a homepage similar to this:




localhost:4080

Firely Server

The FHIR reference server for .NET
by the people who run  SIMPLIFIER.NET

FHIR Endpoint `http://localhost:4080`

CREATED BY



Contact us at server@fire.ly

Download your trial

SERVER

Firely Server 3.10.0.0
This server implements FHIR STU3, R4 and R5-preview #1. It was built with SQL and ASP.NET Core.

license: Test license
licensed to: vonk@fire.ly
until: 2020-12-31

CAPABILITIES

Find out what Firely Server can do by viewing the CapabilityStatement

[View](#)

MORE INFO

[Firely Server product page](#)
[Getting started](#)

Please note that the third example query `/Patient/example` will only work if you first PUT a Patient with the id 'example'. You can get this [example from the specification](#).

2.1 Configuration

The section *Configuring Firely Server* explains how you can configure the Firely Server.

2.2 Running the server

When you have completed your configuration changes, you can run the server. Open a command prompt or Powershell, navigate to your working directory and run:

```
> dotnet .\Firely.Server.dll
```

RELEASE NOTES FIRELY SERVER

3.1 Older Vonk Release notes

3.1.1 Release 2.2.0

Database

1. SQL Server: the index tables have their clustered index on their link to the resources. SQL script '20190919000000_Cluster_Indexes_On_EntryId.sql' (found in the /data folder of the Vonk distribution) must be applied to existing Vonk SQL databases (both to the admin and to the data repositories) to make this change.

Attention: Vonk 2.2.0 (using SQL server) will not start unless this script has been applied to the databases. Please note that running the script can take considerable time, especially for large databases.

Features

1. When running Vonk in IIS, it now profits from the [in-process hosting model](#) that ASP.NET Core offers.
2. Further improved concurrent throughput for SQL Server.

Fix

1. On errors in a transaction, Vonk would not point out the entry if it had no fullUrl. Improved this message by using the resourcetype and id (if present) instead.
2. _include gave a 500 responsecode if a resource contains absolute references.

3.1.2 Release 2.1.0

Database

1. SQL Server: Improved concurrent throughput.

Features

1. Upgrade to HL7.Fhir.Net API 1.3, see the [Older SDK release notes](#).
2. Vonk calls `UseIISIntegration` for better integration with IIS (if present).

Fix

1. Transactions: references to other resources in the transaction were not updated if the resource being referenced was in the transaction as an update (PUT). (this error was introduced in 2.0.0).

3.1.3 Release 2.0.1 hotfix

Fix

1. Supported Interactions were not checked for custom operations. In the *appsettings.json* the custom operations, like `$meta`, were ignored. This has been fixed now.

3.1.4 Release 2.0.0 final

This is the final release of version 2.0.0, so the -beta is off. If you directly upgrade from version 1.1, please also review all the 2.0.0-beta and -beta2 release notes below.

Attention: We upgraded the version of .NET Core used to 2.2. Please get the latest 2.2.x runtime from the [.NET download site](#). The update was needed for several security patches and speed improvements.

Attention: The structure of the Validation section in the settings has changed. See [Validating incoming resources](#) for details.

Attention: This version of Vonk is upgraded to the HL7.Fhir.API version 1.2.0. Plugin- and Facade builders will transitively get this dependency through the Vonk.Core package.

Database

No changes have been made to any of the database implementations.

Fix

1. When you created a StructureDefinition for a new resourcetype on /administration, the corresponding endpoint was not enabled.
2. Vonk does not update references in a transaction when a conditional create is used for an existing resource.
3. Paths in PipelineOptions would interfere if one was the prefix of the other.
4. Indexing a HumanName with no values but just extensions failed.
5. The selflink in a bundle did not contain the sort parameters. In this version the selflink always contains a sort and a count parameter, even if they were not in the request and the default values have been applied.
6. The import of conformance resources from specification.zip yielded warnings on .sch files.
7. Errors introduced in the 2.0.0-beta versions:
 1. Syntax errors in the XML or JSON payload yielded an exception, now they are reported with an `OperationOutcome` upon parsing.
 2. \$expand and other terminology operations caused a `NullReference` exception.
 3. `_element` did not include the mandatory elements.

Feature

1. Vonk supports Custom Resources. See [Custom Resources](#).
2. Operation `$meta` is now supported, to quickly get the tags, security labels and profiles of a resource.
3. /metadata, retrieving the CapabilityStatement performs a lot better (just the initial call for a specific Accept-Type takes a bit longer).
4. Validation can be controlled more detailed. Choose the strictness of parsing independent of the level of validation. With this, the settings section 'Validation' has also changed. See [Validating incoming resources](#).

Plugin and Facade API

1. We upgraded the embedded Fhir.Net API to version 1.2, see the [Older SDK release notes](#).
2. Together with the upgrade to .NET Core 2.2, several libraries were updated as well. Most notably Microsoft.EntityFrameworkCore.*, to 2.2.3.

3.1.5 Release 2.0.0-beta2

Fix

- Fixed RelationalQuery in Vonk.Facade.Relational, so Vonk.Facade.Starter can be used again.

3.1.6 Release 2.0.0-beta

We have refactored Vonk internally to accomodate future changes. There are only minor functional changes to the FHIR Server. Facade and Plugin builders must be aware of a few interface changes, most notably to the IResource interface.

This release is a *beta* release because of the many internal changes, and because we expect to include a few more in the final release. Have a go with it in your test environment to see whether you encounter any trouble. We also encourage you to build your plugin and/or facade against it to prepare for code changes upon the final release.

You can still access the latest final release (1.1.0):

- Binaries: through the [Simplifier downloads page](#), choose 'List previous versions'.
- Docker: `docker pull simplifier/vonk:1.1.0`
- NuGet: `<PackageReference Include="Vonk.Core" Version="1.1.0" />`

Database

No changes have been made to any of the database implementations.

Fix

1. The *\$validate* operation processes the profile parameter.
2. If an update brings a resource 'back to life', Vonk returns statuscode 201 (previously it returned 200).
3. On an initial Administration Import of specification.zip, Vonk found an error in valueset.xml. This file was fixed in the specification.zip that comes with Fhir.NET API 1.1.2.
4. Transaction: references within the transaction are automatically changed to the id's the referenced resources get from Vonk when processing the transaction. This did not happen for references inside extensions. It does now.
5. Administration Import: an Internal Server Error could be triggered with a zip file with nested directories in it.
 - NB: Directories in your zip are still not supported because of [Fhir.NET API issue #883](#), but Vonk will not error on it anymore.
6. Search: The entry.fullUrl for an OperationOutcome in a Search bundle had a relative url.
7. Search: Processed _elements and _summary arguments were not reported in the selflink of the bundle (or any of the paging links).
8. Search: The selflink will include a _count parameter, even if it was not part of the request and hence the default value for _count from the *BundleOptions* was applied.
9. Search on :exact with an escaped comma (e.g. /Patient?name:exact=value1\,value2) was executed as a choice. Now the escape is recognized, and the argument processed as one term.

Feature

1. Upgraded Fhir.NET API to version 1.1.2, see the [Older SDK release notes](#).
2. The Vonk Administration API now allows for StructureMap and GraphDefinition resources to be loaded.
3. The opening page of Vonk (and the only UI part of it) is updated. It no longer contains links that you can only execute with Postman, and it has a button that shows you the CapabilityStatement.
4. We published our custom operations on [Simplifier!](#) And integrated those links into the CapabilityStatement.
5. You can now access older versions of the Vonk binaries through the Simplifier downloads. (This was already possible for the Docker images and NuGet packages through their respective hubs).
6. [Vonk.IdentityServer.Test](#) and [Vonk.Facade.Starter](#) have been integrated into the Continuous Integration system.
7. In JSON, the order of the output has changed:
 1. If id and/or meta elements were added by Vonk (on a create or update), they will appear at the end of the resource.

Plugin and Facade API

1. IResource interface and related classes have had several changes. If you encounter problems with adapting your code, please contact us.
 - It derives from the ISourceNode interface from the Fhir.NET API.
 - Change and Currency are properties that were only relevant in the repository domain, and not in the rest of the pipeline. They have been deprecated. You can access the values still with resource.GetChangeIndicator() and resource.GetCurrencyIndicator(). This is implemented with Annotations on the ISourceNode. All of Vonk's own implementations retain those annotations, but if the relevant annotation is somehow missing, default values are returned (ResourceChange.NotSet resp. ResourceCurrency.Current).
 - The Navigator property is obsolete. The type of it (IElementNavigator) is obsolete in the Fhir.NET API. To run FhirPath you provide type information and run the FhirPath over an ITypedElement:

```
//Have IStructureDefinitionSummaryProvider _schemaProvider injected in the
↳ constructor.
var typed = resource.ToTypedElement(_schemaProvider);
var matchingElements = typed.Select('your-fhirpath-expression');
```

- Id, Version and LastUpdated can no longer be set directly on the IResource instance. IResource has become **immutable** (just like ISourceNode). The alternatives are:

```
var resourceWithNewId = resource.SetId("newId");
var resourceWithNewVersion = resource.SetVersion("newVersion");
var resourceWithNewLastUpdated = resource.SetLastUpdated(DateTimeOffset.UtcNow);
```

- Because the IChangeRepository is responsible for creating new id's and versions, we also included extensions methods on it to update all three fields at once:

```
var updatedeResource = changeRepository.EnsureMeta(resource, KeepExisting.Id /↳
↳ Version / LastUpdated);
var updatedResource = changeRepository.FreshMeta(resource); //replaces all three
```

2. The PocoResource class is obsolete. To go from a POCO (like an instance of the Patient class) to an IResource, use the ToIResource() extension method found in Vonk.Fhir.R3.

3. The PocoResourceVisitor class is obsolete. Visiting can more effectively be done on an ITypedElement:

```
//Have IStructureDefinitionSummaryProvider _schemaProvider injected in the  
↳ constructor.  
var typed = resource.ToTypedElement(_schemaProvider);  
typed.Visit((depth, element) => { //do what you want with element});
```

4. SearchOptions has changed:

- Properties Count and Offset have been removed.
- Instead, use _count and _skip arguments in the IArgumentCollection provided to the SearchRepository.Search method if you need to.

5. We have created a template for a plugin on [GitHub](#). Fetch it for a quick start of your plugin.

3.1.7 Release 1.1.0

Attention: New security issues have been identified by Microsoft. See the [Security notifications for Firely Server](#) for details.

Attention: The setting for the location of the license file has moved. It was in the top level setting LicenseFile. It still has the same name, but it has moved into the section License. See [License](#) for details.

Attention: This version of Vonk is upgraded to the Hl7.Fhir.API version 1.1.1. Plugin- and Facade builders will transitively get this dependency through the Vonk.Core package.

Database

No changes have been made to any of the database implementations.

Feature

1. Vonk will count the number of requests that it processes. See [License](#) for settings on that. Because of this change, the LicenseFile setting has moved from the top level to under License.
2. The plugin folder ([Configuring the Firely Server Pipeline](#)) may now contain subfolders. Plugins will be read from all underlying folders.
3. Vonk supports If-Match on update. See [Managing Resource Contention](#) in the specification for details.
4. Plugins may return non-FHIR content. See [Returning non-FHIR content from a plugin](#).
5. This feature may also be used for [Custom Authentication](#).
6. A [Template for a plugin](#) is added to the documentation.
7. A documentation page on performance is added: [Performance of Firely Server](#).
8. Upgrade of the Hl7.Fhir.API library to 1.1. See the [Older SDK release notes](#).

Fix

1. Transaction: forward references from one resource to another in a Transaction were not correctly resolved.
2. When you set `ValidateIncomingResources` to true, Vonk no longer accepts resources with extensions that are unknown to it. This is now also reflected in the `CapabilityStatement.acceptUnknown`.
3. The links in a bundle response (`Bundle.link`) were relative links. Now they are absolute links.
4. HTTP 500 instead of an OO was returned when trying to update a subscription with an invalid request status.
5. If an error is found in a `SearchParameter` in the Administration database, Vonk logs the (canonical) url of that `SearchParameter` for easier reference.
6. Transaction: Response bundle contained versioned fullUrls. We changed that to unversioned urls.
7. Bundles: Response bundles with an `OperationOutcome` contained a versioned fullUrl for the entry containing the `OperationOutcome`. We changed that to an unversioned url.
8. Deleting a resource from the Administration API that does not exist would lead to an internal server error.

Supported Plugins

1. Several fixes have been done on the [Document plugin](#).

3.1.8 Release 1.0.0

Yes! Vonk version 1.0 is out. It is also the first version that is released without the -beta postfix. It has been very stable from the very first version, and now we think it is time to make that formal.

Release 1.0.0 is functionally identical to 0.7.4.0. But we optimized the deployment process for *Yellow Button - Firely Server for your Simplifier project* and *Docker* in general. The contents of the core specification are now preloaded in the SQLite administration database, so your first startup experience is a lot faster.

3.1.9 Release 0.7.4.0**Database**

1. The index definitions for SQL Server have been updated for improved performance. This should be handled automatically when you start Vonk 0.7.4 and have *AutoUpdateDatabase* enabled.

Fix

1. Posting a resource with an invalid content-type to the regular FHIR endpoint should result in HTTP 415 and not HTTP 400.
2. Warning ‘End method “PocoResourceVisitor.VisitByType”, could not cast entity to PocoResource.’ in the log was incorrect.
3. When running Administration API on SQLite and Vonk on SQL Server, update or delete would fail.
4. Handle quantity with very low precision (e.g. ‘3 times per year’ - 3|http://unitsofmeasure.org/a).
5. POST to <vonk_base>/Administration/* with another Content-Type than application/json or application/xml results in HTTP 500.

Feature

1. Support forward references in a *Transaction bundle*. Previously Vonk would only process references back to resources higher up in the bundle.
2. Performance of Validation and Snapshot Generation has improved by approximately 10 times...
3. ... and correctness has improved as well.
4. Administration API also support the NamingSystem resource.

3.1.10 Release 0.7.3.0

Fix

1. Search on /administration/Subscription was broken
2. Neater termination of the Subscription evaluation process upon Vonk shutdown
3. A Bundle of type batch is now rejected if it contains internal references.
4. Urls in the narrative (href and src) are also updated to the actual location on the server.
5. A system wide search on compartment returns 403, explaining that that is too costly.

3.1.11 Release 0.7.2.1

Fix

1. Delete on /administration was broken.

3.1.12 Release 0.7.2.0

Database

1. Fixes 2 and 3 require a reindex for specific searchparameters, if these parameters are relevant to you.

Features and fixes

1. Fix: Reject a search containing a modifier that is incorrect or not supported.
2. Fix: The definition for searchparameter Encounter.length was unclear. We added the correct definition from FHIR R4 to the errata.zip, so it works for STU3 as well. If this is relevant for you, you may want to reindex for this searchparameter. See *Rebuild the search index for specific searchparameters*, just for 'Encounter.length'.
3. Fix: Error "Unable to index for element of type 'base64Binary'". This type of element is now correctly indexed. One known searchparameter that encounters this type is Device.udi-carrier. If this is relevant to you, you may want to reindex for this searchparameter. See *Rebuild the search index for specific searchparameters*, just for 'Device.udi-carrier'.
4. Fix: Validation would fail on references between contained resources. See also fix #423 in the [Older SDK release notes](#).
5. Fix: E-tag was missing from the response on a delete interaction.

6. Fix: An invalid mimetype in the `_format` parameter (like `_format=application/foobar`) returned response code 400 instead of 415.
7. Fix: If a subscription errors upon execution, not only set the status to error, but also state the reason in `Subscription.error` for the user to inspect.
8. Fix: Search on `/Observation?value-string:missing=false` did not work. As did the missing modifier on other searchparameters on `value[x]` elements.
9. Feature: After `/administration/importResources` (see [Load Conformance Resources on demand](#)), return an `OperationOutcome` detailing the results of the operation.
10. Feature: Upon usage of a wrong value for `_summary`, state the possible, correct values in the `OperationOutcome`.
11. Feature: Allow for multiple deletes with a Conditional Delete, see [Create, read, update, patch, delete](#).
12. Feature: The version of Vonk is included in the log file, at startup.
13. Configuration: Add `Vonk.Smart` to the `PipelineOptions` by default, so the user only needs to set the `SmartAuthorizationOptions.Enabled` to true.
14. Upgrade: We upgraded to the latest C# driver for MongoDB (from 2.4.4 to 2.7.0).

3.1.13 Release 0.7.1.1

Fix

Spinning up a Docker container would crash the container because there was no data directory for SQLite (the default repository). This has been solved now: Vonk will create the data directory when it does not exist.

3.1.14 Release 0.7.1.0

Attention: Fix nr. 8 requires a `reindex/searchparameters` with `include=Resource._id,Resource._lastUpdated,Resource._tag`. Please review [Re-indexing for new or changed SearchParameters](#) on how to perform a reindex and the cautions that go with it. Also note the changes to reindexing in fix nr. 1.

Database

1. We added support for SQLite! See [Using SQLite](#) for details.
2. We also made SQLite the default setting for both the main Vonk database and the [Firely Server Administration API](#).
3. With the introduction of SQLite we advise running the Administration API on SQLite. In the future we will probably deprecate running the Administration API on any of the other databases.
4. Support for CosmosDB is expanded, though there are a *few limitations*.

Facade

1. If you rejected the value for the `_id` searchparameter in your repository, Vonk would report an `InternalServerError`. Now it reports the actual message of your `ArgumentException`.

Features and fixes

1. We sped up *Re-indexing for new or changed SearchParameters*. The request will be responded to immediately, while Vonk starts the actual reindex asynchronously and with many threads in parallel. Users are guarded against unreliable results by blocking other requests for the duration of the reindex. Reindexing is still not to be taken lightly. It is a **very heavy** operation that may take very long to complete. See *Re-indexing for new or changed SearchParameters* for details.
2. A really large bundle could lead Vonk (or more specifically: the validator in Vonk) to a `StackOverflow`. You can now set *limits* to the size of incoming data to avoid this.
3. *Reindexing* is supported on CosmosDB, but it is less optimized than on MongoDB.
4. Using `_include` or `_revinclude` would yield an `OperationOutcome` if there are no search results to include anything on. Fixed that to return 404 as it should.
5. Using the `:not` modifier could return false positives.
6. A batch or transaction with an entry having a value for `IfModifiedSince` would fail.
7. History could not be retrieved for a deleted resource. Now it can.
8. *Reindex* would ignore the generic searchparameters defined on Resource (`_id`, `_lastUpdated`, `_tag`). Because `id` and `lastUpdated` are also stored apart from the search index, this was really only a problem for `_tag`. If you rely on the `_tag` searchparameter you need to reindex **just for the searchparameter** ```Resource._tag```.
9. Vonk logs its configuration at startup. See *Log of your configuration* for details.

3.1.15 Release 0.7.0.0

Database

1. Indexes on the SQL Server repository were updated to improve performance. They will automatically be applied with *AutoUpdateDatabase*.

Facade

1. Release 0.7.0.0 is compatible again with Facade solutions built on the packages with versions 0.6.2, with a few minor changes. Please review the `Vonk.Facade.Starter` project for an example of the necessary adjustments. All the differences can be seen in [this file comparison](#).
2. Fix: The SMART authorization failed when you don't support all the resourcetypes. It will now take into account the limited set of supported resourcetypes.
3. Fix: `Vonk.Facade.Relational.RelationalQueryFactory` would lose a `_count` argument.
4. Documentation: We added documentation on how to implement Create, Update and Delete in a facade on a relational database. See *Enable changes to the repository*. This is also added to the [example Facade solution](#) on GitHub.

Features and fixes

1. Feature: *Vonk FHIR Plugins* has been released. You can now add libraries with your own plugins through configuration.
2. Feature: Through *Vonk FHIR Plugins* you can replace the landing page with one in your own style. We provided an *example* on how to do that.
3. Feature: You can now start Vonk from within another directory than the Vonk binaries directory, e.g. `c:\programs>dotnet .\vonk\vonk.server.dll`.
4. Feature: You can configure the maximum number of entries allowed in a Batch or Transaction, to avoid overloading Vonk. See *Batch and transaction*.
5. Upgrade: We upgraded the FHIR .NET API to version 0.96.0, see the *Older SDK release notes* for details. Mainly #599 affects Vonk, since it provides the next...
6. Fix: Under very high load the FhirPath engine would have concurrency errors. The FhirPath engine is used to extract the search parameters from the resources. This has been fixed.
7. Fix: Search on a frequently used tag took far too long on a SQL Server repository.
8. Fix: The *Patient.deceased* search parameter from the specification had an error in its FhirPath expression. We put a corrected version in the *errata.zip*.
9. Fix: Several composite search parameters on Observation are defined incorrectly in the specification, as is reported in *GForge issue #16001*. Until the specification itself is corrected, we provide corrections in the *errata.zip*.
10. Fix: Relative references in a resource that start with a forward slash (like */Patient/123*) could not be searched on.
11. Fix: System wide search within a compartment looked for the pattern `<base>/Patient/123/?_tag=bla`. Corrected this to `<base>/Patient/123/*?_tag=bla`
12. Fix: When loading *Simplifier resources*, Vonk can now limit this to the changes since the previous import, because the Simplifier FHIR endpoint supports `_lastUpdated`.
13. Fix: *Conformance resources* are always loaded into the Administration API when running on a Memory repository. Or actually, always if there are no StructureDefinitions in the Administration database. To enable this change, imported files are no longer moved to the *AdministrationOptions.ImportedDirectory*.
14. Fix: *Re-indexing for new or changed SearchParameters* would stop if a resource was encountered that could not properly be indexed. It will now continue working and report any errors afterwards in an *OperationOutcome*.
15. Fix: The terms and privacy statement on the default landing page have been updated.
16. Fix: When searching on a search parameter of type date, with an argument precision to the minute (but not seconds), Vonk would reject the argument. It is now accepted.
17. Fix: DateTime fields are always normalized to UTC before they are stored. This was already the case on MongoDB, and we harmonized SQL and Memory to do the same. There is no need to reindex for this change.
18. Fix: When you use accents or Chinese characters in the url for a search, Vonk gives an error.
19. Fix: A reverse chained search on MongoDB sometimes failed with an Internal Server Error.

3.1.16 Release 0.6.5.0

Attention: This version changes the way conformance resources are loaded from zip files and/or directories at startup. They are no longer loaded only in memory, but are added to the Administration API's database. You will notice a delay at first startup, when Vonk is loading these resources into the database. See Feature #1 below.

Attention: 2018-06-07: We updated the Database actions for 0.6.5.0, you should always perform a reindex, see right below.

Database

1. Feature 2, 4 and 14 below require a *reindex/all*, both for MongoDB and SQL Server.

Facade

1. Release 0.6.5.0 is not released on NuGet, so the latest NuGet packages have version 0.6.2-beta. Keep an eye on it for the next release...

Features and fixes

1. Feature: Run Vonk from your Simplifier project! See [Use Firely Server with your Simplifier artifacts](#) for details.
2. Feature: Vonk supports Microsoft Azure CosmosDB, see [Using Microsoft Azure CosmosDB](#). This required a few small changes to the MongoDB implementation (the share the drivers), so please reindex your MongoDB database: *reindex/all*.
3. Feature: Configuration to restrict support for ResourceTypes, SearchParameters and CompartmentDefinitions, see [Restrict supported resources and SearchParameters](#).
4. Feature: Errata.zip: collection of corrected search parameters (e.g. that had a faulty expression in the FHIR Core specification), see [Errata to the specification](#)
5. Upgrade: FHIR .NET API 0.95.0 (see the [Older SDK release notes](#))
6. Fix: a search on `_id:missing=true` was not processed correctly.
7. Fix: better distinction of reasons to reject updates (error codes 400 vs. 422, see [RESTful API specification](#))
8. Fix: recognize `_format=text/xml` and return xml (instead of the default json)
9. Fix: handling of the `:not` modifier in token searches (include resource that don't have a value at all).
10. Fix: handling of the `:not` modifier in searches with choice arguments
11. Fix: `fullUrl` in return bundles cannot be version specific.
12. Fix: evaluate `_count=0` correctly (it was ignored).
13. Fix: correct error message on an invalid `_include` (now Vonk tells you which resourcetypes are considered for evaluating the used searchparameter).
14. Fix: indexing of `Observation.combo-value-quantity` failed for UCUM code for Celcius. This fix requires a *reindex/all* on this searchparameter.
15. Fix: total count in history bundle.

16. Fix: on vonk.fire.ly we disabled validating all input, so you can now create or update resources also if the relevant profiles are not loaded (this was necessary for Crucible, since it references US Core profiles, that are not present by default).
17. Fix: timeout of Azure Web App on first startup of Vonk - Vonk's first startup takes some time due to import of the specification (see [Default Conformance Resources](#)). Since Azure Web Apps are allowed a startup time of about 3 minutes, it failed if the web app was on a low level service plan. Vonk will now no longer await this import. It will finish startup quickly, but until the import is finished it will return a 423 'Locked' upon every request.
18. Fix: improved logging on the import of conformance resources at startup (see [Import of Conformance Resources](#)).

3.1.17 Release 0.6.4.0

Attention: This version changes the way conformance resources are loaded from zip files and/or directories at startup. They are no longer loaded only in memory, but are added to the Administration API's database. You will notice a delay at first startup, when Vonk is loading these resources into the database. See Feature #1 below.

Database

1. Fix #9 below requires a [reindex/all](#).

Facade

1. Release 0.6.4.0 is not released on NuGet, so the latest NuGet packages have version 0.6.2-beta. This release is targeted towards the Administration API and [Terminology services](#), both of which are not (yet) available in Facade implementations. We are working on making the features of the Administration API available to Facade implementers in an easy way.

Features and fixes

1. Feature: Make all loaded conformance resources available through the Administration API.

Previously:

- Only SearchParameter and CompartmentDefinition resources could be loaded from ZIP files and directories;
- And those could not be read from the Administration API.

Now:

- The same set of (conformance) resourcetypes can be read from all sources (ZIP, directory, Simplifier);
- They are all loaded into the Administration database and can be read and updated through the Administration API.

Refer to [Conformance Resources](#) for details.

2. Feature: Experimental support for [Terminology services](#) operations \$validate-code, \$expand, \$lookup, \$compose.
3. Feature: Support for [Compartment Search](#).
4. Feature: Track timing of major dependencies in [Azure Application Insights](#).

5. Feature: *Log settings* can be overridden in 4 levels, just as the appsettings. The logsettings.json file will not be overwritten anymore by a Vonk distribution.
6. Fix: The check for *allowed profiles* is no longer applied to the Administration API. Previously setting Allowed-Profiles to e.g. <http://mycompany.org/fhir/StructureDefinition/mycompany-patient> would prohibit you to actually create or update the related StructureDefinition in the Administration API.
7. Fix: When posting any other resourcetype than the supported conformance resources to the Administration API, Vonk now returns a 501 (Not Implemented).
8. Fix: Support search on Token with only a system (e.g. `<base>/Observation?code=http://loinc.org|`)
9. Fix: Support search on Token with a fixed system, e.g. `<base>/Patient?gender=http://hl7.org/fhir/codesystem-administrative-gender.html|female`. This fix requires a *reindex/all*.
10. Fix: Reindex could fail when a Reference Searchparameter has no targets.
11. Fix: Vonk works as Data Server on *ClinFHIR*, with help of David Hay.
12. Fix: Clearer error messages in the log on configuration errors.
13. Fix: Loading conformance resources from disk in Docker.

Documentation

1. We added documentation on *using IIS or NGINX as reverse proxies* for Vonk.
2. We added documentation on running Vonk on Azure Web App Services.

3.1.18 Release 0.6.2.0

Attention: The loading of appsettings is more flexible. After installing a new version you can simply paste your previous appsettings.json in the Vonk directory. Vonk's default settings are now in appsettings.default.json. see *Firely Server settings* for details.

Database

No changes

Features and fixes

1. Feature: Conditional References in *Transactions* are resolved.
2. Feature: More flexible support for different serializers (preparing for ndjson in Bulkdata)
3. Feature: Improved handling on missing settings or errors in the *Firely Server settings*.
4. Feature: Improved *logging*, including Dependency Tracking on Azure Application Insights, see *Application Insights*
5. Feature: SearchParameter and CompartmentDefinition are now also imported from *Simplifier*, so both Simplifier import and the *Administration API* support the same set of conformance resources: StructureDefinition, SearchParameter, CompartmentDefinition, ValueSet and CodeSystem. See *Conformance resources*.
6. Feature: Loading of appsettings is more flexible, see *Firely Server settings*.
7. Feature: Added documentation on running Vonk behind IIS or NGINX: *Deploy Firely Server on a reverse proxy*.

8. Performance: Improvement in speed of validation, especially relevant if you are *Validating incoming resources*.
9. Fix: If you try to load a SearchParameter (see *Load Conformance Resources from disk*) that cannot be parsed correctly, Vonk puts an error about that in the log.
10. Fix: Results from `_include` and `_revinclude` are now marked with `searchmode: Include` (was incorrectly set to `'Match'` before)
11. Fix: `_format` as one of the parameters in a POST Search is correctly evaluated.
12. Fix: No more errors in the log about a Session being closed before the request has finished (“Error closing the session. System.OperationCanceledException: The operation was canceled.”)
13. Fix: `Subscription.status` is evaluated correctly upon create or update on the Administration API
14. Fix: Token search with only a system is supported (`Observation.code=somesystem|`)
15. Fix: On validation errors like ‘Cannot resolve reference Organization/Organization-example26’ are now suppressed since the validator is set not to follow these references.
16. Fix: New Firely logo in SVG format - looks better
17. Fix: Creating resources with duplicate canonical url’s on the Administration API is prohibited, see *Conformance Resources*.
18. Fix: If a Compartment filter is used on a parameter that is not implemented, Vonk will return an error, see *Compartments*.

3.1.19 Release 0.6.1.0

Name change from Furore to Firely

3.1.20 Release 0.6.0.0

Attention:

- `SearchParametersImportOptions` is renamed to *MetadataImportOptions*.
- *Subscription* can now be disabled from the settings.

Database

1. The MongoDB implementation got a new index. It will be created automatically upon startup.

Features and fixes

1. Feature: *Access control based on SMART on FHIR*.
2. Feature: Vonk can also load `CompartmentDefinition` resources. See *Conformance Resources* for instructions.
3. Feature: `ValueSet` and `CodeSystem` resources can be loaded into the administration endpoint, and loaded from Simplifier. See *Conformance Resources* for instructions.
4. Feature: Be lenient on trailing slashes in the url.
5. Feature: `OperationOutcome` is now at the top of a Bundle result. For human readers this is easier to spot any errors or warnings.

6. Fix: In the *settings for SQL Server* it was possible to specify the name of the Schema to use for the Vonk tables. That was actually not evaluated, so we removed the option for it. It is fixed to 'vonk'.
7. Fix: The OperationOutcome of the *Reset* operation could state both an error and overall success.
8. Fix: If you did not set the CertificatePassword in the appsettings, Vonk would report a warning even if the password was not needed.
9. Fix: *Loading conformance resources* in the SQL Server implementation could lead to an error.
10. Fix: Clearer error messages if the body of the request is mandatory but empty.
11. Fix: Clearer error message if the Content-Type is missing.
12. Fix: GET on [base]/ would return the UI regardless of the Accept header. Now if you specify a FHIR mimetype in the Accept header, it will return the result of a system wide search.
13. Fix: In rare circumstances a duplicate logical id could be created.
14. Fix: GET [base]/metadat would return status code 200 (OK). But it should return a 400 and an OperationOutcome stating that 'metadat' is not a supported resourcetype.

Documentation

1. We consolidated documentation on loading conformance resources into *Conformance Resources*.

3.1.21 Release 0.5.2.0

Attention: Configuration setting SearchOptions is renamed to BundleOptions.
--

Features and fixes

1. Fix: When you specify LoadAtStartup in the *ResourceLoaderOptions*, an warning was displayed: "WRN No server base configured, skipping resource loading."
2. Fix: *Conditional create* that matches an existing resource returned that resource instead of an OperationOutcome.
3. Fix: *_has*, *_type* and *_count* were in the CapabilityStatement twice.
4. Fix: *_elements* would affect the stored resource in the Memory implementation.
5. Fix: Getting a resource with an invalid id (with special characters or over 64 characters) now returns a 404 instead of 501.
6. Feature: *Re-indexing for new or changed SearchParameters* now also re-indexes the Administration API database.
7. Fix: modifier *:above* for parameter type Url now works on the MongoDB implementation.
8. Fix: Vonk would search through inaccessible directories for the specification.zip.
9. Fix: Subscription could not be posted if 'Database' was not one of the SearchParametersImportOptions.
10. Fix: *_(rev)include=** is not supported but was not reported as such.
11. Fix: In a searchresult bundle, the references to other resources are now made absolute, referring to the Vonk server itself.
12. Fix: *BundleOptions* (previously: SearchOptions) settings were not evaluated.

13. Fix: Different responses for invalid resources when you change ValidateIncomingResources setting (400 vs. 501)
14. Fix: Better reporting of errors when there are invalid modifiers in the search.
15. Fix: Creating a resource that would not fit MongoDB's document size resulted in an inappropriate error.
16. Fix: There was no default sort order in the search, resulting in warnings from the SQL implementation. Added default sort on _lastUpdated (desc).
17. Fix: Preliminary disposal of LocalTerminology server by the Validator.

Facade

1. Fix: _include/_revinclude on searchresults having contained resources triggered a NotImplementedException.

3.1.22 Release 0.5.1.1

Facade

We released the Facade libraries on [NuGet](#) along with *getting started documentation*.

No features have been added to the Vonk FHIR Server.

3.1.23 Release 0.5.0.0

Database

1. Long URI's for token and uri types are now supported, but that required a change of the SQL Server database structure. If you have AutoUpdateDatabase enabled (see *Using SQL server*), Vonk will automatically apply the changes. As always, perform a backup first if you have production data in the database.
2. To prevent duplicate resources in the database we have provided a unique index on the Entry table. This update does include a migration. It can happen that that during updating of your database it cannot apply the unique index, because there are duplicate keys in your database (which is not good). Our advise is to empty your database first (with <vonk-endpoint>/administration/reset, then update Vonk with this new version and then run Vonk with AutoUpdateDatabase=true (for the normal and the administration databases).

If you run on production and encounter this problem, please contact our support.

Features and fixes

1. Feature: POST on _search is now supported
2. Fix: Statuscode of <vonk-endpoint>/administration/preload has changed when zero resources are added. The statuscode is now 200 instead of 201.
3. Fix: OPTIONS operation returns now the capability statement with statuscode 200.
4. Fix: A search operation with a wrong syntax will now respond with statuscode 400 and an OperationOutcome. For example GET <vonk-endpoint>/Patient?birthdate<1974 will respond with statuscode 400.
5. Fix: A statuscode 501 could occur together with an OperationOutcome stating that the operation was successful. Not anymore.
6. Fix: An OperationOutcome stating success did not contain any issue element, which is not valid. Solved.

7. Improvement: In the configuration on *Load Conformance Resources from simplifier.net* the section `ArtifactResolutionOptions` has changed to `ResourceLoaderOptions` and a new option has been introduced under that section named `LoadAtStartup` which, if set to true, will attempt to load the specified resource sets when you start Vonk
8. Improvement: the Memory implementation now also supports `SimulateTransactions`
9. Improvement: the option `SimulateTransactions` in the configuration defaults to false now
10. Feature: You can now add `SearchParameters` at runtime by POSTing them to the Administration API. You need to apply *Re-indexing for new or changed SearchParameters* to evaluate them on existing resources.
11. Fix: The batch operation with search entries now detects the correct interaction.
12. Fix: ETag header is not sent anymore if it is not relevant.
13. Fix: Searching on a String `SearchParameter` in a MongoDB implementation could unexpectedly broaden to other string parameters.
14. Fix: If `Reference.reference` is empty in a `Resource`, it is no longer filled with Vonks base address.
15. Feature: Search operation now supports `_summary`.
16. Fix: Paging is enabled for the history interaction.
17. Fix: Conditional updates won't create duplicate resources anymore when performing this action in parallel.
18. Fix: Indexing of `CodeableConcept` has been enhanced.
19. Fix: Search on reference works now also for an absolute reference.
20. Fix: Long uri's (larger than are 128 characters) are now supported for `Token` and `Uri SearchParameters`.
21. Improvement: The configuration of IP addresses in *Limited access* has changed. The format is no longer a comma-separated string but a proper JSON array of strings.

3.1.24 Release 0.4.0.1

Database

1. Long URL's for absolute references are now supported, but that required a change of the SQL Server database structure. If you have `AutoUpdateDatabase` enabled, Vonk will automatically apply the changes. As always, perform a backup first if you have production data in the database.
2. Datetime elements have a new serialization format in MongoDB. After installing this version, you will see warnings about indexes on these fields. Please perform *Re-indexing for new or changed SearchParameters*, for all parameters with `<vonk-endpoint>/administration/reindex/all`. After the operation is complete, restart Vonk and the indexes will be created without errors.

Features and fixes

1. Fix: `SearchParameters` with a hyphen ('-', e.g. `general-practitioner`) were not recognized in (reverse) chains.
2. Fix: `CapabilityStatement` is more complete, including (rev)includes and support for generic parameters besides the `SearchParameters` (like `_count`). Also the `SearchParameters` now have their canonical url and a description.
3. Improvement: *Preloading resources* gives more informative warning messages.
4. Fix: *Re-indexing for new or changed SearchParameters* did not handle contained resources correctly. If you have used this feature on the 0.3.3 version, please apply it again with `<vonk-endpoint>/administration/reindex/all` to correct any errors.

5. Improvement: *Loading resources from Simplifier* now also works for the Memory implementation.
6. Improvements on *Validation*:
 - profile parameter can also be supplied on the url
 - if validation is successful, an OperationOutcome is still returned
 - it always returns 200, and not 422 if the resource could not be parsed
7. Feature: support for Conditional Read, honouring if-modified-since and if-none-match headers.
8. Fix: Allow for url's longer than 128 characters in Reference components.
9. Fix: Allow for an id in a resource on a Create interaction (and ignore that id).
10. Fix: Allow for an id in a resource on a Conditional Update interaction (and ignore that id).
11. Fix: Include Last-Modified header on Capability interaction.
12. Fix: Format Last-Modified header in `httpdate` format.
13. Fix: Include version in `bundle.entry.fullUrl` on the History interaction.
14. Fix: Update `_sort` syntax from DSTU2 to STU3. Note: `_sort` is still only implemented for `_lastUpdated`, mainly for the History interaction.
15. Improvement: If the request comes from a browser, the response is sent with a Content-Type of `application/xml`, to allow the browser to render it natively. Note that most browsers only render the narrative if they receive xml.

3.1.25 Release 0.3.3.0

Attention: We upgraded to .NET Core 2.0. For this release you have to install .NET Core Runtime 2.0, that you can download from dot.net.

Hosting

The options for enabling and configuring HTTPS have moved. They are now in `appsettings.json`, under 'Hosting':

```
"Hosting": {
  "HttpPort": 4080,
  "HttpsPort": 4081, // Enable this to use https
  "CertificateFile": "<your-certificate-file>.pfx", //Relevant when HttpsPort is
  present
  "CertificatePassword" : "<cert-pass>" // Relevant when HttpsPort is present
},
```

This means you have to adjust your environment variables for `CertificateFile` and `CertificatePassword` (if you had set them) to:

```
VONK_Hosting:CertificateFile
VONK_Hosting:CertificatePassword
```

The setting 'UseHttps' is gone, in favour of `Hosting:HttpsPort`.

Database

There are no changes to the database structure.

Features and fixes

1. Feature: Subscription is more heavily checked on create and update. If all checks pass, status is set to active. If not, the Subscription is not stored, and Vonk returns an `OperationOutcome` with the errors.
 - Criteria must all be supported
 - Endpoint must be absolute and a correct url
 - Enddate is in the future
 - Payload mimetype is supported
2. Feature: use `_elements` on Search
3. Feature: *load profiles from your Simplifier project* at startup.
4. Feature: Content-Length header is populated.
5. Fix: PUT or POST on `/metadata` returned 200 OK, but now returns 405 Method not allowed.
6. Fix: Sometimes an error message would appear twice in an `OperationOutcome`.
7. Fix: `_summary` is not yet implemented, but was not reported as ‘not supported’ in the `OperationOutcome`. Now it is. (Soon we will actually implement `_summary`.)
8. Fix: If-None-Exist header was also processed on an update, where it is only defined for a create.
9. Fix: Set `Bundle.entry.search.mode` to ‘outcome’ for an `OperationOutcome` in the search results.
10. UI: Display software version on homepage.

3.1.26 Release 0.3.2.0

1. Fix: `_include` and `_revinclude` could include too many resources.

3.1.27 Release 0.3.1.0

1. IP address restricted access to Administration API functions.
2. Fix on Subscriptions:
 1. Accept only Subscriptions with a channel of type `rest-hook` and the payload (if present) has a valid mime-type.
 2. Set them from requested to active if they are accepted.

3.1.28 Release 0.3.0.0

1. Database changes

If you have professional support, please consult us on the best way to upgrade your database.

1. The schema for the SQL Database has changed. It also requires re-indexing all resources.
2. The (implicit) schema for the documents in the MongoDB database has changed.
3. The Administration API requires a separate database (SQL) or collection (MongoDb).

2. New features:

1. *Custom Search Parameters*

2. Support for Subscriptions with rest-hook channel
3. Preload resources from a zip.
4. Reset database
5. Conditional create / update / delete
6. Support for the prefer header
7. Validation on update / create (can be turned on/off)
8. Restrict creates/updates to specific profiles.
9. Configure supported interactions (turn certain interactions on/off)

3. New search features:

1. `_has`
2. `_type` (search on system level)
3. `_list`
4. `_reinclude`

4. Enhancements

1. `:exact`: Correctly search case (in)sensitive when the `:exact` modifier is (not) used on string parameters.
2. Enhanced reporting of errors and warnings in the `OperationOutcome`.
3. Custom profiles / `StructureDefinitions` separated in the Administration API (instead of in the regular database).
4. Full FHIRPath support for Search Parameters.
5. Fixed date searches on dates without seconds and timezone
6. Fixed evaluation of modifier `:missing`
7. Correct total number of results in search result bundle.
8. Fix paging links in search result bundle
9. Better support for mimetypes.

5. DevOps:

1. New *Firely Server Administration API*
2. Enabled logging of the SQL statements issued by Vonk (see *Log settings*)
3. Migrations for SQL Server (auto create database schema, also for the Administration API)
6. Performance

1. Added indexes to MongoDB and SQL Server implementations.

3.2 Security notifications for Firely Server

3.2.1 January 2021

Microsoft has a new Security Advisory regarding ASP.NET Core:

- Microsoft Security Advisory CVE-2020-1161 | ASP.NET Core Denial of Service Vulnerability in all ASP.NET Core applications on runtime 3.1.3 or lower (#416). If you are not already up-to-date, you should install the latest runtime version from <https://dotnet.microsoft.com/download/dotnet-core/3.1>

3.2.2 July 2020

Microsoft has published several newer Security Advisories regarding ASP.NET Core:

- Microsoft.ApplicationInsights.AspNetcore 2.12 was vulnerable to CVE-2005-2224. We upgraded it to 2.14.
- Microsoft Security Advisory CVE-2020-0602 : ASP.NET Core Denial of Service Vulnerability, #402 and
- Microsoft Security Advisory CVE-2020-0603 : ASP.NET Core Remote Code Execution Vulnerability, #403. These affect applications running SignalR. Vonk does not use SignalR. Nevertheless we recommend to follow Microsoft's advice: For machines running .NET Core 3.1, you should download and install Runtime 3.1.1 or SDK 3.1.101 from <https://dotnet.microsoft.com/download/dotnet-core/3.1>
- Microsoft Security Advisory | MessagePack Denial of Service, #405. This only affect applications using MessagePack, which Vonk does not use.

3.2.3 September 2019

Updates regarding previous Security Advisories:

- Please upgrade the ASP.NET Core runtime to at least version 2.2.7, from the [runtimes download page](#). This solves:
 - #334
 - #373
 - #384
 - #385
- #295: Vonk has been upgraded to ASP.NET Core 2.2, and is therefore no longer vulnerable to this issue. It is nevertheless advised to run a publicly exposed Vonk *behind a proxy* or on an Azure Web App.
- #335: no longer relevant to Vonk since it runs on ASP.NET Core 2.2

Microsoft has published several newer Security Advisories regarding ASP.NET Core:

- #325: This is not applicable yet to Vonk, since it affects AspNetCoreModuleV2 and Vonk still works on AspNetCoreModule (implicitly V1). We will upgrade to V2 shortly though, so we advise you to install the latest AspNetCoreModulev2 anyway.
- #359: Not relevant to Vonk, it does not use SignalR.

3.2.4 January 2019

Microsoft has published two Security Advisories regarding ASP.NET Core:

- If you run Vonk behind Internet Information Server (IIS), you may be vulnerable to “Microsoft Security Advisory CVE-2019-0548: ASP.NET Core Denial Of Service Vulnerability”. Refer to the related [Github issue #335](#) for details and the fix.
- When using older versions of some of the ASP.NET Core packages you may be vulnerable to “Microsoft Security Advisory CVE-2019-0564: ASP.NET Core Denial of Service Vulnerability”. Refer to the related [Github issue #334](#) for details. Vonk FHIR Server up until version 1.1.0 uses versions of the packages involved that are not affected (older than the vulnerable versions). In a future version we will upgrade beyond the vulnerable version upto secure versions. No action is required by the administrator of Vonk.

3.2.5 April 2018

Microsoft has published two Security Advisories regarding ASP.NET Core:

- If you run Vonk behind Internet Information Server (IIS), you may be affected by “Microsoft Security Advisory CVE-2018-0808: ASP.NET Core Denial Of Service Vulnerability”. Refer to the related [GitHub issue #294](#) for details and the fix.
- If you expose Vonk directly to the internet, or host it behind a proxy which does not validate or restrict host headers to known good values, you may be affected by “Microsoft Security Advisory CVE-2018-0787: ASP.NET Core Elevation Of Privilege Vulnerability”. Refer to the related [GitHub issue #295](#) for details and the correct way of hosting Vonk. This ‘host validating middleware’ mentioned by this issue is not a part of Vonk. We advise you to run a publicly exposed Vonk *behind a proxy* or on an Azure Web App.

3.3 Upgrading Firely Server

See *How to upgrade Firely Server?* for information on how to upgrade to a new version of Firely Server.

3.4 Public Endpoint Announcement 8 July 2021

The default FHIR version of the [public Firely Server endpoint](#) is now R4.

3.5 Release 4.4.0

3.5.1 Database

1. MongoDB

1. To improve the performance of deletes, the definition of the index `ix_container_id` is redefined. Firely Server 4.4.0 will automatically change the definition.

2. SQL Server

1. Improved query behind `_include` to leverage an index. No changes to the database schema involved. This only affects the new implementation (available since 4.3.0).

3.5.2 Fix

1. Improved automatic upgrading of terminology settings from pre-4.1.0 instances.
2. Added `CapabilityStatement.status` for R4
3. The default `SmartAuthorizationOptions` in `appsettings.default.json` only have the Filter for 'Patient' enabled. The rest is now commented out as those are generally not used.

3.6 Release 4.3.0

3.6.1 Database

1. SQL Server
 1. To improve the performance of searching we have rewritten a large part of our SQL Server implementation. To be able to use the new implementation go to section `PipelineOptions` in `appsettings.default.json` (or `appsettings.instance.json` if you have overridden the default pipeline options) and add `"Vonk.Repository.Sql.Raw.KSearchConfiguration"`. See [Using SQL server](#) for more details.
 2. We have identified two indexes that needed a fix to increase query performance for certain searches. The upgrade procedure will try to fix these indexes automatically. If your database is large, this may take too long and the upgrade process will time out. If that happens you need to run the upgrade script manually. The script can be found in `sqlserver/FS_SchemaUpgrade_Data_v19_v20.sql`. If you use SQL Server as your Administration database, Firely Server will try to update it automatically as well. If you prefer a manual update, you can run the following script: `sqlserver/FS_SchemaUpgrade_Admin_v18_v19.sql`.

3.6.2 Feature

1. Firely Server now allows you to execute a `ValueSet` expansion of large `ValueSets` (> 500 included concepts). Previously, Firely Server would log an error outlining that the expansion was not possible. The `appsettings` now contain a setting in the Terminology section allowing to select the `MaxExpansionSize`. See [Options](#) for more details.

3.6.3 Fix

1. Fixed a `NullPointerException` which occurred when indexing UCUM quantities that contained more than one annotation (e.g. `"{reads}/{base}"`).
2. Fixed a bug where it was possible to accidentally delete a resource with a different information model then the request. Firely Server will now check the information model of the request against the information model of the resource for conditional delete and delete requests.
3. `$subsumes` returned HTTP 501 - Not implemented for a POST request (instance-level) even if the operation was enabled in the `appsettings`.
4. The `_type` filter on `$everything` and Bulk data export didn't allow for resources that are not within the Patient compartment. The operations would return an empty result set.
5. Added a clarification to the documentation that `$everything` and Bulk data export do not export Device resources by default. Even though the resource contains a reference to Patient, the corresponding compartment definition for Patient does not include Device as a linked resource. It is possible to export Device resources by adding the resource type to "AdditionalResources" settings of the operations.

3.7 Release 4.2.1 hotfix

3.7.1 Database

Note: We found an issue in version 4.2.0, which affects the query performance for Firely Server running on a SQL Server database. If your are running FS v4.2.0 on SQL Server you should upgrade to v4.2.1 or if that is not possible, [Contact us](#).

Attention:

The upgrade procedure will execute a SQL script try to validate the foreign key constraints. If your database is large, this may take too long and the upgrade process will time out. If that happens you need to run the upgrade script manually, The script can be found in `data/20210720085032_EnableCheckConstraintForForeignKey.sql`.

Here are some guidelines:

- We tested it on a database with about 15k Patient records, and 14 million resources in total. The upgrade script took about 20 seconds to complete on a fairly powerful laptop.
- As always, make sure you have a backup of your database that has been tried and tested before you begin the upgrade.
- If you expect the upgrade to time out, you can choose to run the SQL script manually beforehand. Please make sure that Firely Server is shutdown before you execute the script.

3.7.2 Fix

1. Fixed a bug where some of the Foreign Keys in SQL Server had become untrusted. This bug has an impact on the query performance since the the SQL Server query optimizer will not consider FKs when they are not trusted. This has been fixed, all Foreign Keys have been validated and are trusted again.

3.8 Release 4.2.0

3.8.1 Database

Attention: For SQL Server users: this version of Firely Server running on SQL Server has a bug where some of the Foreign Keys became untrusted. This has an impact on the query performance. Please upgrade to version 4.2.1 or if that is not possible, [Contact us](#). Please note that users running Firely Server running either MongoDB, CosmoDb, or SQLite are not affected by this issue.

Attention: For SQL Server we changed the datatype of the primary keys. The related upgradescrypt (`data/20210519072216_ChangePrimaryKeyTypeFromIntToBigint.sql`) can take a lot of time if you have many resources loaded in your database. Therefore some guidelines:

- We tested it on a database with about 15k Patient records, and 14 mln resources in total. Migrating that took about 50 minutes on a fairly powerful laptop.
- Absolutely make sure you create a backup of your database first.
- If you haven't done so already, first upgrade to version 4.1.x.
- If you already expect the migration might time out, you can run it manually upfront. Shut down Firely Server, so no other users are using the database, and then run the script from SQL Server Management Studio (or a similar tool).
- Running the second script (`20210520102224_ChangePrimaryKeyTypeFromIntToBigintBDE.sql`) is optional - that should also succeed when applied by the automigration.

3.8.2 Feature

1. Terminology operation `$lookup` is now also connected to remote terminology services, if enabled. See [Terminology services](#).
2. We provided a script to 'purge' data from a SQL Server database. See `data/20210512_Purge.sql`. You can filter on the resourcetype only. Use with care and after a backup. If you need more elaborate support for hard deletes, please [Contact us](#).

3.8.3 Fix

1. Firely Server could run out of primary keys on the index tables in SQL Server. Fixed by upgrading to bigint, see warning above.
2. Nicer handling of SQL Server migration scripts that time out on startup. It will now kindly ask you to run the related script manually if needed (usually depends on the size of your database).
3. The Patient-everything (`$everything`) operation was not mentioned in the CapabilityStatement.
4. License expired one day too early.
5. Dependencies have been upgraded to the latest versions compatible with .NET Core 3.1.
6. PATCH did not allow adding to a repeating element.
7. If your license does not allow usage of SMART on FHIR, authorization was disabled, emitting a warning in the log. Possibly causing unauthorized access without the administrator noticing it. This specific case will now block the startup of Firely Server.

3.9 Release 4.1.3 hotfix

3.9.1 Fix

1. Fixed a bug where a number of concurrent `$transform` requests on a freshly started Firely Server could lead to Internal Server Error responses.
2. Upgraded the Mapping plugin to `fhir_mapper_docs:mapping_releasenotes_071`.

3.10 Release 4.1.2 hotfix

3.10.1 Fix

1. Fixed a bug when trying to delete multiple resources at once (bulk delete, see [Configuration](#) for configuration options). The operation would take a while and eventually return a 204 No Content without actually deleting any resources. This is fixed, the bulk delete operation now deletes the resources.

3.11 Release 4.1.1 hotfix

3.11.1 Feature

1. SMART configuration: Some identity providers use multiple endpoints with different base addresses for its authorization operations. Added an extra configuration option `AdditionalEndpointBaseAddresses` to define additional base endpoints addresses next to the main authority endpoint to accommodate this. See [Configuration](#) for further details.

3.11.2 Fix

1. Fixed an error in SQL script `data/20210226200007_UpdateIndexesTokenAndDatetime_Up.sql` that is used when manually updating the database to v4.1.0. We also made the script more robust by checking if the current version the database is suitable for the manual upgrade.

3.12 Release 4.1.0

Attention: We have found an issue with SMART on FHIR and searching with `_(rev)include`. And fixed it right away, see Fix nr 1 below. Your Firely Server might be affected if:

- you enabled SMART on FHIR
- and used `patient/read.*` scopes together with a patient compartment

What happens? Patient A searches Firely Server with a patient launch scope that limits him to his own compartment. If any of the resources in his compartment links to *another* patient (let's say for Observation X, the performer is Patient B), Patient A could get to Patient B with `<base>/Observation?_include=Observation.performer`. If you host Group or List resources on your server, a `_revinclude` on those might give access to other Patient resources within the same Group or List.

If you think you might be affected you can:

- upgrade to version 4.1.0
- or if that is not possible, [Contact us](#).

3.12.1 Database

1. SQL Server

1. A new index table was added. The upgrade procedure will try to fill this table based on existing data. If your database is large, this may take too long and time out. Then you need to run the upgrade script found in `data/20210303100326_AddCompartmentComponentTable.sql` manually.
2. A new SQL Server index was added to improve query times when searching with date parameters. The upgrade procedure will try to build this index. If your database is large, this may take too long and time out. Then you need to run the upgrade script found in `data/20210226200007_UpdateIndexesTokenAndDatetime_Up.sql` manually.
3. In both cases you may also run the script manually beforehand.
4. As always: make sure you have a backup of your database that is tested for restore as well.

3.12.2 DevOps

Attention: Because of a change in the devops pipeline there is no longer a `Firely.Server.exe` (formerly `Vonk.Server.exe`) in the distribution zip file. You can run the server as always with `dotnet ./Firely.Server.dll`

3.12.3 Features

1. Inferno, The ONC test tool: Firely Server now passes all the tests in this suite! With version 4.1.0 we specifically added features to pass the ‘Multi-patient API’ tests. Do you want a demo of this? [Contact us!](#).
2. Terminology support has been revamped. Previously you needed to choose between using the terminology services internal to Firely Server *or* external terminology services like from OntoServer or Loinc. With this version you can use both, and based on the codesystem or valueset involved the preferred terminology service is selected and queried.
 1. This works for terminology operations like `$validate-code` and `$lookup`
 2. It also works for validation, both explicitly with `$validate` and implicitly, when validating resources sent to Firely Server.
 3. The CodeSystem, ValueSet and ConceptMap resources involved are conformance resources and therefore always retrieved from the Administration database.
 4. Responses may differ on details from previous versions of Firely Server, but still conform to the specification.
 5. See [Terminology services](#) for further details.
3. `$everything`: We now support the [Patient \\$everything](#) operation for single Patients. (For multiple patients, there is the Bulk Data Export feature.)
4. Performance of `$everything`, Bulk Data Export and authorization on compartments improved. We added a special index to the database that keeps track which resource belongs to which compartment. First in SQL Server, MongoDB has less need for it.
5. SMART on FHIR: Support for token revocation. Reference tokens can be revoked, and Firely Server can check for the revocation.

3.12.4 Fixes

1. SMART on FHIR: We have found ourselves that the authorization restrictions were bypassed when using `_include` or `_revinclude` in a FHIR Search. We solved this security issue immediately.
2. Firely Server transparently translates absolute urls to relative urls (for internal storage) and back. There was a performance gain to be made in this part, which we did. This is mostly notable on large transaction or batch bundles.
3. Batch bundles are not allowed to have links between the resources in the entries. Firely Server will now reject batch bundles that have these links. If you need links, use a transaction bundle instead.

3.12.5 Plugin and Facade

1. We upgraded the Firely .NET SDK to version [3.0.0](#). This SDK version is almost fully compatible with 2.9, but it brings significant simplifications to its use because the Parameters and OperationOutcome resource POCOs are no longer FHIR-version specific.

Note: Every new version of the SDK brings new versions of the `specification.zip` files. So upon upgrade these new files will be read into the Administration database. See [Conformance Resources](#) for more background.

3.13 Release 4.0.0

This major version introduces a new name: **Firely Server** instead of Vonk. Other than that, this release contains some significant code changes, which could impact you if you run Firely Server with your own plugins.

3.13.1 Features

1. Name change Vonk -> Firely Server:
 1. The main entry point dll (formerly: `Vonk.Server.dll`) and executable (formerly: `Vonk.Server.exe`) names have been changed to `Firely.Server.dll` and `Firely.Server.exe` respectively.
 2. The name was changed in the `CapabilityStatement.name`.
 3. The name of the download zip (from Simplifier) has changed from `vonk_distribution.zip` to `firely-server-latest.zip`. Likewise the versioned zip files have changed as well.
2. We have implemented FHIR Bulk Data Access (`$export`) to allow for fast, asynchronous ndjson data exports. The [Bulk Data Export documentation](#) can help you to get started.
3. Firely Server now uses Firely .NET SDK 2.0.2 (formerly: FHIR .NET API)

Attention: If you are running Firely Server with your own self-made plugins, you will likely encounter package versioning problems and need to upgrade your NuGet Firely Server package references (package names starting with `Vonk.`) to version 4.0.0. You also need to upgrade any Firely .NET SDK package references (package names starting with `Hl7.Fhir.`) to version 2.0.2. The [Firely .NET SDK release notes](#) and [Breaking changes in Firely SDK 2.0](#) can give you an idea of the changes you may encounter in the SDK.

4. SMART on FHIR can now recognize prefixes to the claims, see its [Configuration](#).

5. The smart-configuration endpoint (`<url>/.well-known/smart-configuration`) relays the signature algorithms configured in the authorization server.

3.13.2 Fixes

1. Application Insights has now been disabled by default. If you need Application Insights, you can enable it in your log settings file by including the entire section mentioned in [Application Insights log settings](#).
2. When validating a resource, a non-existing code would lead to an `OperationOutcome.issue` with the code `code-invalid`. That issue code has been changed to `not-supported`.
3. On a batch or transaction bundle errors were not reported clearly if the entry in error had no `fullUrl` element. We fixed this by referring to the index of the entry in the entry array, and the `resourcetype` of the resource in the entry (if any).
4. The `import[R4]` folder allows for importing custom `StructureDefinition` resources. If any of them had no id, the error on that caused an exception. Fixed that.
5. If a Facade returned a resource without an id from the Create method, an error was caused by a log statement. Fixed that.
6. Indexing `Subscription.channel[0].endpoint[0]` failed for R4. Fixed that. This means you can't search for existing Subscriptions by `Subscription.url` on the `/administration` endpoint for FHIR R4.
7. Postman was updated w.r.t. acquiring tokens. We adjusted the [documentation on that](#) accordingly.
8. If a patient claim was included in a SMART on FHIR access token, the request would be scoped to the Patient compartment regardless of the scope claims. We fixed this by allowing "user" scopes to access FHIR resources outside of the Patient compartment regardless of the patient claim. See [Launch context arrives with your access_token](#) for more background information.

3.13.3 Plugin and Facade

1. The mapping plugin is upgraded to the Mapping Engine 0.6.0, see its [releasenotes](#).
2. As announced in [Release 3.0.0](#) we removed support for creating a Facade as a standalone ASP.Net Core project. You can now only build a Facade as a plugin to Firely Server. See [Firely Server Facade](#) on how to do that.
3. The order of some plugins has changed. This way it is possible to add a plugin between `PreValidation` and `UrlMapping`:
 - [UrlMapping](#): from 1230 to 1235
 - [Prevalidation](#): from 4320 to 1228
4. A Facade based on `Vonk.Facade.Relational` no longer defaults to STU3

Attention: If you developed a facade plugin based on `Vonk.Facade.Relational`, you need to override `RelationalQueryFactory.EntryInformationModel(string informationModel)` in your implementation to allow the FHIR version you wish to target (see [Deciding on a FHIR version](#))

5. We took the opportunity of a major version upgrade to clean up a list of items that had been declared `Obsolete` already. Others have become obsolete now. This is the full list:

`Obsolete`, now deleted:

```
# Vonk.Core.Common.DeletedResource # Vonk.Core.Common.IResource.Currency, Change and
Clone(), also in VonkResource. # Vonk.Core.Common.IResourceExtensions.ToIResource(this
ISourceNode original, ResourceChange change, ResourceCurrency cur-
rency = ResourceCurrency.Current) (the overload defaulting to STU3) #
Vonk.Core.Context.Guards.SupportedInteractionOptions.SupportsCustomOperationOnLevel()
# Vonk.Core.Context.Internal.BatchOptions # Vonk.Core.Operations.Validation.ValidationOptions #
Vonk.Core.Pluggability.InteractionHandlerAttribute.Tag # Vonk.Core.Pluggability.ModelOptions #
Vonk.Core.Repository.SearchOptions.LatestOne # Vonk.Core.Support.LogHelpers.TryGetTelemetryClient,
both overloads. # Vonk.Core.Support.SpecificationZipLocator.ctor(IHostingEnvironment...) #
Vonk.Fhir.R3.IResourceVisitor + extensions # Vonk.Fhir.R3.Configuration.ModelContributorsFacadeConfiguration
# Vonk.Fhir.R3.FhirExtensions.AsIResource() # Vonk.Fhir.R3.FhirPropertyIndex + FhirProp-
ertyInfo + FhirPropertyIndexBuilder # Vonk.Fhir.R3.IConformanceBuilder + BaseC-
onformanceBuilder + HarvestingConformanceBuilder + extensions + IConformance-
Contributor # Vonk.Fhir.R3.CompartmentDefinitionLoader + (I)SearchParameterLoader
# Vonk.Fhir.R3.MetadataImportOptions + MetadataImportSet + ImportSource #
Vonk.Fhir.R3.PocoResource + PocoResourceVisitor # Vonk.Core.InformationModelAttribute
(actually made internal)
```

Obsolete since this version:

```
# Vonk.Core.Configuration.CoreConfiguration: allows for integrating Vonk components
in your own ASP.NET Web server, discouraged per 3.0 (see these releasenotes). #
Vonk.Fhir.R3.FhirR3FacadeConfiguration: see above.
```

3.14 Release 3.9.3 hotfix

Attention: We changed the behaviour of resthook notifications on Subscriptions. See Fix nr 1 below.

3.14.1 Database

1. SQL Server: The migration that adds the indexes described in [Release 3.9.2 hotfix](#) might run longer than the timeout period of 30 seconds. Therefore we added scripts to apply and revert this migration manually. If you encounter the timeout during upgrade: shut down vonk, run the script using SQL Server Management Studio or any similar tool, then start Vonk 3.9.3 again. In both scripts you only need to provide the databasename for the database that you want to upgrade. If you run your administration database on SQL Server you can but probably do not need to run this script on it. The administration database is typically small enough to complete the script within 30 seconds.

1. apply: <vonk-dir>/data/2021211113200_AddIndex_ForCountAndUpdateCurrent_Up.sql
2. revert: <vonk-dir>/data/2021211113200_AddIndex_ForCountAndUpdateCurrent_Down.sql

3.14.2 Fix

1. *Subscriptions*: A resthook notification was sent as a FHIR create operation, using POST. This was not compliant with the specification that states it must be an update, using PUT. We changed the default behaviour to align with the specification. In order to avoid breaking changes in an existing deployments, you may set the new setting `SubscriptionEvaluatorOptions:SendRestHookAsCreate` to `true` - that way Vonk will retain the (incorrect) behaviour from the previous versions.

3.15 Release 3.9.2 hotfix

3.15.1 Fix

All fixes are relevant to SQL Server only.

1. The 3.9.0 fix that “Improved the handling of concurrent updates on the same resource.” decreased the performance of concurrent transaction handling. We implemented another solution that does not affect performance.
2. Improved read performance by adding an index.

3.16 Release 3.9.1 hotfix

3.16.1 Fix

1. Fixed a bug introduced with 3.9.0 where Vonk would throw the following exception on start-up `System.InvalidOperationException: Unable to resolve service for type 'Vonk.Core.Conformance.IDefinitionProvider' while attempting to activate 'Vonk.Fhir.R3.SnapshotGeneration.SnapshotGeneratorR3`
2. Fixed a breaking change to public search API with the implementation of `_total` parameter. We had introduced a new parameter to the `Next` method in `ResultPage`, effectively breaking backwards compatibility. This has been fixed.

3.17 Release 3.9.0

3.17.1 Features

1. We have made Subscriptions more robust. See *Subscriptions* for details. In summary, if an evaluation of a Subscription fails, Vonk will retry the evaluation periodically for a number amount of tries. You can control the retry period and the maximum number of retries in the subscription settings:
 - `RetryPeriod` is expressed in milliseconds. Default `300000` (30 sec).
 - `MaximumRetries` is the maximum amount of times Vonk will retry to send the resources. Default 3 retries.
2. We have implemented the `_total` parameter for options `none` and `accurate`. Omitting the `_total` parameter is equivalent to `_total=accurate`. If the total number of resources is not relevant, using `_total=none` in the request results in better performance when searching.
3. It is no longer necessary for the `:type` parameter to always be provided to distinguish between multiple reference targets. The parameter does not need to be provided anymore when the search only applies to a single target. For example: `GET <base>/AllergyIntolerance?patient=xyz` The `patient:Patient` type parameter does not have to be supplied. The ‘patient’ search parameter on `AllergyIntolerance` has two possible targets.

It may reference either a Patient or a Group resource. However, the fhirpath statement that goes with it, selects 'AllergyIntolerance.patient', and that reference element may only target a Patient resource.

3.17.2 Fixes

1. Indexing values for a string search parameter threw an exception when there was no value but only an extension. This has been corrected.
2. We made the Provenance.target available as a revInclude Parameter in the CapabilityStatement. Previously, Vonk did not account for the case that a reference is allowed to ANY resource type, which incorrectly resulted in Provenance.target to not be shown in the CapabilityStatement.

All the following fixes are only relevant for SQL Server:

1. Improved the handling of concurrent updates on the same resource.
2. Upgraded the version of the SqlClient library to fix issues when running on Linux.
3. Fixed missing language libraries for SQL Server when running on Docker.

3.18 Release 3.8.0

3.18.1 Database

- We added an important note to the [3.6.0 release notes](#) for MongoDB users.
- Because of the changes in searching for Quantities (feature 2 below), you will need to do a [reindex](#) in order to make use of this. You may limit the reindex to only the searchparameters of type 'quantity' that you actually use (e.g. Observation.value-quantity).

3.18.2 Features

1. We upgraded the FHIR .NET API to 1.9, see the [1.9 releasenotes](#). This will trigger an automatic *import of the Conformance Resources* at startup.
2. We upgraded the [Fhir.Metrics library](#) to 1.2. This allows for a more uniform search on Quantities (mainly under the hood)
3. We upgraded the FHIR Mapping plugin to support the FHIR Mapper version 0.5. See its FHIR Mapper releasenotes.
4. The *built-in terminology services* now support the includeDesignations parameter.
5. The *IVonkContext* now lets you access the original HttpContext.
6. The CapabilityStatement now lists the profiles that are known to Vonk (in its Administration database) under CapabilityStatement.rest.resource.supportedProfile (>= R4 only) and the base profile for a resource under CapabilityStatement.rest.resource.profile.
7. We extended the security extension on the CapabilityStatement to include the endpoints for register, manage, introspect and revoke.
8. IAdministrationSearchRepository and IAdministrationChangeRepository interfaces are no publicly available. Use with care.

3.18.3 Fixes

1. If the server configured as authorization endpoint in the Smart setting is not reachable, Vonk will log a proper error about that.
2. An error message for when a query argument has no value is improved.
3. When *SMART-on-FHIR* is enabled, and the received token contains a launch context, the *_history* operation is no longer available. Because Vonk does not retain the search parameter index for historical resources, it cannot guarantee that these resources fall within the launch context (at least not in a performant way). To avoid information leakage we decided to disable this case altogether.
4. A Create interaction without an id in the resource, with *SMART-on-FHIR* enabled, resulted in an exception.
5. You can now escape the questionmark '?' in a query argument by prepending it with a backslash ' '.
6. A Quantity search using 'lt' on MongoDB resulted in too many results.

3.19 Release 3.7.0

3.19.1 Database

Attention: To accomodate for feature #2 below there is an automatic migration carried out for SQL Server and SQLite. This migration might take some time, so please test it first. For MongoDB, you will have to *Rebuild the whole search index*. If this is not feasible for your database, please *Contact us* for assistance.

3.19.2 Features

1. Patch: We implemented FHIR Patch. You can now update a resource having only partial data for it. See *Create, read, update, patch, delete*.
2. Search on accents and combined characters: we improved searching with and on accents and combined characters. Note the database change above.
3. API 1.7: We upgraded Vonk to use the FHIR .NET API 1.7, having its own *releasenotes*.
4. Security: The Docker image is now based on the Alpine image for .NET Core. This has far less security issues than the Ubuntu image that we used before. The base image is aspnet:3.1-alpine:3.11 (newest version 3.12 has an open bug related to SQLite).
5. Security: We revisited the list of security vulnerabilities, see *Security notifications for Firely Server*.
6. Administration: ConceptMaps are now *imported* at startup.

3.19.3 Fixes

1. Searching on *_lastUpdated* could be inaccurate when time zone differences are in play. We fixed that.
2. Search arguments for a quantity search weren't allowed to be greater than 999.

3.20 Release 3.6.1

3.20.1 Features

1. ConceptMap resources can be stored at the Administration endpoint, both through *import* and through the *RESTful API*.

Attention: Previous versions of Vonk did not include the ConceptMap resources in the import so they will currently not be in your Administration database. If you run your Administration database on SQL Server or MongoDB and want to use the ConceptMap resources from the spec, be sure to rerun the import of the specification resources. You can force Vonk to do so by deleting the `.vonk-import-history.json` file from the ImportedDirectory (see the settings under `AdministrationImportOptions`). If you use SQLite, you can simply use the pre-built `./data/admin.db` from the binaries.

3.20.2 Plugins

1. The FHIR Mapper plugin is upgraded to version 0.3.6.
2. The FHIR Mapper plugin now fully works on the *Administration* endpoint.

3.21 Release 3.6.0

3.21.1 Database

Attention: For MongoDB users: We implemented feature #1 below using the Aggregation Pipeline. This makes an existing issue in MongoDB - *SERVER-7568* <<https://jira.mongodb.org/browse/SERVER-7568>> - a more urgent problem. MongoDB has solved this problem in version 4.4. Therefore we advise you to upgrade to MongoDB 4.4.

3.21.2 Feature

1. Sort: The *sorting* that was implemented for the SQL/SQLite repositories in the previous version is now also implemented for MongoDB.
2. Terminology: The *local terminology service*, built in to the Vonk Administration API, is upgraded to support R4 and R5 (and still R3 of course).
3. Vonk can now index and search on searchparameters that reference a nested resource, like `Bundle.message`.

Attention: Note that any nested resources have to be indexed by Vonk. For new data that is done automatically. But if you want to use this on existing data, you have to *reindex for the searchparameters* you want to use it on. Those will most notably be `Bundle.message` and `Bundle.composition`.

4. If you accidentally provide a body in a GET or DELETE request, Vonk will now ignore that body instead of returning an error.

3.21.3 Fix

1. CapabilityStatement (rev)includes now use ‘:’ as a separator instead of ‘.’.

3.21.4 Plugins

1. The *BinaryWrapper plugin* is upgraded to 0.3.1, where the included BinaryEncodeService is made more reusable for other plugins (most notably the *FHIR Mapper*).

3.22 Release 3.5.0

3.22.1 Feature

1. Search reference by identifier: FHIR R4 allows you to [search a reference by its identifier](#). We added support for this in Vonk. Note that any identifiers in reference elements have to be indexed by Vonk. For new data that is done automatically. But if you want to use this on existing data, you have to [reindex for the searchparameters](#) you want to use it on. E.g. Observation.patient.
2. AuditEvent logging: In release 3.3.0 we already added support for logging audit information to a file. With this release we add to that logging that same information in AuditEvent resources. These resources are written to the Vonk Data database (not the Administration database). Users are not allowed to update or delete these resources. See [Auditing](#) for more background.
3. Audit logging: We added [Request] or [Response] to the log lines so you can distinguish them better.
4. Sort: We started implementing [sorting](#). This release provides sorting for searchparameters of the types string, number, uri, reference, datetime and token, on the repositories SQL, SQLite and Memory. On the roadmap is extending this support to MongoDB and to quantity searchparameters.
5. *Terminology Integration*: You can configure Vonk to route the terminology operations to external terminology servers. You can even configure a preferred server for certain code systems like LOINC or Snomed-CT. On the roadmap is to also allow you to use these servers for validation of codes and for token searches.
6. We implemented [\\$meta-delete](#), see [Meta plugins](#).
7. Loading plugins can lead to unexpected errors. We made the process and the log friendlier, so you can spot configuration errors more easily:
 - The log outputs the version of each of the plugins
 - If a duplicate .dll file is found, Vonk tells you which two dlls are causing this and then exits.
 - If you configured a plugin that you are not licensed to use, this is logged with a friendly hint to acquire a license that does allow you to use it.
8. The log is now by default configured to use asynchronous logging so Vonk is not limited by the speed of the logging sinks (like the Console and the log file). Please update your logsettings.instance.json if you created your own log settings in that. See [Log settings](#) for more background.

3.22.2 Fix

1. You could load invalid XML in the Resource.text through a JSON payload. When that resource was then retrieved in XML, it would fail with an InternalServerError. Vonk will now return an OperationOutcome telling you what the problem is. You can then correct it by using JSON.
2. Composite searchparameters were not parsed correctly. Now they are. So you don't see warnings like Composite SearchParameter 'CodeSystem.context-type-quantity' doesn't have components. anymore.
3. Indexing for the _profile searchparameter was broken for R4 since Vonk 3.2.1. We fixed it. If you added new resources with Vonk 3.2.1 - 3.4.0, you need to *reindex for the Resource._profile* parameter.
4. Audit log: %temp% in the path setting was evaluated as <current directory>\%temp%. Fixed that to evaluate to the systems temporary directory.
5. The logsettings.json configured the Serilog RollingFile sink by default. That is deprecated, so we replaced it with the File sink.
6. *Rebuild the search index for specific searchparameters* now returns an error if you forget to actually specify a searchparameter.
7. An InternalServerError was returned when you validate a resource that is syntactically incorrect. Like a Patient with multiple id's. Vonk now returns an OperationOutcome with the actual problem.
8. The configuration for the FHIR Mapper was simplified. You only need to include Vonk.Plugin.Mapping. Check appsettings.default.json for the new pipeline.
9. Maybe you got accustomed to ignoring a list of warnings at startup of Vonk. We cleaned up the list so that if there is a warning, it is worthwhile investigating the cause of it.
10. The appsettings and logsettings can contain relative file paths for several settings, like the License:LicenseFile. These were evaluated against the current working directory, but that could lead to problems if that was *not* the Vonk directory. We solved that: all relative paths are evaluated against the Vonk directory.
11. The docker image for version 3.4.0 was tagged 3.4.0-. With 3.5.0 we removed the superfluous hyphen at the end.
12. We updated the documentation on *Using Firely Server on Docker* on SQL Server to be clearer about the order of the steps to take.
13. We updated the documentation on *Firely Server Plugin example - Create a new landing page* to match .NET Core 3.1.

3.22.3 Plugins & Facade

1. *FHIR Mapper*
 - Has been upgraded to version 0.3.4.

3.23 Release 3.4.0

3.23.1 Feature

1. Upgraded to FHIR .NET API 1.6.0, that features a couple of changes for working with CDA logical models. See the [release notes of the API](#).
2. Included the FHIR Mapper in the distribution. It is only enabled however when you include the mapping plugin in your license. See `mappingengine_index` for more information about the FHIR Mapper.

3.23.2 Fix

1. When prevalidation is set to the level 'Core', Vonk no longer complains about extensions that are not known if they are not core extensions (i.e. having a url starting with '<http://hl7.org/fhir/StructureDefinition/>').

3.24 Release 3.3.0

Attention: To use the new features for auditing and R5, you need a new license file including the tokens for those plugins. For evaluation and community editions you can retrieve them from Simplifier.net. If you need these updates in your production license, please contact us.

3.24.1 Feature

1. Vonk was upgraded to FHIR .NET API 1.5.0. See the [release notes of the API](#).
2. Vonk can now log audit lines in a separate file. This can help you achieve HIPAA/GDPR compliancy. See [Auditing](#) for more info.
3. Failed authorization attempts are now logged from the *SMART on FHIR* plugin.
4. Support for `_include:iterate` and `_revinclude:iterate`, see [Search](#).
5. The *BinaryWrapper plugin* is now two-way. So you can POST binary content and have it stored as a Binary resource, and GET a Binary resource and have it returned in its original binary format.
6. Experimental support for R5 is now included in the Vonk distribution. For enabling it, see [Support for R5 \(experimental!\)](#).

3.24.2 Fix

1. Indexing of a quantity in resource could fail with a Statuscode 500 if it had no `.value` but only extensions.
2. The use of a SearchParameter of type `reference` having no `target` failed. These searchparameters are now signalled upon import.
3. Since R4 it is valid to search for a quantity without specifying the unit. Vonk now accepts that.
4. A transaction response bundle could contain an empty `response.etag` element, which is invalid.
5. *Preloading resources* was not working since the upgrade to .NET Core 3.0. That has been fixed. It is still only available for STU3 though.

6. Administration import would state that it moves a file to history when it had imported it. That is no longer true, so we removed this incorrect statement from the log.
7. `$validate-code` could cause a `NullReference` exception in some case.
8. The generated `CapabilityStatement` for R4 failed constraint `cpb-14`.
9. Content negotiation favoured a mediatype with quality < 1 over a mediatype without quality. But the default value is 1, so the latter is now favoured.
10. *Validate an instance from the database* did not account for the `informationmodel` (aka FHIR version) of the resource.

3.24.3 Plugins & Facade

1. *Document Operation*
 - Has been upgraded to Vonk 3.2.0.
 - Was assigned a license token
 - Assigns an id and lastUpdated to the result bundle
2. *Conversion*
 - Has been upgraded to Vonk 3.2.0.
 - Was assigned a license token.
3. `Vonk.Facade.Starter` has been upgraded to Vonk 3.2.1 and as a consequence also to `EntityFrameworkCore` 3.1.0.

3.25 Release 3.2.1

3.25.1 Fix

1. SMART plugin now understands multiple scopes per access token.
2. SMART plugin now understands `Resource.*` claims, in addition to already understanding `Resource.read` and `Resource.write`.

3.26 Release 3.2.0

Attention: Vonk 3.2.0 is upgraded to .NET Core 3.1.0, ASP.NET Core 3.1.0 and EntityFramework Core 3.1.0.

- For running the server: install the ASPNET.Core runtime 3.1.0.
- For developing or upgrading Facades that use `Vonk.Facade.Relational`: upgrade to EF Core 3.1.0.
- Plugins that target `NetStandard 2.0` need not be upgraded.

3.26.1 Database

1. There are no changes to the databases. The upgrade of EntityFramework Core does not affect the structure of the SQL Server or SQLite databases, just the access to it.

3.26.2 Fix

1. *Supported interactions* were not enforced for custom operations like e.g. \$convert.
2. If a resource failed *Validating incoming resources*, the OperationOutcome also contained issues on not supported arguments.
3. A search with ?summary=count failed.
4. Added support for FhirPath hasValue() method.
5. Resolution of canonical `http://hl7.org/fhir/v/0360|2.7` failed.
6. CapabilityStatement.rest.resource.searchInclude used ‘.’ as separator, fixed to use ‘:’ in <resource>:<search parameter code>
7. Changed default value of License:LicenseFile to vonk-license.json, aligned with the default naming if you download a license from Simplifier.
8. *Reindexing* always interpreted a resource as STU3. Now it correctly honours the actual FHIR version of the resource.

3.26.3 Feature

1. *BinaryWrapper plugin* can now be restricted to a list of mediatypes on which to act.
2. Vonk used to sort on _lastUpdated by default, and add this as extra sort argument if it was not in the request yet. Now you can configure the element to sort on by default in BundleOptions.DefaultSort. Although Vonk FHIR Server does not yet support sorting on other elements, this is useful for Facade implementations that may support that (and possibly not support sort on _lastUpdated). See also *Search and History*.
3. Implemented \$versions operation
4. Extended the documentation on:
 - *The order of plugins*
 - *Constructing bundles*
 - several smaller additions
5. The SMART authorization plugin can now be configured to *not* check the audience. Although not recommended, it can be useful in testing scenarios or a highly trusted environment.

Attention: We changed the default value for the setting SmartAuthorizationOptions.Audience from vonk to empty, or ‘not set’. This is to avoid awkward syntax to override it with ‘not-set’. But if you rely on the value vonk, please override this setting in your appsettings[.instance].json or environment variables as described in *Changing the settings*.

3.26.4 Plugin and Facade API

1. Vonk.Facade.Relational now supports the use of the `.Include()` function of EntityFramework Core. To do so, override `RelationalQuery.GetEntitySet(DbContext dbContext)`.
2. Vonk.Facade.Relational now supports sorting. Override `RelationalQueryFactory.AddResultShape(SortShape sortShape)` and return a `RelationalShorShape` using the extension method `SortQuery()`.

3.27 Release 3.1.3 hotfix

3.27.1 Fix

1. Fixed behaviour on conditional updates in transactions. In odd circumstances Vonk could crash on this.

3.28 Release 3.1.0

Please also note the changes in [3.0.0](#) (especially the one regarding the SQL server database)

3.28.1 Fix

1. Validation on multi-level profiled resources no longer fails with the message “*Cannot walk into unknown StructureDefinition with canonical*”
2. Improved documentation on [upgrading Vonk](#), the [Vonk pipeline](#), [CORS support](#), [plugins](#) and [IIS deployment](#)
3. Using multiple parameters in `_sort` led to an error for all repositories
4. Vonk UI capability statement view now works for self-mapped endpoints like `/R3` or `/R4`
5. A saved resource reference (e.g. `Patient.generalPractitioner`) on a self-mapped endpoint (e.g. `/R4/...`) would have its relative path duplicated (`/R4/R4/...`)

Attention: If you have used [self-mapped endpoints](#) (appsettings: `InformationModel.Mapping.Map` in the ‘Path’ mapping mode) and you have saved resources containing references, it is possible that your database now contains some resources with broken references. Please contact us if this is the case

3.28.2 Feature

1. The new experimental FHIR mapping engine, which is currently exclusively available on our public FHIR server <http://vonk.fire.ly>
2. New licensing system, supporting the *community edition*
3. Simplifier projects are now imported for FHIR R4 as well
4. The following plugins have been bundled with the Vonk release (compare your appsettings with the new appsettings.default.json to activate them)
 1. The `$document` operation (see [Documents](#))
 2. The `$convert` operation (see [Conversion](#))

3. The binary wrapper (see [Binary](#))

3.28.3 Plugin and Facade API

1. Vonk.Facade.Starter has been upgraded to work with Vonk 3.1.0
2. IConformanceContributor and IConformanceBuilder have moved from Vonk.Core.Pluggability to Vonk.Fhir.R3.Metadata. It is also deprecated, as Vonk.Core.Metadata.ICapabilityStatementContributor is now preferred instead. See [Capabilities](#) for more information
3. Implementations of ISearchRepository can now sort on multiple parameters (in BaseQuery.Shapes). Previously this would result in an error.
4. Improved documentation on the [Important classes and interfaces](#)
5. See [Release 3.0.0](#) for some additional issues you may encounter upgrading your plugins

3.29 Release 3.0.0

3.29.1 Database

Please also note the changes in [3.0.0-beta1](#)

1. SQL Server: SQL script '20190919000000_Cluster_Indexes_On_EntryId.sql' (found in the /data folder of the Vonk distribution) must be applied to existing Vonk SQL databases (both to the admin and to the data repositories)

Attention: Vonk 3.0.0 (using SQL server) will not start unless this script has been applied to the databases. Please note that running the script can take considerable time, especially for large databases.

3.29.2 Feature

1. Information model (= FHIR version) settings
 1. Although Vonk now supports multiple information models (STU3 and R4) simultaneously, an unused model can be disabled (see [Configuring the Firely Server Pipeline](#))
 2. You can set the default (or fallback) information model (previously: STU3), which is used when Vonk can not determine the information model from context (see [Information model](#))
 3. You can map a path or a subdomain to a specific information model (see [Information model](#)), mitigating the need to specify it explicitly in a request
2. Vonk now uses [FHIR .NET API 1.4.0](#)
3. Several performance enhancements have been made for SQL server and IIS setups
4. Added R4-style [Conditional Update](#) to both STU3 and R4

3.29.3 Fix

1. Circular references within resources are now detected, cancelling validation for now. We will re-enable validation for these resources when the FHIR .NET API has been updated
2. An \$expand using incorrect data returned a 500 (instead of the correct 400)
3. Vonk now returns a 406 (Not Acceptable) when the Accept header contains an unsupported format
4. Deletes did not work for R4
5. Search parameters
 1. Search parameters were read twice (at startup and upon the first request)
 2. Search parameter 'CommunicationRequest.occurrence' is not correctly specified in the specification. We provide a correct version.
6. _history
 1. _history was not usable in a multi information model setup
 2. The resulting Bundle.entry in an STU3 _history response contained the unallowed response field
 3. Added Bundle.entry.response to the R4 _history entry
7. Batches
 1. Valid entries in batches also containing invalid entries were not processed
 2. Duplicate fullUrls are no longer accepted in a batch request, which previously led to a processing error
 3. An R4 transaction resulted in STU3 entries
 4. Transactional errors did not include fullUrl

3.29.4 Plugin and Facade API

1. Improved the message you get when the sorting/shaping operator is not implemented by your facade
2. VonkOutcome (and VonkIssue) has been simplified
3. VonkConstants has moved from Vonk.Core.Context to Vonk.Core.Common
4. IResourceChangeRepository.Delete requires a new second parameter: `string informationModel`. Information model constants can be found in `Vonk.Core.Common.VonkConstants.Model`
5. Exclude Vonk.Fhir.R3 or Vonk.Fhir.R4 from the PipelineOptions if you don't support it in your Facade.
6. Updated the minimal PipelineOptions for a Facade Plugin in the [example appsettings.json](#):
 - updated `Vonk.Core.Operations.SearchConfiguration` to `Vonk.Core.Operations.Search`
 - removed `Vonk.UI.Demo`
 - removed `Vonk.Core.Operations.Validate.SpecificationZipSourceConfiguration` from the `Exclude`
 - updated `Vonk.Core.Operations.Terminology` to `Vonk.Plugins.Terminology`

Note: Early Facade implementations were built with by using Vonk services and middleware in a self-built ASP.NET Core web server. This can be seen in the `Vonk.Facade.Starter` project in the [repository](#) with the same name. Due to changes in Vonk this does not work with Vonk 3.0.0. It will be fixed in 3.1.0. But after that such projects cannot

be upgraded anymore and will have to be refactored to a proper plugin (as the ViSi.Repository project in the same repository). Please [contact](#) us in case of any questions.

3.30 Release 3.0.0-beta2

Attention: We updated the *Security notifications for Firely Server*.

3.30.1 Database

Note the changes in *3.0.0-beta1*, but there are no new changes in beta2.

3.30.2 Feature

1. *Subscriptions* works for R4 also. Note that a Subscription will only be activated for resource changes in the same FHIR version as the Subscription itself.
2. *Load Conformance Resources from disk* works for R4 also. Use a directoryname that ends with .R4 for R4 conformance resources.
3. *Re-indexing for new or changed SearchParameters* works for R4 also. Issue a reindex with a fhirVersion parameter in the Accept header, and it will be executed for the SearchParameters defined for that FHIR version.
4. Allow for non-hl7 prefixed canonical urls for conformance resources (since sdf-7 is lifted). See *Custom Resources*.
5. Custom Resources can be validated, both individually and as part of a bundle. See *Custom Resources*.
6. If the Accept header lacks a *fhirVersion parameter*, it will fall back to the fhirVersion parameter of the Content-Type header and vice versa. If both are missing, Vonk will default to STU3.

3.30.3 Fix

1. `_include` did not work for R4.
2. `_include` gave a 500 responsecode if a resource contains absolute references.
3. A resource with unknown elements could result in an uncaught `Hl7.Fhir.ElementModel.StructuralTypeException`.
4. The homepage stated that Vonk was only for STU3. Fixed that.
5. Bundle.timestamp element (new in R4) was not populated in bundles returned from Search and History operations.
6. Some operations could return an `OperationOutcome` with an issue *and* a success message.
7. Better error message if a resource without any meta.profile is not accepted by *Validating incoming resources*.
8. Requesting an invalid FHIR version resulted in a `ArgumentNullException`.

3.30.4 Plugin and Facade API

1. NuGet package `Vonk.Fhir.R4` had a dependency on `Vonk.Administration.API`, but the latter is not published. We removed the dependency.
2. `IResourceExtensions.UpdateMetadata` did not update the id of the resource.
3. `VonkOutcome.RemoveIssue()` method has been removed.

3.30.5 Examples

1. Plugin example (`Vonk.Plugin.ExampleOperation`):
 1. Added an example of middleware directly interacting with the `HttpContext` (instead of just the `VonkContext`), see the file `VonkPluginMiddleware.cs`
 2. `CapabilityStatementBuilder` was not called.
2. DocumentOperation (`Vonk.Plugin.DocumentOperation`):
 1. Composition ID was not determined correctly when using POST.

3.31 Release 3.0.0-beta1

Vonk 3.0.0 is a major upgrade that incorporates handling FHIR R4. This runs in the same server core as FHIR STU3. See [Multiple versions of FHIR](#) for background info.

Attention: If you have overridden the `PipelineOptions` in your own settings, you should review the new additions to it in the `appsettings.default.json`. In particular we added `Vonk.Fhir.R4` that is needed to support FHIR R4.

Attention: MacOS: you may need to clean your temp folder from previous `specification.zip` expansions. Find the location of the temp folder by running `echo $TMPDIR`.

3.31.1 Database

1. SQL Server, SQLite:
 1. `vonk.entry` got a new column 'InformationModel', set to 'Fhir3.0' for existing resources.
 2. `vonk.ref` got a new column 'Version'.
 3. Database indexes have been updated accordingly.

Vonk will automatically update both the Administration and the Data databases when you run Vonk 3.0.0.
2. MongoDB / CosmosDb:
 1. The documents in the `vonkentries` collection got a new element `im` (for `InformationModel`), set to 'Fhir3.0' for existing resources.
 2. The documents in the `vonkentries` collection got a new element `ref.ver` (for `Version`).
 3. Database indexes have been updated accordingly.

3. MongoDB / CosmosDb: Got a light mechanism of applying changes to the document structure. A single document is added to the collection for that, containing `VonkVersion` and `LatestMigration`.
4. MongoDB: The default name for the main database was changed from 'vonkstu3' to 'vonkdata'. If you want to continue using an existing 'vonkstu3' database, override `MongoDbOption:DatabaseName`, see [Hierarchy of settings](#).

3.31.2 Feature

1. Support for FHIR R4 next to FHIR STU3. Vonk will choose the correct handling based on the `fhirVersion` parameter in the `mimetype`. The `mimetype` is read from the `Accept` header and (for POST/PUT) the `Content-Type` header. See [Multiple versions of FHIR](#) for background info.
2. Upgrade to HL7.Fhir.Net API 1.3, see its [releasenotes](#).
3. Administration API imports both STU3 and R4 conformance resources, see [Conformance Resources](#)
 1. Note: [Terminology operations](#) are still only available for STU3.
 2. Note: [Subscriptions](#) are still only available for STU3.
4. Conditional delete on the Administration API. It works just as on the root, see [Create, read, update, patch, delete](#).
5. Defining a custom `SearchParameter` on a [Custom ResourceType](#) is now possible.
6. Canonical uris are now recognized when searching on references ([specification](#))
7. Vonk calls `UseIISIntegration` for better integration with IIS (if present).

3.31.3 Fix

1. In the settings, `PipelineOptions.Branch.Path` for the root had to be `/`. Now you can choose your own base (like e.g. `/fhir`)
2. `$meta`:
 1. enabled on history endpoint (e.g. `/Patient/123/_history/v1`)
 2. disabled on type and system level
 3. returned empty `Parameters` resource if resource had no `meta.profile`, now returns the resources `meta` element.
 4. when called on a non-existing resource, returns 404 (was: empty `Parameters` resource)
 5. added to the `CapabilityStatement`
3. History on non-existing resource returned `OperationOutcome` instead of 404.
4. The setting for `SupportedInteractions` was not enforced for custom operations.
5. `CapabilityStatement.name` is updated from Vonk beta conformance to Vonk FHIR Server <version> `CapabilityStatement`.
6. [Terminology services](#):
 1. `$lookup` did not work on `GET /CodeSystem`
 2. `$lookup` did not support the `coding` parameter
 3. `$expand` did not fill in the expansion element.
 4. Operations were not listed in the `CapabilityStatement`.

5. Namespace changed to Vonk.Plugins.Terminology, and adjusted accordingly in the default PipelineOptions.
7. A SearchParameter of type token did not work on an element of type string, e.g. CodeSystem.version.
8. Search with POST was broken.
9. If a long running task is active (responsecode 423, see *Import of Conformance Resources* and *Re-indexing for new or changed SearchParameters*), the OperationOutcome reporting that will now hide issues stating that all the arguments were not supported (since that is not the cause of the error).
10. Overriding an array in the settings was hard - it would still inherit part of the base setting if the base array was longer. We changed this: an array will always overwrite the complete base array. Note that this may trick you if you currently override a single array element with an environment variable. See *Hierarchy of settings*.
11. The element meta.source cannot be changed on subsequent updates to a resource (R4 specific)
12. SearchParameter StructureDefinition.ext-context yielded many errors in the log because the definition of the fhirpath in the specification is not correct. We provided a corrected version in errataFhir40.zip (see *Errata to the specification*).
13. *Enable or disable interactions* was not evaluated for custom operations.
14. Delete of an instance accepted searchparameters on the url.
15. Transactions: references to other resources in the transaction were not updated if the resource being referenced was in the transaction as an update (PUT). (this error was introduced in 2.0.0).

3.31.4 Plugin and Facade API

1. A new NuGet package is introduced: Vonk.Fhir.R4.
2. VonkConstants moved to the namespace Vonk.Core.Common (was: Vonk.Core.Context)
3. IResource.Navigator element is removed (was already obsolete). Instead: Turn it into an ITypedElement and use that for navigation with FhirPath.
4. InformationModel element is added to
 1. IResource: the model in which the resource is defined (VonkConstants.Model.FhirR3 or VonkConstants.Model.FhirR4)
 2. IVonkContext: the model that was specified in the Accept header
 3. IModelService: the model for which this service is valid (implementations are available for R3 and R4)
 4. InteractionHandler attribute: to allow you to specify that an operation is only valid for a specific FHIR version. This can also be done in the fluent interface with the new method AndInformationModel. See *Interaction Handling*
5. Dependency injection: if there are implementations of an interface for R3 and R4, the dependency injection in Vonk will automatically inject the correct one based on the InformationModel in the request.
6. If you want to register your own service just for one informationmodel, do that as follows:

Add a ContextAware attribute to the implementation class:

```
[ContextAware (InformationModels = new[] {VonkConstants.Model.FhirR3})]
public class MySearchRepository{...}
```

Then register the service as being ContextAware:

```
services.TryAddContextAware<ISearchRepository, MySearchRepository>(ServiceLifeTime.
↳ Scoped);
```

7. `FhirPropertyIndexBuilder` is moved to `Vonk.Fhir.R3` (and was already marked obsolete - avoid using it)
8. Implementations of the following that are heavily dependent upon version specific `HL7.Fhir` libraries have been implemented in both `Vonk.Fhir.R3` and `Vonk.Fhir.R4`.
 1. `IModelService`
 2. `IStructureDefinitionSummaryProvider` (to add type information to an `IResource` and turn it into an `ITypedElement`)
 3. `ValidationService`
9. `IConformanceContributor` is changed to `ICapabilityStatementContributor`. The methods on it have changed slightly as well because internally they now work on a version-independent model. Please review your `IConformanceContributor` implementations.

3.31.5 Examples

1. Document plugin:
 1. `Document Bundle` does not contain an identifier
 2. Missing unit test for custom resources
 3. Upgraded to Vonk 2.0.0 libraries (no, not yet 3.0.0-beta1)
2. Facade example
 1. Added support for searching directly on a reference from Observation to Patient (e.g. `/Observation?patient=Patient/3`).
 2. Fixed support for `_revinclude` of Observation on Patient (e.g. `/Patient?_revinclude:Observation:subject:Patient`).
 3. Upgraded to Vonk 2.0.0 libraries (no, not yet 3.0.0-beta1)
3. Plugin example
 1. Added examples for pre- and post handlers.

3.31.6 Known to-dos

1. *Re-indexing for new or changed SearchParameters*: does not work for R4 yet.
2. *Preloading resources*: does not work for R4 yet.
3. *Subscriptions*: do not work for R4 yet.
4. *Terminology services*: operations do not work for R4.
5. During *Import of Conformance Resources*: Files in the import directory and Simplifier projects are only imported for R3.

HOW TO UPGRADE FIRELY SERVER?

The process for upgrading Firely Server depends on whether you have a vanilla Firely Server, you added your own plugins or are running a Facade. This page describes the general process for each situation. Please refer to the [Release notes Firely Server](#) for details per each released version of Firely Server.

4.1 Upgrading Firely Server

4.1.1 Using the binary distribution

1. Download the latest version of Firely Server, see [Getting Started](#), and extract it to where you want it installed.
2. Copy your `appsettings.instance.json` and `logsettings.instance.json` files from the current installation to the new installation.
3. Check the [Release notes Firely Server](#) for any new settings that you may want to apply or change from their defaults.
4. Check the [Release notes Firely Server](#) for any actions that you need to take specifically for this upgrade.
5. Make sure the new installation can find the license file (see [License](#), general advice is to put the license file outside of the installation directory).
6. Create a backup of your current databases, both the main Resource database and the Administration database. See [Repository](#) to find the details on your configured database connection.
7. Stop the running instance of Firely Server (Ctrl + C if running from the console).
8. Switch to the new installation directory and start Firely Server from there (`> dotnet ./Vonk.Server.dll`)
9. Firely Server will now do several upgrade tasks, during which any web request will be responded to with 423 - Locked:
 1. If needed, an update is applied to the database structure.
 2. If Firely Server introduces a new version of the FHIR .NET API, Firely Server will load a new set of Conformance Resources from the `specification.zip` into the Administration database, for both FHIR STU3 and FHIR R4. In a specific case you can [prevent this step from happening](#).
10. When Firely Server is done with the tasks above, it is again available to process requests.
11. Check the log for warnings stating that you use obsolete settings. If so, adjust them and restart Firely Server.

If anything went wrong, go back:

1. Stop the (new) running instance of Firely Server.
2. Restore both databases from your backup.

3. Switch to the old installation directory and start the old version of Firely Server from there (`> dotnet .\Vonk.Server.dll`)
4. It should start as it did before you began the upgrade.
5. Report the problem to the Firely Server helpdesk, see [Contact us](#).

You may be able to avoid the import of `specification.zip` if:

- The Administration database is in SQLite and
- You have not made alterations to the Administration API through the Web API.

In this case you can simply replace the old database (usually with the filename `vonkadmin.db`) with the one from the new installation directory (in `./data/vonkadmin.db`). Do so *before* you start the new Firely Server installation. Anything specified in [AdministrationImportOptions](#) will be re-imported into the new database.

4.1.2 Using Docker

Revisit [Using Firely Server on Docker](#).

1. Stop the running container for Firely Server: `> docker stop vonk.server`.
2. Pull the latest image for Firely Server: `> docker pull simplifier/vonk`
3. Check the [Release notes Firely Server](#) for any new settings that you may want to apply or change from their defaults, and apply that to the `environment` setting in the docker-compose file.
4. Check the [Release notes Firely Server](#) for any action that you need to take specifically for this upgrade.
5. Create a backup of your current databases, both the main Resource database and the Administration database. See [Repository](#) and your docker-compose file to find the details on where your databases are.
6. Start the new version (see [Using Firely Server on Docker](#) for the various commands to run the Firely Server container).
7. Firely Server will now do several upgrade tasks, during which any web request will be responded to with 423 - Locked:
 1. If needed, an update is applied to the database structure.
 2. If Firely Server introduces a new version of the FHIR .NET API, Firely Server will load a new set of Conformance Resources from the `specification.zip` into the Administration database, for both FHIR STU3 and FHIR R4. In a specific case you can [prevent this step from happening](#).
8. When Firely Server is done with the tasks above, it is again available to process requests.
9. Check the log for warnings stating that you use obsolete settings. If so, adjust them and restart Firely Server.

If anything went wrong, go back:

1. Stop the (new) running container of Firely Server.
2. Restore both databases from your backup.
3. Specify your previous image of Firely Server in the docker command or in the docker-compose file: `simplifier\vonk:<previous-version-tag>`
4. Start the container based on this previous image.
5. It should start as it did before you began the upgrade.
6. Report the problem to the Firely Server helpdesk, see [Contact us](#).

4.2 Upgrading Plugins

Since a Plugin runs in the context of a Firely Server we advise you to start by upgrading your Firely Server, without loading your Plugin. Check the section on [Configuring the Firely Server Pipeline](#) to see how you can exclude your plugin from the pipeline.

Attention: We do not guarantee that a plugin built against version x.y.z of Firely Server can be run within a newer or older version as-is. Between minor versions recompilation is usually enough to update your plugin. Between major versions you should prepare for breaking changes in the public programming API. Sometimes we choose to apply such changes even on a minor version update, if we are fairly sure that you will not or only slightly be affected by it.

Upgrade the references in your plugin:

1. Open the source code of your plugin, and open the project file (`yourplugin.csproj`).
2. Change the references to the Firely Server.* packages to the version that you want to upgrade to.
3. Build and check the errors.
4. Check the list of breaking changes for the new Firely Server version in the [Release notes Firely Server](#). Applying the changes should fix the errors.
5. With some releases Firely Server is also upgraded to a newer version of the Firely .NET SDK. That is then mentioned in the release notes. If this is the case, also check the [SDK release notes](#) for breaking changes.
6. Still errors? Maybe we have overlooked a change. Please report it to us, see [Contact us](#). And if it is easy to fix - do so :-)
7. Build and publish your plugin.
8. Put the resulting dll's in the plugin directory of the new installation of Firely Server.
9. Re-include your plugin in the pipeline.
10. (Re)start Firely Server and test the working of your plugin.

4.3 Upgrading Facades

A Facade implementation is technically also a plugin, but one that only adds repository access services. For this it makes no sense to try to run Firely Server without the Facade as is described for plugins. So start with upgrading the references right away.

Especially for Facades to relational databases: match the version of EntityFrameworkCore that the new version of Firely Server is using. Check the list of changes to see whether we upgraded.

FIRELY SERVER ROADMAP

This page lists the features and changes we have planned for the foreseeable future. This planning is volatile, so changes will happen. We will update this page accordingly. You are also very welcome to provide input to us on features or fixes that you would like to see prioritized.

Disclaimer: No rights can be derived from this roadmap.

5.1 2021

5.1.1 Q1

- Full ONC/CMS compliance
- Firely Server: ONC Edition
 - Downloadable version of Firely Server that is fully compliant with the ONC Cures Act Final Rule
- CAR factory/files
 - Greatly reduces start-up time of Firely Server and other services
- Firely Validator (CLI)
 - Separate validation service as a CLI tool

5.1.2 Q2

- Bulk data export for MongoDB
- Bulk data import tool for MS SQL Server
 - CLI tool for bulk ingestion of data into Firely Server
- Firely Validator for cloud environments
 - Separate validation service optimized for usage in cloud environments
- Firely Server: CMS Edition
 - Downloadable version of Firely Server that is fully compliant with the CMS Final Rule
- Full compliance with MedMij standard

5.1.3 Q3

- Firely Server: MedMij Edition
 - Downloadable version of Firely Server that is fully compliant with the MedMij standards
- Bulk data export as a separate service - deploy and scale it independently of Firely Server itself.
- Bulk data import tool for MongoDB
 - CLI tool for bulk ingestion of data into Firely Server

5.1.4 Q4

- FHIR transaction support for MongoDB
- Enhanced support for subscriptions

FREQUENTLY ASKED QUESTIONS

... and known issues.

6.1 Conflicting resources upon import

When importing specification.zip for R4, Firely Server will report errors like these:

```
::
```

Artifact C:\Users<user>\AppData\Local\Temp\FhirArtifactCache-1.5.0-HI7\Fhir.R4.Specificationspecification_Fhir4_0dataelements.xml could not be imported. Error message: Found multiple conflicting resources with the same resource uri identifier.

Url: <http://hl7.org/fhir/StructureDefinition/de-Quantity.value>

File: C:\UsersChristiaan\AppData\Local\Temp\FhirArtifactCache-1.5.0-HI7\Fhir.R4.Specificationspecification_Fhir4_0dataelements.xml File: C:\UsersChristiaan\AppData\Local\Temp\FhirArtifactCache-1.5.0-HI7\Fhir.R4.Specificationspecification_Fhir4_0dataelements.xml
File: C:\UsersChristiaan\AppData\Local\Temp\FhirArtifactCache-1.5.0-HI7\Fhir.R4.Specificationspecification_Fhir4_0dataelements.xml

The error message is actually correct, since there *are* duplicate fullUrls in dataelements.xml in the specification. This has been reported in [Jira issue FHIR-25430](#).

6.2 Searchparameter errors for composite parameters

When importing specification.zip for various FHIR versions, Firely Server will report errors like these:

```
Composite SearchParameter 'CodeSystem.context-type-quantity' doesn't have components.
```

A searchparameter of type 'composite' should define which components it consists of. Firely Server checks whether all the components of such a composite searchparameter are present. If no components are defined at all - that is, SearchParameter.component is empty - it will display this error. This indicates an error in the definition of the searchparameter and should be solved by the author of it.

However, the implementation of this check seems to have an error so too many composite parameters are reported as faulty. We will address this issue in the next release.

6.3 .NET SDK not found

Since version 4.0 Vonk was renamed to Firely Server, including the main entrypoint. It changed from `vonk.server.dll` to `firely.server.dll`.

If you now still run `dotnet vonk.server.dll` on .NET runtime 3.1 it will state this error:

```
::
```

```
It was not possible to find any installed .NET Core SDKs Did you mean to run .NET Core SDK  
commands? Install a .NET Core SDK from: https://aka.ms/dotnet-download
```

This is very misleading. The actual error is that you probably tried to run `dotnet vonk.server.dll` but this dll no longer exists.

The same error can happen if you have built a Docker image of your own with `dotnet vonk.server.dll` as entry-point.

.NET 5 fixed this and more clearly states that the dll is missing.

CONFIGURING FIRELY SERVER

In this section we assume you have downloaded and installed the Firely Server binaries, and have obtained a license file. If not, please see the [Getting Started](#) and follow the steps there first.

The steps you followed to get started will provide you with a basic Firely Server, that runs on a standard port and keeps the data in a SQLite database.

If you need to adjust the port, or want to use a MongoDB, SQL or CosmosDB database, you can configure Firely Server by adjusting the [Firely Server settings](#).

If you want to change the way Firely Server logs its information, you can adjust the [Log settings](#).

7.1 Firely Server settings

Firely Server settings are controlled in json configuration files called `appsettings(.*).json`. The possible settings in these files are all the same and described below. The different files are read in a hierarchy so you can control settings on different levels. All appsettings files are in the Firely Server distribution directory, next to `Firely.Server.dll`. We go through all the sections of this file and refer you to detailed pages on each of them.

You can also control [Firely Server settings with Environment Variables](#).

Changes to the settings require a restart of Firely Server.

7.1.1 Hierarchy of settings

Firely Server reads its settings from these sources, in this order:

appsettings.default.json

Installed with Firely Server, contains default settings and a template setting if no sensible default is available.

appsettings.json

You can create this one for your own settings. Because it is not part of the Firely Server distribution, it will not be overwritten by a next Firely Server version.

environment variables

See [Firely Server settings with Environment Variables](#).

appsettings.instance.json

You can create this one to override settings for a specific instance of Firely Server. It is not part of the Firely Server distribution. This file is especially useful if you run multiple instances on the same machine.

Settings lower in the list override the settings higher in the list (think CSS, if you're familiar with that).

Warning: JSON settings files can have arrays in them. The configuration system can NOT merge arrays. So if you override an array value, you need to provide all the values that you want in the array. In the Firely Server settings this is relevant for e.g. `Validation.AllowedProfiles` and for the `PipelineOptions`.

Note: By default in ASP.NET Core, if on a lower level the array has more items, you will still inherit those extra items. We fixed this in Firely Server, an array will always overwrite the complete base array. To nullify an array, add the value with an array with just an empty string in it:

```
"PipelineOptions": {
  "Branches": [
    {
      "Path": "myroot",
      "Exclude": [""]
    }
  ]
}
```

This also means you cannot override a single array element with an environment variable. (Which was tricky anyway - relying on the exact number and order of items in the original array.)

7.1.2 Changing the settings

In general you do not change the settings in `appsettings.default.json` but create your own overrides in `appsettings.json` or `appsettings.instance.json`. That way your settings are not overwritten by a new version of Firely Server (with a new `appsettings.default.json` therein), and you automatically get sensible defaults for any new settings introduced in `appsettings.default.json`.

Settings after first install

After you installed Firely Server (see *Getting Started*), either:

- copy the `appsettings.default.json` to `appsettings[.instance].json` and remove settings that you do not intend to alter, or
- create an empty `appsettings[.instance].json` and copy individual parts from the `appsettings.default.json` if you wish to adjust them.

Adjust the new `appsettings[.instance].json` to your liking using the explanation below.

When running *Firely Server on Docker* you probably want to adjust the settings using the *Environment Variables*.

Settings after update

If you install the binaries of an updated version of Firely Server, you can:

- copy the new binaries over the old ones, or
- deploy the new version to a new directory and copy the `appsettings[.instance].json` over from the old version.

In both cases, check the [Release notes Firely Server](#) to see if settings have changed, or new settings have been introduced. If you want to adjust a changed / new setting, copy the relevant section from `appsettings.default.json` to your own `appsettings[.instance].json` and then adjust it.

Commenting out sections

JSON formally has no notion of comments. But the configuration system of ASP.Net Core (and hence Firely Server) accepts double slashes just fine:

```
"Administration": {
  "Repository": "SQLite", //Memory / SQL / MongoDB
  "SqlDbOptions": {
    "ConnectionString": "connectionstring to your Firely Server Admin SQL Server,
    ↳database (SQL2012 or newer); Set MultipleActiveResultSets=True",
    "SchemaName": "vonkadmin",
    "AutoUpdateDatabase": true,
    "MigrationTimeout": 1800 // in seconds
    // "AutoUpdateConnectionString" : "set this to the same database as
    ↳'ConnectionString' but with credentials that can alter the database. If not set,
    ↳defaults to the value of 'ConnectionString'"
  },
}
```

This will ignore the `AutoUpdateConnectionString`.

7.1.3 Log of your configuration

Because the hierarchy of settings can be overwhelming, Firely Server logs the resulting configuration. To enable that, the loglevel for `Vonk.Server` must be `Information` or more detailed. That is set for you by default in `logsettings.default.json`. Refer to [Log settings](#) for information on setting log levels.

7.1.4 Administration

```
"Administration": {
  "Repository": "SQLite", //Memory / SQL / MongoDB are other options, but SQLite is
  ↳advised.
  "MongoDbOptions": {
    "ConnectionString": "mongodb://localhost/vonkadmin",
    "EntryCollection": "vonkentries"
  },
  "SqlDbOptions": {
    "ConnectionString": "connectionstring to your Firely Server Admin SQL Server,
    ↳database (SQL2012 or newer); Set MultipleActiveResultSets=True",
    "SchemaName": "vonkadmin",
  },
}
```

(continues on next page)

(continued from previous page)

```

    "AutoUpdateDatabase": true,
    "MigrationTimeout": 1800 // in seconds
    // "AutoUpdateConnectionString" : "set this to the same database as
    ↪ 'ConnectionString' but with credentials that can alter the database. If not set, ↪
    ↪ defaults to the value of 'ConnectionString'"
  },
  "SQLiteDbOptions": {
    "ConnectionString": "Data Source=./data/vonkadmin.db",
    "AutoUpdateDatabase": true
  },
  "Security": {
    "AllowedNetworks": [ "::1" ], // i.e.: ["127.0.0.1", "::1" (ipv6 localhost), "10.1.
    ↪ 50.0/24", "10.5.3.0/24", "31.161.91.98"]
    "OperationsToBeSecured": [ "reindex", "reset", "preload" ]
  }
},

```

The Administration section is to *Configure the Administration API* and its repository.

7.1.5 License

```

"License": {
  "LicenseFile": "firelyserver-trial-license.json",
  "RequestInfoFile": "./.vonk-request-info.json",
  "WriteRequestInfoFileInterval": 15 // in minutes
}

```

The *Getting Started* explains how to obtain a licensefile for Firely Server. Once you have it, put the path to it in the LicenseFile setting. Note that in json you either use forward slashes (/) or double backward slashes (\\) as path separators.

Other settings:

- RequestInfoFile sets the location of the file with request information. This file will be used in future releases.
- WriteRequestInfoFileInterval sets the time interval (in minutes) to write aggregate information about processed requests to the RequestInfoFile.

7.1.6 Repository

```

"Repository": "SQLite", //Memory / SQL / MongoDB / CosmosDb

```

1. Repository: Choose which type of repository you want. Valid values are:

1. Memory
2. SQL, for Microsoft SQL Server
3. SQLite
4. MongoDB
5. CosmosDb

Memory

```
"MemoryOptions": {
  "SimulateTransactions": "false"
},
```

Refer to *Using the In-Memory storage* for configuring the In-Memory storage.

MongoDB

```
"MongoDbOptions": {
  "ConnectionString": "mongodb://localhost/vonkstu3",
  "EntryCollection": "vonkentries",
  "SimulateTransactions": "false"
},
```

Refer to *Using MongoDB* for configuring the connection to your MongoDB databases.

SQL

```
"SqlDbOptions": {
  "ConnectionString": "connectionstring to your Firely Server SQL Server database.
↳ (SQL2012 or newer); Set MultipleActiveResultSets=True",
  "SchemaName": "vonk",
  "AutoUpdateDatabase": true,
  "MigrationTimeout": 1800 // in seconds
  //"AutoUpdateConnectionString" : "set this to the same database as 'ConnectionString
↳ ' but with credentials that can alter the database. If not set, defaults to the value.
↳ of 'ConnectionString'"
},
```

Refer to *Using SQL server* for configuring access to your SQL Server databases.

SQLite

```
"SQLiteDbOptions": {
  "ConnectionString": "Data Source=./data/vonkdata.db",
  "AutoUpdateDatabase": true
},
```

Refer to *Using SQLite* for configuring access to your SQLite Server databases.

CosmosDb

```
"CosmosDbOptions": {
  "ConnectionString": "mongodb://<password>@<server>:10255/vonk?ssl=true&
↪replicaSet=globaldb",
  "EntryCollection": "vonkentries"
},
```

Refer to *Using Microsoft Azure CosmosDB* for configuring access to your CosmosDb databases.

7.1.7 http and https

```
"Hosting": {
  "HttpPort": 4080,
  //"HttpsPort": 4081, // Enable this to use https
  //"CertificateFile": "<your-certificate-file>.pfx", //Relevant when HttpsPort is
↪present
  //"CertificatePassword" : "<cert-pass>" // Relevant when HttpsPort is present
},
```

Refer to *Configure http and https* for enabling https and adjusting port numbers.

7.1.8 Validation

```
"Validation": {
  "Parsing": "Permissive", // Permissive / Strict
  "Level": "Off", // Off / Core / Full
  "AllowedProfiles": []
},
```

Refer to *Validating incoming resources*.

7.1.9 Search and History

```
"BundleOptions": {
  "DefaultCount": 10,
  "MaxCount": 50,
  "DefaultSort": "-_lastUpdated"
},
```

The Search and History interactions return a bundle with results. Users can specify the number of results that they want to receive in one response with the `_count` parameter.

- `DefaultCount` sets the number of results if the user has not specified a `_count` parameter.
- `MaxCount` sets the number of results in case the user specifies a `_count` value higher than this maximum. This is to protect Firely Server from being overloaded.
- `DefaultCount` should be less than or equal to `MaxCount`
- `DefaultSort` is what search results are sorted on if no sort order is specified in the request. If a sort order is specified, this is still added as the last sort clause.

7.1.10 Batch and transaction

```
"BatchOptions": {
  "MaxNumberOfEntries": 100
},
```

This will limit the number of entries that are accepted in a single Batch or Transaction bundle.

Note: This setting has been moved to the `SizeLimits` setting as of Firely Server (Vonk) version 0.7.1, and the logs will show a warning that it is deprecated when you still have it in your appsettings file.

7.1.11 Protect against large input

```
"SizeLimits": {
  "MaxResourceSize": "1MiB",
  "MaxBatchSize": "5MiB",
  "MaxBatchEntries": 150
},
```

- `MaxResourceSize` sets the maximum size of a resource that is sent in a create or update.
- `MaxBatchSize` sets the maximum size of a batch or transaction bundle. (Note that a POST `http(s)://<firely-server-endpoint>/Bundle` will be limited by `MaxResourceSize`, since the bundle must be processed as a whole then.)
- `MaxBatchEntries` limits the number of entries that is allowed in a batch or transaction bundle.
- The values for `MaxResourceSize` and `MaxBatchSize` can be expressed in b (bytes, the default), kB (kilobytes), KiB (kibibytes), MB (megabytes), or MiB (mebibytes). Do not put a space between the amount and the unit.

7.1.12 SearchParameters and other Conformance Resources

```
"AdministrationImportOptions": {
  "ImportDirectory": "./vonk-import",
  "ImportedDirectory": "./vonk-imported", //Do not place ImportedDirectory *under*
↳ ImportDirectory, since an import will recursively read all subdirectories.
  "SimplifierProjects": [
    {
      "Uri": "https://stu3.simplifier.net/<your-project>",
      "UserName": "Simplifier user name",
      "Password": "Password for the above user name",
      "BatchSize": 20
    }
  ]
}
```

See *Conformance Resources* and *Custom Search Parameters*.

7.1.13 Cache of Conformance Resources

```
"Cache": {  
  "MaxConformanceResources": 5000  
}
```

Firely Server caches StructureDefinitions and other conformance resources that are needed for (de)serialization and validation in memory. If more than `MaxConformanceResources` get cached, the ones that have not been used for the longest time are discarded. If you frequently encounter a delay when requesting less used resource types, a larger value may help. If you are very restricted on memory, you can lower the value.

7.1.14 Reindexing for changes in SearchParameters

```
"ReindexOptions": {  
  "BatchSize": 100,  
  "MaxDegreeOfParallelism": 10  
},
```

See *Re-index Configuration*.

7.1.15 Restrict supported resources and SearchParameters

```
"SupportedModel": {  
  "RestrictToResources": [ "Patient", "Observation" ],  
  "RestrictToSearchParameters": ["Patient.active", "Observation.patient", "Resource._id",  
  ↪ "StructureDefinition.url"],  
  "RestrictToCompartments": ["Patient"]  
},
```

By default, Firely Server supports all ResourceTypes, SearchParameters and CompartmentDefinitions from the specification. They are loaded from the *specification.zip*. If you want to limit support, you can do so with the configuration above. This is primarily targeted towards Facade builders, because they have to provide an implementation for everything that is supported.

Be aware that:

- support for `_type` and `_id` must not be disabled
- the Administration API requires support for the 'url' SearchParameter on the conformance resourcetypes
- this uses the search parameter names, not the path within the resource - so for example to specify `Patient.address.postalCode` as a supported location, you'd use `"Patient.address-postalcode"`.

7.1.16 Enable or disable interactions

By default, the value `SupportedInteractions` contains all the interactions that are implemented in Firely Server. But you can disable interactions by removing them from these lists.

```
"SupportedInteractions": {
  "InstanceLevelInteractions": "read, vread, update, delete, history, conditional_
↪delete, conditional_update, $validate",
  "TypeLevelInteractions": "create, search, history, $validate, $snapshot, conditional_
↪create",
  "WholeSystemInteractions": "capabilities, batch, transaction, history, search,
↪$validate"
},
```

If you implement a custom operation in a plugin, you should also add the name of that operation at the correct level. E.g. add `$convert` to `TypeLevelInteractions` to allow `<base>/<resourcetype>/$convert`.

7.1.17 Subscriptions

```
"SubscriptionEvaluatorOptions": {
  "Enabled": true,
  "RepeatPeriod": 20000,
  "SubscriptionBatchSize" : 1
},
```

See *Subscriptions*.

7.1.18 Information model

Firely Server supports the use of multiple information models (currently FHIR STU3 and R4) simultaneously. The `InformationModel` section contains the related settings. By default, Firely Server serves both versions from the root of your web service, defaulting to STU3 when the client does not use `Accept` or `_format` to specify either one. Mapping a path or a subdomain to a specific version creates an additional URI serving only that particular version.

```
"InformationModel": {
  "Default": "Fhir4.0", // For STU3: "Fhir3.0". Information model to use when none is_
↪specified in either mapping, the _format parameter or the ACCEPT header.
  "Mapping": {
    "Mode": "Off"
    // "Mode": "Path", // yourserver.org/r3 => FHIR STU3; yourserver.org/r4 => FHIR R4
    // "Map": {
    //   "/R3": "Fhir3.0",
    //   "/R4": "Fhir4.0"
    // }
    // "Mode": "Subdomain", // r3.yourserver.org => FHIR STU3; r4.yourserver.org => FHIR_
↪R4
    // "Map":
    // {
    //   "r3": "Fhir3.0",
    //   "r4": "Fhir4.0"
    // }
```

(continues on next page)

(continued from previous page)

```
}  
},
```

See *Multiple versions of FHIR*.

7.1.19 Patient Everything Operation

```
"PatientEverythingOperation": {  
  "AdditionalResources": [ "Organization", "Location", "Substance", "Medication", "Device"  
↪  ] // included referenced resources, additional to the Patient compartment resources  
},
```

The Patient \$everything operation returns all resources linked to a patient that are listed in the Compartment Patient. This section allows you to define additional resources that will be included in the resulting searchset bundle.

See *Patient \$everything*.

7.1.20 FHIR Capabilities

```
"FhirCapabilities": {  
  "ConditionalDeleteOptions": {  
    "ConditionalDeleteType": "Single", // Single or Multiple,  
    "ConditionalDeleteMaxItems": 1  
  }  
},
```

See *Create, read, update, patch, delete*.

7.1.21 History size

```
"HistoryOptions": {  
  "MaxReturnedResults": 100  
}
```

See *History*.

7.1.22 Configuring the Firely Server Pipeline

You can add your own plugins to the Firely Server pipeline, or control which of the standard Firely Server plugins are used for your Firely Server, by changing the PipelineOptions.

```
"PipelineOptions": {  
  "PluginDirectory": "./plugins",  
  "Branches": [  
    {  
      "Path": "/",  
      "Include": [  
        "Vonk.Core",
```

(continues on next page)

(continued from previous page)

```

        "Vonk.Fhir.R3",
        "Vonk.Fhir.R4",
        // etc.
    ],
    "Exclude": [
    ]
},
{
    "Path": "/administration",
    "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",
        "Vonk.Fhir.R4",
        // etc.
    ],
    "Exclude": [
        "Vonk.Core.Operations"
    ]
}
]
}

```

It is possible to disable a specific information model by removing Vonk.Fhir.R3 or Vonk.Fhir.R4 from the pipeline

Please note the warning on merging arrays in [Hierarchy of settings](#).

See [Firely Server Plugins](#) for more information and an example custom plugin.

7.2 Firely Server settings with Environment Variables

7.2.1 Environment Variables for appsettings

All the settings in *Firely Server settings* can be overridden by environment variables on your OS. This can be useful if you want to deploy Firely Server to several machines, each having their own settings for certain options. For [Using Firely Server on Docker](#) using environment variables in the docker-compose file is currently the only way to pass settings to the container. Or if you don't want a database password in the appsettings.json file.

The format for the environment variables is:

```
VONK_<setting_level_1>[:<setting_level_n>]*
```

So you start the variable name with the prefix 'VONK_', and then follow the properties in the json settings, separating each level with a colon ':'. Some examples:

appsettings.json:

```
"Repository" : "SQL"
```

environment variable:

```
VONK_Repository=SQL
```

To access an embedded value, using the ':' separator:

appsettings.json:

```
"Administration" : {
  "SqlDbOptions" : {
    "ConnectionString" : "<some connectionstring>"
  }
}
```

environment variable:

```
VONK_Repository:SqlDbOptions:ConnectionString=<some connectionstring>
```

To access an array item, use 0-based indexing:

```
VONK_PipelineOptions:Branches:0:Exclude:0=Vonk.Repository.Memory
VONK_PipelineOptions:Branches:0:Exclude:1=Vonk.Repository.Sql
```

Arrays in Environment Variables

Sometimes the appsettings allow for an array of values, e.g. in the setting for AllowedProfiles in *Validating incoming resources*. You can address them by appending an extra colon and an index number.

appsettings.json:

```
"Validation": {
  "ValidateIncomingResources": "true",
  "AllowedProfiles":
  [
    http://hl7.org/fhir/StructureDefinition/daf-patient,
    http://hl7.org/fhir/StructureDefinition/daf-allergyintolerance
  ]
},
```

environment variables:

```
VONK_Validation:ValidateIncomingResources=true
VONK_Validation:AllowedProfiles:0=http://hl7.org/fhir/StructureDefinition/daf-patient
VONK_Validation:AllowedProfiles:1=http://hl7.org/fhir/StructureDefinition/daf-
↪allergyintolerance
```

7.2.2 Log settings with Environment Variables

You can control the *Log settings* with Environment Variables the same way as the *Environment Variables for appsettings* above. The difference is in the prefix. For the log settings we use 'VONKLOG_'.

logsettings.json

```
"Serilog": {
  "MinimumLevel": {
    "Override": {
      "Vonk.Configuration": "Information",
```

environment variable:

```
VONKLOG_Serilog:MinimumLevel:Override:Vonk.Configuration=Information
```

7.2.3 Changing Environment Variables on Windows

In Windows you can change the Environment Variables with Powershell or through the UI. Based on the first example above:

- In Powershell run: `> $env:VONK_Repository="SQL"`
- or go to your *System*, open the *Advanced system settings* → *Environment variables* and create a new variable with the name `VONK_Repository` and set the value to `"SQL"` (you don't need to enter the quotes here).

7.3 Configure the Administration API

This configuration is part of *Firely Server settings*.

```
"Administration": {
  "Repository": "SQLite", //Memory / SQL / MongoDB
  "MongoDbOptions": {
    "ConnectionString": "mongodb://localhost/vonkadmin",
    "EntryCollection": "vonkentries"
  },
  "SqlDbOptions": {
    "ConnectionString": "connectionstring to your Firely Server Admin SQL Server_
→ database (SQL2012 or newer); Set MultipleActiveResultSets=True",
    "SchemaName": "vonkadmin",
    "AutoUpdateDatabase": true,
    "MigrationTimeout": 1800 // in seconds
    // "AutoUpdateConnectionString" : "set this to the same database as 'ConnectionString
→ ' but with credentials that can alter the database. If not set, defaults to the value_
→ of 'ConnectionString'"
  },
  "SQLiteDbOptions": {
    "ConnectionString": "Data Source=./data/vonkadmin.db",
    "AutoUpdateDatabase": true,
    "MigrationTimeout": 1800 // in seconds
  },
  "Security": {
    "AllowedNetworks": [ ":::1" ], // i.e.: ["127.0.0.1", ":::1" (ipv6 localhost), "10.1.
→ 50.0/24", "10.5.3.0/24", "31.161.91.98"]
    "OperationsToBeSecured": [ "reindex", "reset", "preload" ]
  }
},
```

7.3.1 Choosing your storage

1. Repository: Choose which type of repository you want. Valid values are:
 1. Memory
 2. SQL
 3. SQLite
 4. MongoDB
1. MongoDBOptions: Use these with "Repository": "MongoDb", see *Using MongoDB* for details.
2. SqlDbOptions: Use these with "Repository": "SQL", see *Using SQL server* for details.
3. SQLiteDbOptions: Use these with "Repository": "SQLite", see *Using SQLite* for details.

7.3.2 Limited access

1. Security: You can restrict access to the operations listed in `OperationsToBeSecured` to only be invoked from the IP addresses listed in `AllowedNetworks`.
 - Operations that can be secured are:
 - `reindex` (see *Re-indexing for new or changed SearchParameters*)
 - `reset` (see *Reset the database*)
 - `preload` (see *Preloading resources*)
 - `StructureDefinition` (restrict both read and write)
 - `SearchParameter` (restrict both read and write)
 - `ValueSet` (restrict both read and write)
 - `CodeSystem` (restrict both read and write)
 - `CompartmentDefinition` (restrict both read and write)
 - `Subscription`: (restrict both read and write, see *Subscriptions*)
 - The `AllowedNetworks` have to be valid IP addresses, either IPv4 or IPv6, and masks are allowed.

7.4 Conformance Resources

Firely Server uses [Conformance Resources](#) along with some [Terminology Resources](#) for various operations:

- `SearchParameter`: For indexing resources and evaluating *search* interactions.
- `StructureDefinition`: For *snapshot generation*, and of course – along with `ValueSet` and `CodeSystem` – for *validation*.
- `CompartmentDefinition`: For *access control* and `Compartment Search`.
- `ValueSet` and `CodeSystem`: For *Terminology services* operations.
- `StructureMap` and `ConceptMap`: for mapping

You can control the behaviour of Firely Server for these interactions by loading resources of these types into Firely Server. There are two ways of doing this:

1. With regular FHIR interactions (create, update, delete) on the *Firely Server Administration API*.

2. With the *Import of Conformance Resources*.

No matter which method you use, all Conformance resources are persisted in the Administration API database (see *Configure the Administration API* for configuring that database), and available through the Administration API endpoint (`<firely-server-endpoint>/administration`)

For each resourcetype the base profile is listed in the CapabilityStatement under `CapabilityStatement.rest.resource.profile` and (since FHIR R4) all the other profiles are listed under `CapabilityStatement.rest.resource.supportedProfile`. So by requesting the *CapabilityStatement* you can easily check whether your changes to the StructureDefinitions were correctly processed by Firely Server.

Attention: Please be aware that Conformance Resources have to have a **unique canonical url** within the FHIR Version they are loaded, in their url element. Firely Server does not allow you to POST two conformance resources with the same canonical url. For SearchParameter resources, the combination of base and code must be unique.

Attention: Creates or updates of **SearchParameter** resources should be followed by a *re-index*.

Before you delete a SearchParameter, be sure to remove it from the index first, see the `exclude` parameter in *re-index*.

Changes to the other types of resources have immediate effect.

Attention: A StructureDefinition can only be posted in the context of a FHIR Version that matches the StructureDefinition.fhirVersion. See *Multiple versions of FHIR*.

Note: The import process will retain any id that is already in the resource, or assign a new id if there is none in the resource. For FHIR versions other than STU3, a postfix is appended to the id to avoid collisions between FHIR versions. See also *Conformance resources*.

7.4.1 Import of Conformance Resources

The import process of conformance resources runs on every startup of Firely Server, and *on demand*.

The process uses these locations on disk:

- ImportDirectory;
- ImportedDirectory;
- a read history in the file `.vonk-import-history.json`, written in ImportedDirectory.

Attention: Please make sure that the Firely Server process has write permission on the ImportedDirectory.

The process follows these steps for each FHIR version (currently STU3 and R4, and experimentally for R5)

1. Load the *Default Conformance Resources*, if they have not been loaded before.
2. Load the *Errata to the specification*, if they have not been loaded before.
3. *Load Conformance Resources from disk*. After reading, the read files are registered in the read history.

4. *Load Conformance Resources from simplifier.net*. After reading, the project is registered in the read history. Subsequent reads will query only for resources that have changed since the last read.

Loading the conformance resources from the various sources can take some time, especially on first startup when the *Default Conformance Resources* have to be imported. During the import Firely Server will respond with 423 'Locked' to every request to avoid storing or retrieving inconsistent data.

The read history keeps a record of files that have been read, with an MD5 hash of each. If you wish to force a renewed import of a specific file, you should:

- manually edit the read history file and delete the entry about that file;
- provide the file again in the ImportDirectory (if you deleted it previously - Vonk does not delete it).

7.4.2 Retain the import history

If you run the Administration database on SQL Server or MongoDB it is important to *retain* the `.vonk-import-history` file. This means that if you run Firely Server on something stateless like a Kubernetes pod, or a webapp service, you need to attach file storage on which to store this file. If you do not do that, Firely Server will import all the conformance resources *on every start*.

7.4.3 Running imports with multiple instances

If you run multiple instances of Firely Server each will have its own `/administration` pipeline. So you need to make sure that only 1 instance will perform the import. The import at startup will happen when:

- we upgraded to a new version on the FHIR .NET API (always mentioned in the *releasenotes*)
- you add new resources to the `ImportDirectory`
- resources retrieved from Simplifier are renewed.

To ensure that only one instance runs the import you can do two things:

1. Make sure only 1 instance is running:
 1. Stop Firely Server
 2. Scale down to 1 instance
 3. Upgrade Firely Server (by referring to a newer image, or installing newer binaries)
 4. Start Firely Server
 5. Let it do the import
 6. Then scale back up to multiple instances.
2. Exclude the namespace `Vonk.Administration.Api.Import` from the *PipelineOptions* in branch `administration` on all but one instance.

If you want to use the manual import (`<url>/administration/import`) you are advised to apply solution nr. 1 above. In the second solution the call may or may not end up on an instance having the Import functionality.

We are aware that this can be a bit cumbersome. On the *Firely Server Roadmap* is therefore the story to host the Administration API in its own microservice.

7.4.4 Default Conformance Resources

Firely Server comes with the specification.zip file from the HL7 FHIR API. It contains all the Conformance resources from the specification. These are loaded and used for validation and snapshot generation by default.

Some of the conformance resources (especially SearchParameters) contain errors in the core specification. We try to correct all errors in *Errata to the specification*. You can also override them yourself by:

- updating them through the administration api, as described below;
- providing an altered version in the ImportDirectory, with the same id and canonical url.

Attention: The Core Specification provides almost 4000 Conformance Resources. Depending on the machine it may take a few minutes to load and index them.

7.4.5 Load Conformance Resources from disk

Firely Server can read SearchParameter and CompartmentDefinition resources from a directory on disk at startup. The AdministrationImportOptions in the *Firely Server settings* control from which directory resources are loaded:

```
"AdministrationImportOptions": {
  "ImportDirectory": "<path to the directory you want to import from, default ./vonk-
↳import>",
  "ImportedDirectory": "<path to the directory where imported files are moved to,
↳default ./vonk-imported>"
},
```

ImportDirectory

All files and zip files will be read, and any conformance resources in them will be imported. By default, STU3 is assumed. If you have R4 conformance resources, place them in a sibling directory that has the same name as your “ImportDirectory” with .R4 appended to it – so for example ./vonk-import.R4.

ImportedDirectory

This directory will contain the read history in the .vonk-import-history.json file. Please note, that this information is stored directly in the administration database when running on SQLite.

Note that in json you either use forward slashes (/) or double backward slashes (\\) as path separators.

7.4.6 Load Conformance Resources from simplifier.net

You are encouraged to manage and publish your profiles and related Conformance Resources on simplifier.net. If you do that, you can have Firely Server read those. You configure this in the *Firely Server settings*:

```
"AdministrationImportOptions": {
  "SimplifierProjects": [
    {
      "Uri": "FHIR endpoint for retrieving StructureDefinitions",
      "UserName": "UserName for retrieving the StructureDefinitions",
      "Password": "Password for the above user name",
      "BatchSize": "<number of resources imported at once, optional - default is 20>"
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
  ],
}
```

Uri

must point to a Simplifier project endpoint, see below on how to get this

UserName

your username, if you access a private Simplifier project

Password

password with the username

BatchSize

you normally don't need to change this parameter

You can load from multiple Simplifier projects by adding them to the list. The environment variable version of this is:

```
VONK_Administration:SimplifierProjects:0:Uri=<FHIR endpoint for retrieving_
↳ StructureDefinitions>
```

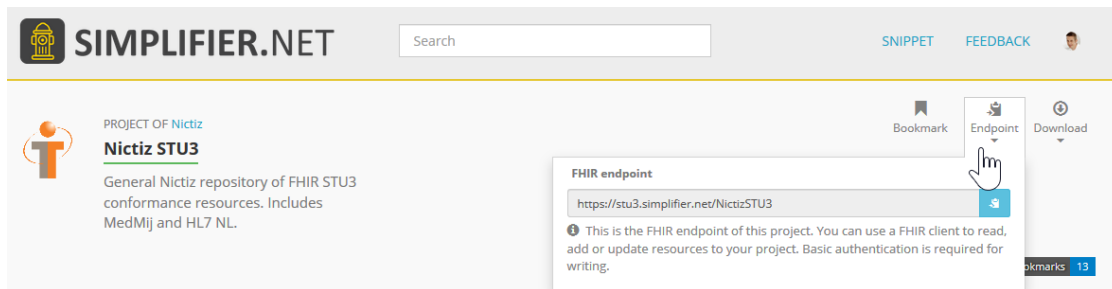
Vonk automatically finds the FHIR version for each project and imports it only for the matching FHIR version.

Get a FHIR endpoint for a Simplifier project

Open the project of your choice on <https://simplifier.net>. There are two limitations:

1. You must have access to the project (so either public or private but accessible to you)
2. The project must be STU3

Then on the overview page of the project click 'Endpoint' and copy the value you see there:



By default the endpoint is `https://stu3.simplifier.net/<projectname>`

7.4.7 Load Conformance Resources on demand

It can be useful to reload the profiles, e.g. after you have finalized changes in your project. Therefore you can instruct Firely Server to actually load the profiles from the source(s) with a separate command:

```
POST http(s)://<firely-server-endpoint>/administration/importResources
```

The operation will return an OperationOutcome resource, containing details about the number of resources created and updated, as well as any errors that occurred. Please note that this will also respect the history of already read files, and not read them again.

7.4.8 Manage Conformance Resources with the Administration API

The *Firely Server Administration API* has a FHIR interface included, on the `https://<firely-server-endpoint>/administration` endpoint. On this endpoint you can do most of the FHIR interactions (create, read, update, delete, search) on these resourcetypes:

- SearchParameter
- StructureDefinition
- ValueSet
- CodeSystem
- CompartmentDefinition

If you are *not permitted* to access the endpoint for the resource you want to manage (e.g. `<firely-server-endpoint>/administration/StructureDefinition`), Firely Server will return status-code 403.

Note: You can also do the same interactions on the same resourcetypes on the normal (or 'data') FHIR endpoint `https://<firely-server-endpoint>`. This will only result in storing, updating or deleting the resource. But it will not have any effect on the way Firely Server operates.

Example

To add a StructureDefinition to Firely Server

```
POST <firely-server-endpoint>/administration/StructureDefinition
```

- In the body provide the StructureDefinition that you want to add.
- The Content-Type header must match the format of the body (application/fhir+json or application/fhir+xml)

If you prefer to assign your own logical id to e.g. StructureDefinition 'MyPatient', you can use an update:

```
PUT <firely-server-endpoint>/administration/StructureDefinition/MyPatient
```

7.5 Validating incoming resources

You can have Firely Server validate all resources that are sent in for create or update. The setting to do that is like this:

```
"Validation": {
  "Parsing": "Permissive", // Permissive / Strict
  "Level": "Off", // Off / Core / Full
  "AllowedProfiles":
  [
    "http://hl7.org/fhir/StructureDefinition/daf-patient",
    "http://hl7.org/fhir/StructureDefinition/daf-allergyintolerance"
  ]
},
```

7.5.1 Parsing

Every incoming resource - xml or json - has to be syntactically correct. That is not configurable.

Beyond that, you can choose between Permissive or Strict parsing. Permissive allows for:

- empty elements (not having any value, child elements or extensions)
- the `fhir_comments` element
- errors in the xhtml of the Narrative
- **json specific:**
 - array with a single value instead of just a value, or vice versa (json specific)
- **xml specific:**
 - repeating elements interleaved with other elements
 - elements out of order
 - mis-representation (element instead of attribute etc.)

7.5.2 Validation

You can choose the level of validation:

- Off: no validation is performed.
- Core: the resource is validated against the core definition of the resourcetype.
- Full: the resource is validated against the core definition of the resourcetype and against any profiles it its `meta.profile` element.

7.5.3 Allow for specific profiles

To enable this feature, set `Level` to `Full`.

If you leave the list of `AllowedProfiles` empty, any resource will be allowed (provided it passes the validations set in `Parsing` and `Level`).

When you add canonical urls of `StructureDefinitions` to this list, Firely Server will:

- check whether the incoming resource has any of these profiles listed in its `meta.profile` element
- validate the resource against the profiles listed in its `meta.profile` element.

So in the example above, Firely Server will only allow resources that conform to either the DAF Patient profile or the DAF AllergyIntolerance profile.

Note that the resource has to declare conformance to the profile in its `meta.profile` element. Firely Server will *not* try to validate a resource against all the `Validation.AllowedProfiles` to see whether the resource conforms to any of them, only those that the resource claims conformance to.

7.5.4 Upgrading from < 2.0

Previous to version 2.0.0, the setting was "ValidateIncomingResources": "true" // or false. The corresponding settings since version 2.0.0 are:

- true / no AllowedProfiles => "Level": "Core"
- true / with AllowedProfiles => "Level": "Full", keep the AllowedProfiles as it is.
- false => "Level": "Off"

7.6 Using the In-Memory storage

- Navigate to your Firely Server working directory
- Changing a setting means overriding it as described in *Changing the settings*.
- Find the Repository setting:

```
"Repository": "Memory",
```

- If it is not already set to Memory, do so now.
- You can set SimulateTransactions to “true” if you want to experiment with [FHIR transactions](#). The In-Memory implementation does not support real transactions, so in case of an error already processed entries will NOT be rolled back:

```
"MemoryOptions": {
  "SimulateTransactions": "true"
},
```

7.6.1 Using the In-Memory storage for the Administration API database

This works the same as with the normal Firely Server database, except that you put the settings within the Administration section

E.g.:

```
"Administration": {
  "Repository": "Memory",
  "MemoryOptions": {
    "SimulateTransactions": "false"
  }
}
```

Warning: Using the In-Memory storage for the administration layer will cause Firely Server to load the specification files on each startup. This takes several minutes, and Firely Server will respond with a ‘423 - Locked’ error to all requests during that time. As of version 0.7.1 we have implemented support for SQLite, which we recommend to use instead of the In-Memory storage. See *Prefer SQLite for Firely Server Administration* for more information.

7.7 Using MongoDB

We assume you already have MongoDB installed. If not, please refer to the [MongoDB download](#) pages.

Firely Server can work with MongoDB 4.0 and higher. Since Firely Server (Vonk) version 3.7.0 Firely Server uses the MongoDB Aggregation Framework heavily, and you are advised to upgrade to MongoDB 4.4 (or newer). In this version issue *SERVER-7568* <<https://jira.mongodb.org/browse/SERVER-7568>> is solved, so the most selective index is used more often.

- Navigate to your Firely Server working directory
- Changing a setting means overriding it as described in *Changing the settings*.
- Find the Repository setting:
`"Repository": "SQLite",`
- Change the setting to MongoDB
- If you have your own database in MongoDB already, change the `MongoDbOptions` to reflect your settings:

```
"MongoDbOptions": {  
  "ConnectionString": "mongodb://localhost/vonkstu3",  
  "EntryCollection": "vonkentries",  
  "SimulateTransactions": "false"  
},
```

- If MongoDB does not have a database and/or collection by this name, Firely Server will create it for you.
- Find the section called `PipelineOptions`. Make sure it contains the MongoDB repository in the root path for Firely Server Data:

```
"PipelineOptions" :  
{  
  "Branches" : [  
    "/" : {  
      "Include" : [  
        "Vonk.Repository.MongoDb.MongoDbVonkConfiguration"  
        //...  
      ]  
    }  
  ]  
}
```

- You can set `SimulateTransactions` to “true” if you want to experiment with [FHIR transactions](#). MongoDB does not support real transactions across documents, so in case of an error already processed entries will NOT be rolled back.

7.7.1 Using MongoDB for the Administration API database

Although we encourage you to use *SQLite for Firely Server Administration*, you can still use MongoDB for Firely Server Administration as well.

This works the same as with the normal Firely Server database, except that you:

- put the settings within the **Administration** section
- provide a different `ConnectionString` and/or `EntryCollection`, e.g.:

```
"Administration": {
  "Repository": "MongoDB",
  "MongoDbOptions": {
    "ConnectionString": "mongodb://localhost/vonkstu3",
    "EntryCollection": "vonkadmin",
    "SimulateTransactions": "false"
  }
}
```

- Find the section called **PipelineOptions**. Make sure it contains the MongoDB repository in the administration path for Firely Server Administration:

```
"PipelineOptions" :
{
  "Branches" : [
    "/administration" : {
      "Include" : [
        "Vonk.Repository.MongoDb.MongoDbAdministrationConfiguration"
        //...
      ]
    }
  ]
}
```

Attention: For MongoDB it is essential to retain the `.vonk-import-history.json` file. Please read [Retain the import history](#) for details.

7.7.2 Tips and hints for using MongoDB for Firely Server

1. If searches and/or creates and updates are excessively slow, you may be limited by the IOPS on your MongoDB deployment (e.g. MongoDB Atlas). Try upgrading it and check the timings again.
2. If for any reason you would like to see how Firely Server is interacting with MongoDB, make the following adjustments to the [Log settings](#):
 1. In the section `Serilog.MinimumLevel.Override` add `"Vonk.Repository.DocumentDb": "Verbose"`. Add it before any broader namespaces like `Vonk`.
 2. In the section on the File sink, change the `restrictedToMinimumLevel` to `Verbose`.
3. **With regards to Firely Server version and MongoDB version:**
 1. If you are on a Firely Server (Vonk) version < v3.6, you can keep using MongoDB v4.0 or higher.

2. If you are on Firely Server (Vonk) v3.6 or higher and are unable to migrate to MongoDB 4.4 (relatively soon), please contact us if you need assistance.

7.8 Using SQL server

There are two ways to create the Firely Server database on a SQL Server instance: Have Firely Server create it for you entirely or create an empty database and users yourself and have Firely Server create the schema (tables etc.).

In both cases:

- Prepare an instance of SQL Server 2012 or newer. Any edition - including SQL Server Express - will do. The instance will have a servername and possibly an instancename: `server/instance`.
- Changing a setting means overriding it as described in *Changing the settings*.
- Find the Repository setting:

```
"Repository": "SQLite",
```

- Change the setting to SQL
- Find the section called SqlDbOptions. It has these values by default:

```
"SqlDbOptions": {
  "ConnectionString": "connectionstring to your Firely Server SQL Server database_
↳(SQL2012 or newer); Set MultipleActiveResultSets=True",
  "SchemaName": "vonk",
  "AutoUpdateDatabase": true,
  "MigrationTimeout": 1800 // in seconds
  //"AutoUpdateConnectionString" : "set this to the same database as
↳'ConnectionString' but with credentials that can alter the database. If not set,
↳defaults to the value of 'ConnectionString'"
},
```

- Find the section called PipelineOptions. Make sure it contains the SQL repository in the root path for Firely Server Data. For Firely Server versions older than 4.3.0 use `Vonk.Repository.Sql.SqlVonkConfiguration`. For Firely Server v4.3.0 and above use `Vonk.Repository.Sql.Raw.KSearchConfiguration`:

```
"PipelineOptions" :
{
  "Branches" : [
    "/" : {
      "Include" : [
        //"Vonk.Repository.Sql.SqlVonkConfiguration", // use this for FS_
↳versions < v4.3.0
        "Vonk.Repository.Sql.Raw.KSearchConfiguration", // use this for FS_
↳versions >= v4.3.0
        //...
      ]
    }
  ]
}
```

- The site connectionstrings.com is useful for determining the correct connectionstring for your environment.

- If you will only use Windows Accounts, you can use the (default) Authentication Mode, which is Windows Authentication Mode. But if you also want to use SQL Server accounts, you have to run it in Mixed Mode. Refer to [Authentication in SQL Server](#) for more information.
- Although we encourage you to use *SQLite for Firely Server Administration*, you can still use SQL Server for Firely Server Administration as well:

```
"Administration": {
  "Repository": "SQL",
  "SqlDbOptions": {
    "ConnectionString": "Integrated Security=SSPI;Persist Security Info=False;
↪Initial Catalog=VonkAdmin;Data Source=Server\\Instance;
↪MultipleActiveResultSets=true",
    "SchemaName": "vonk",
    "AutoUpdateDatabase": true,
    "MigrationTimeout": 1800 // in seconds
  }
},
//...
"PipelineOptions" :
{
  "Branches" : [
    "/administration" : {
      "Include" : [
        //"Vonk.Repository.Sqlite.SqliteTaskConfiguration", // use this for_
↪FS versions < v4.3.0
        "Vonk.Repository.Sql.Raw.KAdminSearchConfiguration", // use this_
↪for FS versions >= v4.3.0
        //...
      ]
    }
  ]
}
```

7.8.1 Have Firely Server create your database

This option is mainly for experimentation as it effectively requires sysadmin privileges for the connecting user.

- Prepare a login on SQL Server with the following role:
 - sysadmin
- Set the SqlDbOptions for the Firely Server database as follows (the values are example values for connecting with your own Windows login):

```
"SqlDbOptions": {
  "ConnectionString": "Integrated Security=SSPI;Persist Security Info=False;
↪Initial Catalog=VonkData;Data Source=Server\\Instance;MultipleActiveResultSets=true
↪",
  "SchemaName": "vonk",
  "AutoUpdateDatabase": true,
  "MigrationTimeout": 1800 // in seconds
},
```

- Set the SqlDbOptions under Administration for the Administration database likewise:

```
"Administration": {
  "Repository": "SQL",
  "SqlDbOptions": {
    "ConnectionString": "Integrated Security=SSPI;Persist Security Info=False;
↪Initial Catalog=VonkAdmin;Data Source=Server\\Instance;
↪MultipleActiveResultSets=true",
    "SchemaName": "vonk",
    "AutoUpdateDatabase": true,
    "MigrationTimeout": 1800 // in seconds
  }
}
```

- You don't need to set `AutoUpdateConnectionString` since the `ConnectionString` will already have enough permissions.
- Start Firely Server. It will display in its log that it applied pending migrations. After that the database is created and set up with the correct schema.
- If an upgrade to a new version of Firely requires a migration then a SQL time out might occur, halting the upgrade and resulting in a rollback of the migration. The duration of the SQL time out for migrations can be controlled with `MigrationTimeout`. The default value is 1800 seconds (30 min).

Attention: For SQL Server it is essential to retain the `.vonk-import-history.json` file. Please read [Retain the import history](#) for details.

7.8.2 Create a database and users by script, and have Firely Server create the schema

- Log into SQL Server as the Administrator user.
- From the working directory open `data01-CreateDatabases.sql`
- In SQL Server Management Studio, in the menu select Query|SQLCMD Mode.
- In the script uncomment and adjust the variable names `dbName` and `AdminDbName` as well as any other variables to your own liking.
- Run the script to create both the Firely Server database and the Administration API database.
- From the working directory open `data02-CreatedBUser.sql`
- In SQL Server Management Studio, in the menu select Query|SQLCMD Mode.
- In the script uncomment and adjust the variables at the top names to your own liking.
- Run the script to create two users, one with access to the Firely Server database, the other with access to the Administration database. This script grants the database role `db_ddladmin` to both users, to enable the `AutoUpdateDatabase` feature. Refer to [Overview of permissions](#) for an overview of necessary authorization for different features.
- Set the `SqlDbOptions` for the Firely Server database as follows:

```
"SqlDbOptions": {
  "ConnectionString": "User Id=<dbUserName>;Password=<dbPassword>;Initial Catalog=
↪<DataDbName>;Data Source=server\\instance;MultipleActiveResultSets=True",
  "SchemaName": "vonk",
}
```

(continues on next page)

(continued from previous page)

```
"AutoUpdateDatabase": "true"
}
```

- If you have set up a different user for running the AutoUpdateDatabase feature, you can provide that:

```
"SqlDbOptions": {
  "ConnectionString": "User Id=<dbUserName>;Password=<dbPassword>;Initial Catalog=
↳<DataDbName>;Data Source=server\\instance;MultipleActiveResultSets=True",
  "SchemaName": "vonk",
  "AutoUpdateDatabase": "true"
  "AutoUpdateConnectionString": "User Id=<updateUserName>;Password=
↳<updatePassword>;Initial Catalog=<DataDbName>;Data Source=server\\instance;
↳MultipleActiveResultSets=True",
}
```

- Set the SqlDbOptions under Administration for the Administration database likewise:

```
"Administration" : {
  "Repository": "SQL",
  "SqlDbOptions": {
    "ConnectionString": "User Id=<AdminDbUserName>;Password=<AdminDbPassword>;
↳Initial Catalog=<AdminDbName>;Data Source=server\\instance;
↳MultipleActiveResultSets=True",
    "SchemaName": "vonk",
    "AutoUpdateDatabase": "true"
  }
}
```

- For the administration you can also provide different credentials for performing the auto update:

```
"Administration" : {
  "Repository": "SQL",
  "SqlDbOptions": {
    "ConnectionString": "User Id=<AdminDUserName>;Password=<AdminDbPassword>;
↳Initial Catalog=<AdminDbName>;Data Source=server\\instance;
↳MultipleActiveResultSets=True",
    "SchemaName": "vonk",
    "AutoUpdateDatabase": "true"
    "AutoUpdateConnectionString": "User Id=<updateAdminUserName>;Password=
↳<updateAdminPassword>;Initial Catalog=<AdminDbName>;Data Source=server\\instance;
↳MultipleActiveResultSets=True",
  }
}
```

7.8.3 Overview of permissions

This paragraph lists the permissions needed to perform specific actions on the SQL database.

- To run the AutoUpdateDatabase feature, including creation of the databases:
 - **sysadmin**
- To run the AutoUpdateDatabase feature on an already created (but empty) database:
 - **db_ddladmin** (both for the normal Firely Server Data database and the Administration database)
- To read/write resources:
 - **db_datareader**
 - **db_datawriter**
- To execute the ResetDb feature:
 - **db_ddladmin** only on the normal Firely Server database for the user in the SqlConnectionstring. (no extra permissions are required for the user on the Administration database).

7.9 Using SQLite

SQLite is a file based database engine. The engine itself does not run separately on the server, but in-process in the application, Firely Server in this case.

For more background on SQLite please refer to the [SQLite documentation](#).

SQLite is the default configuration of Firely Server. For the Administration database there is little reason to change this. For the actual runtime data, (the 'Firely Server database') itself, you may run into limitations of SQLite if you put it through its paces. You may find one of the other repositories a better fit then. You can safely use different storage engines for Firely Server Data and Firely Server Administration.

7.9.1 Prefer SQLite for Firely Server Administration

Until Firely Server (Vonk) version 0.7.0 you could use any of the storage engines for both Firely Server Data and Firely Server Administration. Starting with Firely Server (Vonk) 0.7.1 you are encouraged to use SQLite for Firely Server Administration. Over time we will deprecate support for running Firely Server Administration on the SQL Server, MongoDB and Memory storage engines. For Firely Server Data you can of course still use the storage engine of your preference.

Firely Server Administration poses very limited stress on its storage engine, therefore SQLite is adequate. And it provides several advantages:

- Runs out of the box: SQLite requires no installation of a database engine, but still provides durable storage (unlike the Memory storage). Thus, you don't need to setup anything to run Firely Server Administration. And you can download the Firely Server binaries and run them without any further configuration.
- Flexible on updates: Many of the features that we will add to Firely Server require changes to the schema of the Administration database. By only supporting SQLite for this, we can provide these features to you more quickly.
- Readymade database: In the other storage engines, the conformance resources from the specification had to be *imported* before Firely Server could start. This would take a couple of minutes. Because SQLite is file based, we can run the import process for you and provide you with a readymade Administration database.

- Runs with Facades: perhaps the most important feature. If you build a Firely Server Facade, the facade will not provide support for hosting conformance resources. With Firely Server Administration on SQLite the facade has its own storage and you can use Firely Server Administration out of the box. This enables e.g. validation against your custom resources (that can be imported from your Simplifier project), subscriptions, and other use cases.

7.9.2 Settings for using SQLite for Firely Server Data

- Changing a setting means overriding it as described in *Changing the settings*.
- Find the Repository setting and set it to SQLite if it not already set to that:

```
"Repository": "SQLite",
```

- Find the section called SQLiteDbOptions. It has these values by default:

```
"SQLiteDbOptions": {
  "ConnectionString": "Data Source=./data/vonkdata.db",
  "AutoUpdateDatabase": true
},
```

Firely Server will create the database *file*, but please make sure the *directory* already exists.

- Find the section called PipelineOptions. Make sure it contains the SQLite repository in the root path:

```
"PipelineOptions" :
{
  "Branches" : [
    "/" : {
      "Include" : [
        "Vonk.Repository.SQLite.SqliteVonkConfiguration"
        //...
      ]
    },
    //...
  ]
}
```

7.9.3 Settings for using SQLite for Firely Server Administration

- Set the SqlDbOptions under Administration for the Administration database similar to those above:

```
"Administration" : {
  "Repository": "SQLite",
  "SQLiteDbOptions": {
    "ConnectionString": "Data Source=./data/vonkadmin.db",
    "AutoUpdateDatabase": "true"
  }
}
```

Firely Server will create the database *file*, but please make sure the *directory* already exists.

- Find the section called PipelineOptions. Make sure it contains the SQLite repository in the administration path:

```

"PipelineOptions" :
{
  "Branches" : [
    "/" : {
      //...
    },
    "/administration" : {
      "Include" : [
        "Vonk.Repository.SQLite.SqliteAdministrationConfiguration"
        //...
      ]
    }
  ]
}

```

7.9.4 Administration import history in SQLite

When Firely Server *imports Conformance resources*, it keeps record of what it has imported. Unlike the SQL Server and MongoDB engines, the SQLite storage engine does *not* use the `.vonk-import-history.json` file for that. Instead, in SQLite the import history is stored within the Administration database itself.

7.10 Using Microsoft Azure CosmosDB

You can connect Firely Server to CosmosDB the same way you connect to MongoDB. There are a few limitations that we will work out later. They are listed below.

Attention: You cannot use CosmosDb for the Firely Server Administration database. Use *SQLite* instead.

1. Create a CosmosDB account on Azure, see the [Quickstart Tutorial](#)
2. Make sure you choose the MongoDB API
3. In the Azure Portal, open your CosmosDB account and go to the 'Connection Strings' blade. Copy the 'Primary Connection String' to your clipboard.
4. Now on your own machine, navigate to your Firely Server working directory
5. Changing a setting means overriding it as described in [Changing the settings](#).
6. Find the Repository setting:

```
"Repository": "Sqlite",
```

7. Change the setting to CosmosDb
8. If you have your own database in CosmosDB already, change the CosmosDbOptions to reflect your settings:

```

"CosmosDbOptions": {
  "ConnectionString": "<see below>",
  "EntryCollection": "vonkentries",
  "SimulateTransactions": "false"
},

```

Paste the ConnectionString from step 3, and add the databasename that you want to use. The connectionstring looks like this:

```
mongodb://<accountname>:<somerandomstring>==@<accountname>.documents.azure.
↳com:10255?ssl=true&replicaSet=globaldb
```

You can add the databasename after the portnumber, like this:

```
mongodb://<accountname>:<somerandomstring>==@<accountname>.documents.azure.
↳com:10255/vonk?ssl=true&replicaSet=globaldb
```

9. If your CosmosDB account does not have a database or collection by this name, Firely Server will create it for you.
10. You can set SimulateTransactions to “true” if you want to experiment with [FHIR transactions](#). Firely Server does not utilize the CosmosDB way of supporting real transactions across documents, so in case of an error already processed entries will NOT be rolled back.

7.10.1 CosmosDB Request Units

If you upload a lot of data in a short time (as is done on [reindexing](#)), you quickly exceed the default maximum of 1000 Request Units / second. If you encounter its limits, the Firely Server log will contain errors stating ‘Request rate is large’. This is likely to happen upon [reindexing](#) or when using Vonkloader. Solutions are:

- Raise the limit to at least 5000 RU/s. See the [Microsoft documentation](#) for instructions.
- Lower the load
 - on Reindexing, lower the MaxDegreeOfParallelism, see [this warning](#)
 - with Vonkloader, lower the value of the -parallel parameter.

7.10.2 Limitations

1. Request size for insertions to CosmosDB is limited to around 5 MB. Some bundles in the examples from the specification exceed that limit. Then you will get an error stating ‘Request size too large’. You can avoid this by limiting the size of incoming resources in the [SizeLimits](#) setting.
2. The CosmosDB implementation of the MongoDB API is flawed on processing **\$not** on arrays. This inhibits the use of these searches in Firely Server:
 - Using the `:not` modifier
 - Using `:missing=true`

7.11 Configure http and https

You can enable http and/or https and adjust the port numbers for them in [Firely Server settings](#).

7.11.1 Changing the port number

By default Firely Server will run on port 4080 of your system. You can change the port setting by overriding it as described in *Changing the settings*:

- Navigate to your Firely Server working directory
- Find this setting:

```
"Hosting": {  
  "HttpPort": 4080  
}
```

- Change the number to the port number you want

7.11.2 Changing from http to https

If you need your server to run on https instead of http, follow these steps:

- Navigate to the location where you extracted the Firely Server files.
- Find these settings:

```
"Hosting": {  
  "HttpPort": 4080,  
  "HttpsPort": 4081, // Enable this to use https  
  "CertificateFile": "<your-certificate-file>.pfx", //Relevant when  
↪ HttpsPort is present  
  "CertificatePassword" : "<cert-pass>" // Relevant when HttpsPort is  
↪ present  
},
```

- Uncomment the lines for HttpsPort, CertificateFile and CertificatePassword.
- Set the HttpsPort to the port of your liking (standard https port is 443)
- Set CertificateFile to the location of the .pfx file that contains the certificate for your site
- Set CertificatePassword to the password for the certificate file.

Note: We recommend setting this value as an environment variable for security reasons:

```
VONK_Hosting:CertificatePassword=<password>
```

To set this:

- In Powershell run: `> $env:VONK_Hosting:CertificatePassword="my_password"` where *my_password* is the password for the .pfx file
- or go to your *System*, open the *Advanced system settings* → *Environment variables* and create a new variable with the name `VONK_Hosting:CertificatePassword` and the value set to your password
- You can choose to comment-out the `HttpPort` setting, so Firely Server will no longer be available through unsecured http.

7.12 Cross Origin Resource Sharing (CORS)

CORS is enabled in Firely Server. Since Firely Server provides an API that is expected to be consumed by applications from different domains, CORS is enabled for any origin. Currently there is no setting to control this behaviour.

7.13 Log settings

Firely Server uses [Serilog](#) for logging. The logging settings are controlled in json configuration files called `logsettings(*).json`. The files are read in a hierarchy, exactly like the [appsettings files](#) are. Firely Server comes with default settings in `logsettings.default.json`. You can adjust the way Firely Server logs its information by overriding these default settings in `logsettings.json` or `logsettings.instance.json`. You need to create this file yourself.

Alternatively you can control *Log settings with Environment Variables*.

Firely Server by default does not log any Patient Health Information data, regardless of the level of the log. The only PHI part that can be included in the log is the User Name, when running with Smart authorization (so the user is identified) and including this part in the outputTemplate (see below).

7.13.1 Changing the log event level

Serilog defines several levels of log events. From low to high, these are Verbose, Debug, Information, Warning, Error and Fatal. You can set the minimum level you want to log, meaning that events for that level or higher will be logged. By default, Firely Server uses Error as the minimum level of recording information.

To change the level of logging, follow these steps:

- Find this setting:

```
"MinimumLevel": {
  "Default": "Error",
},
```

- Change the setting for Default from Error to the level you need, from the choice of Verbose, Debug, Information, Warning, Error and Fatal.

You can deviate from the default minimum level for specific namespaces. You do this by specifying the namespace and the log event level you would like for this namespace, for example:

```
"MinimumLevel": {
  "Default": "Error",
  "Override": {
    "Vonk": "Warning"
  }
},
```

Some additional namespaces you might want to log are:

- `Vonk.Configuration` to log configuration information on startup
- `Vonk.Core.Licensing` to show license information in your logs
- `Vonk.Repository.EntityFrameworkCore`, `Vonk.Repository.DocumentDb` or `Vonk.Repository.Memory` to log repository events

- `Vonk.Core.Repository.EntryIndexerContext`, set it to "Error" if you have excessive warnings about indexing (mostly when importing *Synthea* <<https://synthea.mitre.org/downloads>> data)
- Microsoft to log events from the Microsoft libraries
- `Microsoft.AspNetCore.Diagnostics` to report request handling times
- System to log events from the System libraries

Please note that the namespaces are evaluated in order from top to bottom, so more generic 'catch all' namespaces should be at the bottom of the list. So this will log events on `Vonk.Repository.Sql` on Information level:

```
"MinimumLevel": {
  "Default": "Error",
  "Override": {
    "Vonk.Repository.Sql": "Information",
    "Vonk": "Warning"
  }
},
```

But in this (purposefully incorrect) example the Warning level on the Vonk namespace will override the Information level on the `Vonk.Repository.Sql` namespace:

```
"MinimumLevel": {
  "Default": "Error",
  "Override": {
    "Vonk": "Warning",
    "Vonk.Repository.Sql": "Information"
  }
},
```

7.13.2 Changing the sink

Another setting you can adjust is `WriteTo`. This tells Serilog which sink(s) to log to. Serilog provides several sinks, and for Firely Server you can use `Console`, `File`, `ApplicationInsights` and `Seq`. All of which can be wrapped in an `Async` sink to avoid blocking Firely Server when waiting for the sink to process the log statements.

Console

The Console sink will write to your shell.

- Find the `WriteTo` setting:

```
"WriteTo": [
  {
    "Name": "Async",
    "Args": {
      "configure": [
        {
          "Name": "Console",
          "Args": {
            "restrictedToMinimumLevel": "Information",
            "outputTemplate": "{Timestamp:yyyy-MM-dd,
↵HH:mm:ss.fff zzz} {UserId} {Username} [{Level}] [ReqId: {RequestId}] {Message}"

```

(continues on next page)

(continued from previous page)

```

↪{NewLine}{Exception}"
                                }
                                }
                        ]
                }
        },
        {
            //Settings for other sinks
        }

```

The Console is notoriously slow at processing log statements, so it is recommended to limit the number of statements for this sink. Use the `restrictedToMinimumLevel` to do so. Also, if you are on Windows, the Powershell command window appears to be faster than the traditional Command Line window.

Settings for the Console sink:

- **outputTemplate**: What information will be in each log line. Besides regular text you can use placeholders for information from the log statement:
 - `{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz}`: When this was logged, with formatting
 - `{UserId}`: Technical id of the logged in user - if applicable
 - `{Username}`: Name of the logged in user - if applicable
 - `{Application}`: Name of the application (in case other applications are logging to the same sink). Is set to Vonk at the bottom of the logsettings file
 - `{Level}`: Level of the log, see the values in [Changing the log event level](#)
 - `{MachineName}`: Name of the machine hosting the Firely Server instance. Especially useful when running multiple instances all logging to the same file.
 - `{RequestId}`: Unique id of the web request, useful to correlate log statements
 - `{Message}`: Actual message being logged
 - `{Exception}`: If an error is logged, Firely Server may include the original exception. That is then formatted here.
 - `{SourceContext}`: The class from which the log statement originated (this is usually not needed by end users).
 - `{NewLine}`: Well, eh, continue on the next line
- **restrictedToMinimumLevel**: Only log messages from this level up are sent to this sink.

File

The File sink will write to a file, possibly rolling it by interval or size.

- Find the `WriteTo` setting:

```

"WriteTo": [
    {
        {
            //Settings for Console
        }
    },

```

(continues on next page)

(continued from previous page)

```

{
  "Name": "Async",
  "Args": {
    "configure": [
      {
        "Name": "File",
        "Args": {
          "path": "%temp%/vonk.log",
          "rollingInterval": "Day",
          "fileSizeLimitBytes": "",
          "retainedFileCountLimit": "7",
          "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.
↪fff zzz} {UserId} {Username} [{Application}] [{Level}] [Machine: {MachineName}]{
↪[ReqId: {RequestId}] {Message}{NewLine}{Exception}",
          "restrictedToMinimumLevel": "Verbose"
        }
      }
    ]
  },
  {
    //Settings for Azure ApplicationInsights
  }
}

```

- Under File, change the location of the logfiles by editing the value for path. For example:

```

{
  "Name": "RollingFile",
  "Args": {
    "path": "c:/logfiles/vonk.log"
  }
},

```

Other values that you can set for the File log are:

- **rollingInterval**: When this interval expires, the log system will start a new file. The start datetime of each interval is added to the filename. Valid values are *Infinite*, *Year*, *Month*, *Day*, *Hour*, *Minute*.
- **fileSizeLimitBytes**: Limit the size of the log file, which is 1GB by default. When it is full, the log system will start a new file.
- **retainedFileCountLimit**: If more than this number of log files is written, the oldest will be deleted. Default value is 31. Explicitly setting it to an empty value means files are never deleted.
- **outputTemplate**: as described for *Console*.
- **restrictedToMinimumLevel**: as described for *Console*.

Application Insights

Firely Server can also log to Azure Application Insights (“Application Insights Telemetry”). What you need to do:

1. Create an Application Insights instance on Azure.
2. Get the InstrumentationKey from the Properties blade of this instance.
3. Add the correct sink to the logsettings.json:

```
"WriteTo": [
  {
    "Name": "ApplicationInsightsTraces",
    "Args": {
      "instrumentationKey": "<the key you copied in step 2>",
      "restrictedToMinimumLevel": "Verbose" //Or a higher level
    }
  },
],
```

4. This also enables Dependency Tracking for access to your database. This works for both SQL Server and MongoDB. And for the log sent to [Seq](#) if you enabled that.
5. If you set the level for Application Insights to **Verbose**, and combine that with [Database details](#), you get all the database commands right into Application Insights.

Seq

[Seq](#) is a web interface to easily inspect structured logs.

For the Seq sink, you can also specify arguments. One of them is the server URL for your Seq server:

```
"WriteTo": [
  {
    "Name": "Seq",
    "Args": { "serverUrl": "http://localhost:5341" }
  }
],
```

- Change `serverUrl` to the URL of your Seq server
- `restrictedToMinimumLevel`: as described for [Console](#).

7.13.3 Database details

Whether you use MongoDB or SQL Server, you can have Firely Server log in detail what happens towards your database. Just set the appropriate loglevel to ‘Verbose’:

```
"MinimumLevel": {
  "Default": "Error",
  "Override": {
    "Vonk.Repository.EntityFrameworkCore": "Verbose",
    "Vonk.Repository.DocumentDb": "Verbose",
    "Vonk": "Warning"
  }
},
```

If you do so you probably don't want all this detail in your console sink, so you can limit the level for that, see [Console](#) above.

FIRELY SERVER DEPLOYMENT OPTIONS

Firely Server can be deployed as:

- Binaries, see *Getting Started*
- As Docker image, see *Using Firely Server on Docker*
- With *Yellow Button - Firely Server for your Simplifier project*

The Binaries can be deployed on your own machines and on cloud services. For deployment on Azure we included *instructions*.

In order to test the *Access control and SMART* you will need to *Set up an Identity Provider*.

8.1 Using Firely Server on Docker

We have created a Docker image for Firely Server, so you can run the server in any environment that supports Docker. For this section we assume you have Docker installed on your system. If you want to install Docker for Windows, please read *docker_win* for specific installation details.

8.1.1 Getting started

Before you can run Firely Server, you will need to pull the Docker Firely Server container and request a license.

1. Open your favourite command line tool and execute this command: `> docker pull simplifier/vonk`
2. Go to the [Simplifier website](#), login and download your evaluation license.
3. Create a working directory for Firely Server and place the license file there.

Warning: If you use Docker, you may want to run multiple instances of Firely Server (e.g. with Kubernetes). Read *Running imports with multiple instances* for caveats with the Administration endpoint.

8.1.2 Running a Docker Firely Server in SQLite mode

The easiest and the default way to run a Docker Firely Server container is to run Firely Server in SQLite repository mode. Note that this is not the most performant mode - see MongoDB and SQL Server options below for that.

Open your command prompt and execute this command: `> docker images simplifier/vonk`

You will get a list that looks like:

```
PS C:\Data\projects\FHIR\FirelyServer> docker images simplifier/vonk
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
simplifier/vonk     latest         9c432eb5c983   13 months ago  466MB
PS C:\Data\projects\FHIR\FirelyServer>
```

Navigate to your working directory for Firely Server and run the container with this command:

- in cmd.exe: `docker run -d -p 8080:4080 --name vonk.server -v %CD%/firelyserver-license.json:/app/firelyserver-license.json simplifier/vonk`
- in Powershell: `docker run -d -p 8080:4080 --name vonk.server -v ${PWD}/firelyserver-license.json:/app/firelyserver-license.json simplifier/vonk`

If your license file has a different name, use that name instead of `firelyserver-license` on the left side of the `-v` parameter in the command above. E.g. `-v ${PWD}/my-fancy-license.json:/app/firelyserver-license.json`

This will spin up a Firely Server container. It maps the host port 8080 to the container port 4080 with the switch `-p 8080:4080`. It will give the container the name `vonk.server` with the switch `--name vonk.server`. Furthermore it mounts the local licensefile into the `/app` directory, which is the directory where Firely Server resides in the container. Finally it will run the container in background mode with the switch `-d`.

To test whether the container is running correctly, type the command: `> docker ps`

```
PS C:\run\vonk> docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED        STATUS        PORTS                    NAMES
aa709bb96599  simplifier/vonk  "dotnet Vonk.Serve..."  8 seconds ago  Up 7 seconds  0.0.0.0:8080->4080/tcp  vonk.server
PS C:\run\vonk>
```

You can also take a look at the logs for Firely Server with: `> docker logs vonk.server`

Open a browser and use the address `http://localhost:8080/`. This will show the landing page of Firely Server.

To stop the container just type: `> docker stop vonk.server` And to start it again: `> docker start vonk.server` To completely remove the container: `> docker rm vonk.server`

Adjust settings when running in a Docker container

In the paragraph above we showed how to use `-v` to mount the license file into the app directory. We can use the same technique to mount a custom settings file or logsettings file. Or a directory to host the SQLite database outside of the container. Or have the logfile written to your working directory. Below is a full example using PowerShell. For creating `appsettings.instance.json` refer to [Firely Server settings](#). For creating `logsettings.instance.json` refer to [Log settings](#).

appsettings.instance.json

```
{
  "SQLiteDbOptions": {
    "ConnectionString": "Data Source=./resourcedata/vonkdata.db;Cache=Shared",
    "AutoUpdateDatabase": true
  }
}
```

logsettings.instance.json

Settings for console and levels not included for brevity.

```
{
  "Serilog": {
    "WriteTo": [
      {
        "Name": "Async",
        "Args": {
          "configure": [
            {
              "Name": "File",
              "Args": {
                "path": "./log/vonk.log",
                "rollingInterval": "Day",
                "fileSizeLimitBytes": "",
                "retainedFileCountLimit": "7",
                "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz}
→{UserId} {Username} [{Application}] [{Level}] [Machine: {MachineName}] [ReqId:
→{RequestId}] {Message}{NewLine}{Exception}",
                "restrictedToMinimumLevel": "Information"
              }
            }
          ]
        }
      }
    ],
    "Enrich": [ "FromLogContext", "WithMachineName", "WithThreadId" ],
    "Properties": {
      "Application": "Firely Server",
      "Environment": "Default"
    }
  }
}
```

Powershell

```
mkdir logs
mkdir resourcedata //do not use 'data' - the administration database is already in that
↳ folder in the container
//create the appsettings.instance.json above
//create the logsettings.instance.json above

docker run -d -p 8080:4080 --name firely.server `
-v ${PWD}/firelyserver-license.json:/app/firelyserver-license.json `
-v ${PWD}/appsettings.instance.json:/app/appsettings.instance.json `
-v ${PWD}/logsettings.instance.json:/app/logsettings.instance.json `
-v ${PWD}/resourcedata:/app/resourcedata `
-v ${PWD}/log:/app/log `
simplifier/vonk:4.0.0
```

You should see a vonkdata.db appear in the ./resourcedata folder, and a log file in the ./log folder. From here you can experiment with other settings. You can also easily keep different settings files side-by-side, mapping the one you want to test into the container, e.g. `-v ${PWD}/some-weird-settings.json:/app/appsettings.instance.json`.

Spinning up with a docker-compose file

Another way to spin up a Firely Server container is to use a docker-compose file. The above example can also be established by the following docker-compose-sqlite.yml:

```
1 version: '3'
2
3 services:
4
5   vonk-web:
6     image: simplifier/vonk
7     ports:
8       - "8080:4080"
9     environment:
10      - VONK_Repository=SQLite
11      - VONK_Administration:Repository=SQLite
12      - VONK_License:LicenseFile=./license/firelyserver-trial-license.json
13     volumes:
14      - ./app/license
```

Save the text above to a file in your working directory with the name `docker-compose.sqlite.yml` and then run the following command: `> docker-compose -f docker-compose.sqlite.yml up -d`

If your license file has a different name, use that name instead of `firelyserver-trial-license` in the text above - but make sure to keep `./license` as that maps to a Docker volume inside the container.

```
PS C:\run\vonk> docker-compose -f docker-compose.memory.yml up -d
Creating vonk_vonk-web_1
PS C:\run\vonk> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
2c99a0648c3e   simplifier/vonk   "dotnet Vonk.Serve..."   5 seconds ago   Up 4 seconds   0.0.0.0:8080->4080/tcp    vonk_vonk-web_1
PS C:\run\vonk>
```

To stop the container, run: `> docker-compose -f docker-compose.sqlite.yml down`

Note: Strictly the settings for `VONK_Repository` and `VONK_Administration` are not needed here, since SQLite is the default setting. The settings are included to show where to configure the type of database to use. Much more information on that topic is in the paragraphs below.

8.1.3 Running Docker with a SQL Server container

Firely Server can use also other repositories than Memory, for example SQL Server. This section describes how to spin up a Firely Server container and a SQL Server container. We will use docker-compose to achieve this.

Warning: SQL Server container uses at least 3.25 GB of RAM. Make sure to assign enough memory to the Docker VM if you're running on Docker for Mac or Windows.

Warning: If you also run the Administration database on SQL Server, please read *Retain the import history*.

```

1 version: '3'
2
3 services:
4   vonk-web:
5     image: simplifier/vonk
6     ports:
7       - "8080:4080"
8     depends_on:
9       - vonk-sqlserver-db
10    environment:
11      - VONK_Repository=SQL
12      - VONK_SqlDbOptions:ConnectionString=Initial Catalog=VonkStu3;Data Source=vonk-
13      ↳sqlserver-db,1433;User ID=sa;Password=SQLServerStrong(!)Password*
14      - VONK_SqlDbOptions:SchemaName=vonk
15      - VONK_SqlDbOptions:AutoUpdateDatabase=true
16      - VONK_SqlDbOptions:AutoUpdateConnectionString=Initial Catalog=VonkStu3;Data
17      ↳Source=vonk-sqlserver-db,1433;User ID=sa;Password=SQLServerStrong(!)Password*
18      - VONK_Administration:Repository=SQL
19      - VONK_Administration:SqlDbOptions:ConnectionString=Initial Catalog=VonkAdmin;Data
20      ↳Source=vonk-sqlserver-db,1433;User ID=sa;Password=SQLServerStrong(!)Password*
21      - VONK_Administration:SqlDbOptions:SchemaName=vonkadmin
22      - VONK_Administration:SqlDbOptions:AutoUpdateDatabase=true
23      - VONK_Administration:SqlDbOptions:AutoUpdateConnectionString=Initial
24      ↳Catalog=VonkAdmin;Data Source=vonk-sqlserver-db,1433;User ID=sa;
25      ↳Password=SQLServerStrong(!)Password*
26      - VONK_License:LicenseFile=./license/firelyserver-trial-license.json
27    volumes:
28      - ./app/license
29
30   vonk-sqlserver-db:
31     image: microsoft/mssql-server-linux
32     ports:
33       - "1433:1433"

```

(continues on next page)

(continued from previous page)

```

29  environment:
30      - ACCEPT_EULA=Y
31      - SA_PASSWORD=SQLServerStrong(!)Password*
32  healthcheck:
33      test: /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P 'SQLServerStrong(!
↪)Password*' -Q 'SELECT 1 FROM VonkSTU3.sys.tables'
34      interval: 1m30s
35      timeout: 10s
36      retries: 3

```

Save the text above to a file in your working directory with the name `docker-compose.mssqlserver.yml`. Make sure your Firely Server license file is named `firelyserver-trial-license.json` and is residing in your working directory (see [Getting started](#) on how to obtain the license), **not** in a subdirectory named `license` (that is an internal directory inside the container). If your license file has a different name, use that name instead of `firelyserver-trial-license` in the text above.

Then use this command to spin up a Firely Server container and SQL container: `> docker-compose -f docker-compose.mssqlserver.yml up -d`

Open a browser and use the address `http://localhost:8080/`. This will show the landing page of Firely Server.

Warning: Wait for about 2 minutes, because it takes a while to fire up the SQL container

8.1.4 Running Docker with a SQL Server on host

Another possibility is to run a Firely Server container with a SQL Server repository on the host. You will need a Microsoft SQL Server running on your host. The version of SQL Server must at least be version 2012.

Warning: If you also run the Administration database on SQL Server, please read [Retain the import history](#).

To run the Firely Server container we will use the following docker-compose file:

```

1  version: '3'
2
3  services:
4
5      vonk-web:
6          image: simplifier/vonk
7          ports:
8              - "8080:4080"
9          environment:
10             - VONK_Repository=SQL
11             - VONK_SqlDbOptions:ConnectionString=Database=VonkStu3;Server=my_host\
↪<myInstanceName>;User ID=<myUser>;Password=<myPassword>
12             - VONK_SqlDbOptions:SchemaName=vonk
13             - VONK_SqlDbOptions:AutoUpdateDatabase=true
14             - VONK_SqlDbOptions:AutoUpdateConnectionString=Database=VonkStu3;Server=my_host\
↪<myInstanceName>;User ID=<DLLUser>;Password=<myPassword>
15             - VONK_Administration:Repository=SQL

```

(continues on next page)

(continued from previous page)

```

16     - VONK_Administration:SqlDbOptions:ConnectionString=Database=VonkAdmin;Server=my_
17 ↪host\<myInstanceName>;User ID=<myUser>;Password=<myPassword>
18     - VONK_Administration:SqlDbOptions:SchemaName=vonkadmin
19     - VONK_Administration:SqlDbOptions:AutoUpdateDatabase=true
20     - VONK_Administration:SqlDbOptions:AutoUpdateConnectionString=Database=VonkAdmin;
21 ↪Server=my_host\<myInstanceName>;User ID=<DLLUser>;Password=<myPassword>
22     - VONK_License:LicenseFile=./license/firelyserver-trial-license.json
23     volumes:
24     - ./app/license
25     extra_hosts:
26     - "my_host:192.0.2.1"

```

Save the text above to a file in your working directory with the name `docker-compose.mssqlserver_host.yml`. Before we spin up the container we have to adjust the `docker-compose.mssqlserver_host.yml`:

- On lines 11, 14, 16 and 19 the connection string to the database server is stated. Change the `Server` to your database server and instance name.
- Also change the `User ID` and `Password` on lines 11, 14, 16 and 19 to your credentials.
- Furthermore we have to tell Docker which IP address the host uses. This is done on line 24. In this case the host (named `my_host`) uses IP address 192.0.2.1. Change this to the appropriate address.

After saving your settings, make sure your Firely Server license file is named `firelyserver-trial-license.json` and is residing in your working directory (see [Getting started](#) on how to obtain the license), **not** in a subdirectory named `license` (that is an internal directory inside the container). Or use the name of your license file instead of `firelyserver-trial-license` in the text above.

You can run the Firely Server container as follows: `> docker-compose -f docker-compose.mssqlserver_host.yml up -d`

A database will automatically be created if it is not already present on the database server. See [this page](#) for an overview of permissions the database user needs for creating the database and/or schema.

Open a browser and use the address <http://localhost:8080/>. This will show the landing page of Firely Server.

Warning: When you have a firewall installed on your host machine, it can block traffic from your Firely Server container to your host. Provide an inbound rule to allow traffic from the container to the host.

8.1.5 Run Docker with a MongoDB container

This section describes how to spin up a Firely Server container and a MongoDB container using a `docker-compose`. We assume you already have MongoDB installed.

Warning: If you also run the Administration database on MongoDB, please read [Retain the import history](#).

```

1 version: '3'
2
3 services:
4
5     vonk-web:

```

(continues on next page)

(continued from previous page)

```

6  image: simplifier/vonk
7  environment:
8      - VONK_Repository=MongoDb
9      - VONK_MongoDbOptions:ConnectionString=mongodb://vonk-mongo-db/vonkstu3
10     - VONK_MongoDbOptions:EntryCollection=vonkentries
11     - VONK_Administration:Repository=MongoDb
12     - VONK_Administration:MongoDbOptions:ConnectionString=mongodb://vonk-mongo-db/
    ↪vonkadmin
13     - VONK_Administration:MongoDbOptions:EntryCollection=vonkentries
14     - VONK_License:LicenseFile=./license/firelyserver-trial-license.json
15  volumes:
16     - ./app/license
17  ports:
18     - "8080:4080"
19
20  vonk-mongo-db:
21  image: mongo

```

Save the text above to a file in your working directory with the name `docker-compose.mongodb.yml`. Make sure your Firely Server license file is named `firelyserver-trial-license.json` and is residing in your working directory (see [Getting started](#) on how to obtain the license), **not** in a subdirectory named `license` (that is an internal directory inside the container). If your license file has a different name, use that name instead of `firelyserver-trial-license` in the text above.

Use this command to spin up a Firely Server container and MongoDB container: `> docker-compose -f docker-compose.mongodb.yml up -d`

Open a browser and use the address <http://localhost:8080/>. This will show the landing page of Firely Server.

8.2 Firely Server deployment on Azure Web App Service

In this section we explain how you can deploy Firely Server in the Azure cloud.


8.2.1 Getting started

Before you can run Firely Server, you will need to download the Firely Server binaries and request a license:

1. Go to the Simplifier website, login and download the Firely Server binaries from <https://simplifier.net/vonk/download>
2. Download the trial license file from the same location.

8.2.2 Deployment

1. Go to Azure (<https://portal.azure.com>) and create a web app:









Web App
Microsoft

Create and deploy web sites in seconds, as powerful as you need them


Leverage your existing tools to create and deploy applications without the hassle of managing infrastructure. Microsoft Azure Web Sites offers secure and flexible development, deployment, and scaling options for any sized web application. Use frameworks and templates to create web sites in seconds. Choose from source control options like TFS, GitHub, and BitBucket. Use any tool or OS to develop your site with .NET, PHP, Node.js or Python.

- Fastest way to build for the cloud
- Provision and deploy fast
- Secure platform that scales automatically
- Great experience for Visual Studio developers
- Open and flexible for everyone
- Monitor, alert, and auto scale (preview)

MyTestGroup

Summary



MyTestGroup

Monitoring

Events in the past week

Alert rules


1 rule(s)

Billing

Resource Costs

mytestsite8

Summary



MyTestGroup

Monitoring

Requests and errors today

0

0

PUBLISHER

Microsoft

Create

- Choose a name for the webapp, we will use the placeholder <webapp>. Fill in an existing resource group or


112

Chapter 8. Firely Server deployment options

create a new one and select Windows for the operation system (OS):





Create Web App

Basics Monitoring Tags Review + create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#) 



Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * 	<div>Visual Studio Professional Subscription </div>
Resource Group * 	<div>(New) firelyserver </div>

[Create new](#)

Instance Details

Name *	<div>firelyserver </div> <small>.azurewebsites.net</small>
Publish *	<input checked="" type="radio"/> Code <input type="radio"/> Docker Container
Runtime stack *	<div>.NET Core 3.1 (LTS) </div>
Operating System *	<input type="radio"/> Linux <input checked="" type="radio"/> Windows

- Add the trial license file (firelyserver-trial-license.json) to the vonk_distribution.zip by dragging the license file into the zipfile.
- Open a webbrowser, navigate to `https://<webapp>.scm.azurewebsites.net/ZipDeployUI` and drag vonk_distribution.zip into the browser window. This will install the Firely Server as a Web App in Azure. In our example the url is `https://<webapp>.scm.azurewebsites.net/ZipDeployUI` This method of deployment does not work in Internet Explorer. It does work in Firefox, Chrome and Edge.
- Open a browser and go to the site `https://<webapp>.azurewebsites.net/` . This will show the Firely Server home page.

8.2.3 Change database

In this example Firely Server is using a memory repository. If you want to change it to another kind of repository then you could change that on the page Application Settings of the Web App. Here you can set *Environment Variables* with the settings for either *SQL Server* or *MongoDB*. For example for MongoDB it will look like this:

The screenshot shows the 'Application settings' page in the Azure portal. The left sidebar contains navigation links: Tags, Diagnose and solve problems, DEPLOYMENT (Quickstart, Deployment credentials, Deployment slots, Deployment options, Continuous Delivery (Preview)), and SETTINGS (Application settings, Authentication / Authorization, Managed service identity, Backups, Custom domains). The 'Application settings' section is active, showing a list of settings:

Setting Name	Value	Slot Setting	Reset
WEBSITE_NODE_DEFAULT_VERSION	6.9.1	<input type="checkbox"/>	✕
VONKLOG_SerilogWriteTo1:ArgspathFormat	./logs/vonk-{Date}.log	<input type="checkbox"/>	✕
VONK_Repository	MongoDb	<input type="checkbox"/>	✕
VONK_MongoDbOptions:ConnectionString	mongodb://vonk:<password>@mongodb.server.example:27017/vonkadmin?ssl=true&authSource=admin	<input type="checkbox"/>	✕
VONK_Administration:Repository	MongoDb	<input type="checkbox"/>	✕
VONK_Administration:MongoDbOptions:ConnectionString	mongodb://vonk:<password>@mongodb.server.example:27017/vonk?ssl=true&authSource=admin	<input type="checkbox"/>	✕

Below the table is a link: + Add new setting.

8.2.4 More information

About Azure zip deployment: <https://docs.microsoft.com/en-us/azure/app-service/app-service-deploy-zip#deploy-zip-file>

8.3 Yellow Button - Firely Server for your Simplifier project

Yellow Button is an easy way to run an instance of Firely Server that is aware of all the profiles, search parameters and other (conformance) resources that are part of a Simplifier project. That can be a project of your own or for instance a project with national profiles.

8.3.1 Prerequisites

Yellow Button provides you with a PowerShell script to start Firely Server in a Docker container. This means that you will need:

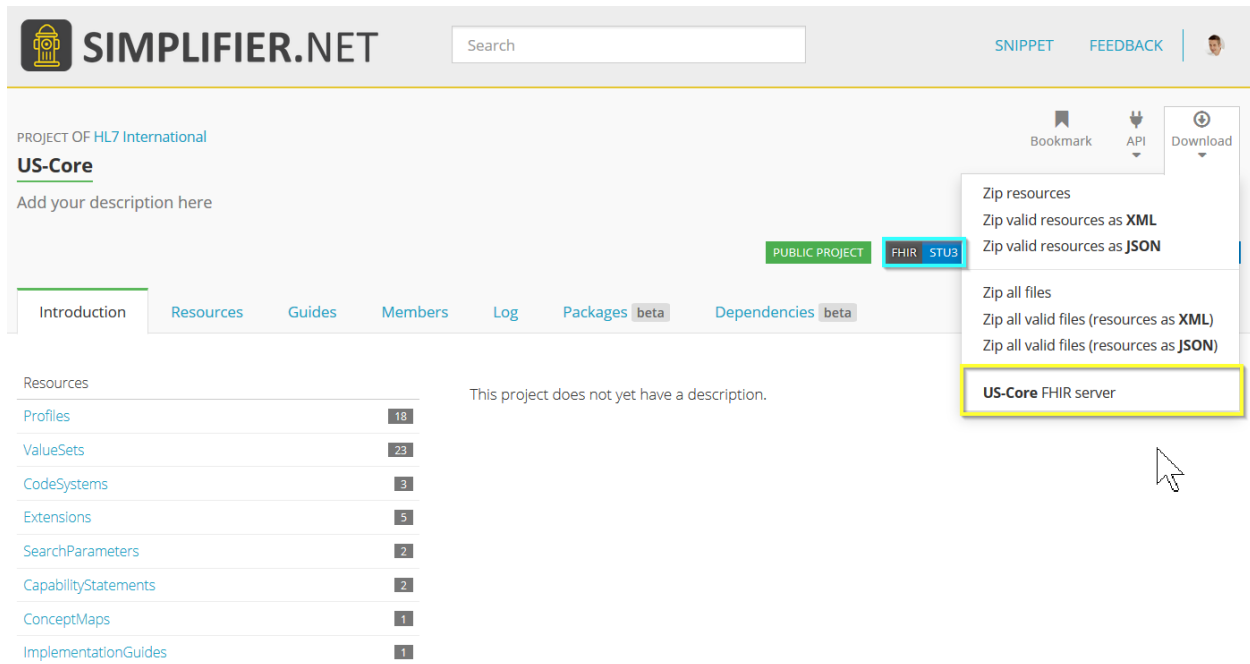
- Windows OS that can run Docker for Windows
- Docker for Windows

If you still need to install Docker for Windows, please read `docker_win` for specific installation details.


8.3.2 Getting the server


Go to [Simplifier](#) and open the project page of the project that has the profiles that are relevant to you. For our example we took the US-Core project. Note that it has to be an *STU3* project.

On the project page, click the Download button, and in the dropdown menu that appears, click 'US-Core FHIR Server' (the name here will reflect the name of the project). It is shown in the image below with the menu item marked in yellow. The light blue marking shows you where to check whether this is a FHIR STU3 project.



When you click the menu item, Simplifier takes you to a page with instructions. It also includes a link to this documentation page, since we can host more detailed instructions here than on the Simplifier site.


SIMPLIFIER.NET

[SNIPPET](#)
[FEEDBACK](#)


PROJECT **US-Core**

US-Core FHIR server

← Back to project

Download and run the project as a FHIR server in Docker.

1

Install docker

Docker is the container engine that will run your FHIR server container. If you have not installed Docker yet, you can [download it here](#). It should run with Linux Containers, and have your Shared Drives enabled. Read the [instructions](#) for more detail.

2

Download and open the zip

Click on the download button to get the zip file. After that unpack it in a folder of your choice.


Download

3

Run the start script

Run the powershell script `start-vonk-server.ps1`. This script will start a VONK FHIR server that is configured with all conformance resources of this project. The database will contain all examples in this project. If Powershell does not let you run the script, [change the ExecutionPolicy](#).

Powered by



Vonk
[Public Vonk FHIR Server](#)
[Vonk Product Information](#)

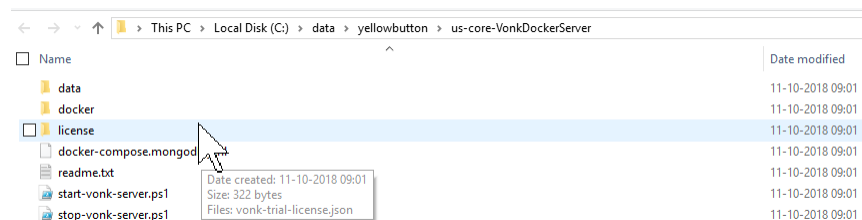
Powershell + Linux Container
 This installer uses Windows Powershell and runs a Docker Linux container. In the future, we will also provide an installer for Linux/Bash.

Documentation
 For more information you can check the [Yellow Button documentation](#).

License
 With this installer, you will be given a trial license to run this VONK FHIR server. For commercial licenses see [here](#).

Questions and Comments
 If you have questions or comments, let us know through the feedback button on the top of this site.

Click the yellow / orange Download button and you will download a zip file named after the server you are installing. Here it is `us-core-VonkDockerServer.zip`. Save it and unpack it in a folder on your harddrive. I unpacked it to `c:\data\yellowbutton\us-core-VonkDockerServer`.



8.3.3 Running the server

Open Windows Powershell (or Windows Powershell ISE if you prefer). Windows Powershell by default does not allow you to run scripts that you downloaded from the internet. To make sure you allow the server script to run, first tell Powershell that that is all right:

```
> Set-ExecutionPolicy Unrestricted -Scope CurrentUser
```

You can be more restrictive by setting it to `RemoteSigned`. See [Documentation on ExecutionPolicy](#) for more information.

Now go to the folder where you unpacked the zip, and run the script `./start-vonk-server.ps1`:

```
> cd c:\data\yellowbutton\us-core-VonkDockerServer
> .\start-vonk-server.ps1
```

Powershell will probably still ask for confirmation before running the script. Type 'R' to allow it to run:

```
Security warning
Run only scripts that you trust. While scripts from the internet can be useful, this script can potentially harm your
computer. If you trust this script, use the Unblock-File cmdlet to allow the script to run without this warning
message. Do you want to run C:\data\yellowbutton\us-core-VonkDocker\Server\start-vonk-server.ps1?
[D] Do not run [R] Run once [S] Suspend [?] Help (default is "D"): R_
```

The script will pull images from the Docker hub as necessary, and then start the Firely Server. Firely Server will load all the conformance resources from the core specification, and from your project into its Administration database. You can read [Conformance Resources](#) if you want to know more about this. Since this may take some time, you will see a progress bar. Firely Server is allowed to finish this task in at most 6 minutes.

```
PS C:\data\yellowbutton\us-core-VonkDockerServer> .\start-vonk-server.ps1

Initialization in Progress
 4% Complete:
[oooo]

Pulling vonk-web ... done
Pulling vonk-mongo-db ... done
Creating network "us-core_default" with the default driver
Creating us-core_vonk-web_1 ... done
Creating us-core_vonk-mongo-db_1 ... done
Initializing Vonk. This can take a few minutes the first time..
```

As soon as Firely Server is ready, the script will finish and it will open your browser on the endpoint of Firely Server: <http://127.0.0.1:8080/>, showing the landing page. For further use of the Firely Server RESTful API you will want to use an API testing tool like Postman.

If the script does not finish in due time, please check the Common errors and mistakes below.

8.3.4 Common errors and mistakes

Error messages

1. Docker is not running

```
error during connect: Get http://%2F%2F.%2Fpipe%2Fdocker_engine/v1.39/
↳containers/json: open //.pipe/docker_engine: The system cannot find the file
↳specified.
In the default daemon configuration on Windows, the docker client must be run
↳elevated to connect. This error may also indicate that the docker daemon is
↳not running.
Docker is not running, now exiting the script. See https://docs.docker.com/
↳docker-for-windows/install/ for more information.
```

Solution: The problem is exactly as stated – your Docker for Windows is probably not running. Start it from the Windows Start menu and try again.

2. Mount failed

```
ERROR: for vonk-web Cannot start service vonk-web: error while creating mount_
↳ source path '/host_mnt/c/data/yellowbutton/us-core-VonkDockerServer/license':
↳ mkdir /host_mnt/c: file exists
```

```
Pulling vonk-web ... done
Pulling vonk-mongo-db ... done
Starting us-core_vonk-mongo-db_1 ...
Recreating us-core_vonk-web_1 ... error
Starting us-core_vonk-mongo-db_1 ... done
yellowbutton/us-core-VonkDockerServer/license': mkdir /host_mnt/c: file exists
ERROR: for vonk-web Cannot start service vonk-web: error while creating mount source path '/host_mnt/c/data/yellowbutton/us-core-VonkDockerServer/license': mkdir /host_mnt/c: file exists
ERROR: Encountered errors while bringing up the project.
Running the Docker containers is failing. Exiting this script.
```

Solution: This may happen at subsequent starts of the Firely Server container. It appears to be an error in Docker for Windows. But it may be fixed by resetting the credentials for Drive Sharing in Docker for Windows (even if you did not change your password). See [docker_win_shared_drives](#) for more information.

3. Network failed

```
ERROR: for vonk-web Cannot start service vonk-web: driver failed programming ↵
↵external connectivity on endpoint ...
```

```
ERROR: for inactiveproject_vonk-web_1 Cannot start service vonk-web: driver failed programming external connectivity on
Creating inactiveproject_vonk-mongo-db_1 ...
userland proxy: mkdir /port/tcp:0.0.0.0:8080:tcp:172.24.0.3:4080: input/output error
ERROR: for vonk-web Cannot start service vonk-web: driver failed programming external connectivity on endpoint inactive
project_vonk-web_1 (8cc52589e67dc2e5f602512b6c0abd31fd907d1443fe8fbf80b4404356165df0): Error starting userland proxy: mk
dir /port/tcp:0.0.0.0:8080:tcp:172.24.0.3:4080: input/output error
F0000: Encountered errors while bringing up the project.
Running the Docker containers is failing. Exiting this script.
```

Solution: This is an issue reported as [Issue 1967 on Docker for Windows](#). It can be solved by restarting Docker on Windows.

Configuration checks

1. Is Docker for Windows configured to run *Linux* containers and not Windows containers? Check the [Docker switching Container type](#) documentation on this if needed.
2. Did you enable Shared Drives on Docker for Windows? Yellow Button needs this to provide the license file to the Docker container. See [docker_win_shared_drives](#) for more information.
3. Did you change your Windows password after sharing your drive in Docker for Windows? If so, you need to reset your credentials in Docker for Windows. See [docker_win_shared_drives](#) for more information.
4. Does Docker for Windows have enough resources to let Firely Server run its initialization within the designated time? You can give it more resources in the [Docker Advanced Settings](#).

Still no luck? Please contact us on server@fire.ly. Please include:

- the output of the Powershell script `./start-vonk-server.ps1`
- version info of Windows
- version info of Docker for Windows
- any other information you think is relevant.

8.3.5 Using the server

When your Firely Server is running, you can check whether your profiles are indeed present in the server by requesting them from the Administration endpoint. In this example we search for the US-Core profiles:

```
GET http://127.0.0.1:8080/administration/StructureDefinition?url:below=http://hl7.org/
↵fhir/us/core
```

Please note that any Conformance resources that influence the behaviour of Firely Server – such as the Validation – must be managed on the Administration API and not the regular FHIR endpoint. See [Firely Server Administration API](#) for more background.

If the project you created the server off of contains any *example* resources, they will be available at the normal FHIR endpoint:

```
GET http://127.0.0.1:8080/<more specific search if you want>
```

8.3.6 Your project in progress

You created the Firely Server off of a Simplifier project. That project may evolve. Either because it is your own and you improve your profiles, or because the maintainer of the project applies changes. Firely Server is connected to the Simplifier project. This means that you can update the conformance resource in Firely Server from the contents of the project by:

- invoking `importResources`:

```
POST http://127.0.0.1:8080/administration/importResources
```

- or restarting Firely Server:

```
> ./stop-vonk-server.ps1
> ./start-vonk-server.ps1
```

8.3.7 Further steps

Yellow Button is an easy way to get started with Firely Server. But there are many other *deployment options* for the server. Besides that you can add your own plugins with *Firely Server Plugins*, or build a *Firely Server Facade* with Firely Server.

8.4 Deploy Firely Server on a reverse proxy

8.4.1 Why

For ASP.NET Core 1.0 Microsoft suggested to always use another web server in front of Kestrel for public websites. For ASP.NET Core 2.0, while this is not a hard constraint anymore there are still a series of advantages in doing so:

- some scenarios like sharing the same IP and port by multiple applications are not yet supported in Kestrel
- helps in limiting the exposed surface area
- provides an additional layer of configuration and defense
- provides process management for the ASP.NET Core application (ensuring it restarts after it crashes)
- in some scenarios a certain web server already integrates very well
- helps simplifying load balancing and SSL setup

Hence using a reverse proxy together with the Kestrel server allows us to get benefits from both technologies at once.

8.4.2 With IIS

A common option on Windows is using IIS: for instructions on how to deploy Firely Server on IIS see *Deploy Firely Server on IIS*.

For a comparison of IIS and Kestrel features at the moment of this writing you can [check here](#).

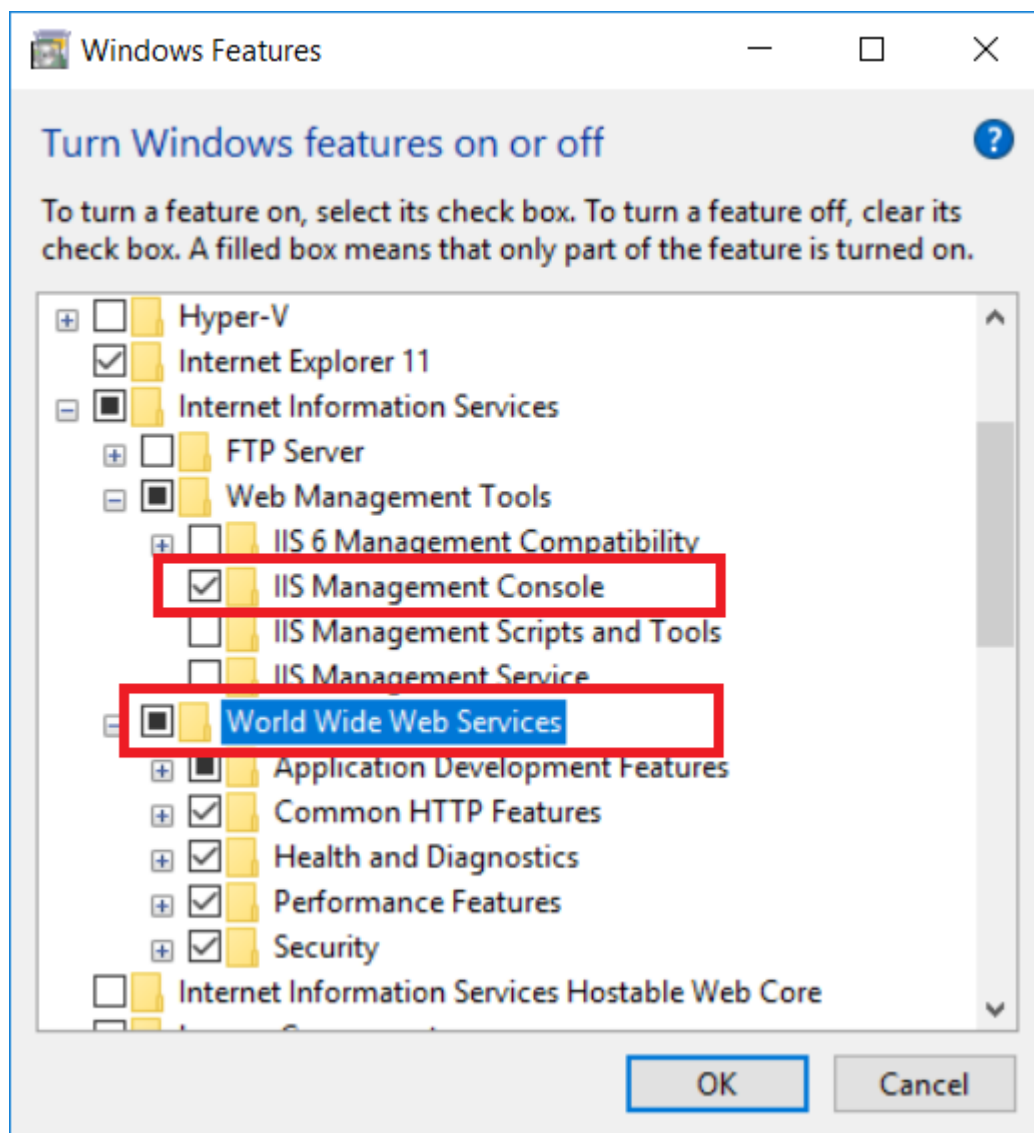
8.4.3 With Nginx

A popular open source alternative is Nginx. For instruction on how to deploy Firely Server on Nginx see [Deploy Firely Server on Nginx on Linux](#)

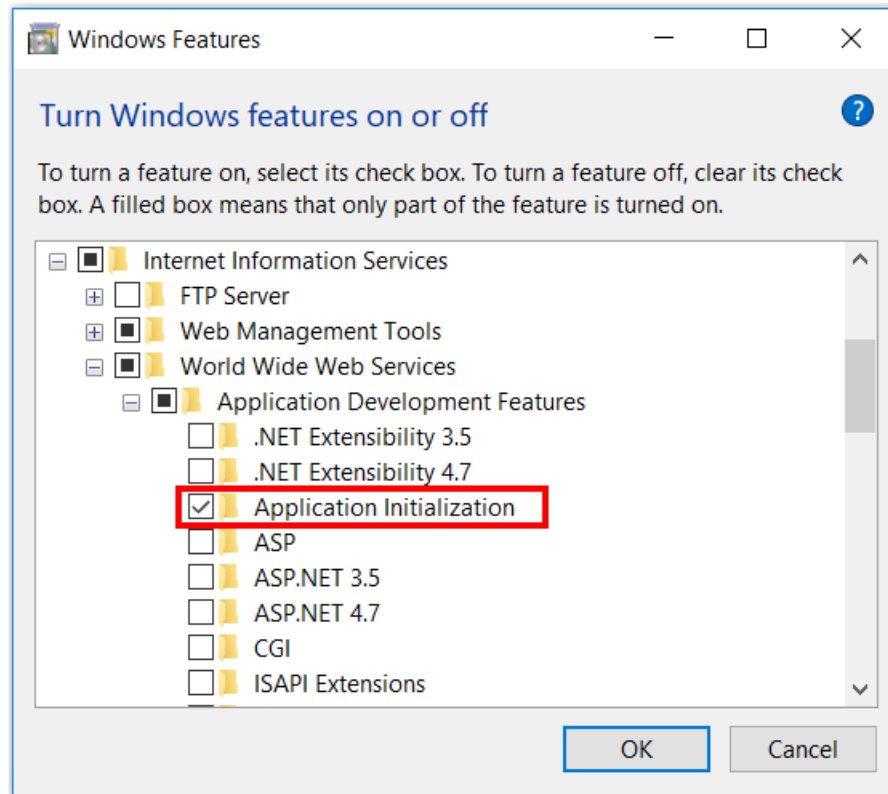
Deploy Firely Server on IIS

Prerequisites

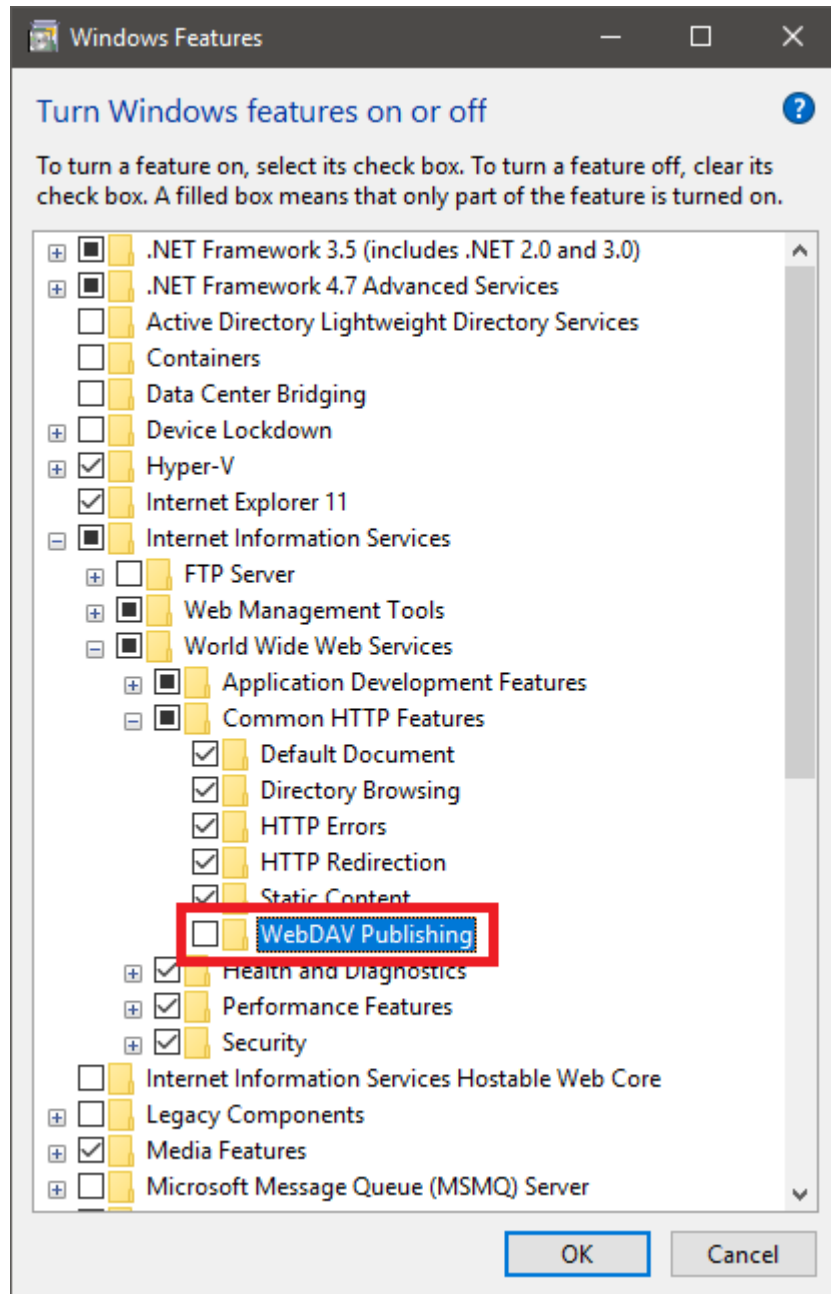
1. The following operating systems are supported: Windows 7 or Windows Server 2008 R2 and later
2. Have IIS windows feature turned on on the hosting machine. You can do this in **Control Panel -> Programs and Features -> Turn Windows features on or off**. You need to check **Internet Information Services -> Web Management Tools -> IIS Management Console** and **Internet Information Services -> World Wide Web Services** to accept the default features or customize IIS to fit your needs.



3. In order to have IIS start the Firely Server right away, you will have to enable the “Application Initialization” setting:



4. To enable PUT and DELETE interactions, you will have to turn off WebDAV:



See <https://stackoverflow.com/questions/6739124/iis-7-5-enable-put-and-delete-for-restful-service-extensionless> for some background information. If you do not want to disable WebDAV for all of IIS, you can also disable it just for Firely Server using a setting in *web.config*, see *Configuration*.

5. Choose a solution to deploy/move the application to the hosting system. Multiple alternatives exist like Web Deploy, Xcopy, Robocopy or Powershell. One popular choice is using Web Deploy in Visual Studio. For using that you will need to install Web Deploy on the hosting system. To install Web Deploy, you can use the Web Platform Installer (<https://www.microsoft.com/web/downloads/platform.aspx>).
6. Install the *.NET Core Runtime & Hosting bundle* on the hosting system. After installing it, you may need to do a “net stop was /y” and “net start w3svc” to ensure all the changes are picked up for IIS. The bundle installs the .NET Core Runtime, .NET Core Library, and the ASP.NET Core Module. ASP.NET Core Module (ANCM) allows you to run ASP.NET Core applications using Kestrel behind IIS. For more information about ANCM check <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/aspnet-core-module>

6. Prepare binaries. You can either download the binaries for Firely Server (see [Getting Started](#)), or create your own solution by building a facade.

If you are deploying a Firely Server facade in your own web server application, take the additional following prerequisites into consideration:

- Make sure you use the **IISIntegration NuGet** package. You can install this as part of one of the metapackages (`Microsoft.AspNetCore` and `Microsoft.AspNetCore.All`) or independently `Microsoft.AspNetCore.Server.IISIntegration`. This is needed for the interoperability between Kestrel and ANCM.
- Provide a `web.config` file for configuring the ANCM module or make sure the selected deploy options generates one for you. Using **dotnet publish** or **Visual studio publish** would generate a `web.config` for you. Check <https://docs.microsoft.com/en-us/aspnet/core/hosting/aspnet-core-module> for guidance on configuring the ANCM.

Create Website in IIS

1. Publish the application to a location on the host server, using the solution selected in the Prerequisites step 3.
2. In IIS Manager create a new website or a new application under existing IIS site. Fill the **Site name**, the **Binding** and link the **Physical path** to the folder created in the previous step, similar to the picture below. The bindings defined in IIS override any bindings set in the application by calling either `Listen` or `UseUrls`.

Add Website

Site name: Vonk_IIS

Application pool: Vonk_IIS Select...

Content Directory

Physical path: D:\Project\Vonk ...

Pass-through authentication

Connect as... Test Settings...

Binding

Type: http IP address: All Unassigned Port: 80

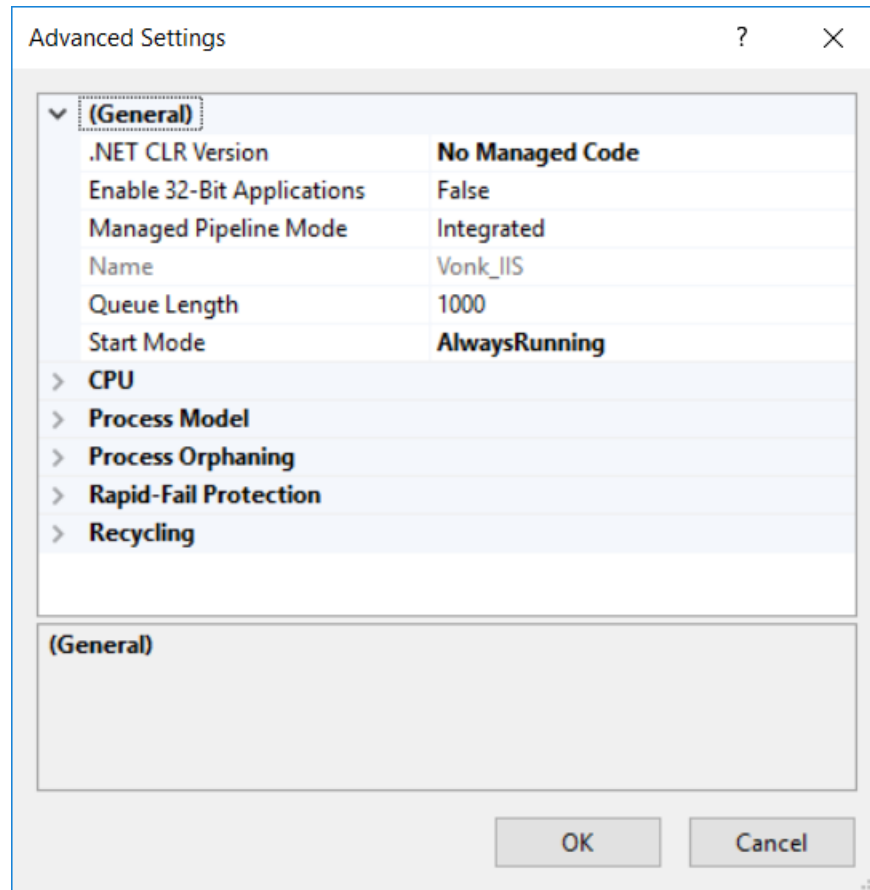
Host name: vonk.mydomain.org

Example: www.contoso.com or marketing.contoso.com

☐ Start Website immediately

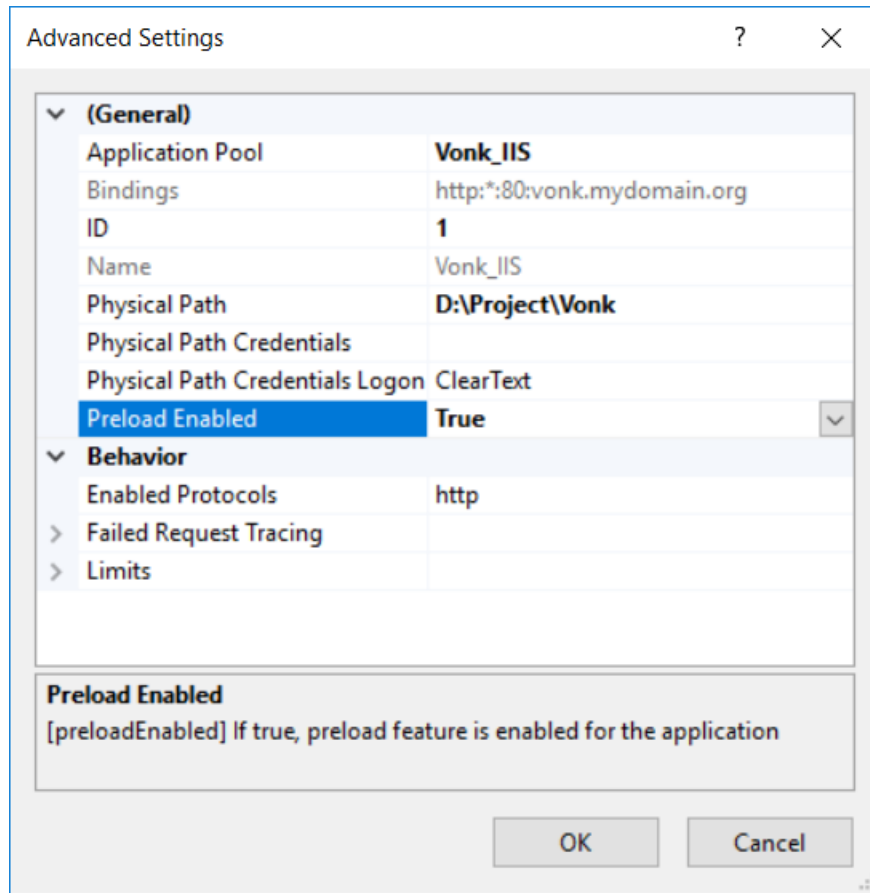
OK Cancel

3. Edit the application pool to set the **.NET CLR VERSION** to **NO Managed Code**, similar to the picture below (we use IIS as a reverse proxy, so it isn't actually executing any .NET code). To edit the application pool, go to the **Application Pools** panel, right-click the website's app pool and select **Advanced Settings...** from the popup menu.



Attention: IIS will pause a website if it is idle for a while. Pausing a dotnet process is the same as shutting it down, so this means that IIS shuts down the Firely Server. This causes problems with each first request that is sent to Firely Server after an idle period. Firely Server needs a couple of seconds to start up again, and answers with a 423 Lock Error while it loads. Make sure to set **Start Mode** to **AlwaysRunning** to prevent IIS from shutting down Firely Server.

4. Go to your IIS Sites, and click on your Firely Server website. On the right, choose **Advanced settings...**. Set **Preload Enabled** to **True** to make IIS load the Firely Server right away, instead of on the very first request. Otherwise this first request results in a 423 Lock Error as described above.



Configuration

- You can use *web.config* to configure ASP.NET Core Module and IIS using the `<system.webServer>` section. Read more about configuring ANCM at <https://docs.microsoft.com/en-us/aspnet/core/hosting/aspnet-core-module>.
- If you need to disable WebDAV for Firely Server, so you can perform PUT and DELETE interactions, add this to the *web.config* file:

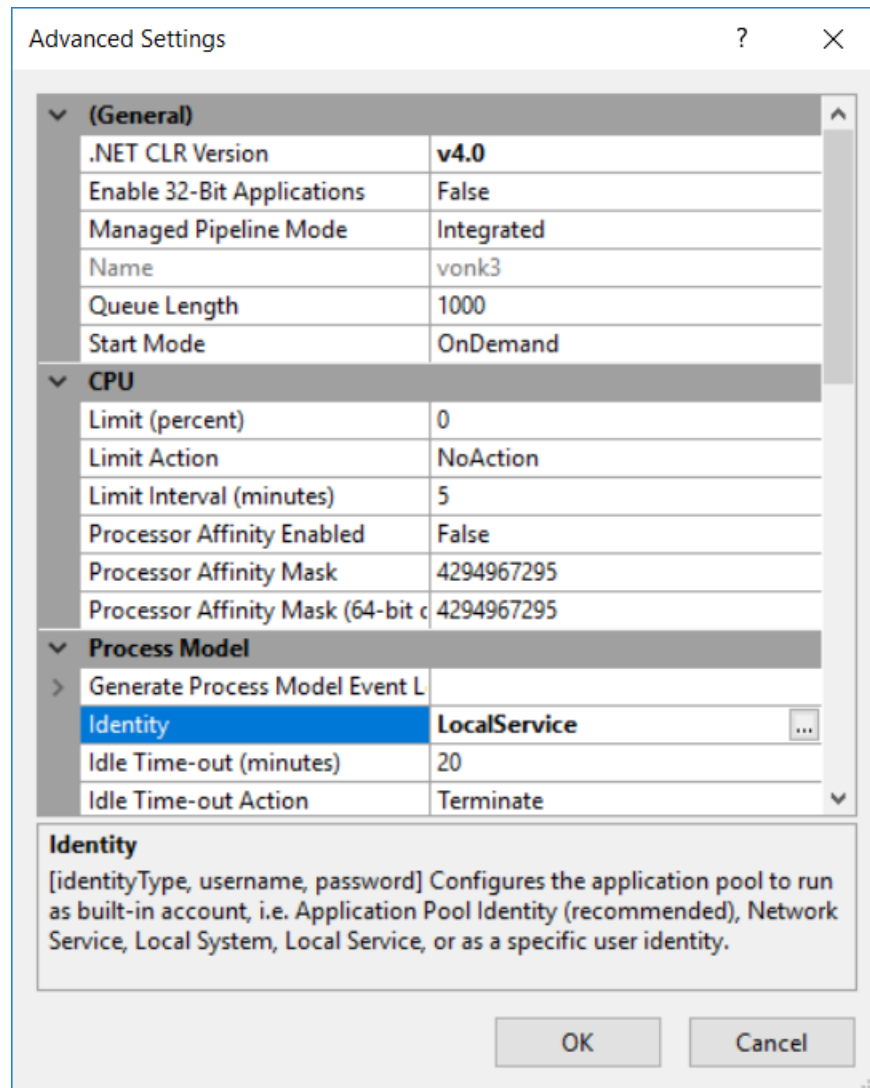
```
<modules>
  <remove name="WebDAVModule" />
</modules>
<handlers>
  <remove name="WebDAV" />
</handlers>
```

- You can configure the Firely Server using the *appsettings.json* file (see *Configuring Firely Server*).
- If you are building a Firely Server facade, you can use *IISOptions* to configure *IISIntegration* service options. You can read more about *IISOptions* at <https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.iisoptions?view=aspnetcore-2.0>.

```
services.Configure<IISOptions>(options =>
{
    ...
});
```

SQL

In order to use the Sql Repository option in IIS you should make sure that the identity of the IIS application pool has rights to use the database considering the provided connection string. To change the identity the application pool is using open IIS Application Pools select your application pool right click and select “Advanced Settings...” You should see something similar to the image below:



Extra

If you'd like to set Firely Server environment variables via an Azure pipelines task, you can do so by setting the application pool's environment variables. For example, to pass the variable TEST to Firely Server that's housed in the fhir application pool, do the following:

```
%systemroot%\system32\inetsrv\APPCMD set config -section:system.applicationHost/
applicationPools /- "[name='fhir'].environmentVariables.[name='VONK_TEST']" /
commit:apphost
%systemroot%\system32\inetsrv\APPCMD set config -section:system.applicationHost/
```

(continues on next page)

(continued from previous page)

```
↪applicationPools /+"[name='fhir'].environmentVariables.[name='VONK_TEST',value='some_↪value_here']" /commit:apphost
```

See also *Firely Server settings with Environment Variables*.

Deploy Firely Server on Nginx on Linux

About Nginx

NGINX is a popular open source web server. It can act as a reverse proxy server for TCP, UDP, HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer and a HTTP cache. You can find the documentation for the Nginx server at <https://nginx.org/en/docs/>.

Prerequisites

1. The following linux distribution are supported: Ubuntu, RHEL, Debian, Fedora, CentOS, SLES
2. Install .Net Core on the machine (see <https://www.microsoft.com/net/learn/get-started/linuxubuntu>)
3. Install Nginx `sudo apt-get install nginx`

Start Kestrel Firely Server

Download the binaries for Firely Server (see *Getting Started*), open a terminal console and start the Firely Server process by using: `dotnet Vonk.Server.dll`. You should be able to reach to home page at `http://localhost:4080` (or a different port if you changed the default configurations)

Configure Nginx as a reverse proxy

To configure Nginx as a reverse proxy to forward requests to our ASP.NET Core application, modify `/etc/nginx/sites-available/default`. Open it in a text editor, and replace the contents with the following:

```
server {  
    listen 80;  
    # Match incoming requests with the following path and forward them to  
    # the location of the Kestrel server.  
    # See http://nginx.org/en/docs/http/nginx_http_core_module.html#location  
    location / {  
        #This should match the location where you deployed the Firely Server binaries.↪  
↪with the Kestrel server.  
        #This can be on the same machine as the Nginx server or on a separate dedicated.↪  
↪machine  
        proxy_pass http://localhost:4080;  
        # The Kestrel web server we are forwarding requests to only speaks HTTP 1.1.  
        proxy_http_version 1.1;  
        proxy_set_header Upgrade $http_upgrade;  
        # Adds the 'Connection: keep-alive' HTTP header.  
        proxy_set_header Connection keep-alive;  
        # Forwards the Host HTTP header.
```

(continues on next page)

(continued from previous page)

```
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
}
}
```

Now you can run the Firely Server.

Configuration

- To configure the Firely Server, you can use the appsettings.json file (see [Configuring Firely Server](#)).
- To configure Nginx you need to add extra options to the /etc/nginx/sites-available/default or to the nginx.conf file.
- To monitor the application you can use systemd and create a service for starting, stopping and managing the process.

8.5 Set up an Identity Provider

8.5.1 About Identity Providers and Firely Server

In order to use *Access control and SMART* you need an Identity Provider that can provide OAuth2 JWT Tokens with claims that conform to *SMART on FHIR*. In a production scenario, you typically already have such a provider. It could be the EHR system, the Active Directory, or a provider set up specifically for let's say a Patient Portal. It is also very well possible that the provider handing the correct claims uses a federated OAuth2 provider to do the authentication.

8.5.2 An Identity Provider for testing

To allow you to test *Access control and SMART*, we provide you with instructions to build and run an Identity Provider in which you can configure the necessary clients, claims and users yourself to test different scenarios. The instructions are based on the excellent [IdentityServer4 project on GitHub](#) by Dominick Baier and Brock Allen.

By default, the configuration is such that you can test many different cases. If you wish to adjust the configuration, that will require a bit of programming.

The Identity Provider is built in Microsoft .NET Core. Therefore it should also run cross-platform, just as Firely Server itself. However, we did not try that.

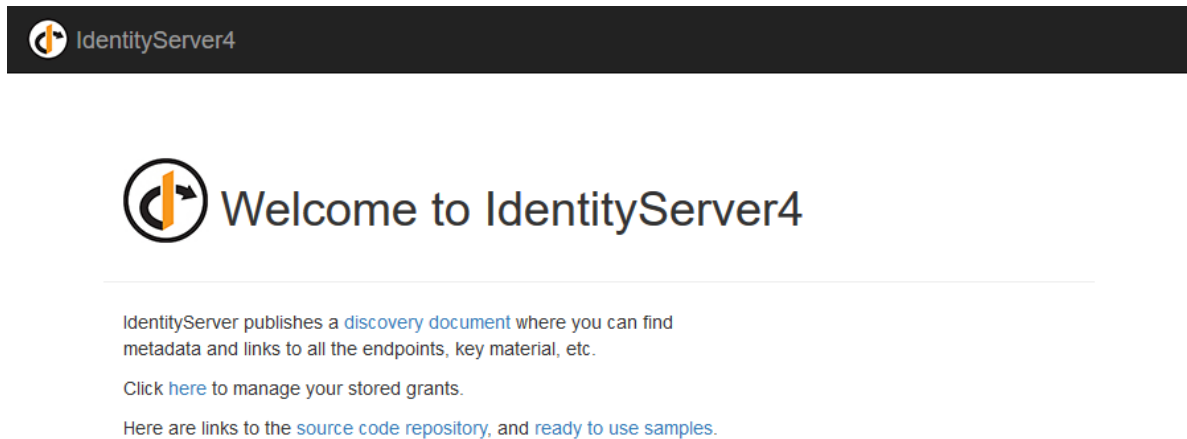
Note: The project below is provided for your convenience. It comes with no warranty and is not supported by Firely.

In order to get tokens from the Identity Provider you need an http client. We included instructions on [Access Control Tokens with Postman](#).

8.5.3 Instructions

1. Clone the project `Vonk.IdentityServer.Test` from GitHub
2. Run the Powershell script `.\scripts\GenerateSSLCertificate.ps1` This will generate an SSL Certificate in `.\Vonk.IdentityServer.Test\ssl_cert.pfx`, with the password 'cert-password'. This is preconfigured in `Program.cs`.
3. Open the solution `Vonk.IdentityServer.Test.sln` in Visual Studio
4. Build the solution
5. Run the `Vonk.IdentityServer.Test` project
6. Visual Studio should automatically open <http://localhost:5100> in your browser.

You should see a page like this.



7. Also try <https://localhost:5101> for the https connection. Your browser will ask you to make a security exception for the self-signed certificate.
8. Get the openid connect configuration at <http://localhost:5100/.well-known/openid-configuration>. You can see all the available scopes in this document.

8.5.4 Configuration

The Identity Server is preconfigured with two users and one client:

Client

ClientId

Postman

Secret

secret

Redirect Uri

<https://www.getpostman.com/oauth2/callback>

This client is allowed to request any of the available scopes.

It is called Postman, since many users use the Postman REST client to test FHIR Servers. If you use another client, you can still use it as the ClientId, or alter the values in Config.cs.

Users

Alice

UserName

Alice

Password

password

Launch context

patient=alice-identifier

Bob

UserName

Bob

Password

password

Launch context

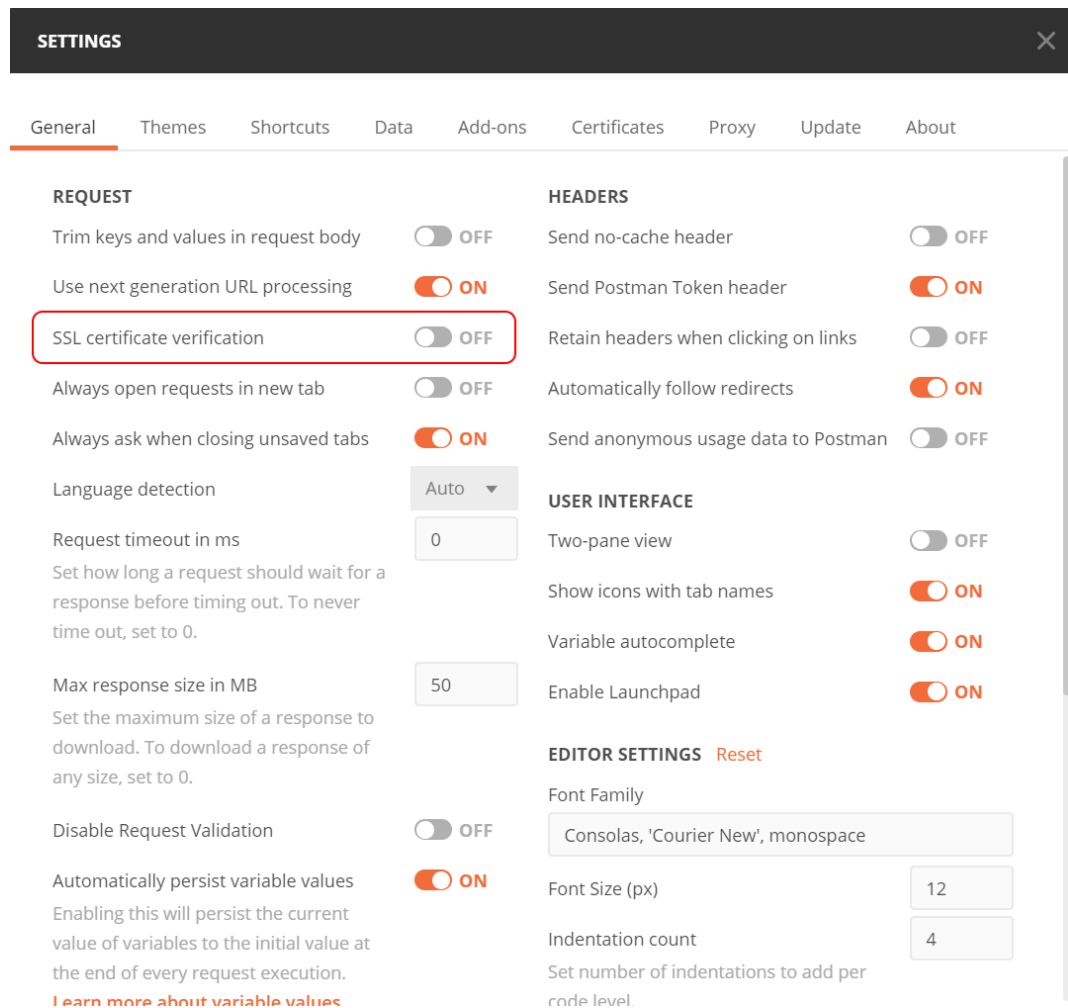
patient=bob-identifier

You can add or alter users in Config.cs.

8.6 Access Control Tokens with Postman

You can use Postman to get a JWT Token from the IdentityServer, and use that in a subsequent request to your local Firely Server instance.

1. Make sure IdentityServer is running (see *Set up an Identity Provider*), I assume at <http://localhost:5100>
2. Open Postman Settings (menu: File | Settings) and turn ssl certificate validation off, otherwise your self-signed certificate will not be accepted.



3. Open a request in Postman, let's say GET /Patient
4. Verify that you get a 401 (smile)
5. Go to the Headers tab and make sure there is no Authorization header (if there is, it might have an outdated token, and you don't want that)
6. Go to the Authorization tab, that looks like this:

GET
{{url}}/Patient
Send
Save

Params
Auth
Headers (7)
Body
Pre-req.
Tests
Settings
Cookies
Code

TYPE

OAuth 2.0

The authorization data will be automatically generated when you send the request. [Learn more about authorization](#)

Add authorization data to

Request Headers

Current Token

Access Token
Available Tokens
Access Token

Header Prefix
Bearer

Configure New Token

Token Name
firelyserver_auth_token

Grant Type
Authorization Code

Callback URL
https://www.getpostman.com/oauth2/callback

☐
Authorize using browser

Auth URL
https://localhost:5101/connect/authorize

Access Token URL
https://localhost:5101/connect/token

Client ID
Postman

Client Secret
secret

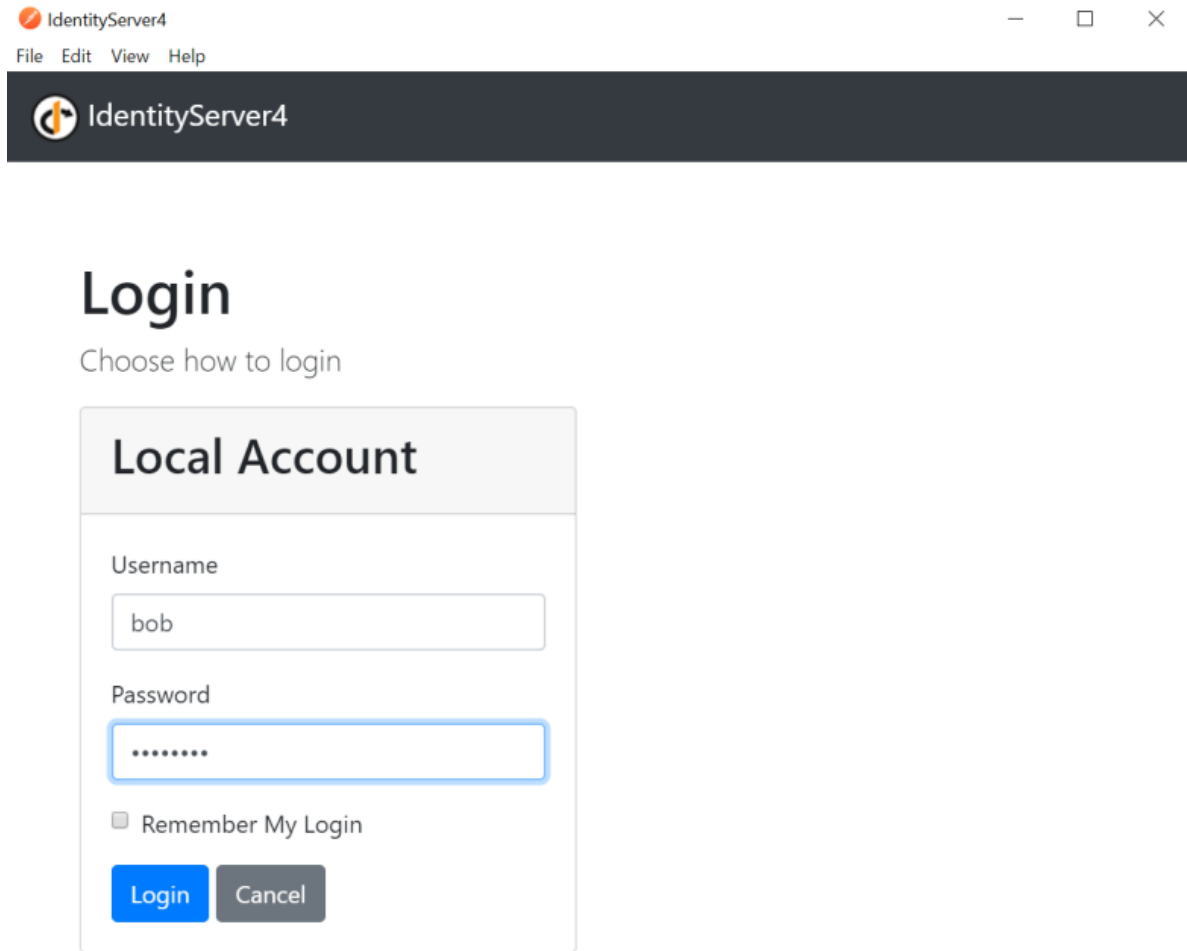
Scope
openid profile user/*read

State
State

Client Authentication
Send as Basic Auth header

Get New Access Token

7. In the 'Type' dropdown choose OAuth2 (SMART uses OpenIdConnect, which is a specialization of OAuth2)
8. In the 'Add authorization data to' dropdown choose 'Request headers' (probably preselected)
9. Now fill in the blank fields under section 'Configure New Token'.
10. Take special care to use https in the AUTH URL and Access Token URL fields.
11. You can alter the values in 'Scope' to get other claims in the token.
12. Click 'Get New Access Token' and you'll be presented with the login screen of IdentityServer:



IdentityServer4

File Edit View Help

IdentityServer4

Login

Choose how to login

Local Account

Username

bob

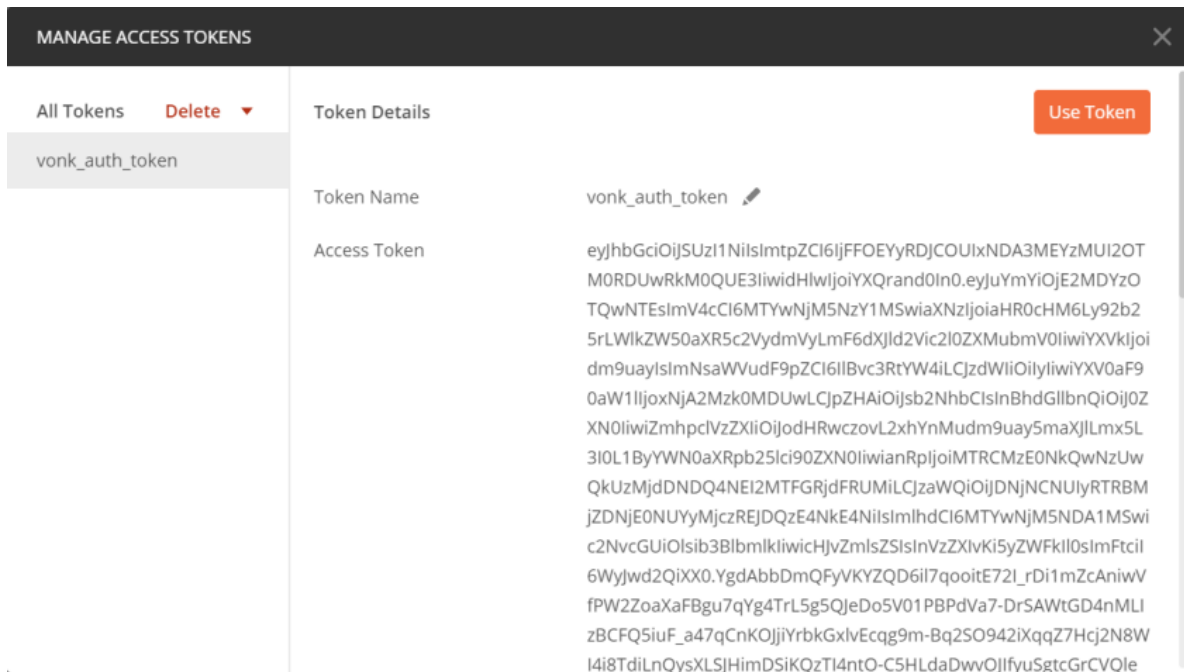
Password

.....

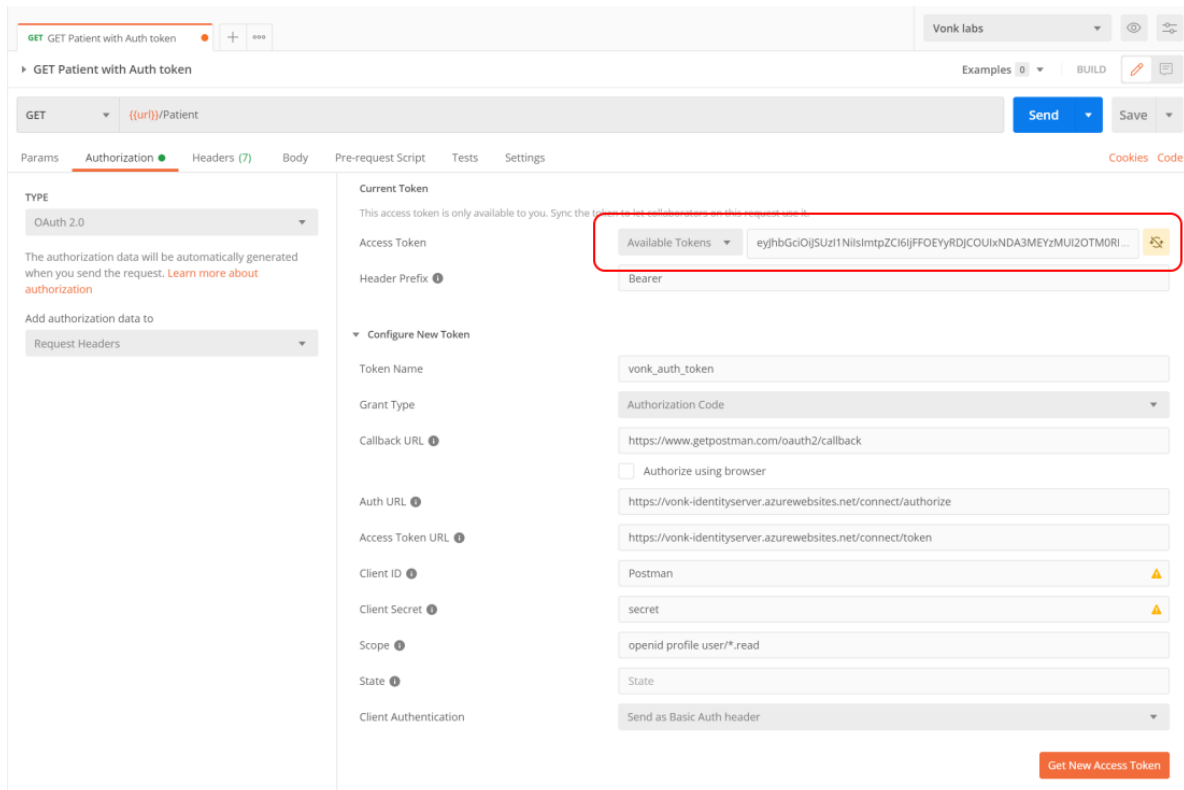
☐ Remember My Login

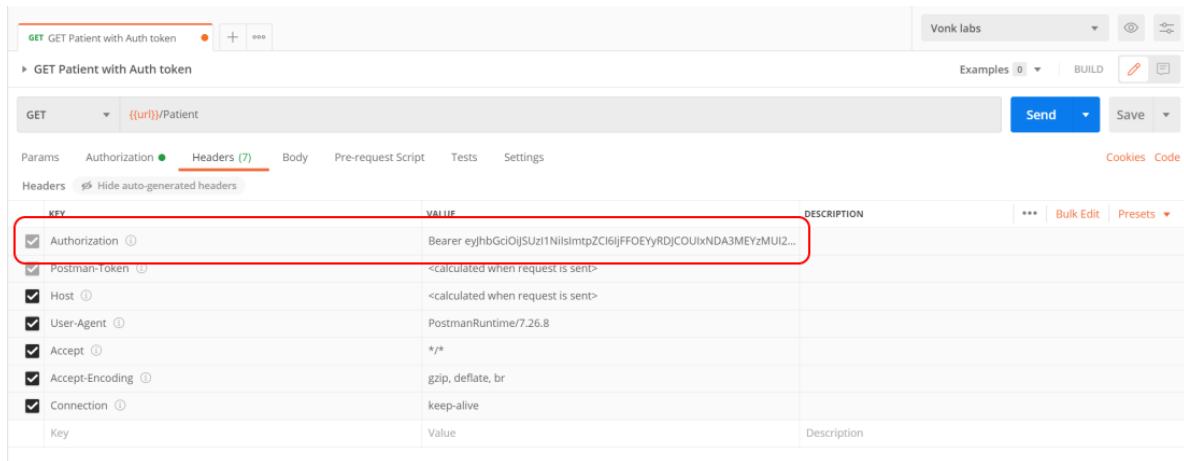
Login Cancel

13. Log in as Bob or Alice and you return to Postman with the newly retrieved token:



14. Optionally, you can copy the value of the access token and paste it into [JWT.io](https://jwt.io). It will show you the contents of the token.
15. Back in Postman, click 'Use Token'.
16. The token will be added as Authorization header to the request (make sure you have disabled 'Hide auto-generated headers' in the Headers tab):





17. Issue the original request again. Provided there is a Patient with the identifier of Bob or Alice (or whomever you chose), it will be in the search results.

8.7 Performance of Firely Server

8.7.1 About Performance

What is the performance of Firely Server? That is a simple question, but unfortunately it has no answer. Or more precisely, the answer depends on many variables. On this page we try to give you insight into those variables, introduce you to testing performance yourself, and finally present the results of the tests that we run ourselves.

8.7.2 Performance variables

Firely Server Configuration

Firely Server can be run as self contained FHIR Server or as a Facade on top of an existing system. The performance of a Facade is heavily dependent on the performance of the system it is built on top of. The self contained server can run on different databases. On top of that you can configure a couple of features in the settings that influence the performance. These are the most important configuration variables to take into account:

Repository

1. **Memory:** Memory is only meant for quick tests, and for use in unittests. Do not use it in any serious scenario, much less for performance critical scenarios.
2. **SQLite:** SQLite is mainly used for the Administration database of Firely Server, but you can also use it for the main database. Deployment is very easy because of the zero footprint of the driver, but be aware of its limits. Firely Server must have very fast access to the database file, so effectively it has to be on a local disk. Multithreading does work, but does not scale as well as other databases.
3. **SQL Server:** Performance tuning of SQL Server is a topic on its own. Firely Server manages the tables it needs, and the indexes on top of it are optimized for the way Firely Server queries them.
4. **MongoDB:** Performance tuning of MongoDB is, as well, a topic on its own. Firely Server manages the collections it needs, and the indexes on top of it are optimized for the way Firely Server queries them. MongoDB is used in our own performance tests, see below.

Prevalidation

By default, a resource that is sent to Firely Server is fully validated against its `StructureDefinition`. This requires extra processing and thus extra time. But you can disable full validation if needed, with the [ValidateIncomingResources](#) setting. We have no tests in place yet to time the difference caused by this setting.

Search Parameters

When a resource is sent to Firely Server for storage, Firely Server indexes the resource for all the search parameters that are applicable to the resource. The more search parameters are known to Firely Server, the more resources will be used for indexing - both time (for the indexing processing) and storage (for storing the values of each search parameter). This also increases the size of the index tables and indexes, and therefore querying times. Thus, if you know you will only use a portion of the predefined search parameters you can choose to delete the others from the Administration API - see [Conformance Resources](#) and [Restrict supported resources and SearchParameters](#).

Pipeline

Firely Server is made up of a pipeline of plugins. You can leave out any plugin that you don't need - so if you don't need conditional processing (create, update, delete), just exclude them from the pipeline. Excluded plugins are not loaded and thus never executed - see [Configuring the Firely Server Pipeline](#).

Platform

Firely Server can run on Windows, Linux and MacOS. Directly or in a Docker container. On real hardware, virtual machine, app service or Kubernetes cluster. And then you can choose the dimensions of the platform, scaling up (bigger 'hard'ware) or scaling out (more (virtual) machines) as you see fit. Each of these choices will influence performance.

Besides the way Firely Server is deployed, the way the database is deployed is an important factor. Firely Server needs a very low latency connection between the Firely Server process(es) and the database server. If you have configured [Application Insights](#), the calls to the database are recorded as separate dependencies so you can check whether this may be a bottleneck.

Firely Server is optimized for multithreaded processing. This means that it will fully benefit from extra processing cores, either in the same machine (multi core processor) or by adding additional machines (and thus processors).

Firely Server is fully stateless, so no data about connections is saved between requests. First of all this helps in scaling out, since you don't need thread affinity. On top of that this reduces the memory requirements of Firely Server.

Usage patterns

How will Firely Server be used in your environment?

1. Mostly for querying, or rather for creating and updating resources? Altering resources requires more processing than reading them. Also see the comment on indexing and search parameters above.
2. How is the distribution of values in the resources that you query on? E.g. if you use only a few types of resources, query them just by tag and the resources have only about 5 different tags, calculating the number of results will take a lot of time. Using more finegrained distributed values to query on solves this.
3. With many individual resources or with (large) batches or transactions? Transactions take a lot longer to process and require more memory, proportionally to the number of resources in them. If many transactions are run in parallel, requests may queue up.

4. Many users with a low request rate each, or a few heavy users? Since Firely Server is stateless, this has little influence. The total request rate is what counts.

8.7.3 Testing performance yourself

Because of all the variables mentioned above the best way to find out whether Firely Server's performance is sufficient for your use is: test it yourself.

We provide an evaluation license that you can use for any testing, including performance testing. See [Getting Started](#).

Variables

Before you start testing, study the variables above and provide answers to them. Then you can configure your platform and your tests in a way that comes closest to the expected real use.

Requests

You need a set of requests that you want to test. Based on your use case, identify the 5 (or more) most frequent requests. For extra realism you should provide the parameters to the requests from a dataset (like a .csv file with search parameter values).

What to measure?

There are essentially two questions that you can investigate:

1. Given this deployment, (mix of) requests and an expected request rate, what are the response times?
2. Given this deployment and (a mix of) requests, how many requests can Firely Server handle before it starts returning time-outs?

Besides response times more insight can be gained by measuring the load on the server (processor / memory usage, disk and network latency, for both the Firely Server and the database server) as well as the machine you are generating the requests from (to ensure that is not bottlenecked).

Always make sure to use at least 2 separate machines for testing: one for Firely Server, and a separate one for generating the requests. Testing Firely Server on the same machine as you're generating requests from will make Firely Server compete with the load testing tool for resources which'll hamper the legitimacy of the test results.

Based on the answers you can retry with different parameters (e.g. add/remove hardware) to get a sense of the requirements for real use deployment.

Data

Performance testing is best done with data as realistic to your situation as possible. So if you happen to have historic data that you can transform to FHIR resources, that is the best data to test with.

But if you don't have data of your own, you can use synthesized data. We use data from the Synthea project for our own tests. And we provide VonkLoader to upload the collection bundles from Synthea to Firely Server (or any FHIR Server for that matter).

If you build a Facade, the historical data is probably already in a test environment of the system you build the Facade on. That is a perfect start.

Test framework

To run performance tests you need a framework to send the requests in parallel and measure the response times. Test automation is a profession in itself so we cannot go into much detail here. You can search for 'REST Performance test tools' to get some options.

8.7.4 Available performance figures

We are in the process of setting up performance tests as part of our Continuous Integration and Deployment. Here we describe how this test is currently set up. Because of the beta phase this is in, the output is not yet complete nor fully reliable. Nevertheless we share the preliminary results to give you a first insight.

Firely Server performance test setup

1. Configuration
 1. Repository: MongoDB, both for Administration and for the main database.
 2. Prevalidation: off
 3. Search parameters: support all types of resources and all search parameters from the FHIR specification.
 4. Pipeline: load all available plugins except authorization.
2. Platform
 1. Azure Kubernetes Service, 2 nodes.
 2. Each node: Standard F2s (2 vcpus, 4 GB memory), running Linux
 3. 1 MongoDB pod and 2 Firely Server pods, plus the Kubernetes manager
3. Usage pattern - we created a simple mix of requests
 1. Upload the first 100 Synthea bundles from the precalculated set, each collection bundle transformed to a Batch.
 2. A 'general' test, consisting of:
 1. Query Patient by name: GET {url}/Patient?name=...
 2. Query Patient by name and maximum age: GET {url}/Patient?name={name}&birthdate=ge{year}
 3. Query all Conditions: GET {url}/Condition
 4. Query a Patient by identifier, with Observations: GET {url}/Patient?identifier={some identifier}&_revinclude=Observation:subject
 5. Query a Patient by identifier, with Observations and DiagnosticReports: GET {url}/Patient?identifier={some identifier}&_revinclude=Observation:subject&_revinclude=DiagnosticReport:patient
 3. Page through all the CarePlan resources: GET {url}/CarePlan?_count=10, and follow next links.
 4. Page through 1/5 of the Patient resources and delete them: DELETE {url}/Patient/{id}
 5. 20 concurrent users, randomly waiting up to 1 second before issuing the next request.
 6. Test run of 5 minutes
4. Test framework
 1. Locust for defining and running tests

2. Telegraf agents for collection metrics
3. InfluxDB for storing results
4. Grafana for displaying results

Test results

1. Upload: not properly timed yet.
2. General test: 75 percentile of response times around 200 ms. Note that the responses on queries with ‘_revinclude’ contain over 30 resources on average, sometimes over 100.
3. Page through all CarePlan resources: 75 percentile of response times around 110 ms.
4. Delete patients: This test always runs with 40 concurrent users, and 75 percentile of response times are around 350ms. Note that in Firely Server a delete is essentially an update, since all old versions are retained.

SECURITY

This section lists some general recommendations and steps to configure SMART on FHIR on Firely Server, to help secure access to the data in your server.

9.1 Access control and SMART

Contents

- *Concepts*
- *Identification and Authentication*
- *Authorization*
- *Access Control Engine*
- *Custom Authentication*
- *Other forms of Authorization*
- *Configuration*
- *Compartments*
- *Tokens*
- *Access Control Decisions*
- *Testing*

9.1.1 Concepts

This explanation of access control and SMART in Firely Server requires basic understanding of the *architecture* of Firely Server, so you know what is meant by middleware components and repository interfaces. It also presumes general knowledge about authentication and OAuth2.

Access control generally consists of the following parts, which will be addressed one by one:

- Identification: Who are you? – usually a user name, login, or some identifier.
- Authentication: Prove your identification – usually with a password, a certificate or some other (combination of) secret(s) owned by you.
- Authorization: What are you allowed to read or change based on your identification?

- Access Control Engine: Enforce the authorization in the context of a specific request.

9.1.2 Identification and Authentication

Firely Server does not authenticate users itself. It is meant to be used in an [OAuth2](#) environment in which an [OAuth2 provider](#) is responsible for the identification and authentication of users. Typically, a user first enters a Web Application, e.g. a patient portal, or a mobile app. That application interactively redirects the user to the OAuth2 provider - which is the authorization server and may or may not handle authentication as well - to authenticate, and receives an OAuth2 token back. Then, the application can do an http request to Firely Server to send or receive resource(s), and provide the OAuth2 token in the http Authentication header, thereby acting on behalf of the user. Firely Server can then read the OAuth2 token and validate it with the OAuth2 authorization server. This functionality is not FHIR specific.

9.1.3 Authorization

Authorization in Firely Server by default is based on [SMART on FHIR](#) and more specifically the [Scopes and Launch Context](#) defined by it. SMART specifies several claims that can be present in the OAuth2 token and their meaning. These are examples of scopes and launch contexts that are recognized by Firely Server:

- `scope=user/Observation.read`: the user is allowed to read Observation resources
- `scope=user/Encounter.write`: the user is allowed to write Encounter resources
- `scope=user/*.read`: the user is allowed to read any type of resource
- `scope=user/*.write`: the user is allowed to write any type of resource
- `scope=[array of individual scopes]`
- `patient=123`: the user is allowed access to resources in the compartment of patient 123 – see [Compartments](#).

SMART on FHIR also defines scopes starting with ‘patient/’ instead of ‘user/’. In Firely Server these are evaluated equally. But with a scope of ‘patient/’ you are required to also have a ‘patient=...’ launch context to know to which patient the user connects. It is also possible to apply a launch context to a user scope, for example the scope can look like “launch user/*.read”. In your authorization server you can specify the resources that are in the launch context parameter.

The assignment of these claims to users, systems or groups is managed in the OAuth2 authorization server and not in Firely Server. Firely Server does, however, need a way to access these scopes - so if your OAuth server is issuing a self-encoded token, ensure that it has a `scope` field with all of the granted scopes inside it.

9.1.4 Access Control Engine

The Access Control Engine in Firely Server evaluates two types of authorization:

1. Type-Access: Is the user allowed to read or write resource(s) of a specific resourcetype?
2. Compartment: Is the data to be read or written within the current compartment (if any)?

As you may have noticed, Type-Access aligns with the concept of scopes, and Compartment aligns with the concept of launch context in SMART on FHIR.

The Firely Server `SmartContextMiddleware` component extracts the claims defined by SMART on FHIR from the OAuth2 token, and puts it into two classes that are then available for Access Control Decisions in all the interaction handlers (e.g. for read, search, create etc.)

SMART on FHIR defines launch contexts for Patient, Encounter and Location, extendible with others if needed. If a request is done with a Patient launch context, and the user is allowed to see other resource types as well, these other resource types are restricted by the [Patient CompartmentDefinition](#).

9.1.5 Custom Authentication

You may build a plugin with custom middleware to provide authentication in a form that suits your needs. One example could be that you want to integrate [ASP.NET Core Identity](#) into Firely Server. Then you don't need the OAuth2 middleware, but instead can use the Identity framework to authenticate your users. See [Custom authorization plugin](#) for more details.

9.1.6 Other forms of Authorization

In [Access Control in Plugins and Facades](#) you can find the interfaces relevant to authorization in Firely Server. If your environment requires other authorization information than the standard SMART on FHIR claims, you can create your own implementations of these interfaces. You do this by implementing a [custom plugin](#). All the standard plugins of Firely Server can then use that implementation to enforce access control.

9.1.7 Configuration

You will need to add the Smart plugin to the Firely Server pipeline. See [Firely Server Plugins](#) for more information. In `appsettings[.instance].json`, locate the pipeline configuration in the `PipelineOptions` section, or copy that section from `appsettings.default.json` (see also [Changing the settings](#)):

```
"PipelineOptions": {
  "PluginDirectory": "./plugins",
  "Branches": [
    {
      "Path": "/",
      "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",
        ...
      ]
    }
  ]
}
```

Add `Vonk.Smart` to the list of included plugins. When you restart Firely Server, the Smart service will be added to the pipeline.

You can control the way Access Control based on SMART on FHIR behaves with the `SmartAuthorizationOptions` in the [Firely Server settings](#):

```
"SmartAuthorizationOptions": {
  "Enabled": true,
  "Filters": [
    {
      "FilterType": "Patient", //Filter on a Patient compartment if a 'patient' launch
      ↳scope is in the auth token
      "FilterArgument": "identifier=#patient#" //... for the Patient that has an
      ↳identifier matching the value of that 'patient' launch scope
    },
    {
      "FilterType": "Encounter", //Filter on an Encounter compartment if an 'encounter'
      ↳launch scope is in the auth token
      "FilterArgument": "identifier=#encounter#" //... for the Encounter that has an
      ↳identifier matching the value of that 'encounter' launch scope
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    "FilterType": "RelatedPerson", //Filter on a RelatedPerson compartment if a
    ↪ 'relatedperson' launch scope is in the auth token
    "FilterArgument": "identifier=#relatedperson#" //... for the RelatedPerson that
    ↪ has an identifier matching the value of that 'relatedperson' launch scope
  },
  {
    "FilterType": "Practitioner", //Filter on a Practitioner compartment if a
    ↪ 'practitioner' launch scope is in the auth token
    "FilterArgument": "identifier=#practitioner#" //... for the Practitioner that has
    ↪ an identifier matching the value of that 'practitioner' launch scope
  },
  {
    "FilterType": "Device", //Filter on a Device compartment if a 'device' launch
    ↪ scope is in the auth token
    "FilterArgument": "identifier=#device#" //... for the Device that has an
    ↪ identifier matching the value of that 'device' launch scope
  }
],
"Authority": "url-to-your-identity-provider",
"AdditionalEndpointBaseAddresses": ["different-base-url-that-is-also-used-by-your-
    ↪ identity-provider"], // optional, only needed for certain identity provider setups
"Audience": "name-of-your-fhir-server" //Default this is empty
// "ClaimsNamespace": "http://smarthealthit.org",
"RequireHttpsToProvider": false, //You want this set to true (the default) in a
    ↪ production environment!
"Protected": {
  "InstanceLevelInteractions": "read, vread, update, delete, history, conditional_
    ↪ delete, conditional_update, $validate",
  "TypeLevelInteractions": "create, search, history, conditional_create",
  "WholeSystemInteractions": "batch, transaction, history, search"
}
}

```

- **Enabled:** With this setting you can disable ('false') the authentication and authorization altogether. When it is enabled ('true'), Firely Server will also evaluate the other settings. The default value is 'false'. This implies that authorization is disabled as if no SmartAuthorizationOptions section is present in the settings.
- **Filters:** Defines how different launch contexts are translated to search arguments. See [Compartment](#) for more background.
 - **FilterType:** Both a launch context and a CompartmentDefinition are defined by a resourcetype. Use FilterType to define for which launch context and related CompartmentDefinition this Filter is applicable.
 - **FilterArgument:** Translates the value of the launch context to a search argument. You can use any supported search parameter defined on FilterType. It should contain the name of the launch context enclosed in hashes (e.g. #patient#), which is substituted by the value of the claim.
- **Authority:** The base url of your identity provider, such that {{base_url}}/.well-known/openid-configuration returns a valid configuration response ([OpenID Connect Discovery documentation](#)). At minimum, the jwks_uri, token_endpoint and authorization_endpoint keys are required in addition to the keys required by the specification. See [Set up an Identity Provider](#) for more background.
- **AdditionalEndpointBaseAddresses:** Optional configuration setting. Add additional base authority endpoints that your identity provider also uses for operations that are listed in the .well-known document.
- **Audience:** Defines the name of this Firely Server instance as it is known to the Authorization server. Default is

‘firelyserver’.

- **ClaimsNamespace:** Some authorization providers will prefix all their claims with a namespace, e.g. `http://my.company.org/auth/user/*.read`. Configure the namespace here to make Firely Server interpret it correctly. It will then strip off that prefix and interpret it as just `user/*.read`. By default no namespace is configured.
- **RequireHttpsToProvider:** Token exchange with an Authorization server should always happen over https. However, in a local testing scenario you may need to use http. Then you can set this to ‘false’. The default value is ‘true’.
- **Protected:** This setting controls which of the interactions actually require authentication. In the example values provided here, `$validate` is not in the `TypeLevelInteractions`. This means that you can use `POST [base-url]/Patient/$validate` without authorization. Since you only read Conformance resources with this interaction, this might make sense.

9.1.8 Compartments

In FHIR a [CompartmentDefinition](#) defines a set of resources ‘around’ a focus resource. For each type of resource that is linked to the focus resource, it defines the reference search parameters that connect the two together. The type of the focus-resource is in `CompartmentDefinition.code`, and the relations are in `CompartmentDefinition.resource`. The values for param in it can be read as a [reverse chain](#).

An example is the [Patient CompartmentDefinition](#), where a Patient resource is the focus. One of the related resource-types is Observation. Its params are subject and performer, so it is in the compartment of a specific Patient if that Patient is either the subject or the performer of the Observation.

FHIR defines CompartmentDefinitions for Patient, Encounter, RelatedPerson, Practitioner and Device. Although Firely Server is functionally not limited to these five, the specification does not allow you to define your own. Firely Server will use a CompartmentDefinition if:

- the CompartmentDefinition is known to Firely Server, see [Conformance Resources](#) for options to provide them.
- the OAuth2 Token contains a claim with the same name as the `CompartmentDefinition.code` (but it must be lowercase).

So some of the launch contexts mentioned in SMART on FHIR map to CompartmentDefinitions. For example, the launch context ‘launch/patient’ and ‘launch/encounter’ map to the compartment ‘Patient’ and ‘Encounter’. Please note that launch contexts can be extended for any resource type, but not all resource types have a matching CompartmentDefinition, e.g. ‘Location’.

A CompartmentDefinition defines the relationships, but it becomes useful once you combine it with a way of specifying the actual focus resource. In SMART on FHIR, the launch context can do that, e.g. `patient=123`. As per the [SMART Scopes and Launch Context](#), the value ‘123’ is the value of the Patient.id. Together with the Patient CompartmentDefinition this defines a – what we call – Compartment in Firely Server:

- Patient with id ‘123’
- And all resources that link to that patient according to the Patient CompartmentDefinition.

There may be cases where the logical id of the focus resource is not known to the authorization server. Let’s assume it does know one of the Identifiers of a Patient. The Filters in the [Configuration](#) allow you to configure Firely Server to use the identifier search parameter as a filter instead of `_id`. The value in the configuration example does exactly that:

```
"Filters": [
  {
    "FilterType": "Patient", //Filter on a Patient compartment if a 'patient' launch
    scope is in the auth token
    "FilterArgument": "identifier=#patient#" //... for the Patient that has an
```

(continues on next page)

(continued from previous page)

```

    ↪ identifier matching the value of that 'patient' launch scope
  },
  ...
]

```

Please notice that it is possible that more than one Patient matches the filter. This is intended behaviour of Firely Server, and it is up to you to configure a search parameter that is guaranteed to have unique values for each Patient if you need that. You can always stick to the SMART on FHIR default of `_id` by specifying that as the filter:

```

"Filters": [
  {
    "FilterType": "Patient", //Filter on a Patient compartment if a 'patient' launch
    ↪ scope is in the auth token
    "FilterArgument": "_id=#patient#" //... for the Patient that has an identifier
    ↪ matching the value of that 'patient' launch scope
  },
  ...
]

```

But you can also take advantage of it and allow access only to the patients from a certain General Practitioner, of whom you happen to know the Identifier:

```

"Filters": [
  {
    "FilterType": "Patient", //Filter on a Patient compartment if a 'patient' launch
    ↪ scope is in the auth token
    "FilterArgument": "general-practitioner.identifier=#patient#" //... for the Patient
    ↪ that has an identifier matching the value of that 'patient' launch scope
  },
  ...
]

```

In this example the claim is still called 'patient', although it contains an Identifier of a General Practitioner. This is because the `CompartmentDefinition` is selected by matching its code to the name of the claim, regardless of the value the claim contains.

If multiple resources match the `Compartment`, that is no problem for Firely Server. You can simply configure the Filters according to the business rules in your organization.

9.1.9 Tokens

When a client application wants to access data in Firely Server on behalf of its user, it requests a token from the authorization server (configured as the Authority in the [Configuration](#)). The configuration of the authorization server determines which claims are *available* for a certain user, and also for the client application. The client app configuration determines which claims it *needs*. During the token request, the user is usually redirected to the authorization server, which might or might not be the authentication server as well, logs in and is then asked whether the client app is allowed to receive the requested claims. The client app cannot request any claims that are not available to that application. And it will never get any claims that are not available to the user. This flow is also explained in the [SMART App Authorization Guide](#).

The result of this flow should be a JSON Web Token (JWT) containing zero or more of the claims defined in SMART on FHIR. The claims can either be scopes or a launch context, as in the examples listed in [Authorization](#). This token is encoded as a string, and must be sent to Firely Server in the Authorization header of the request.

A valid access token for Firely Server at minimum will have:

- the iss claim with the base url of the OAuth server
- the aud the same value you've entered in `SmartAuthorizationOptions.Audience`
- the scope field with the scopes granted by this access token
- optionally, the compartment claim, if you'd like to limit this token to a certain compartment. Such a claim may be requested by using a context scope or launching a part of an EHR launch. See [Requesting context with scopes](#) for more details. For example, in case of Patient data access where the `launch/patient` scope is used, include the `patient` claim with the patient's id or identifier (*Compartments*) and make sure to request the `patient/<permissions>` scope permissions. Compartment claims combined with `user/<permissions>` claims are not taken into account.

9.1.10 Access Control Decisions

In this paragraph we will explain how Access Control Decisions are made for the various FHIR interactions. For the examples assume a Patient Compartment with identifier=123 as filter.

1. Search

a. Direct search on compartment type

Request

GET [base]/Patient?name=fred

Type-Access

User must have read access to Patient, otherwise a 401 is returned.

Compartment

If a Patient Compartment is active, the Filter from it will be added to the search, e.g. GET [base]/Patient?name=fred&identifier=123

b. Search on type related to compartment

Request

GET [base]/Observation?code=x89

Type-Access

User must have read access to Observation, otherwise a 401 is returned.

Compartment

If a Patient Compartment is active, the links from Observation to Patient will be added to the search. In pseudo code: GET [base]/Observation?code=x89& (subject:Patient.identifier=123 OR performer:Patient.identifier=123)

c. Search on type not related to compartment

Request

GET [base]/Organization

Type-Access

User must have read access to Organization, otherwise a 401 is returned.

Compartment

No compartment is applicable to Organization, so no further filters are applied.

d. Search with include outside the compartment

Request

GET [base]/Patient?_include=Patient:organization

Type-Access

User must have read access to Patient, otherwise a 401 is returned. If the user has read access to Organization, the `_include` is evaluated. Otherwise it is ignored.

Compartment

Is applied as in case 1.a.

e. Search with chaining

Request

GET `[base]/Patient?general-practitioner.identifier=123`

Type-Access

User must have read access to Patient, otherwise a 401 is returned. If the user has read access to Practitioner, the search argument is evaluated. Otherwise it is ignored as if the argument was not supported. If the chain has more than one link, read access is evaluated for every link in the chain.

Compartment

Is applied as in case 1.a.

f. Search with chaining into the compartment

Request

GET `[base]/Patient?link:Patient.identifier=456`

Type-Access

User must have read access to Patient, otherwise a 401 is returned.

Compartment

Is applied to both Patient and link. In pseudo code: GET `[base]/Patient?link:(Patient.identifier=456&Patient.identifier=123)&identifier=123`
In this case there will probably be no results.

2. Read: Is evaluated as a Search, but implicitly you only specify the `_type` and `_id` search parameters.
3. VRead: If a user can Read the current version of the resource, he is allowed to get the requested version as well.
4. Create

a. Create on the compartment type

Request

POST `[base]/Patient`

Type-Access

User must have write access to Patient. Otherwise a 401 is returned.

Compartment

A Search is performed as if the new Patient were in the database, like in case 1.a. If it matches the compartment filter, the create is allowed. Otherwise a 401 is returned.

b. Create on a type related to compartment

Request

POST `[base]/Observation`

Type-Access

User must have write access to Observation. Otherwise a 401 is returned. User must also have read access to Patient, in order to evaluate the Compartment.

Compartment

A Search is performed as if the new Observation were in the database, like in case 1.b. If it matches the compartment filter, the create is allowed. Otherwise a 401 is returned.

- c. Create on a type not related to compartment

Request

POST [base]/Organization

Type-Access

User must have write access to Organization. Otherwise a 401 is returned.

Compartment

Is not evaluated.

5. Update

- a. Update on the compartment type

Request

PUT [base]/Patient/123

Type-Access

User must have write access *and* read access to Patient, otherwise a 401 is returned.

Compartment

User should be allowed to Read Patient/123 and Create the Patient provided in the body. Then Update is allowed.

- b. Update on a type related to compartment

Request

PUT [base]/Observation/xyz

Type-Access

User must have write access to Observation, and read access to both Observation and Patient (the latter to evaluate the compartment)

Compartment

User should be allowed to Read Observation/123 and Create the Observation provided in the body. Then Update is allowed.

6. Delete: Allowed if the user can Read the current version of the resource, and has write access to the type of resource.
7. History: Allowed on the resources that the user is allowed to Read the current versions of (although it is theoretically possible that an older version would not match the compartment).

9.1.11 Testing

Testing the access control functionality is possible on a local instance of Firely Server. It is not available for the [publicly hosted test server](#).

You can test it using a dummy authorization server and Postman as a REST client. Please refer to these pages for instructions:

- [Set up an Identity Provider](#)
- [Access Control Tokens with Postman](#)

You might also find it useful to enable more extensive authorization failure logging - Firely Server defaults to a secure setup and does not show what exactly went wrong during authorization. To do so, set the `ASPNETCORE_ENVIRONMENT` environment variable to `Development`.

FEATURES

Firely Server offers many features as defined in the FHIR Specification and beyond.

10.1 FHIR RESTful API

Firely Server supports most of the features in the [FHIR RESTful API](#).

10.1.1 FHIR Versions

All the operations below can be called for FHIR STU3 or FHIR R4. Firely Server supports the `fhirVersion` mimetype parameter and `fhir version` endpoint mappings for that purpose. See [Multiple versions of FHIR](#) for more information.

10.1.2 Create, read, update, patch, delete

These five operations to manage the contents of the Firely Server, commonly referenced by the acronym CRUD, are implemented as per the specification. Patch is implemented as [FHIR Patch](#), as this is the most versatile one. This includes version-read and the conditional variations. Only a few limitations apply.

Firely Server enables create-on-update: If you request an update and no resource exists for the given id, the provided resource will be created under the provided id.

Firely Server can reject a resource based on [Validating incoming resources](#).

Configuration

A conditional delete interaction may match multiple resources. You can configure the server to delete all matches, or reject the operation (effectively only allowing single matches to be deleted). Allowing multiple deletes requires support for transactions on the database (SQL Server or SQLite). If you allow for multiple deletes, you have to specify a maximum number of resources that can be deleted at once, to save you from accidentally deleting too many resources.

```
"FhirCapabilities": {
  "ConditionalDeleteOptions": {
    "ConditionalDeleteType": "Single", // Single or Multiple,
    "ConditionalDeleteMaxItems": 1
  }
}
```

Limitations on CRUD

1. Simultaneous conditional creates and updates are not entirely transactionally safe:
 - Two conditional updates may both result in a `create`, although the result of one may be a match to the other.
 - Two conditional creates may both succeed, although the result of one may be a match to the other.
 - A conditional create and a simultaneous conditional update may both result in a `create`, although the result of one may be a match to the other.
2. Parameter `_pretty` is not yet supported.
3. XML Patch and JSON Patch are not supported.

10.1.3 Versioning

Firely Server keeps a full version history of every resource, including the resources on the *Firely Server Administration API*.

10.1.4 Search

Search is supported as per the specification, with a few *Limitations on search*.

In the default configuration the SearchParameters from the *FHIR specification* are available. But Firely Server also allows *Custom Search Parameters*.

Chaining and reverse chaining is fully supported.

Quantity search on UCUM quantities automatically converts units to a canonical form. This means you can have kg in an Observation and search by lbs, or vice versa.

Compartment Search is supported.

Firely Server also supports `_include:iterate` and `_revinclude:iterate`, as well as its STU3 counterparts `_include:recurse` and `_revinclude:recurse`. See *the specification* for the definition of those. You can configure the maximum level of recursion:

```
"FhirCapabilities": {
  "SearchOptions": {
    "MaximumIncludeIterationDepth": 1
  }
},
```

Sorting

`_sort` is implemented for searchparameters of types:

- string
- number
- uri
- reference
- datetime

- token

for the repositories:

- SQL
- SQLite
- Memory

How is sort evaluated?

- A searchparameter may be indexed with multiple values for a single resource. E.g. Patient.name for Angelina Jolie would have name=Angelina and name=Jolie. And George Clooney: name=George and name=Clooney. As the FHIR Specification phrases it: “In this case, the sort is based on the item in the set of multiple parameters that comes earliest in the specified sort order when ordering the returned resources.” Here is an example of how Firely Server evaluates this.

- In ascending order: Patient?_sort=name

Name values	Asc. per resource	Asc. resources
Angelina	Angelina	<i>Angelina Jolie</i>
Jolie	Jolie	
George	Clooney	<i>George Clooney</i>
Clooney	George	

- Now in descending order: Patient?_sort=-name

Name values	Desc. per resource	Desc. resources
Angelina	Jolie	<i>Angelina Jolie</i>
Jolie	Angelina	
George	George	<i>George Clooney</i>
Clooney	Clooney	

- The searchparameter to sort on may not be indexed at all for some of the resources in the resultset. E.g. a Patient without any identifier will not be indexed for Patient.identifier. Resources not having that parameter always end up last (both in ascending and descending order). This is similar to the ‘nulls last’ option in some SQL languages.
- Token parameters are sorted only on their code element. The system element is ignored in the sorting.
- Firely Server uses the default collation as configured on the database server. This collation defines the ordering of characters.

Limitations on search

The following parameters and options are not yet supported:

1. _text
2. _content
3. _query
4. _containedType
5. _filter

6. `:approx` modifier on a quantity `SearchParameter`
7. `:text` modifier on a string `SearchParameter`
8. `:above`, `:below`, `:in`, `:not-in` modifiers on a token `SearchParameter`
9. `:above` on a uri `SearchParameter` (`:below` is supported)
10. `*` wildcard on `_include` and `_revinclude`
11. `_pretty`

Furthermore:

1. Paging is supported, but it is not isolated from intermediate changes to resources.

10.1.5 History

History is supported as described in the specification, on the system, type and instance level. The `_since` and `_count` parameters are also supported.

Configuration

```
"HistoryOptions": {  
  "MaxReturnedResults": 100  
}
```

If a `_history` call would result in more than `MaxReturnedResults`, Firely Server asks the user to be more specific. Use this to avoid overloading the server or the connection.

Limitations on history

1. `_at` parameter is not yet supported.
2. Paging is supported, but it is not isolated from intermediate changes to resources.

10.1.6 Batch

Batch is fully supported on the usual endpoint. You can limit the number of entries accepted in a single batch. See [Batch and transaction](#).

Note that batches are not supported in the `/administration` endpoint.

10.1.7 Transaction

Transactions are supported, with these limitations:

1. Of the three storage implementations, only SQL Server and SQLite truly support transactions. On [MongoDB](#) and [Memory](#), transaction support can be simulated at the FHIR level, but not be enforced on the database level.
2. References between resources in the transaction can point backwards or forwards. Only circular references are not supported.
3. The `/administration` endpoint does not support transactions.

You can limit the number of entries accepted in a single transaction. See [Batch and transaction](#).

10.1.8 Capabilities

On the Capabilities interaction (<firely-server-endpoint>/metadata) Firely Server returns a CapabilityStatement that is built dynamically from the supported ResourceTypes, SearchParameters and interactions. E.g. if you *Configure Search Parameters*, the SearchParameters that are actually loaded appear in the CapabilityStatement.

10.1.9 Not supported interactions

These interactions are not yet supported by Firely Server:

1. HEAD

Besides that, Firely Server does not yet return the date header as specified in [HTTP return values](#)

10.2 Bulk Data Export

Firely Server provides the option to export resources with the Bulk Data Export Service. The Bulk Data Export Service enables the \$export operation from the Fhir specification. Read more about the [\\$export request flow](#)

10.2.1 Appsettings

To start using the Bulk Data Export Service (BDE) you will first have to add the plugin to the PipelineOptions in the appsettings.

```
"PipelineOptions": {
  "PluginDirectory": "./plugins",
  "Branches": [
    {
      "Path": "/",
      "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",
        "Vonk.Fhir.R4",
        //"Vonk.Fhir.R5",
        "Vonk.Repository.Sql.SqlVonkConfiguration",
        "Vonk.Repository.Sqlite.SqliteVonkConfiguration",
        "Vonk.Repository.MongoDb.MongoDbVonkConfiguration",
        "Vonk.Repository.CosmosDb.CosmosDbVonkConfiguration",
        "Vonk.Repository.Memory.MemoryVonkConfiguration",
        "Vonk.Subscriptions",
        "Vonk.Smart",
        "Vonk.UI.Demo",
        "Vonk.Plugin.DocumentOperation.DocumentOperationConfiguration",
        "Vonk.Plugin.ConvertOperation.ConvertOperationConfiguration",
        "Vonk.Plugin.BinaryWrapper",
        "Vonk.Plugin.Audit",
        "Vonk.Plugins.TerminologyIntegration",
        "Vonk.Plugin.BulkDataExport"
      ],
      "Exclude": [
        "Vonk.Subscriptions.Administration"
      ]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
    ]  
  }, ...etc...
```

Note: We did not implement BDE for all database types. Make sure the admin database is configured for either SQL Server or SQLite and the data database is configured for SQL Server.

BDE introduces two new parts to the appsettings, namely TaskFileManagement and BulkDataExport.

```
"TaskFileManagement": {  
  "StoragePath": "./taskfiles"  
},  
"BulkDataExport": {  
  "RepeatPeriod" : 60000, //ms  
  "AdditionalResources": [ "Organization", "Location", "Substance", "Device",  
↪ "BodyStructure", "Medication", "Coverage" ]  
},
```

In StoragePath you can configure the folder where the exported files will be saved to. Make sure the server has write access to this folder.

In RepeatPeriod you can configure the polling interval (in milliseconds) for checking the Task queue for a new export task.

A patient-based or group-based Bulk Data Export returns resources based on the Patient compartment definition (<https://www.hl7.org/fhir/compartimentdefinition-patient.html>). These resources may reference resources outside the compartment as well, such as a Practitioner who is the performer of a Procedure. Using the *AdditionalResources*-setting, you can determine which types of referenced resources are exported in addition to the compartment resources.

10.2.2 \$export

There are three different levels for which the \$export operation can be called:

System

url: [firely-server-base]/\$export

This will create a system level export task, exporting all resources in the Firely Server database to a .ndjson file per resourcetype.

Patient

url: [firely-server-base]/Patient/\$export

This will create a type level export task, exporting all resources included in the Patient Compartment in the Firely Server database to an .ndjson file per resourcetype.

Group

url: [firely-server-base]/Group/<group-id>/\$export

This will create an instance level export task. For each Patient in the Group, the task will export all resources included in the Patient Compartment in the Firely Server database to an .ndjson file per resourcetype.

Note: For now we only support inclusion in a Group through Group.member.

Making an \$export request will create a new task in the database with status “Queued”. The request should return an absolute **\$exportstatus** URL in the Content-Location header and the OperationOutcome in the body.

10.2.3 \$exportstatus

The \$export request should return the \$exportstatus url for your export task. This url can be used to request the current status of the task through a GET request, or to cancel the task through a DELETE request.

There are six possible status options:

1. Queued
 2. Active
 3. Complete
 4. Failed
 5. CancellationRequested
 6. Cancelled
- If a task is Queued or Active, GET \$exportstatus will return the status in the X-Progress header
 - If a task is Complete, GET \$exportstatus will return the results with a **\$exportfilerequest** url per exported .ndjson file. This url can be used to retrieve the files per resourcetype. If there were any problems with parts of the export, an url for the generated OperationOutcome resources can be found in the error section of the result.
 - If a task is Failed, GET \$exportstatus will return HTTP Statuscode 500 with an OperationOutcome.
 - If a task is on status CancellationRequested or Cancelled, GET \$exportstatus will return HTTP Statuscode 410 (Gone).

10.2.4 \$exportfilerequest

If a task has the Complete status, the GET \$exportstatus request should return one or more \$exportfilerequest urls. Performing a GET request on this \$exportfilerequest url returns a body of FHIR resources in newline delimited json (ndjson).

Note: The Accept header for this request has to be:

```
application/fhir+ndjson
```

10.3 Patient \$everything

10.3.1 Description

This plugin implements the Patient \$everything operation, as described in <https://www.hl7.org/fhir/operation-patient-everything.html>. This operation returns all resources associated with a single patient. The resources returned are determined by the Patient compartment, defined in <https://www.hl7.org/fhir/compartementdefinition-patient.html>. Currently, the functionality is only available if you use SQL server for your data.

```
GET <base-url>/Patient/<patient-id>/$everything
Accept: <any supported FHIR media type>
```

Optional parameters:

- `_since`: Get only resources changed since this moment
- `_type`: Limit the returned resource types to only the types in this list

Please note that other defined operation parameters have not been implemented (yet).

So if you would want to fetch only Patient 1 and their Observations, changed since the 1st of January, 2021 in FHIR JSON format, you would use:

```
GET <base-url>/Patient/1/$everything?_type=Patient,Observation&_since=2021-01-01
Accept: application/fhir+json
```

10.3.2 Configuration

Many resources in the Patient compartment reference resources outside the compartment. For example: An Observation might have a performer which is a Practitioner. As Practitioner itself is not in the Patient compartment, the resource would normally not be returned. But using a setting you can control which additional resource types are returned if they are referenced by any of the resources you requested.

```
"PatientEverythingOperation": {
  "AdditionalResources": [ "Organization", "Location", "Substance", "Device",
    ↪ "Medication" ]
}
```

This is the default value for the setting. As you can see, Practitioner is not included by default out of privacy considerations but you can change that by overriding the setting.

Note: Device was added as an additional resource as it includes a reference to a patient but is not listed in the patient's compartment yet. As soon as the specification is updated, it will be removed from the appsettings and returned by default.

To include the plugin in your pipeline, add the following extra Include:

```
"PipelineOptions": {
  "Branches": [
    {
      "Path": "/",
      "Include": [
        ...
        "Vonk.Plugin.PatientEverything"
```

(continues on next page)

(continued from previous page)

```

    ],
    },
    ...
  ]
}

```

10.3.3 License

The Patient \$everything operation is licensed. To use it, you may need to renew your license.

10.4 Custom Operations

10.4.1 Validation

Firely Server can validate a resource against a profile as defined in the `$validate` operation.

You can call validate on three levels:

1. *Validate on the system level*
2. *Validate on the ResourceType level*
3. *Validate an instance from the database*

Besides that you can configure Firely Server to validate every incoming resource and even filter on specific profiles. See the section on *Validating incoming resources*. In all cases, the *Precondition* is that Firely Server must have access to all relevant StructureDefinitions.

Validation has some *Limitations*.

Note: The very first validation call will take a considerable amount of time, typically around 5 seconds. This is because Firely Server maintains a cache of validation information, and on the first call that cache is still empty. Subsequent calls are much faster.

Validate on the system level

```
POST <firely_server_endpoint>/$validate[?profile=<canonical-url-of-structuredefinition>]
```

There are two ways of calling \$validate:

1. With a Resource or a Bundle of resources as body, and optionally, a profile parameter on the url.
2. With a Parameters resource as body, having
 - a parameter element with the Resource to validate in the resource parameter;
 - (optionally) the profile to validate against in the profile parameter

In both cases the request must have a Content-Type header matching the format of the body (`application/fhir+json` or `application/fhir+xml`).

If you do not specify a profile parameter, Firely Server will validate the Resource against any profiles mentioned in `meta.profile` as well as the base profile from the FHIR Specification.

If you call \$validate on the system level, Firely Server will make no assumptions about the ResourceType of the Resource to validate.

Validate on the ResourceType level

```
POST <firely_server_endpoint>/<resourcetype>/$validate[?profile=<canonical-url-of-  
↳structuredefinition>]
```

You can call \$validate in the same two ways as with *Validate on the system level*.

If you call \$validate on the ResourceType level, Firely Server will check whether the Resource to validate is of the same <resourcetype> as provided in the url.

Validate an instance from the database

```
GET <firely_server_endpoint>/<resourcetype>/<id>/$validate[?profile=<canonical-url-of-  
↳structuredefinition>]
```

This time you can only use the (optional) profile parameter on the url to specify a StructureDefinition to validate against.

Precondition

Firely Server must be aware of all the StructureDefinitions referenced directly via parameter or indirectly by a profile in meta.profile. Refer to the *Conformance Resources* for more information.

Limitations

1. The mode parameter is not yet supported.
2. Implicit ValueSets (ones that use the .filter property) are not supported - create explicit ones instead (without the .filter property).

10.4.2 Snapshot generation

Firely Server is capable of generating a snapshot for a StructureDefinition. This operation is not defined in the FHIR Specification.

You can invoke this operation with

```
POST <firely-server-endpoint>/StructureDefinition/$snapshot
```

- The body must contain the StructureDefinition that you want filled with a fresh snapshot. The StructureDefinition may contain an existing snapshot, it will be ignored.
- The Content-Type header must match the format of the body (application/fhir+json or application/fhir+xml)

Firely Server will return the same StructureDefinition, but with the snapshot element (re-)generated.

Note: The very first call to \$snapshot will take a considerable amount of time, typically around 5 seconds. This is because Firely Server maintains a cache of StructureDefinition information, and on the first call that cache is still empty. Subsequent calls are much faster.

Precondition

Firely Server must be aware of all the other StructureDefinitions that are referred to by the StructureDefinition in the body of the request. Refer to the [Conformance Resources](#) for more information.

10.4.3 \$meta

Firely Server provides an implementation of the \$meta operation as defined in the [FHIR Specification](#).

By default the operation is only enabled on the level of a resource instance. It can also be enabled on the level of a resourcetype or system wide, but the cost of execution will then be high. On sufficient customer demand an optimized implementation is possible.

The FHIR Specification [operations framework](#) allows for the definition of custom operations and defines how to offer them in the [FHIR RESTful API](#). Firely Server offers two custom operations out of the box.

1. *Validation*
2. *Snapshot generation*

10.5 Custom Resources

Custom Resources are not formally defined in the FHIR Specification. To Firely Server a Custom Resource is a resource with a definition that is a specialization of DomainResource, but that is not in the Core FHIR Specification. Firely Server can handle these, provided it knows about the StructureDefinition that defines it. This page explains how to register such a StructureDefinition and store custom resources.

Warning: Custom Resources are not interoperable. Do not use them for exchanging resources outside your own programming control boundaries.

10.5.1 What to use them for?

Firely Server can be used as a platform to build apps on. In these apps, structures arise outside of the FHIR Specification or even the Health domain. Still, it would be useful to also use Firely Server to store, search and version these structures. Note that this is only for internal use in the app.

10.5.2 Register the definition

Just like any resourcetype, the definition for a custom resource is formalized as a StructureDefinition. Firely Server will recognize it as the definition of a custom resourcetype if and only if:

- base = DomainResource
- derivation = specialization
- kind = resource
- abstract = false

This also means that a Logical Model as-is cannot be used as the definition of a custom resourcetype.

Examples of these can be found in the specification: each resourcetype is defined this way. The easiest way to get started is with the definition of Basic (in [xml](#) or [json](#)), and adjust that:

1. Choose a name for the type, let's say 'Foo'.
2. Choose a url for the type. In STU3 this has to start with <http://hl7.org/fhir/StructureDefinition/> (constraint sdf-7), so <http://hl7.org/fhir/StructureDefinition/Foo> makes sense. Note that in R4 you are encouraged to use a url in a domain that you control and *not* within hl7.org.
3. Make sure the id, name and type elements align with the name 'Foo'.
4. Adjust the description
5. Make sure all the elements in the differential start with 'Foo.'
6. (Recommended) Store your definition in Simplifier.net for version management, comments and collaboration.

If you have created the StructureDefinition, register it in Firely Server using any of the methods mentioned in [Conformance Resources](#). As an example we will issue an update interaction on the Administration API:

```
PUT <base-url>/administration/StructureDefinition/Foo
Content-Type=application/fhir+json; fhirVersion=3.0
```

By using an update we can choose the id and hence the location of this StructureDefinition. Firely Server does this by default for all the resourcetypes defined by the specification as well.

10.5.3 Use a resource

To test whether you can actually use the endpoint associated with your custom resourcetype, try a search on it: GET <base-url>/Foo. This should return an empty bundle. If you get an OperationOutcome with the text "Request for not-supported ResourceType(s) Foo", the registration of the definition is not correct.

Note: The CapabilityStatement will not list the custom definition. This is because the element CapabilityStatement.rest.resource.type has a Required binding to the ResourceType valueset. And obviously this valueset does not contain our 'Foo' resourcetype.

Now use your favorite editor to create a resource that conforms to the Foo StructureDefinition. And then create it on Firely Server: POST <base-url>/Foo.

All the operations on specification-defined resourcetypes are also available for custom resources. You can also use them in a batch or transaction bundle. Custom Resources can also be validated. This also means that [Validating incoming resources](#) can be used in conjunction with Custom Resources.

10.5.4 Search parameters on a custom resource

In Firely Server you can define your own custom search parameters on any type of resource (see [Custom Search Parameters](#)). This includes Custom Resources. Just use the type name of the Custom Resource in the SearchParameter.base.

10.6 Terminology services

Firely Server provides support for Terminology operations and validation against terminologies. This is done through a local implementation based on the Administration database ('Local Terminology Service') and - if configured - extended with external FHIR Terminology Servers ('Remote Terminology Services'). The configuration allows you to set preferred services for each CodeSystem and ValueSet. Firely Server will then transparently select and query either the Local or one of the Remote Terminology Services.

Of the operations listed below the following can be supported by the Local Terminology Service: \$validate-code, \$expand, \$lookup, \$find-matches. Note that it only supports simple ValueSets and CodeSystems like the ones part of the FHIR specification. It cannot support complex terminologies like LOINC or SNOMED-CT.

The terminology operations can be invoked for different FHIR versions as specified in *Multiple versions of FHIR*.

10.6.1 Terminology Integration

In earlier versions of Firely Server, local terminology services were separately configured from so-called Terminology Integration. These are now merged.

Operations

ValueSet \$validate-code

definition

<http://www.hl7.org/implement/standards/fhir/valueset-operation-validate-code.html>

notes

- Available on the type level <firely-server-endpoint>/administration/ValueSet/\$validate-code and the instance level <firely-server-endpoint>/administration/ValueSet/<id>/\$validate-code.
- Only the parameters url, valueSet, valueSetVersion, code, system, display, coding, codeableConcept, abstract are supported.
- The url and valueSetVersion input parameters are only taken into consideration if no valueSet resource was provided in the body. So the valueSet in the body takes priority.
- Both GET and POST interactions are available.

ValueSet \$expand

definition

<http://www.hl7.org/implement/standards/fhir/valueset-operation-expand.html>

notes

- Available on the type level <firely-server-endpoint>/administration/ValueSet/\$expand and the instance level <firely-server-endpoint>/administration/ValueSet/<id>/\$expand.
- Only the parameters url, valueSet, valueSetVersion and includeDesignations are supported.
- The url and valueSetVersion input parameters are only taken into consideration if no valueSet resource was provided in the body. So the valueSet in the body takes priority.
- Both GET and POST interactions are available.

CodeSystem \$lookup

definition

<http://www.hl7.org/implement/standards/fhir/codesystem-operation-lookup.html>

notes

- Available on the type level `<firely-server-endpoint>/administration/CodeSystem/$lookup`.
- Only the parameters code, system, version, coding and date are supported.
- Code & system combination takes priority over the coding parameter.
- Both GET and POST interactions are available.

CodeSystem \$find-matches / \$compose

definition

<http://www.hl7.org/implement/standards/fhir/codesystem-operation-find-matches.html>

notes

- Available on the type level `<firely-server-endpoint>/administration/CodeSystem/$find-matches` and the instance level `<firely-server-endpoint>/administration/CodeSystem/<id>/$find-matches`.
- Only the parameters system, exact, version, property.code and property.value are supported.
- The url and valueSetVersion input parameters are only taken into consideration if no valueSet resource was provided in the body. So the valueSet in the body takes priority.
- Both GET and POST interactions are available.
- \$find-matches was named \$compose in FHIR STU3. The operation is supported with both names.

CodeSystem \$subsumes

definition

<http://www.hl7.org/implement/standards/fhir/codesystem-operation-subsumes.html>

notes

- Available on the type level `<firely-server-endpoint>/administration/CodeSystem/$subsumes`.
- Only the parameters codeA, codeB, system, and version are supported.
- The system input parameters is only taken into when called on the type level.
- Both GET and POST interactions are available.

ConceptMap \$closure

definition

<http://www.hl7.org/implement/standards/fhir/conceptmap-operation-closure.html>

notes

- Available on the system level <firely-server-endpoint>/administration/\$closure.
- This operation is passed on to a Remote Terminology Service supporting it. It supports any parameters that the Remote service supports.
- Only POST interactions are available.

ConceptMap \$translate

definition

<http://www.hl7.org/implement/standards/fhir/conceptmap-operation-translate.html>

notes

- Available on the instance level <firely-server-endpoint>/administration/ConceptMap/[id]/\$translate and the type level <firely-server-endpoint>/administration/ConceptMap/\$translate.
- Only the parameters url, conceptMap (on POST), conceptMapVersion, code, system, version, source, target, targetsystem and reverse are supported.
- Both GET and POST interactions are available.

Configuration

10.6.2 Pipeline

Make sure to add the `Vonk.Plugins.Terminology` plugin to the `PipelineOptions` in `appsettings` in order to make use of the `TerminologyIntegration` plugin. Additionally, to the “/administration” pipeline, `Vonk.Plugins.Terminology` can be used on the regular FHIR pipeline “/”. Please note that in this case, `CodeSystems` and `ValueSets` are resolved from the Administration repository when executing a terminology operation and the corresponding resource is not provided as part of the request as a parameter.

```
"PipelineOptions": {
  "PluginDirectory": "./plugins",
  "Branches": [
    {
      "Path": "/",
      "Include": [...]
    },
    {
      "Path": "/administration",
      "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",
        "Vonk.Fhir.R4",
        "Vonk.Administration",
        ...
        "Vonk.Plugins.Terminology"
      ]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    ],
    "Exclude": [
        "Vonk.Subscriptions.Administration"
    ]
}, ...etc...

```

To include or exclude individual operations in the pipeline, see the available plugins under *Terminology*.

Also make sure that the terminology operations are allowed at all in the SupportedInteractions section:

```

"SupportedInteractions": {
  "InstanceLevelInteractions": "$validate-code, $expand, $compose, $translate, $subsumes
  ↪ ",
  "TypeLevelInteractions": "$validate-code, $expand, $lookup, $compose, $translate,
  ↪ $subsumes",
  "WholeSystemInteractions": "$closure"
},

```

Lastly, operation on the administration endpoint can be limited to specific IP addresses:

```

"Administration": {
  "Security": {
    "AllowedNetworks": [ "127.0.0.1", "::1" ], // i.e.: [ "127.0.0.1", "::1" (ipv6 ↪
  ↪ localhost), "10.1.50.0/24", "10.5.3.0/24", "31.161.91.98" ]
    "OperationsToBeSecured": [ "$validate-code", "$expand", "$compose", "$translate", "
  ↪ $subsumes", "$lookup", "$closure" ]
  }
},

```

10.6.3 Options

You can enable the integration with one or more external terminology services by setting the required options in the appsettings file. There is a block for the Local Terminology Service and one for each Remote Terminology Service.

For each terminology service you can set the following options:

Order

The order of the terminology service, or the priority. If multiple Terminology services could be used for a request, Firely Server will use the priority to select a service. Terminology services are arranged in a ascending order: so 1 will be selected over 2.

PreferredSystem

If a request is directed at a specific code system, Firely Server will choose this terminology server over other available services. A system matches one of the preferred systems if the system starts with the preferred system. So `http://loinc.org` will match any CodeSystem or ValueSet with a canonical that starts with that url.

SupportedInteractions

The operations supported by the terminology service. Firely Server will only select this service if the operation is in this list. Valid values:

```

"ValueSetValidateCode"
"CodeSystemValidateCode"
"Expand"

```

(continues on next page)

(continued from previous page)

```
"FindMatches" / "Compose"
"Lookup"
"Translate"
"Subsumes"
"Closure"
```

SupportedInformationModels

The FHIR versions supported by the terminology service. Valid values:

```
"Fhir3.0"
"Fhir4.0"
"Fhir5.0"
```

Endpoint

The endpoint url where Firely Server can redirect the requests to.

Username

If the terminology service uses Basic Authentication, you can set the required username here.

Password

If the terminology service uses Basic Authentication, you can set the required password here.

MediaType

Default Media-Type that should be used for serialization of the Parameters resources forwarded to the external terminology service

Notes:

- The Endpoint, Username and Password settings are not valid for the Local Terminology Server, just for the Remote services.
- If a Remote Terminology Service has different endpoints for different FHIR versions, configure each endpoint separately.
- The SupportedInformationModels cannot be broader than the corresponding Fhir.Rx plugins configured in the PipelineOptions.

A sample Terminology section in the appsettings can look like this:

```
"Terminology": {
  "MaxExpansionSize": 650,
  "LocalTerminologyService": {
    "Order": 10,
    "PreferredSystems": [ "http://hl7.org/fhir" ],
    "SupportedInteractions": [ "ValueSetValidateCode", "Expand" ],
    "SupportedInformationModels": [ "Fhir3.0", "Fhir4.0", "Fhir5.0" ]
  },
  "RemoteTerminologyServices": [
    {
      "Order": 20,
      "PreferredSystems": [ "http://snomed.info/sct" ],
      "SupportedInteractions": [ "ValueSetValidateCode", "Expand", "Translate", "Subsumes",
        ↪ "Closure" ],
      "SupportedInformationModels": [ "Fhir4.0" ],

```

(continues on next page)

(continued from previous page)

```
    "Endpoint": "https://r4.ontoserver.csiro.au/fhir/",
    "MediaType": "application/fhir+xml"
  },
  {
    "Order": 30,
    "PreferredSystems": [ "http://loinc.org" ],
    "SupportedInteractions": [ "ValueSetValidateCode", "Expand", "Translate" ],
    "SupportedInformationModels": [ "Fhir3.0", "Fhir4.0" ],
    "Endpoint": "https://fhir.loinc.org/",
    "Username": "",
    "Password": ""
  }
],
},
```

This means if you execute a terminology operation request, Firely Server will check whether the request is correct, redirect it to the preferred terminology service and finally return the result.

Additionally to the remote and local terminology services, you can configure the maximum number of concepts that are allowed to be included in a local ValueSet expansion (MaxExpansionSize). ValueSets stored in the local administration database larger than the configured setting will not be expanded, hence they cannot be used for \$validate-code, \$validate or \$expand.

License

The Terminology plugin itself is licensed with the license token <http://fire.ly/vonk/plugins/terminology>.

When you configure Remote Terminology Services it is your responsibility to check whether you are licensed to use those services.

10.7 Custom Search Parameters

10.7.1 Configure Search Parameters

You can control which search parameters are known to Firely Server. This is managed in the same way as all the conformance resources, see *Conformance Resources*.

10.7.2 Re-indexing for new or changed SearchParameters

Firely Server extracts values from resources based on the available search parameters upon create or update. This means that if you already had resources in your database before adding a custom search parameter, those resources will not be indexed for that parameter. If you on the other hand removed a previously used search parameter, the index will contain superfluous data.

To fix that, you should re-index (repeat the extraction) for these parameters.

In short, both reindex operations below will:

- Return an Operation Outcome stating that the reindex procedure was started successfully.
- Run the actual reindex asynchronously, using a configured number of threads, thereby using most of the hardware resources available to Firely Server.

- Block any other requests for the duration of the reindex.
- Log progress in the log.

Caution: This is a possibly lengthy operation, so use it with care.

- Always try the reindex on a representative (sub)set of your data in a test environment to assess how long the operation may take in the production environment.
- Always make a backup of your data before performing a reindex.

Warning: During the re-index operation, all other operations are blocked and responded to with response code '423 - Locked'.

Reindexing and FHIR versions

Reindexing is also controlled by the `fhirVersion` parameter (see [Multiple versions of FHIR](#)) in the Accept header or the version-mapped endpoint. It will then reindex only for SearchParameters and resources *in that FHIR version*. So for a full reindex of everything you may need to issue the command twice, once for each `fhirVersion`.

Rebuild the whole search index

This is only needed if we changed something very significant to the way Firely Server searches, like

- The way values are extracted for all or many searchparameters.
- The structure in which Firely Server stores the search index.

To re-index all resources for all search parameters, use:

```
POST http(s)://<firely-server-endpoint>/administration/reindex/all
Accept=application/fhir+json (or xml); fhirVersion=3.0 (or 4.0)
```

This will delete any previously indexed data and extract it again from the resources.

Rebuild the search index for specific searchparameters

This is needed if:

- The definition (usually the `expression`) of a searchparameter has changed.
- A searchparameter was added.
- A searchparameter was removed and you want the search index to be tidy and not have this parameter in it anymore.

To re-index all resources for certain search parameters, use:

```
POST http(s)://<firely-server-endpoint>administration/reindex/searchparameters
Accept=application/fhir+json (or xml); fhirVersion=3.0 (or 4.0)
```

In the body of the POST, you put the name of the search parameters to actually re-index as form parameters:

```
include=Patient.name,Observation.code  
exclude=Organization.name
```

`include` means that resources will be re-indexed only for those search parameters. You use this if you added or changed one or few search parameters.

`exclude` means that any existing index data for those search parameters will be erased. You use this when you removed a search parameter.

Remember to adjust the Content-Type header: `application/x-www-form-urlencoded`.

If you are *not permitted* to perform the reindex, Firely Server will return statuscode 403.

Re-index Configuration

Firely Server will not re-index the resources in the database all at once, but in batches. The re-index operation will process all batches until all resources are re-indexed. You can control the size of the batches in the *Firely Server settings*. Besides that you can also control how many threads run in parallel to speed up the reindex process. The configured value is a maximum, since Firely Server will also be limited by the available computing resources.

```
"ReindexOptions": {  
  "BatchSize": 100,  
  "MaxDegreeOfParallelism": 10  
},
```

Use any integer value ≥ 1 .

Warning: CosmosDB in its default configuration (and on the CosmosDB emulator) is fairly limited in its throughput. If you encounter errors stating 'Request rate is large', you will have to:

- lower the MaxDegreeOfParallelism,
- restart Firely Server
- and start a the reindex operation again.

10.7.3 Limitations

Every search parameter has to have either:

- a valid FhirPath in it's Expression property, or
- be a Composite search parameter and specify at least one component.

10.8 Errata to the specification

The FHIR Specification is good, but not perfect. Some of the SearchParameters have errors. If we find these errors, we report them in the issue tracking system of HL7. But it takes time until the fix is applied to the specification. In the meantime Firely Server provides you with updated versions of the resources that have errors, so you can use them already while we await the fixes in the specification.

These corrections come with the Firely Server installation, in the files:

- `errataFhir3.0.zip`, with corrections to the STU3 version of the Specification
- `errataFhir4.0.zip`, with corrections to the R4 version of the Specification

These files are imported automatically during startup, as are other conformance resources, see [Conformance Resources](#).

Currently the `errata.zip` file contains the following corrections:

clinical-patient

This parameter incorrectly specified that both Patient and Group were target resource types for the patient search parameter. For DeviceUseStatement-patient this was correct, so we created a separate file for this parameter, still listing the Group as a valid target type.

search parameters with FhirPath expression `.as(DateTime)`

Several search parameters had an incorrect FhirPath expression using `.as(DateTime)` instead of `.as(dateTime)`. As a result, Firely Server could not index the fields correctly and searches on the dates would not work. The search parameters that were corrected are: `clinical-date`, `DeviceRequest-event-date`, `Observation-code-value-date`, `Observation-value-date` and `patient-death-date`.

Resource.<xyz> expressions

The FhirPath library did not support polymorphism yet, so all the search parameters defined with an expression of `Resource.<xyz>` – for example `Resource.meta.lastUpdated` – did not work correctly. We have changed the expression to have just the `<xyz>` part – for example `meta.lastUpdated`.

StructureDefinition.ext-context (R4 only)

The FhirPath expression ended on a BackboneElement that cannot be indexed. Changed to the expression `StructureDefinition.context.where(type='element').expression`.

10.9 Subscriptions

Subscriptions can be managed in the [Firely Server Administration API](#), on the `/administration/Subscription` endpoint. If you post a Subscription to the regular FHIR endpoint, it will be stored but not evaluated. Subscriptions posted to the `/administration` endpoint will be processed and evaluated for each POST/PUT to the server.

Firely Server currently only supports STU3/R4-style Subscriptions with a Channel of type `rest-hook`.

If you are *not permitted* to access the `/Subscription` endpoint, Firely Server will return statuscode 403.

See [Subscriptions in the specification](#) for more background on Subscriptions.

10.9.1 FHIR versions

You POST a Subscription with a `fhirVersion` parameter (see [Multiple versions of FHIR](#)) or to a version specific endpoint. It will then respond to changes on resources *in that FHIR version*. So if you need a Subscription on both STU3 and R4 resources, POST that Subscription for both FHIR versions.

10.9.2 Configuration

Firely Server evaluates the active Subscriptions periodically, and in batches (x at a time, until all the active Subscriptions have been evaluated). You can control the period and the batchsize. If an evaluation of a Subscription fails, Firely Server will retry the evaluation periodically for a maximum amount of tries. You can control the retry period and the maximum number of retries.

```
"SubscriptionEvaluatorOptions": {  
  "Enabled" : true  
  "RepeatPeriod": 20000,  
  "SubscriptionBatchSize" : 1,  
  "RetryPeriod": 60000,  
  "MaximumRetries": 3,  
  "SendRestHookAsCreate": false  
},
```

- **Enabled** allows you to quickly enable or disable the evaluation of Subscriptions. Default value is 'false', which implies that Subscription evaluation is also off if this section is left out of the settings.
- **RepeatPeriod** is expressed in milliseconds. In the example above the period is set to 20 seconds, meaning that after a change a subscriber will be notified in at most 20 seconds.
- **SubscriptionBatchSize** is expressed in number of Subscriptions that is retrieved and evaluated at once. Default is 1, but you can set it higher if you have a lot of Subscriptions.
- **RetryPeriod** is expressed in milliseconds. In the example above the period is set to 60 seconds, meaning that Firely Server will retry to send the resources after a minimum of 60 seconds. Retry is included in the normal evaluation process, so the **RetryPeriod** cannot be smaller than **RepeatPeriod**.
- **MaximumRetries** is the maximum amount of times Firely Server will retry to send the resources.
- **SendRestHookAsCreate**: in versions < 3.9.3, Vonk sent RestHook notifications as a create operation using a POST. That was not compliant with the specification that requires an update operation using a PUT. The default value of `false` provides compliant behaviour. Only set it to `true` if you need Firely Servder to keep sending create operations as it did previously.

10.10 Auditing

Firely Server can log access through the RESTful API for auditing purposes. It has 3 features:

1. Write requests and responses to a separate audit logfile.
2. Include user id and name from the JWT token (if present) in the audit log lines.
3. Write the audit information to AuditEvent resources in the Firely Server Data database.

All features can be enabled by including `Vonk.Plugins.Audit` in the pipeline. See [Configure the pipeline](#) for details on how to do that.

You can enable specific features by narrowing the namespace that you include in the pipeline, see the available plugins listed under [Auditing](#).

10.10.1 Audit log file configuration

Configure where to put the audit log file and the format of its lines in the appsettings (see *Firely Server settings*):

```
"Audit": {
  "PathFormat": "./audit/AuditLog-{Date}.log"
  "OutputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Application}] [Audit]␣
↪[Machine: {MachineName}] [ReqId: {RequestId}] [IP-Address: {Ip}] [Connection:
↪{ConnectionId}] [UserId: {UserId}] [UserName: {UserName}] [Path: {Path}] [Action:
↪{Action}] [Resource: {Resource} Key:{ResourceKey}] [StatusCode: {StatusCode}] {NewLine}
↪"
},
```

The OutputTemplate listed here contains all the properties that can be logged:

- RequestId: unique id of this request, use this to correlate request and response
- Ip: IP Address of the client
- ConnectionId: use this to correlate requests from the same client
- UserId: user id from the JWT token (if present)
- UserName: user name from the JWT token (if present)
- Path: request url
- Action: interaction that was requested (like instance_read or type_search)
- Resource: resourcetype involved
- ResourceKey: 'key' of the resource involved (if any), consisting of the resourcetype and the id, formatted as "resourcetype/id"
- StatusCode: statuscode of the response at the time of logging (by default '-1' when the request is not handled yet)

For transactions and batches, the audit plugin will write a line for the transaction/batch as a whole *and* one for every entry in the transaction/batch.

10.10.2 AuditEvent logging

There is no further configuration for AuditEvent logging. If you include it in the pipeline, it will start generating AuditEvent resources.

For transactions and batches the audit plugin will create an AuditEvent for the transaction/batch as a whole *and* one for every entry in the transaction/batch.

Firely Server does not allow you to update or delete the AuditEvent resources through the RESTful API so the Audit log cannot be tampered with. You can of course still manipulate these resources directly on the database, for instance to offload a surplus of old AuditEvent resources elsewhere. Please [Contact us](#) for details if you want to do this.

10.11 Reset the database

If you have set up Firely Server as a reference server in a testing environment, it can be useful to reset the database. You would usually do this in combination with *Preloading resources*.

To reset the database, execute:

```
POST http(s)://<firely-server-endpoint>/administration/reset
```

Firely Server will return statuscode 200 if the operation succeeded.

If you are *not permitted* to perform the reset, Firely Server will return statuscode 403.

Note: On a large database this operation may take a while.

An alternative, if you have direct access to the database server, is to delete the database altogether and have Firely Server recreate it again.

- If you run on SQL Server, see *Using SQL server* for the `AutoUpdateDatabase` feature.
- If you run on MongoDB, Firely Server will recreate the collection by default if it is not present.

Although the operation requires no further arguments, it requires a POST, since it is certainly not side-effect free.

10.12 Preloading resources

Caution: Preload in Firely Server (Vonk) 3.0.0 is only available for STU3.

If you have set up Firely Server as a reference server in a testing environment, it can be useful to load it with an ‘iron test set’ of examples. You can do that with the preload feature. Usually you will want to *Reset the database* first.

To preload a set of resources, execute:

```
POST http(s)://<firely-server-endpoint>/administration/preload
Content-Type: application/octet-stream
Body: a zip file with resources, each resource in a separate file (xml or json).
```

Firely Server will return statuscode 200 if the operation succeeded.

If you are *not permitted* to preload resources into the database, Firely Server will return statuscode 403.

Note: The operation can take quite long if the zip contains many resources. E.g. when uploading the *examples-json.zip* from the specification, it took about a minute on MongoDB and about 7 minutes on SQL Server on a simple test server.

Attention: This feature is not meant for bulk uploading really large sets of resources. Firely Server currently has no special operation for bulk operations.

10.13 Multiple versions of FHIR

Since version 3.0.0 Firely Server can run multiple versions of FHIR side-by-side in the same server. This page explains how this works and what the consequences are.

10.13.1 Requests

The FHIR Specification explains the `mimetype` parameter that distinguishes one FHIR version from another in the paragraph on the [FHIR Version parameter](#). Firely Server uses this approach to let you choose the model for your request. Below are examples on how to use the `fhirVersion` parameter and how it influences the behaviour of Firely Server. Accepted values for the parameter are:

- `fhirVersion=3.0`, for FHIR STU3
- `fhirVersion=4.0`, for FHIR R4

You can add the `fhirVersion` to the `Accept` and/or the `Content-Type` header. If you specify it on both, the `fhirVersion` parameters have to be the same.

The examples below explain the behaviour with STU3, but if you replace `fhirVersion` with 4.0, it works exactly the same on R4.

Note: If you do not specify a `fhirVersion` parameter, Firely Server will use `fhirVersion=3.0` (STU3) as a default. This way the behaviour is compatible with previous versions of Firely Server. If you like, you can change the [Default Information model](#)

Note: If you use both an `Accept` header and a `Content-Type` header, the `fhirVersion` parameter for both must be the same. So this would be *invalid*:

```
POST <base>/Patient
Accept=application/fhir+json; fhirVersion=3.0
Content-Type=application/fhir+json; fhirVersion=4.0
```

Search for all Patients in STU3. In Firely Server this means Patient resources that were also stored as STU3. There is no automatic conversion of resources that were stored as R4 to the STU3 format (or vice versa).

```
GET <base>/Patient
Accept=application/fhir+json; fhirVersion=3.0
```

Search for Patients with the name 'Fred' in STU3. The search parameters used in the query must be valid in STU3.

```
GET <base>/Patient?name=Fred
Accept=application/fhir+json; fhirVersion=3.0
```

Create a Patient resource in STU3. This will only be retrievable when accessed with STU3:

```
POST <base>/Patient
Content-Type=application/fhir+json; fhirVersion=3.0
Accept=application/fhir+json; fhirVersion=3.0

{<valid Patient JSON body>}
```

Update a Patient resource in STU3.:

```
PUT <base>/Patient/123
Content-Type=application/fhir+json; fhirVersion=3.0
Accept=application/fhir+json; fhirVersion=3.0

{<valid Patient JSON body with id: 123>}
```

1. If no resource with this id existed before: it will be created with this id. (This was already always the behaviour of Firely Server.)
2. If a resource with this id existed before, in STU3: update it.
3. If a resource with this id already exists in R4: you will get an error with an `OperationOutcome` saying that a resource with this id already exists with a different `informationmodel`.

Note: Id's still have to be unique within a `resourcetype`, regardless of the FHIR version.

Delete a Patient resource.:

```
DELETE <base>/Patient/123
Accept=application/fhir+json; fhirVersion=3.0
```

This will delete Patient/123, regardless of its FHIR version. The Accept header is needed for Firely Server to know how to format an `OperationOutcome` if there is an error.

10.13.2 Conformance resources

Conformance resources like `StructureDefinition` and `SearchParameter` are registered *per FHIR version*. This implies:

1. Conformance resources will be imported during *Import of Conformance Resources* for both STU3 and R4. To avoid id clashes (see note above), the id's in R4 are appended with '-Fhir4.0'
1. So the `StructureDefinition` for Patient will be available for STU3 and R4 respectively like this:

```
GET <base>/StructureDefinition/Patient
Accept=application/fhir+json; fhirVersion=3.0

GET <base>/StructureDefinition/Patient-Fhir4.0
Accept=application/fhir+json; fhirVersion=4.0
```

2. If you add a `StructureDefinition` or `SearchParameter` via the Administration API, you can decide for yourself whether to append the FHIR version to the id or not. Just note that you cannot use the same id for different FHIR versions.
3. Depending on the `fhirVersion` parameter Firely Server evaluates whether a `resourcetype` or `searchparameter` is valid in that FHIR version. E.g. 'VerificationResult' is only valid in R4, but 'DataElement' is only valid in R3.
4. For validation, the `StructureDefinitions` and terminology resources needed are only searched for in the FHIR version of the resource that is being validated.
5. When you *Manage Conformance Resources with the Administration API*, a `StructureDefinition` can only be posted to the Administration API in the context of a FHIR Version that matches the `StructureDefinition.fhirVersion`. So this works:

```

POST <base>/administration/StructureDefinition
Accept=application/fhir+json; fhirVersion=4.0
Content-Type=application/fhir+json; fhirVersion=4.0

{
  "resourcetype": "StructureDefinition"
  ...
  "fhirVersion": "4.0.0" //Note the FHIR version matching the Content-Type
}

```

But it would not work if "fhirVersion"="3.0.1"

6. If you *Load Conformance Resources on demand*, this will be done for all the importfiles described above, regardless of the fhirVersion in the Accept header.

10.13.3 Running a single version

To use only a single version you set the Default information model in *Information model* to the version you want to use. In addition, you can exclude the namespace of the version you don't need (Vonk.Fhir.R3 or Vonk.Fhir.R4) from the *PipelineOptions* to disable its use. If you exclude a namespace, make sure to exclude it from all branches.

10.13.4 Running different versions on different endpoints

To assign endpoints to different versions, create a mapping in *Information model*. Use the Mode switch to select either a path or a subdomain mapping, assigning your endpoints in the Map array. Mapped endpoints will only accept the version you have specified. The web service root ('/' and '/administration/') will still accept all supported versions.

Assigning an endpoint to a FHIR version is exactly equivalent to adding that particular fhirVersion MIME parameter to every single request sent to that endpoint. So using these settings:

```

"InformationModel": {
  "Default": "Fhir4.0",
  "Mapping": {
    "Mode": "Path",
    "Map": {
      "/R3": "Fhir3.0",
      "/R4": "Fhir4.0"
    }
  }
}

```

The call

```

GET http://myserver.org/Patient
Accept=application/fhir+json; fhirVersion=3.0

```

is equivalent to

```

GET http://myserver.org/R3/Patient

```

and the call

```
GET http://myserver.org/Patient (defaults to R4)
```

is equivalent to

```
GET http://myserver.org/R4/Patient
```

and the administration call

```
GET http://myserver.org/administration/StructureDefinition (defaults to R4)
```

is equivalent to

```
GET http://myserver.org/administration/R4/StructureDefinition (/R4 is a postfix to '/  
↪administration')
```

As you can see, on a mapped endpoint it is never necessary to use a FHIR `_format` parameter or a `fhirVersion` MIME parameter in a `Content-Type` or `Accept` header.

10.13.5 Support for R5 (experimental!)

By default the binaries for supporting R5 are included in the Firely Server distribution (since Firely Server (Vonk) 3.3.0). But also by default these binaries are not loaded. See the `PipelineOptions` in `appsettings.default`, where `Vonk.Fhir.R5` is commented out.

Re-enable these in your `appsettings.instance` and you are good to go.

Note that there is not yet an `errata_Fhir5.0.zip` and Firely Server will complain about that in the log. You can ignore that message.

10.14 HIPAA compliance

Firely Server is a well-tested, secure HL7 FHIR® server that enables you to comply with the Technical Safeguards of the HIPAA Security Rule.

On this page we will detail how you can achieve compliance for your Firely Server deployment. To ensure your organisation's specific usecase, environment, and deployment are compliant, feel free to [contact us](#): we'd be happy to help.

10.14.1 164.312(a)(1) Standard: Access control

Implement technical policies and procedures for electronic information systems that maintain electronic protected health information to allow access only to those persons or software programs that have been granted access rights as specified in 164.308(a)(4).

There are several ways to approach this:

1. ensure Firely Server is deployed in a secure environment where only those with correct permissions are able to access it,
2. use SMART on FHIR as a means of controlling access,
3. or add custom authentication based on a plugin.

Deploying in a secure environment (1) would mean access to Firely Server is controlled by third-party software or policy, placing this scenario outside the scope of this guide.

For scenario (2), Firely Server implements support for [Smart on FHIR](#), a sibling specification to FHIR for securely connecting third-party applications to Electronic Health Record data. See [Access control and SMART](#) on how to configure Firely Server with it.

You may also wish to setup custom authentication (3). Given how Firely Server is based on a pipeline architecture, you can insert a plugin at the start of the pipeline to call out to your authentication service(s) prior to handling the request. See [this gist](#) as an example.

10.14.2 164.312(c)(1) Standard: Integrity

Implement policies and procedures to protect electronic protected health information from improper alteration or destruction.

The same solutions apply to this point as [164.312\(a\)\(1\) Standard: Access control](#) and [164.312\(b\) Standard: Audit control](#).

10.14.3 164.312(d) Standard: Person or entity authentication

Implement procedures to verify that a person or entity seeking access to electronic protected health information is the one claimed.

The same solutions apply to this point as [164.312\(a\)\(1\) Standard: Access control](#).

10.14.4 164.312(a)(2)(i) Unique user identification

Assign a unique name and/or number for identifying and tracking user identity.

The same solution applies to this point as [164.312\(b\) Standard: Audit control](#). For Firely Server to be able to log the identity of the user, this identity must be present in or derivable from the authentication token, and it must be added to the log properties. If you use the SMART on FHIR plugin, that is automatically configured. If you want to do this from within a custom authentication plugin, feel free to contact us for details.

10.14.5 164.312(b) Standard: Audit control

Implement hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information.

With the use of the [Audit Event log](#) plugin, Firely Server will thoroughly log every interaction as a note in a log file and/or in an AuditEvent resource. Logged information will be a trace record of all system activity: viewing, modification, deletion and creation of all Electronic Protected Health Information (ePHI).

The audit trail can track the source IP, event type, date/time, and more. If a JWT token is provided (for SMART on FHIR), the user/patient identity can be logged as well.

10.14.6 164.312(e)(1, 2) Standard: Transmission security

Implement technical security measures to guard against unauthorized access to electronic protected health information that is being transmitted over an electronic communications network.

Implement a mechanism to encrypt electronic protected health information whenever deemed appropriate.

Transmission security in Firely Server can be achieved by encrypting the communications with TLS/SSL. Standard industry practice is to use a reverse proxy (e.g. nginx or IIS) for this purpose. If you'd like, you can also enable secure connections in Firely Server *directly* without a proxy as well.

Firely Server is regularly updated with the latest versions of ASP.NET to ensure that the latest cryptographic algorithms are available for use.

10.14.7 164.312(e)(2)(ii) Encryption

Implement a mechanism to encrypt electronic protected health information whenever deemed appropriate.

The recommended way to ensure that e-PHI is encrypted as necessary is to use disk encryption, and there are several solutions for this depending on your deployment environment. If you're deploying in the cloud - see your vendors options for disk encryption, as most have options for encrypted disks already. If you're deploying locally, look into BitLocker on Windows or dm-crypt/LUKS for Linux.

Disk encryption is preferred over individual database field encryption as the latter would severely impact the search performance.

10.14.8 164.312(a)(2)(ii) Emergency access procedure

Establish (and implement as needed) procedures for obtaining necessary electronic protected health information during an emergency.

This depends on the solution you went with for *164.312(a)(1) Standard: Access control*.

In case you went with SMART on FHIR, add an authorization workflow that grants emergency access rights - essentially, a "super" access token. The application can then use this token with Firely Server, just like any other token.

If you went with a custom authentication scheme, add a special measure to handle this scenario.

10.14.9 164.312(a)(c) Implementation specification: Mechanism to authenticate electronic protected health information

Implement electronic mechanisms to corroborate that electronic protected health information has not been altered or destroyed in an unauthorized manner.

Firely Server does not allow you to delete resources through its RESTful API. Old versions of resources are retained by default. The only way to alter or destroy resources is through direct database access.

Therefore database-level safety mechanisms must ensure that information is not altered or destroyed unless it's desired.

FIRELY SERVER ADMINISTRATION API

Besides the regular FHIR endpoint, Firely Server also exposes an Administration API. The endpoint for this is:

```
http(s)://<firely-server-endpoint>/administration
```

11.1 Functions

The following functions are available in the Administration API.

- *Conformance Resources*
- *Subscriptions*
- *Re-indexing for new or changed SearchParameters*
- *Reset the database*
- *Preloading resources*
- *Terminology services*

11.2 Configuration

You can *Configure the Administration API*, including restricting access to functions of the Administration API to specific ip addresses.

11.3 Database

The Administration API uses a database separately from the main ‘Firely Server Data’ database. Historically, SQL Server, MongoDB and Memory are supported as databases for the Administration API. As of Firely Server (Vonk) version 0.7.1, SQLite is advised for this, and we have made that the default configuration. See *Using SQLite* on how to configure for this.

FIRELY SERVER PLUGINS

Firely Server Plugins is the means to adjust a Firely Server to your own special needs, beyond the configuration. Please have a look at the *Overview of Firely Server, Plugins and Facades* to see how plugins fit in the Firely Server. *Architecture* provides more detail on that.

12.1 Configure the pipeline

Configuration of the pipeline in Firely Server is done with `PipelineOptions` in combination with `SupportedInteractions`. A default setup is installed with Firely Server in `appsettings.default.json`, and it looks like this:

```
"PipelineOptions": {
  "PluginDirectory": "./plugins",
  "Branches": [
    {
      "Path": "/",
      "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",
        "Vonk.Fhir.R4",
        "Vonk.Repository.Sql.SqlVonkConfiguration",
        "Vonk.Repository.Sqlite.SqliteVonkConfiguration",
        "Vonk.Repository.MongoDb.MongoDbVonkConfiguration",
        "Vonk.Repository.CosmosDb.CosmosDbVonkConfiguration",
        "Vonk.Repository.Memory.MemoryVonkConfiguration",
        "Vonk.Subscriptions",
        "Vonk.Smart",
        "Vonk.UI.Demo",
        "Vonk.Plugin.DocumentOperation.DocumentOperationConfiguration",
        "Vonk.Plugin.ConvertOperation.ConvertOperationConfiguration",
        "Vonk.Plugin.BinaryWrapper.BinaryWrapperConfiguration",
        "Vonk.Plugin.MappingToStructureMap.MappingToStructureMapConfiguration",
        "Vonk.Plugin.TransformOperation.TransformOperationConfiguration"
      ],
      "Exclude": [
        "Vonk.Subscriptions.Administration"
      ]
    },
    {
      "Path": "/administration",
```

(continues on next page)

(continued from previous page)

```

    "Include": [
      "Vonk.Core",
      "Vonk.Fhir.R3",
      "Vonk.Fhir.R4",
      "Vonk.Repository.Sql.SqlAdministrationConfiguration",
      "Vonk.Repository.Sqlite.SqliteAdministrationConfiguration",
      "Vonk.Repository.MongoDb.MongoDbAdminConfiguration",
      "Vonk.Repository.Memory.MemoryAdministrationConfiguration",
      "Vonk.Subscriptions.Administration",
      "Vonk.Plugins.Terminology",
      "Vonk.Administration"
    ],
    "Exclude": [
      "Vonk.Core.Operations",
      "Vonk.Core.Licensing.LicenseRequestJobConfiguration"
    ]
  }
},
"SupportedInteractions": {
  "InstanceLevelInteractions": "read, vread, update, delete, history, conditional_delete,
→ conditional_update, $validate, $validate-code, $expand, $compose, $meta, $meta-add,
→ $document",
  "TypeLevelInteractions": "create, search, history, conditional_create, compartment_
→ type_search, $validate, $snapshot, $validate-code, $expand, $lookup, $compose,
→ $document",
  "WholeSystemInteractions": "capabilities, batch, transaction, history, search,
→ compartment_system_search, $validate, $convert"
},

```

PluginDirectory:

You can put plugins of your own (or third party) into this directory for Firely Server to pick them up, without polluting the Firely Server binaries directory itself. For a list of available plugins in Firely Server, see [Plugins available for Firely Server](#). The directory in the default setting of `./plugins` is not created upon install, you may do this yourself if you want to add a plugin.

PluginDirectory.Branches:

A web application can branch into different paths, and Firely Server has two by default:

- `/`: the root branch, where the main *FHIR RESTful API* is hosted;
- `/administration`: where the *Firely Server Administration API* is hosted.

Branches contains a subdocument for each of the defined paths:

Path

The path for this branch. This is the part after the base URL that Firely Server is hosted on.

Include

(Prefixes of) *Configuration classes* that add services and middleware to Firely Server.

Exclude

(Prefixes of) *Configuration classes* that may not be executed. Exclude overrides Include and is useful if you want to use all but one configuration class from a namespace.

SupportedInteractions:

A comma-separated list of all interactions Firely Server should enable on `[base]/[type]/[id]` (In-

stanceLevelInteractions), [base]/[type] (TypeLevelInteractions), and [base] (WholeSystemInteractions) levels. Firely Server will use this list to enable/disable supported interactions and reflect it in /metadata accordingly.

If you'd like to limit what operations your Firely Server supports, remove them from this list.

If you've added a custom plugin that enables a new interaction, make sure to load the plugin (see PluginDirectory above) and enable the interaction in this list. For example, if you've added the Vonk.Plugin.ConvertOperation \$convert plugin in PipelineOptions.Branches.Include, make sure to enable the operation \$convert as well:

```
"WholeSystemInteractions": "$convert, capabilities, batch, transaction, history,
↪search, compartment_system_search, $validate"
```

12.2 Configuration classes

A configuration class is a static class with two public static methods having the signature as below, that can add services to the Firely Server dependency injection system, and add middleware to the pipeline.

```
[VonkConfiguration (order: xyz)] //xyz is an integer
public static class MyVonkConfiguration
{
    public static void ConfigureServices(IServiceCollection services)
    {
        //add services here to the DI system of ASP.NET Core
    }

    public static void Configure(IApplicationBuilder builder)
    {
        //add middleware to the pipeline being built with the builder
    }
}
```

As you may have noticed, the methods resemble those in an ASP.NET Core Startup class. That is exactly where they are ultimately called from. We'll explain each of the parts in more detail.

VonkConfiguration

This is an attribute defined by Firely Server (package Vonk.Core, namespace Vonk.Core.Pluggability). It tells Firely Server to execute the methods in this configuration class. The order property determines where in the pipeline the middleware will be added. You can see the order of the plugins in the *log* at startup.

MyVonkConfiguration

You can give the class any name you want, it will be recognized by Firely Server through the attribute, not the classname. We do advise you to choose a name that actually describes what is configured. It is also better to have multiple smaller configuration classes than one monolith adding all your plugins, so you allow yourself to configure your plugins individually afterwards.

ConfigureServices

The main requirements for this method are:

- It is public static;
- It has a first formal argument of type Microsoft.Extensions.DependencyInjection.IServiceCollection;

- It is the only method in this class matching the first two requirements.

This also means that you can give it a different name. Beyond that, you may add formal arguments for services that you need during configuration. You can only use services that are available from the ASP.NET Core hosting process, not any services you have added yourself earlier. Usual services to request are:

- IConfiguration
- IHostingEnvironment

These services will be injected automatically by Firely Server.

Configure

The main requirements for this method are:

- It is public static;
- It has a first formal argument of type `Microsoft.AspNetCore.Builder.IApplicationBuilder`;
- It is the only method in this class matching the first two requirements.

This also means that you can give it a different name. Beyond that, you may add formal arguments for services that you may need during configuration. Here you can use services that are available from the ASP.NET Core hosting process *and* any services you have added yourself earlier. For services in request scope please note that this method is not run in request scope. These services will be injected automatically by Firely Server.

We provided an *example* of this: creating your own landing page.

12.3 Detailed logging of loading plugins

If your plugin or any of the Firely Server plugins appears not to be loaded correctly, you may inspect what happens in more detail in the log. See *Log settings* for where you can find the log file. You can vary the log level for `Vonk.Core.Pluggability.VonkConfigurer` to hide or reveal details.

On the Information level, Firely Server will tell you which assemblies are loaded and searched for `VonkConfiguration` attributes:

```
Looking for Configuration in these assemblies:
C:\data\dd18\vonk_preview\Vonk.Administration.Api.dll
C:\data\dd18\vonk_preview\Vonk.Core.dll
C:\data\dd18\vonk_preview\Vonk.Fhir.R3.dll
C:\data\dd18\vonk_preview\Vonk.Fhir.R4.dll
C:\data\dd18\vonk_preview\Vonk.Repository.Generic.dll
C:\data\dd18\vonk_preview\Vonk.Repository.Memory.dll
C:\data\dd18\vonk_preview\Vonk.Repository.MongoDb.dll
C:\data\dd18\vonk_preview\Vonk.Repository.Sql.dll
C:\data\dd18\vonk_preview\vonk.server.dll
C:\data\dd18\vonk_preview\Vonk.Server.PrecompiledViews.dll
C:\data\dd18\vonk_preview\Vonk.Smart.dll
C:\data\dd18\vonk_preview\Vonk.Subscriptions.dll
C:\data\dd18\vonk_preview\Vonk.UI.Demo.dll
C:\data\dd18\vonk_preview\plugins\Visi.Repository.dll
C:\data\dd18\vonk_preview\plugins\Vonk.Facade.Relational.dll
```


Also on the Information level, Firely Server will show the services and middleware as it has loaded, in order. The list below is also the default pipeline as it is configured for Firely Server.

Configuration:

```

/
  FhirR3Configuration           [100] | Services: V | Pipeline: X
  FhirR4Configuration           [101] | Services: V | Pipeline: X
  MetadataConfiguration         [110] | Services: V | Pipeline: X
  LicenseConfiguration          [120] | Services: V | Pipeline: X
  SerializationConfiguration     [130] | Services: V | Pipeline: X
  RepositorySearchSupportConfiguration [140] | Services: V | Pipeline: X
  RepositoryIndexSupportConfiguration [141] | Services: V | Pipeline: X
  PluggabilityConfiguration      [150] | Services: V | Pipeline: X
  ViSiConfiguration             [240] | Services: V | Pipeline: X
  DemoUIConfiguration           [800] | Services: V | Pipeline: V
  VonkToHttpConfiguration       [1110] | Services: V | Pipeline: V
  VonkFeaturesExtensions        [1120] | Services: X | Pipeline: V
  FormatConfiguration           [1130] | Services: V | Pipeline: V
  LongRunningConfiguration      [1170] | Services: V | Pipeline: V
  VonkCompartmentsExtensions    [1210] | Services: X | Pipeline: V
  SupportedInteractionConfiguration [1220] | Services: V | Pipeline: V
  UrlMappingConfiguration       [1230] | Services: V | Pipeline: V
  ElementsConfiguration         [1240] | Services: V | Pipeline: V
  FhirBatchConfiguration        [3110] | Services: V | Pipeline: V
  FhirTransactionConfiguration   [3120] | Services: V | Pipeline: V
  SubscriptionConfiguration     [3200] | Services: V | Pipeline: V
  ValidationConfiguration       [4000] | Services: V | Pipeline: V
  DefaultShapesConfiguration    [4110] | Services: V | Pipeline: V
  CapabilityConfiguration       [4120] | Services: V | Pipeline: V
  IncludeConfiguration          [4210] | Services: V | Pipeline: X
  SearchConfiguration           [4220] | Services: V | Pipeline: V
  ProfileFilterConfiguration     [4310] | Services: V | Pipeline: V
  PrevalidationConfiguration    [4320] | Services: V | Pipeline: V
  ReadConfiguration             [4410] | Services: V | Pipeline: V
  CreateConfiguration           [4420] | Services: V | Pipeline: V
  UpdateConfiguration           [4430] | Services: V | Pipeline: V
  DeleteConfiguration           [4440] | Services: V | Pipeline: V
  ConditionalCreateConfiguration [4510] | Services: V | Pipeline: V
  ConditionalUpdateConfiguration [4520] | Services: V | Pipeline: V
  ConditionalDeleteConfiguration [4530] | Services: V | Pipeline: V
  HistoryConfiguration          [4610] | Services: V | Pipeline: V
  VersionReadConfiguration      [4620] | Services: V | Pipeline: V
  InstanceValidationConfiguration [4840] | Services: V | Pipeline: V
  SnapshotGenerationConfiguration [4850] | Services: V | Pipeline: V
/administration
  SqlVonkConfiguration          [220] | Services: V | Pipeline: X
  SqlAdministrationConfiguration [221] | Services: V | Pipeline: X
  DatabasePluggabilityConfiguration [300] | Services: V | Pipeline: X
  VonkToHttpConfiguration       [1110] | Services: V | Pipeline: V
  VonkFeaturesExtensions        [1120] | Services: X | Pipeline: V
  FormatConfiguration           [1130] | Services: V | Pipeline: V
  SecurityConfiguration         [1150] | Services: V | Pipeline: V
  AdministrationOperationConfiguration [1160] | Services: V | Pipeline: V

```

(continues on next page)

(continued from previous page)

LongRunningConfiguration	[1170]	Services: V Pipeline: V
VonkCompartmentsExtensions	[1210]	Services: X Pipeline: V
SupportedInteractionConfiguration	[1220]	Services: V Pipeline: V
UrlMappingConfiguration	[1230]	Services: V Pipeline: V
ElementsConfiguration	[1240]	Services: V Pipeline: V
DefaultShapesConfiguration	[4110]	Services: V Pipeline: V
AdministrationSearchConfiguration	[4221]	Services: V Pipeline: V
ValidationConfiguration	[4310]	Services: V Pipeline: X
SubscriptionValidationConfiguration	[4330]	Services: V Pipeline: V
ChangeInterceptionConfiguration	[4390]	Services: X Pipeline: V
AdministrationReadConfiguration	[4411]	Services: V Pipeline: V
AdministrationCreateConfiguration	[4421]	Services: V Pipeline: V
AdministrationUpdateConfiguration	[4431]	Services: V Pipeline: V
AdministrationDeleteConfiguration	[4441]	Services: V Pipeline: V
AdministrationImportConfiguration	[5000]	Services: V Pipeline: V
CodeSystemLookupConfiguration	[5110]	Services: V Pipeline: V
ValueSetValidateCodeInstanceConfiguration	[5120]	Services: V Pipeline: V
ValueSetValidateCodeTypeConfiguration	[5130]	Services: V Pipeline: V
ValueSetExpandInstanceConfiguration	[5140]	Services: V Pipeline: V
ValueSetExpandTypeConfiguration	[5150]	Services: V Pipeline: V
CodeSystemComposeInstanceConfiguration	[5160]	Services: V Pipeline: V
CodeSystemComposeTypeConfiguration	[5170]	Services: V Pipeline: V

It shows all the configuration classes it found, and whether a `ConfigureServices` and / or a `Configure` method was found and executed. It also displays the value of the `order` property of the `VonkConfiguration` attribute for each configuration class. This allows you to determine an appropriate order for your own configuration class.

On the Verbose level, Firely Server will also tell you why each configuration class that is found is being included or excluded. An example:

```
2018-07-02 12:58:10.586 +02:00 [Firely Server] [Verbose] [Machine: XYZ] [ReqId: ] ↳
↳ Searching for configurations in assembly "Vonk.Core, Version=0.7.0.0, Culture=neutral,
↳ PublicKeyToken=null"
2018-07-02 12:58:10.625 +02:00 [Firely Server] [Verbose] [Machine: XYZ] [ReqId: ] "Vonk.
↳ Core.Serialization.SerializationConfiguration" was included on "/" because it matches
↳ the include "Vonk.Core"
2018-07-02 12:58:10.625 +02:00 [Firely Server] [Verbose] [Machine: XYZ] [ReqId: ] "Vonk.
↳ Core.Serialization.SerializationConfiguration" was not included on "/administration"
↳ because it did not match any include
```

12.4 The order of plugins

Firely Server is organized as a pipeline of components - called **Middleware**. Every request travels through all the components until one of the components provides the response to the request. After that, the response travels back through all the components, in reverse order. Components that come *after* the responding component in the pipeline are not visited at all.

So let's say you issue a FHIR read interaction, `GET <base-url>/Patient/example`. The component implementing this interaction sits in the pipeline after search but before create. So the request will visit the search middleware (that will ignore it and just pass it on) but will never visit the create middleware.

So many components implement an interaction and provide a response to that interaction. In Firely Server those are

called Handlers. Some components may not provide responses directly, but read or alter the request on the way in. Such a component is called a PreHandler. Reversely, a component may read or alter the response on the way back. Such a component is called a PostHandler.

A plugin can configure its own component in this pipeline but as you may understand by now it makes a difference *where* in the pipeline you put that component. Especially if it is a Pre- or PostHandler. To control the position in the pipeline, Firely Server uses the concept of ‘Order’.

The *VonkConfiguration attribute* lets you define an Order for your component. This page explains how to choose a suitable number for that order.

12.4.1 Inspect order numbers in use

When you start Firely Server, the log lists all the loaded plugins, with their order. You can see an example [here](#). Also the list of *Plugins available for Firely Server* includes the order number chosen for each of those plugins.

12.4.2 Minimum order

Registering services

The order is mainly relevant for middleware that you register in the pipeline, in the `Configure(IApplicationBuilder app)` method. Some plugins, like e.g. a Facade implementation, only need to register services for the dependency injection framework, in the `ConfigureServices(IServiceCollection services)` method. If that is the case, the order is only relevant if you need to *override* a registration done by Firely Server. There are two ways:

1. Choose an order before Firely Server’s default registration. Firely Server in general uses `TryAddSingleton` or `TryAddScoped` to register an implementation of an interface. This means that if an implementation is already registered, the `TryAdd...` will not register a second implementation.

As an example: if you want to override the registration of `IReadAuthorizer`: that is registered from the *RepositorySearchSupport* plugin, with order 140. So you would choose an order lower than 140.

2. Choose a high order (e.g. > 10000) and make sure your registration overwrites any existing registration.

```
services.AddOrReplace<IReadAuthorizer, MyReadAuthorizer>(ServiceLifetime.Scoped);
```

The latter method is the least error prone and therefore recommended.

If you only need to register interfaces and/or classes defined by your plugin, the order is not relevant, so pick any number. All service registrations are done before the pipeline itself is configured.

Registering middleware

For middleware it is more important where exactly it ends up in the pipeline. This depends mostly on what type of handler it is, see below at *Handlers and pre- and posthandlers*.

No matter what handler you have, it probably wants to act on the *IVonkContext*. Then it is important to be in the pipeline *after* the *HttpToVonk* plugin (order: 1110), since this plugin translates information from the `HttpContext` to an `IVonkContext` and adds the latter as a feature to the `HttpContext.Features` collection.

Also, you probably want to set your response on the `IVonkContext.Response` and not directly on the `HttpContext.Response`. Then, you will need the *VonkToHttp* plugin (order: 1120) to translate the `IVonkContext` back to the `HttpContext`.

So in general, the minimum order you need for your plugin will be higher than 1120.

If you want your middleware to act on all the entries in a Batch or Transaction, you need to choose an order higher than that of the *Transaction* plugin, which is 3120.

12.4.3 Order collisions

If two plugins have the same order number, it is not defined in what order the plugins will be put in the pipeline. As long as those plugins act on disjoint sets of requests that may not be a problem. But it is recommended to avoid this by checking the orders already in use.

12.4.4 Handlers and pre- and posthandlers

In Firely Server you can define different types of middleware:

- Handler - acts on requests of a certain type, provides the response to it and ends the pipeline.
- Prehandler - acts on requests of certain type(s), may modify the request and sends the request further down the pipeline.
- Posthandler - lets the request pass by to be handled further down the pipeline. When the response passes on the way back, it acts on requests or responses of certain type(s), and may modify the response.

This is explained in the [session on Plugins](#) from DevDays 2018.

What type of middleware you want your service to be is defined by your use of one of the **Handle...* methods from the *VonkAppBuilder extension methods* or the *IApplicationBuilder extension methods*.

Prehandler

A Prehandler needs to act *before* the actual handler will provide a response. So the order of it must be lower than any Handler that may handle the requests that this Prehandler is interested in.

So if you want a Prehandler to intercept all create interactions, you should choose an order lower than that of the *Create* plugin, which is 4420.

An example of this is the *Prevalidation* plugin. It needs to validate all resources that get handled by the Create, Update, Conditional Create and Conditional Update plugins. Of these, Create has the lowest order: 4420. So it must be below 4420. But it also needs to act on each resource in a *Batch* or *Transaction*, so it must be higher than these two, which means higher than 3120. So this is why we have chosen 4320 as order for Prevalidation.

Posthandler

A Posthandler needs to act *after* the actual handler provided a response. But due to the nature of the processing pipeline that means it must have an order *lower* than that of the handler(s) it wants to post-process. The idea is that the posthandler sits in the pipeline and lets the request pass through. Then one of the handlers provides the response and sends it back through the pipeline. It will pass through the posthandler again (now 'backwards'), and then the posthandler will do its processing.

So if you want a Posthandler to process the responses of all create interactions (e.g. for logging purposes), you should choose an order lower than that of the *Create* plugin.

An example for this is the *Include* plugin. This must act on the response of the *Search* plugin. So the Include has order 4210, right before Search which has 4220.

12.5 Important classes and interfaces

If you want to develop a plugin for Firely Server, there are a couple of classes that you will probably interact with. These classes are listed under *Reference - Programming API*.

12.6 Template for a plugin

Attention: A complete template for a Firely Server plugin can be found on [Github](#). It covers all details on how to create a custom operation and how to use Firely Server services internally.

A regular Firely Server plugin acts on the *IVonkContext* and its *IVonkRequest* and *IVonkResponse* properties.

You don't have to create ASP.NET Core middleware yourself. You just need to create a service acting on *IVonkContext*. In the configuration you can specify when the service should be called, and in which position in the pipeline it should be put. See *Interaction Handling* for details on that.

You can use the following code as a template for a plugin:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.DependencyInjection.Extensions;
using System.Threading.Tasks;
using Vonk.Core.Context;
using Vonk.Core.Context.Features;
using Vonk.Core.Pluggability;
using Vonk.Fhir.R3;
using F = HL7.Fhir.Model;

namespace com.mycompany.vonk.myplugin
{
    public class MyPluginService
    {
        public async Task Act(IVonkContext vonkContext)
        {
            var (request, args, response) = vonkContext.Parts();
            //do something with the request
            //write something to the response
            response.Payload = new F.Patient{Id = "pat1"}.ToIResource();
            response.HttpResult = 200;
        }
    }

    [VonkConfiguration(order: 5000)]
    public static class MyPluginConfiguration
    {
        public static IServiceCollection AddMyPluginServices(IServiceCollection services)
        {
            services.TryAddScoped<MyPluginService>();
            return services;
        }
    }
}
```

(continues on next page)

(continued from previous page)

```

    }

    public static IApplicationBuilder UseMyPlugin(IApplicationBuilder app)
    {
        app.OnCustomInteraction(VonkInteraction.system_custom, "myOperation").
        ↳HandleAsyncWith<MyPluginService>((svc, context) => svc.Act(context));
        return app;
    }
}

```

12.7 Returning non-FHIR content from a plugin

Some plugins may need to return content that is not a FHIR Resource. You currently cannot do that through the `IVonkResponse`. But there is another way: write directly to the `HttpContext.Response`. The plugin with order 1110 makes the `IVonkContext` accessible. That means if you pick an order higher than 1110 you can read the `IVonkContext.Request` and `.Arguments`, and write directly to the `HttpContext.Response`.

If you write the response yourself, you also need to set the `StatusCode` and `Content-Type` on the `HttpContext.Response`.

Note: If you write to the `HttpContext.Response` directly, the payload of the `IVonkContext.Response` is ignored.

The steps to take are:

1. Configure your plugin with an order higher than 1110
2. Write the `HttpContext.Response.Body` directly
3. Set other properties of the `HttpContext.Response` (like `StatusCode`) yourself.

An example of such a plugin would look like this. Note that this is now regular ASP.NET Core Middleware, not a service like in *Template for a plugin*.

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using System.Text;
using System.Threading.Tasks;
using Vonk.Core.Context;
using Vonk.Core.Context.Features;
using Vonk.Core.Pluggability;

namespace com.mycompany.vonk.myplugin
{
    public class MyPluginMiddleware
    {
        private readonly RequestDelegate _next;

        public MyPluginMiddleware(RequestDelegate next)
        {
            _next = next;
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

    }

    public async Task Invoke(HttpContext httpContext)
    {
        var vonkContext = httpContext.Vonk();
        var (request, args, _) = vonkContext.Parts();
        if (VonkInteraction.system_custom.HasFlag(request.Interaction))
        {
            //write something directly to the HttpContext.Response. Now you also
            ↪have to set the Content-Type header and the Content-Length yourself.
            var message = "This is a response that is not a FHIR resource";
            string contentLength = Encoding.UTF8.GetByteCount(message).ToString();

            httpContext.Response.Headers.Add("Content-Type", "text/plain;
            ↪charset=utf-8");
            httpContext.Response.Headers.Add("Content-Length", contentLength);
            httpContext.Response.StatusCode = 200;
            await httpContext.Response.WriteAsync("This is a response that is not a
            ↪FHIR resource");
        }
        else
        {
            await _next(httpContext);
        }
    }
}

[VonkConfiguration(order: 1115)] //note the order: higher than 1110
public static class MyPluginConfiguration
{
    public static IServiceCollection AddMyPluginServices(IServiceCollection services)
    {
        //No services to register in this example, but if you create services to do
        ↪the actual work - register them here.
        return services;
    }

    public static IApplicationBuilder UseMyPlugin(IApplicationBuilder app)
    {
        app.UseMiddleware<MyPluginMiddleware>(); //You cannot use the extension
        ↪methods that allow you to filter the requests.
        return app;
    }
}
}

```

12.7.1 Custom authorization plugin

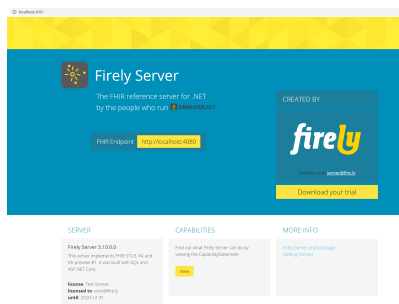
This feature can also be used to implement custom authorization. You can find a template for that in this [gist](#).

If you just return a statuscode, you could use the regular `IVonkContext.Response`. If you want to return e.g. a custom json object, you should use the method described above. Do not forget to set the Content-Type (to `application/json` for a custom json object).

For more information about access control in plugins and facades, see *Access Control in Plugins and Facades*.

12.8 Firely Server Plugin example - Create a new landing page

As a minimal example of how to use Firely Server Plugins we will show you how to create a library with your own landing page, and use it to replace the landing page that is provided by Firely Server. The landing page is the webpage you see when you access Firely Server's endpoint with a browser. By default it looks like this:



12.8.1 Create a new ASP.NET Core web application

In Visual Studio create a new project of type ASP .NET Core Web Application:

Create a new project


Recent project templates


- ASP.NET Core Web Application C#
- xUnit Test Project (.NET Core) C#
- MSTest Test Project (.NET Core) C#
- Class Library (.NET Standard) C#
- Console App (.NET Core) C#


asp.net core


Clear all


C# All platforms All project types

**ASP.NET Core Web Application**
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
C# Linux macOS Windows Cloud Service Web

**Console App (.NET Core)**
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Console

**MSTest Test Project (.NET Core)**
A project that contains MSTest unit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Test

**NUnit Test Project (.NET Core)**
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Desktop Test Web

**WPF App (.NET Core)**
Windows Presentation Foundation client application
C#

Back Next


Press OK to continue.

Configure your new project

ASP.NET Core Web Application C# Linux macOS Windows Cloud Service Web

Project name
WebApplication2

Location
C:\source\repos

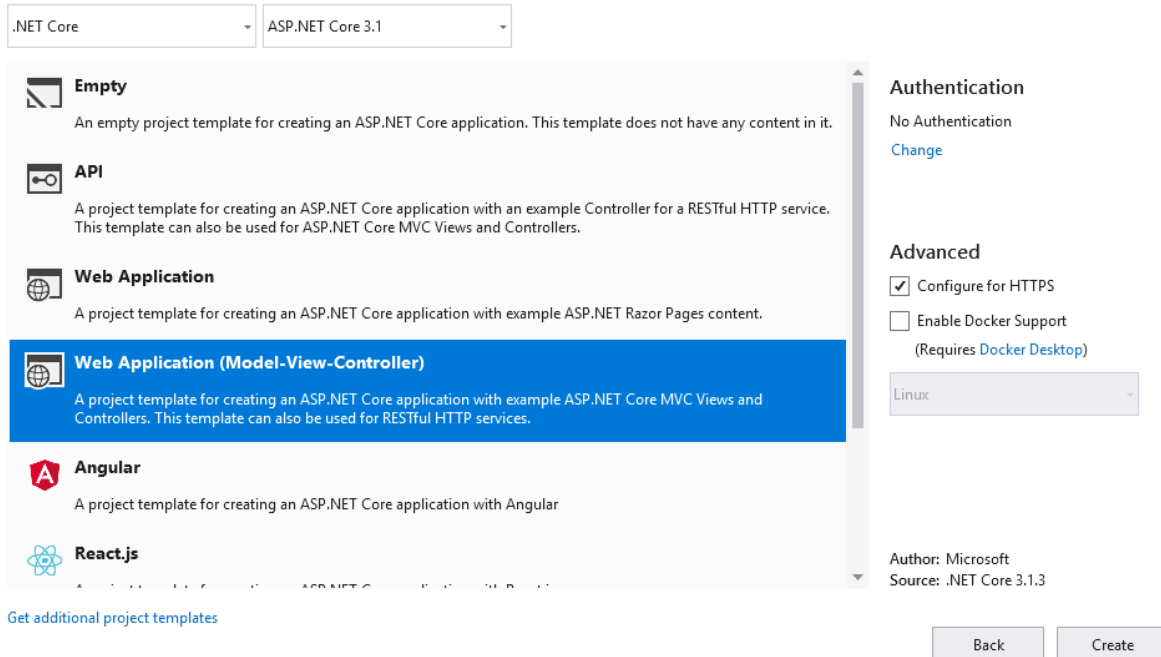
Solution name 
WebApplication2

☐ Place solution and project in the same directory

Back Create

Choose a name for your project and solution. Click Create to continue.

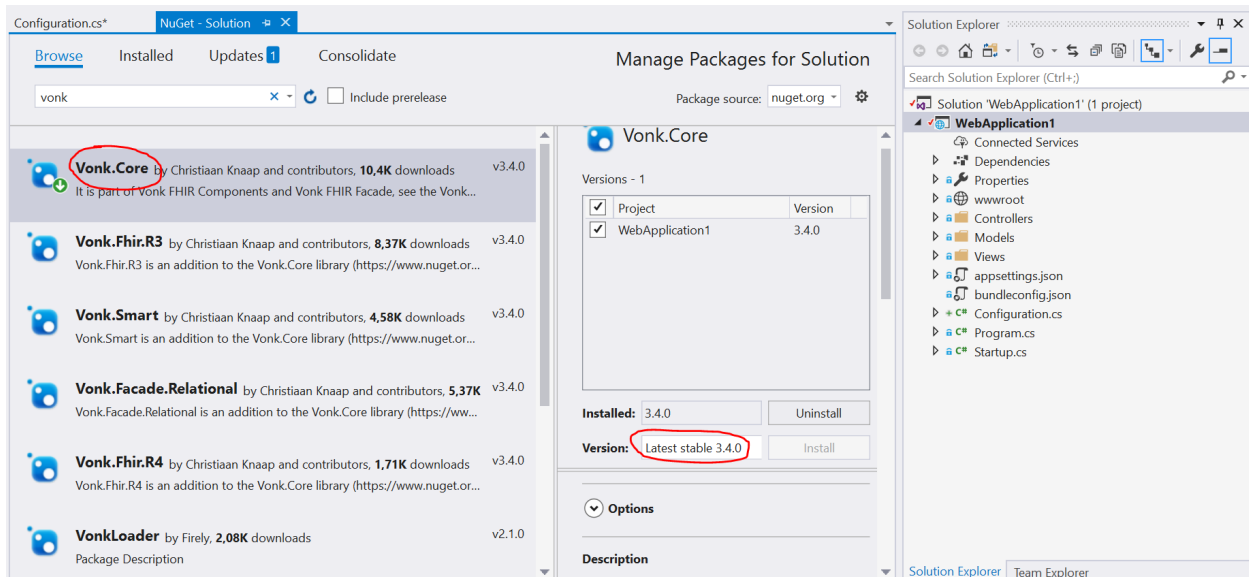
Create a new ASP.NET Core web application



Choose ASP.NET Core 3.1 and select Web Application (Model-View-Controller). Press OK.

12.8.2 Add Firely Server Package

Add Vonk.Core via the Nuget Package Manager:



This will give you access to all the core components of Firely Server, including the Vonk.Core.Pluggability.

VonkConfiguration attribute.

12.8.3 Adjust project file

Add wwwroot and Views as an Embedded resource in the project file (that is necessary for Firely Server to pick them up from a library dll). To edit the project file, right click on the project file and select Edit <projectname>.cproj:

```
<ItemGroup>
  <EmbeddedResource Include="wwwroot\**\*" />
  <EmbeddedResource Include="Views\**" />
</ItemGroup>
```

The project file will look like this:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Vonk.Core" Version="3.4.0" />
  </ItemGroup>

  <ItemGroup>
    <EmbeddedResource Include="wwwroot\**\*" />
    <EmbeddedResource Include="Views\**" />
  </ItemGroup>

</Project>
```

Save the project file.

12.8.4 Create the configuration class

Next, add a new file for the configuration class, as described in *Configuration classes*. Annotate it with [VonkConfiguration(order: 802)]. In the example below the class is named UIConfiguration. Then add the static methods as prescribed:

```
public static IServiceCollection AddUIServices(IServiceCollection services)
{
    var thisAssembly = typeof(UIConfiguration).GetTypeInfo().Assembly;
    services
        .AddMvc(option => option.EnableEndpointRouting = false)
        .AddRazorRuntimeCompilation()
        .AddApplicationPart(thisAssembly)
        .AddControllersAsServices();

    var embeddedFileProvider = new EmbeddedFileProvider(
        thisAssembly,
        thisAssembly.GetName().Name
    );
}
```

(continues on next page)

(continued from previous page)

```
services.Configure<MvcRazorRuntimeCompilationOptions>(options =>
{
    options.FileProviders.Clear();
    options.FileProviders.Add(embeddedFileProvider);
});
return services;
}
```

```
public static IApplicationBuilder UseUI(IApplicationBuilder app)
{
    var thisAssembly = typeof(UIConfiguration).GetTypeInfo().Assembly;
    var embeddedStaticFileProvider = new EmbeddedFileProvider(
        thisAssembly,
        thisAssembly.GetName().Name + ".wwwroot"
    );

    app.UseStaticFiles(new StaticFileOptions() { FileProvider =
        ↪embeddedStaticFileProvider });

    return app.MapWhen(ctx => ctx.IsBrowserRequest(), ab => ab.UseMvcWithDefaultRoute());
}
```

The source file will then look like this:

```

namespace WebApplication2
{
    [VonkConfiguration(order: 802)]
    2 references
    public static class UIConfiguration
    {
        0 references
        public static IServiceCollection AddUIServices(IServiceCollection services)
        {
            var thisAssembly = typeof(UIConfiguration).GetTypeInfo().Assembly;
            services
                .AddMvc(option => option.EnableEndpointRouting = false)
                .AddRazorRuntimeCompilation()
                .AddApplicationPart(thisAssembly)
                .AddControllersAsServices();

            var embeddedFileProvider = new EmbeddedFileProvider(
                thisAssembly,
                thisAssembly.GetName().Name
            );

            services.Configure<MvcRazorRuntimeCompilationOptions>(options =>
            {
                options.FileProviders.Clear();
                options.FileProviders.Add(embeddedFileProvider);
            });
            return services;
        }

        0 references
        public static IApplicationBuilder UseUI(IApplicationBuilder app)
        {
            var thisAssembly = typeof(UIConfiguration).GetTypeInfo().Assembly;
            var embeddedStaticFileProvider = new EmbeddedFileProvider(
                thisAssembly,
                thisAssembly.GetName().Name + ".wwwroot"
            );

            app.UseStaticFiles(new StaticFileOptions { FileProvider = embeddedStaticFileProvider });

            return app.MapWhen(ctx => ctx.IsBrowserRequest(), ab => ab.UseMvcWithDefaultRoute());
        }
    }
}

```

12.8.5 Deploy and Configure

Build this project in Release mode and copy the produced dll (located in <src>\bin\Release\netcoreapp3.1) to the plugin directory of Firely Server, as configured in the *PipelineOptions:PluginDirectory*.

Go to the *Firely Server settings* of Firely Server, and replace the namespace of the landingpage (Vonk.UI.Demo) in the include of the PipelineOptions:

```

"PipelineOptions": {
  "PluginDirectory": "./plugins",
  "Branches": [
    {
      "Path": "/",
      "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",

```

(continues on next page)

(continued from previous page)

```

        "Vonk.Fhir.R4",
        //"Vonk.Fhir.R5"
        "Vonk.Repository.SqlVonkConfiguration",
        "Vonk.Repository.SqliteVonkConfiguration",
        "Vonk.Repository.MongoDbVonkConfiguration",
        "Vonk.Repository.MemoryVonkConfiguration",
        "Vonk.Subscriptions",
        "Vonk.Smart",
        "WebApplication2" //This is the adjustment you make.
        "Vonk.Plugin.DocumentOperation.DocumentOperationConfiguration",
        "Vonk.Plugin.ConvertOperation.ConvertOperationConfiguration",
        "Vonk.Plugin.BinaryWrapper",
        "Vonk.Plugin.MappingToStructureMap.MappingToStructureMapConfiguration",
        "Vonk.Plugin.TransformOperation.TransformOperationConfiguration",
        "Vonk.Plugin.Audit"
    ],
    "Exclude": [
        "Vonk.Subscriptions.Administration"
    ]
},
{
    "Path": "/administration",
    "Include": [
        "Vonk.Fhir.R3",
        "Vonk.Fhir.R4",
        //"Vonk.Fhir.R5"
        "Vonk.Repository.SqlAdministrationVonkConfiguration",
        "Vonk.Repository.SqliteAdministrationVonkConfiguration",
        "Vonk.Repository.MongoDbAdministrationVonkConfiguration",
        "Vonk.Repository.MemoryAdministrationVonkConfiguration",
        "Vonk.Subscriptions.Administration",
        "Vonk.Plugins.Terminology",
        "Vonk.Plugin.Audit",
        "Vonk.Administration"
    ],
    "Exclude": [
        "Vonk.Core.Operations",
        "Vonk.Core.Licensing.LicenseRequestJobConfiguration"
    ]
}
]
}

```

12.8.6 Run and admire

Now run Firely Server from the commandline or Powershell window with

```
> dotnet .\Vonk.Server.dll
```

Open a browser and visit the homepage of Firely Server (<http://localhost:4080>) to admire your own landingpage.

12.9 Firely Server Plugin example - \$document operation

As a more elaborate example we implemented the \$document operation from the FHIR Specification as a plugin.

You can find the code on [GitHub](#).

12.10 BinaryWrapper plugin

12.10.1 Description

Enables you to send binary content to Firely Server and have it stored as a Binary resource, as well as the reverse: get a Binary resource and have it returned in its original binary format. The contents are Base64 encoded and stored inside the resource in the Firely Server database. Therefore this plugin is only suitable for small binary files.

Sending binary content example request:

```
POST <base-url>/Binary
Content-Type = application/pdf
Accept = application/fhir+json; fhirVersion=4.0
Body: enclose a file with the actual contents
```

- This can also be done with a PUT: PUT <base-url>/Binary/example
- The Content-Type must be the mediatype of the actual contents. It will only be accepted by Firely Server if it is one of the mediatypes listed in the RestrictToMimeTypes below.
- The Accept header can be either a fhir mediatype, with any of the supported FHIR versions. You could also set it equal to the Content-Type in which case you will be returned the same contents again.

Getting binary content example request:

```
GET <base-url>/Binary/example
Accept = application/pdf; fhirVersion=4.0
```

- The Accept header should match the mediatype of the actual contents. If you don't know the mediatype, you could request the binary resource in FHIR format first and examine its contentType element.

12.10.2 Configuration

```
"Vonk.Plugin.BinaryWrapper":{
  "RestrictToMimeTypes": ["application/pdf", "text/plain", "image/png", "image/jpeg"]
},
"SizeLimits": {
  "MaxResourceSize": "1MiB", // b/kB/KiB/Mb/MiB, no spaces
},
"PipelineOptions": {
  "Branches": [
    {
      "Path": "/",
      "Include": [
        ...
        "Vonk.Plugin.BinaryWrapper"
      ]
    },
    ...
  ]
}
```

- `RestrictToMimeTypes` protects Firely Server from arbitrary content.
- This plugin honours the Firely Server setting for maximum resource size. This protects Firely Server from binary contents that are too large to store in the database.
- The namespace `Vonk.Plugin.BinaryWrapper` configures both encoding and decoding of binary contents. You can configure them separately as well:

```
"PipelineOptions": {
  "Branches": [
    {
      "Path": "/",
      "Include": [
        ...
        "Vonk.Plugin.BinaryWrapper.BinaryEncodeConfiguration",
        "Vonk.Plugin.BinaryWrapper.BinaryDecodeConfiguration"
      ]
    },
    ...
  ]
}
```


12.10.3 Relationships

The TransformOperation plugin relies on the BinaryWrapper to encode the contents to be mapped, so the *IVonkContext* then contains a proper Binary resource in its payload to work with.

12.10.4 Release notes

Version 0.3.0

- Built against Firely Server (Vonk) 3.2.0
- Compatible with Firely Server (Vonk) 3.2.0, 3.2.1, 3.3.0
- Introduces the decoding of Binary resources, so you can GET a Binary resource in its original binary format.

Version 0.2.0

- Build against Firely Server (Vonk) 3.0.0
- Compatible with Firely Server (Vonk) 3.0.0
- Functionally equivalent to version 0.1.0

Version 0.1.0

- Build against Firely Server (Vonk) 2.1.0
- Compatible with Firely Server (Vonk) 2.1.0
- Introduces the encoding of Binary resources, so you can POST binary contents and have it stored as a Binary resource.

12.11 Convert plugin

12.11.1 Description

Enables you to convert between json and xml representation of a resource using the \$convert operation. It can not convert between different FHIR versions.

Convert example request:

```
POST <base-url>/$convert
Content-Type = application/fhir+json; fhirVersion=4.0
Accept = application/fhir+xml; fhirVersion=4.0
Body: a resource in JSON format
```

- This can also be done reversely, with a body in XML format

12.11.2 Configuration

```
"PipelineOptions": {
  "Branches": [
    {
      "Path": "/",
      "Include": [
        ...
        "Vonk.Plugin.ConvertOperation"
      ]
    },
    ...
  ]
}
```

12.11.3 Release notes

Version 0.2.0

- Build against Firely Server (Vonk) 3.2.0
- Compatible with Firely Server (Vonk) 3.2.0, 3.3.0
- Functionally equivalent to version 0.1.0

Version 0.1.0

- Build against Firely Server (Vonk) 3.0.0
- Compatible with Firely Server (Vonk) 3.0.0, 3.1.0
- Introduces the implementation of \$convert for conversion between json and xml.

FIRELY SERVER FACADE

Firely Server Facade is a means to use the Firely Server implementation of the FHIR RESTful API on top of an existing repository. This repository may be a relational database, a nosql database or even another web api.

This chapter details the two options for setting up a Firely Server Facade, and provides you with an exercise to get some hands-on experience with a Facade solution.

13.1 Facade setup configuration

To set up a Firely Server Facade, you will need to create a library with your own implementation of the interfaces for reading and/or writing FHIR resources and provide that as a plugin to Firely Server.

13.1.1 Provide a plugin to Firely Server

This leverages the capabilities of *Firely Server Plugins*. With this setup you:

- create a new ASP.NET Core library
- include Firely Server NuGet packages
- implement your own repository backend to interface with your data store (can be SQL server or any other medium)
- configure the PipelineOptions to use your library instead of Firely Server's own repository implementation
- configure the PipelineOptions to limit the plugins to those that are supported by your repository implementation.

The benefit of using this approach is that you automatically get to use all of Firely Server's configuration, logging, Application Insights integration, the *Administration API*, etc. described in the other sections of this documentation.

The *exercise* below uses this setup.

Note: Although we take care to try and avoid breaking changes, please be prepared to retest and update your plugins when you choose to update Firely Server.

13.2 Exercise: Build your first Facade

The best way to get experience in developing a Firely Server Facade is by following the exercise - build your first facade. This exercise builds a facade on a simple relational database, by creating a plugin and inserting that into the Firely Server pipeline.

Using a Firely Server Facade allows you to open up legacy systems to the FHIR ecosystem, or add a whole new database backend.

In this exercise you will use Firely Server Facade libraries to build an ASP.NET Core library implementing a FHIR RESTful API on top of an existing database.

The existing database contains two simple tables 'Patient' and 'BloodPressure'. In the exercise we refer to it as the 'ViSi' system, short for 'VitalSigns'.

13.2.1 Git repository Vonk.Facade.Starter

This repository contains the completed exercise. You can find the repository at [Github](#). Of course we do recommend to try and do the exercise yourself first, before looking at the final result.

13.3 Prerequisites and Preparations

- Experience with programming ASP.NET (Core) libraries.
- Basic understanding of the [FHIR RESTful API](#) and FHIR servers.
- Visual Studio 2017 or newer
 1. get a free community edition at <https://www.visualstudio.com/downloads/>
 2. be sure to select the components for C# ASP.NET Core web development
- .NET Core 2.0 SDK, from <https://www.microsoft.com/net/download/windows>
 1. this is probably installed along with the latest Visual Studio, but needed if your VS is not up-to-date.
- SQL Server 2012 or newer:
 1. get a free developer or express edition at <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>
 2. add SQL Server Management Studio from <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms>
- Postman, Fiddler or a similar tool to issue http requests and inspect the responses.

13.3.1 Installing the Firely Server

Before you can start implementing your facade, you need to have the Firely Server installed. See [Getting Started](#) on how to download the binaries and license key.

13.3.2 Preparing the database

Download the `CreateDatabase` script, and create a SQL Server database with it.

It creates a database 'ViSi' with two tables: Patient and BloodPressure. You can familiarize yourself with the table structure and contents to prepare for the mapping to FHIR later on.

Proceed to the next step to start your facade project.

13.4 Starting your project

In this step, you will create and set up your project to start building the facade.

13.4.1 Create new project

1. Open Visual Studio 2017
2. File | New | Project
 - Choose Class Library (.NET Core)
 - Project name and directory at your liking; Click OK

13.4.2 Add Firely Server Packages

1. Tools > NuGet Package Manager > Package Manager Console
 - Run `Install-Package Vonk.Core`
 - Run `Install-Package Vonk.Fhir.R3` (if you want to use R3)
 - Run `Install-Package Vonk.Fhir.R4` (if you want to use R4)

Note: You can install the latest beta release of the Firely Server packages by adding `-IncludePrerelease` to the install command.

13.5 Mapping the database

In this step you will start mapping the existing database model to FHIR resources.

13.5.1 Reverse engineer the database model

To use [EF Core](#), install the package for the database provider(s) you want to target. This walkthrough uses SQL Server. For a list of available providers see [Database Providers](#).

- Tools NuGet Package Manager Package Manager Console
- Run `Install-Package Microsoft.EntityFrameworkCore.SqlServer`

We will be using some Entity Framework Tools to create a model from the database. So we will install the tools package as well:

- Run `Install-Package Microsoft.EntityFrameworkCore.Tools`

Note: The current version of Firely Server uses the latest EF Core libraries. If you are developing for an older Firely Server version, please check the version of `Microsoft.EntityFrameworkCore.SqlServer.dll` in your Firely Server distribution folder. Add `-Version <version>` to the commands above to use the same version in your Facade implementation.

Now it's time to create the EF model based on your existing database.

- Tools NuGet Package Manager Package Manager Console
- Run the following command to create a model from the existing database. Adjust the Data Source to your instance of SQL Server. If you receive an error stating The term 'Scaffold-DbContext' is not recognized as the name of a cmdlet, then close and reopen Visual Studio.:

```
Scaffold-DbContext "MultipleActiveResultSets=true;Integrated Security=SSPI;Persist_
↪Security Info=False;Initial Catalog=ViSi;Data Source=localhost" Microsoft.
↪EntityFrameworkCore.SqlServer -OutputDir Models
//For localdb: Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=ViSi;
↪Trusted_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir_
↪Models
//For SQLEXPRESS: Scaffold-DbContext "Data Source=(local)\SQLEXPRESS;Initial_
↪Catalog=ViSi;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -
↪OutputDir Models
```

You can also generate the scaffolding using the [EF CLI tools](#) which are crossplatform:

```
dotnet ef dbcontext scaffold "User ID=SA;Password=<enter your password here>;
↪MultipleActiveResultSets=true;Server=tcp:.;Connect Timeout=5;Integrated Security=false;
↪Persist Security Info=False;Initial Catalog=ViSi;Data Source=localhost" Microsoft.
↪EntityFrameworkCore.SqlServer --output-dir Models
```

The reverse engineer process creates entity classes (`Patient.cs` & `BloodPressure.cs`) and a derived context (`ViSiContext.cs`) based on the schema of the existing database.

The entity classes are simple C# objects that represent the data you will be querying and saving. Later on you will use these classes to define your queries on and to map the resources from.

13.5.2 Clean up generated code

- To avoid naming confusion with the FHIR Resourcetype Patient, rename both files and classes:
 - Patient `ViSiPatient`
 - BloodPressure `ViSiBloodPressure`

In `ViSiContext.cs`, ensure that the EF objects mapping our class to the database table are correct and without prefixes (since it's just our local classes that have them):

```
public virtual DbSet<ViSiBloodPressure> BloodPressure { get; set; }
public virtual DbSet<ViSiPatient> Patient { get; set; }
```

- The Scaffold command puts your connectionstring in the `ViSiContext` class. That is not very configurable. Later in the exercise, we will add it as 'DbOptions' to the `appsettings.instance.json` file in [3. Configure your Firely Server Facade](#).

- Rename the default `Class1` class to `DbOptions`, and add this to interpret the setting:

```
public class DbOptions
{
    public string ConnectionString { get; set; }
}
```

- Remove the empty constructors from the `ViSiContext` class
- Use the options in your `ViSiContext` class, by adding:

```
private readonly IOptions<DbOptions> _dbOptionsAccessor;

public ViSiContext(IOptions<DbOptions> dbOptionsAccessor)
{
    _dbOptionsAccessor = dbOptionsAccessor;
}
```

- Change the existing `OnConfiguring` method that contains the connectionstring to:

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(_dbOptionsAccessor.Value.ConnectionString);
    }
}
```

13.5.3 Create your first mapping

1. Add a new public class `ResourceMapper` to the project
2. Add usings for `Vonk.Core.Common`, for `Hl7.Fhir.Model`, for `Hl7.Fhir.Support` and for `<your project>.Models`
3. Add a method to the class `public IResource MapPatient(ViSiPatient source)`
4. In this method, put code to create a FHIR Patient object, and fill its elements with data from the `ViSiPatient`:

```
var patient = new Patient
{
    Id = source.Id.ToString(),
    BirthDate = source.DateOfBirth.ToFhirDate()
};
patient.Identifier.Add(new Identifier("http://mycompany.org/patientnumber",
    source.PatientNumber));
// etc.
```

For more examples of filling the elements, see the FHIR API documentation: [FHIR-model](#).

5. Then return the created Patient object as an `IResource` with `patient.ToIResource()`.

`IResource` is an abstraction from actual `Resource` objects as they are known to specific versions of the `Hl7.Fhir.Net` API. See [IResource](#).

13.6 Enable Search

Enabling search involves two major steps:

1. Creating a query to the database based on the bits and pieces in the search url
2. Getting a count and actual data from the database with that query, and map it to a `SearchResult`

The next paragraphs will walk you through these steps.

13.6.1 1. Create a query

Firely Server Facade is meant to be used across all kinds of database paradigms and schemas. Or even against underlying web services or stored procedures. This means Firely Server cannot prescribe the way your query should be expressed. After all, it could be an http call to a webservice, or a json command to MongoDB.

In our case we will build a LINQ query against our ViSi model, that is translated by Entity Framework to a SQL query. Because this is a quite common case, Firely Server provides a basis for it in the package `Vonk.Facade.Relational`.

- Go back to the NuGet Package Manager Console and run `Install-Package Vonk.Facade.Relational`

Note: If you did this previously for the other Firely Server packages, you can install the latest beta release of this package as well by adding `-IncludePrerelease` to the install command.

Adding classes for the query

You usually create a query class per `ResourceType`. The `Query` object is used to capture the elements of the search that are provided to the `QueryFactory`.

In this exercise we start with resource type `Patient`, and will create a `PatientQuery` and `PatientQueryFactory` class. Because `PatientQuery` has no specific content of its own, we will include both in one file.

- Add a new class `PatientQueryFactory` to the root of the project
- Add using statements for `Vonk.Facade.Relational`, `Microsoft.EntityFrameworkCore`, and `<your project>.Models`
- Above the actual `PatientQueryFactory` class insert the `PatientQuery` class:

```
public class PatientQuery: RelationalQuery<ViSiPatient>
{}
```

- Now flesh out the `PatientQueryFactory`:

```
public class PatientQueryFactory: RelationalQueryFactory<ViSiPatient, PatientQuery>
{}
```


Adding a constructor

You have to provide a constructor for the factory class. With this you tell Firely Server for which resource type this QueryFactory is valid. The DbContext is used for retrieving DbSetes for related entities, as we will see later:

```
public PatientQueryFactory(DbContext onContext) : base("Patient", onContext) { }
```

Deciding on a FHIR version

You need to explicitly tell Firely Server for which FHIR version(s) you wish to return resources. If you don't override EntryInformationModel, any search will fail with a 501 Not Implemented. The following override will allow searches for any possible FHIR version to be handled by your facade:

```
public override PatientQuery EntryInformationModel(string informationModel)
{
    return default(PatientQuery);
}
```

If you wish to implement search only for a single FHIR version or for a limited set of versions you can override the method like this:

```
public override PatientQuery EntryInformationModel(string informationModel)
{
    if (informationModel == VonkConstants.Model.FhirR4)
    {
        return default(PatientQuery);
    }

    throw new NotImplementedException($"FHIR version {informationModel} is not supported
    ↪");
}
```

Handling the search request

Each of the searchparameters in the search request triggers a call to the Filter method. This method takes a parameterName and IFilterValue as its arguments.

The parameterName is the searchparameter as it was used in the search url. This name corresponds with the code field in a SearchParameter resource. The IFilterValue value is one of 10 possible implementations, one for each type of SearchParameter. See *IFilterValue implementations* for a short description of these possibilities.

By default the Filter method dispatches the call to a suitable overload of AddValueFilter, based on the actual type of the value parameter. It is up to you to override the ones you support any parameters for.

- Override the method PatientQuery AddValueFilter(string parameterName, TokenValue value) in the PatientQueryFactory class to implement support for the _id parameter, which is a token type parameter.

The _id parameter must be matched against the ViSiPatient.Id property. So we have to:

- Parse the Token.Code to an integer (ViSiPatient.Id is of type int)
- Create a query with a predicate on ViSiPatient.Id.

This is how:

```

if (parameterName == "_id")
{
    if (!int.TryParse(value.Code, out int patientId))
    {
        throw new ArgumentException("Patient Id must be an integer value.");
    }
    else
    {
        return PredicateQuery(vp => vp.Id == patientId);
    }
}
return base.AddValueFilter(parameterName, value);

```

Note: The `ArgumentException` in this code will automatically result in setting the argument status to error, so the Firely Server will send a response with an error code and `OperationOutcome`. See the information about the `IArgumentCollection` and `IArgument` classes in *IVonkContext*.

That's it for now, we will add support for another parameter later.

IFilterValue implementations

There are 10 possible implementations you can use as value for the `IFilterValue` parameter in the Query. The first 7 are the *general search parameter types*: `StringValue`, `DateTimeValue`, `TokenValue`, `NumberValue`, `QuantityValue`, `UriValue` and `ReferenceValue`.

Besides that there are two special values for chaining and reverse chaining: `ReferenceToValue` and `ReferenceFromValue`.

And finally there is a special value for when Firely Server does not know the `SearchParameter` and hence not the type of it: `RawValue`.

This is the second step for enabling search.

13.7 2. Get the data and map to FHIR

Getting the data happens in the implementation of the `ISearchRepository`. It has only one method, `Search`. The `Vonk.Facade.Relational` package has an abstract implementation of it that you can use as a starting point. This implementation assumes that you can support searching for exactly one `ResourceType` at a time. The gist of the implementation is to switch the querying based on the `ResourceType`. The querying itself then looks pretty much the same for every type of resource.

- Add the new class `ViSiRepository` to the root of the project:

```
public class ViSiRepository : SearchRepository
```

- You have to provide a constructor that gets a `QueryContext`. We'll get to that later. Apart from that you will need your `DbContext` (`ViSiContext`) to query on, and the `ResourceMapper` to perform the mapping of the results. So put all of that in the constructor:

```
private readonly ViSiContext _visiContext;
private readonly ResourceMapper _resourceMapper;
```

(continues on next page)

(continued from previous page)

```

public ViSiRepository(QueryContext queryContext, ViSiContext visiContext,
↳ ResourceMapper resourceMapper) : base(queryContext)
{
    _visiContext = visiContext;
    _resourceMapper = resourceMapper;
}

```

- You will have to implement the abstract method `Task<SearchResult> Search(string resourceType, IArgumentCollection arguments, SearchOptions options)`.

- First, let's inspect the parameters:

resourceType

The ResourceType that is being searched for, e.g. Patient in `<firely-server-endpoint>/Patient?...`

arguments

All the arguments provided in the search, whether they come from the path (like 'Patient'), the querystring (after the '?'), the headers or the body. Usually you don't have to inspect these yourself.

options

A few hints on how the query should be executed: are deleted or contained resources allowed etc. Usually you just pass these on as well.

- The pattern of the implementation is:

1. switch on the resourceType
2. dispatch to a method for querying for that resourceType

Naturally we do this async, since in a web application you should never block a thread while waiting for the database.

- To implement this, add this to the class:

```

protected override async Task<SearchResult> Search(string resourceType,
↳ IArgumentCollection arguments, SearchOptions options)
{
    switch (resourceType)
    {
        case "Patient":
            return await SearchPatient(arguments, options);
        default:
            throw new NotImplementedException($"ResourceType {resourceType} is
↳ not supported.");
    }
}

```

- Now we moved the problem to `SearchPatient`, so this method needs to be implemented. The pattern here is:
 1. Create a query - in this case, a `PatientQuery` via `PatientQueryFactory`.
 2. Execute the query against the `DbContext` (our `_visiContext`) to get a count of matches.
 3. Execute the query against the `DbContext` to get the current page of results.
 4. Map the results using the `_resourceMapper`

The implementation of this looks like:

```
private async Task<SearchResult> SearchPatient(IArgumentCollection _arguments, SearchOptions options)
{
    var query = _queryContext.CreateQuery(new PatientQueryFactory(_visiContext), arguments, options);

    var count = await query.ExecuteCount(_visiContext);
    var patientResources = new List<IResource>();

    if (count > 0)
    {
        var visiPatients = await query.Execute(_visiContext).ToListAsync();

        foreach (var visiPatient in visiPatients)
        {
            patientResources.Add(_resourceMapper.MapPatient(visiPatient));
        }
    }
    return new SearchResult(patientResources, query.GetPageSize(), count);
}
```

What happens behind the scenes is that the QueryBuilderContext creates a QueryBuilder that analyzes all the arguments and options, and translates that into calls into your PatientQueryFactory. This pattern offers maximum assistance in processing the search, but also gives you full control over the raw arguments in case you need that for anything. Any argument that is reported as in Error, or not handled will automatically show up in the OperationOutcome of the Firely Server response.

In the next paragraph you will configure your Firely Server to use your Facade, and can – finally – try out some searches. The paragraph after that expands the project to support ViSiBloodPressure Observations, and details how to add custom search parameters.

13.8 Finalizing search

In the previous steps you have created search support for the `_id` parameter on a Patient resource type. In order to test if your Facade implementation works correctly, you will need to perform a couple of steps:

1. Create a configuration class for the [ASP .Net Core pipeline](#)
2. Plug the Facade into the Firely Server
3. Configure the Firely Server to use your repository

13.8.1 1. Add configuration class

To add your repository service to the Firely Server pipeline, you will need to add a configuration class that sets the order of inclusion, and adds to the services. For background information, see [Configuration classes](#).

- Add a static class to your project called ViSiConfiguration
- Add the following code to it:

```
[VonkConfiguration(order: 240)]
public static class ViSiConfiguration
{
    public static IServiceCollection AddViSiServices(this IServiceCollection
↪services, IConfiguration configuration)
    {
        services.AddDbContext<ViSiContext>();
        services.TryAddSingleton<ResourceMapper>();
        services.TryAddScoped<ISearchRepository, ViSiRepository>();

        services.Configure<DbOptions>(configuration.GetSection(nameof(DbOptions)));
        return services;
    }
}
```

13.8.2 2. Create your Facade plugin

- First, build your project
- Find the resulting dll and copy that to the plugins folder in the working directory of your Firely Server

Note: If your Firely Server working directory does not contain a plugins folder yet, you can create one. Within it, you can create subfolders, which can be useful if you work with multiple plugins.

You can also configure the name and location of this folder with the `PipelineOptions.PluginDirectory` setting in the appsettings file.

13.8.3 3. Configure your Firely Server Facade

- Create an appsettings.instance.json file in your Firely Server working directory.

Tip: See [Firely Server settings](#) for more information about the hierarchy of the appsettings(.*).json files and the settings that can be configured.

- Add a setting for the connectionstring to the appsettings.instance.json file:

```
"DbOptions" : { "ConnectionString" : "<paste the connection string to your ViSi
↪database here>" },
```

- Add the SupportedInteractions section. You can look at [Enable or disable interactions](#) to check what this section should contain. For now you only need "WholeSystemInteractions": "capabilities", "InstanceLevelInteractions": "read" and "TypeLevelInteractions": "search":

```
"SupportedInteractions": {  
  "InstanceLevelInteractions": "read",  
  "TypeLevelInteractions": "search",  
  "WholeSystemInteractions": "capabilities"  
},
```

- Add the `SupportedModel` section to indicate which resource types and search parameters you support in your Facade implementation:

```
"SupportedModel": {  
  "RestrictToResources": [ "Patient" ],  
  "RestrictToSearchParameters": ["Resource._id", "StructureDefinition.url"]  
},
```

- You will need to add your repository to the Firely Server pipeline, and remove the existing repository implementations. The standard settings for the pipeline configuration can be found in the `appsettings.default.json` file, or see [Configure the pipeline](#) for an example.
 - Copy the whole `PipelineOptions` section to your `appsettings.instance.json` file (both `/` and `/administration`)
 - To the `Include` part of the branch with `"Path": "/"` add your namespace, and remove the `Vonk.Repository.*` lines from it:

```
{  
  "Path": "/",  
  "Include": [  
    "Vonk.Core",  
    "Vonk.Fhir.R3",  
    "Vonk.Subscriptions",  
    "Vonk.Smart",  
    "Vonk.UI.Demo",  
    "ViSiProject" // fill in (a prefix of) the namespace of your project here  
  ]  
},
```

- Remove the `PipelineOptions` from `appsettings.default.json`, because of the warning mentioned on the [Hierarchy of settings](#).

13.8.4 Test your work

Proceed to the next section to test your Facade, and for some helpful tips about debugging your code.

13.9 Debugging the Facade

- Start your Firely Server

Note: If this is your first startup of Firely Server, it will take a while to load in all of the specification files.

- You can inspect the console log to see if the pipeline is configured to include your repository. See [Detailed logging of loading plugins](#) for more details.

- To test your Facade, open Postman, or Fiddler, or use curl to request GET `http://localhost:4080/metadata`
The resulting CapabilityStatement should list only the Patient resource type in its `.rest.resource` field, and – among others – the `_id` search parameter in the `.rest.searchParam` field.
- Now you can test that searching patients by `_id` works: GET `http://localhost:4080/Patient?_id=1`
Requesting the resource ‘normally’ should automatically work as well: GET `http://localhost:4080/Patient/1`

Important: If it works, congratulations! You now have a Firely Server Facade running!

13.9.1 Testing during implementation

Follow these steps if you want to test your work during the implementation phase without having to build, copy and start Firely Server each time, or with the ability to set break points in your code and debugging it:

- In the project properties, click on the **Build** tab.
- Set the **Output** path to your Firely Server plugins directory.
- Go to the **Debug** tab and set **Launch** to **Executable**.
- Point the **Executable** field to your `dotnet.exe`.
- Set the **Application arguments** to `<your-Firely-Server-working-directory>/Firely.Server.dll`.
- Set the **Working directory** to your Firely Server working directory.

Now, whenever you click to start debugging, Firely Server will start from your project and your project dll will be automatically built to the Firely Server plugins directory.

Next part of the exercise

You can proceed to the next section to add support for Observations as well.

13.10 Finalizing your project

13.10.1 Add support for the ViSiBloodPressure Observations

First, follow similar steps as above to support ViSiBloodPressure:

1. Add a mapping method in the `ResourceMapper` class to map from a `ViSiBloodPressure` to a FHIR Observation resource, and return that as an `IResource`.
2. Create a `BloodPressureQuery` query class.
3. Add a `BPQueryFactory` extending `RelationalQueryFactory<ViSiBloodPressure, BloodPressureQuery>`.
4. Implement support for the `_id` parameter by overriding `public virtual BloodPressureQuery AddValueFilter(string parameterName, TokenValue value)`.
5. Add the Observation type to the `SupportedModel` section in Firely Server’s `appsettings.instance.json`:
`"RestrictToResources": ["Patient", "Observation"]`

When you have completed these steps, build your project again and copy the dll to your Firely Server plugins folder. After you (re)start Firely Server, you will be able to request an Observation through your Facade: GET http://localhost:4080/Observation?_id=1 or GET <http://localhost:4080/Observation/1>.

Since you do not always want to request Observations by their technical id, but more often might want to request Observations from a specific patient, the next part will describe implementing support that. The Patient resource is referenced by the Observation in its subject field. The corresponding search parameter is either subject or patient.

Add support for chaining

To add support for searching on `Observation?subject:Patient._id` we need to override the `AddValueFilter` overload receiving a `ReferenceToValue` parameter in the query factory for BloodPressure (`BPQueryFactory`).

The `ReferenceToValue` type contains the possible `Targets` for the chain search parameter as parsed from the query string. We are currently interested only on the `Patient` type so we can restrict the implementation to that target. The `ReferenceToValue` type also has an extension method `CreateQuery` that expects an implementation of the `RelationalQueryFactory` of the referenced target. This will generate the query to obtain the resources referenced by it.

Searching on chained parameters involves the following steps:

1. Retrieve all patient ids based on the chained parameter. You can use the `ReferenceToValue.CreateQuery` extension method to get the query and run the query with its `Execute` method.
2. Create a `PredicateQuery` with the condition that `ViSiBloodPressure.PatientId` is included in the ids retrieved at the previous step.

The final code should look similar to this:

```
public override BloodPressureQuery AddValueFilter(string parameterName,   
    ↪ReferenceToValue value)   
{   
    if (parameterName == "subject" && value.Targets.Contains("Patient"))   
    {   
        var patientQuery = value.CreateQuery(new   
    ↪PatientQueryFactory(OnContext));   
        var patIds = patientQuery.Execute(OnContext).Select(p => p.Id);   
   
        return PredicateQuery(bp => patIds.Contains(bp.PatientId));   
    }   
    return base.AddValueFilter(parameterName, value);   
}
```

Note: `patIds` is of type `IQueryable`, so the resulting `BloodPressureQuery` will still be executed as a single command to the database.

3. Add support for the `Observation.subject` search parameter in the Firely Server appsettings similar to how we did it for `_id`.

At this point you should be able to search for GET http://localhost:4080/Observation?subject:Patient._id=1

Add support for reverse chaining

Adding support for `Patient?_has:Observation:subject:_id=1` is similar. You just need to use the `AddValueFilter` overload receiving a `ReferenceFromValue`.

The `ReferenceFromValue` type has a `Source` property filled in with the source of the search parameter. It also has an extension method `CreateQuery` that given the corresponding `RelationalQueryFactory` implementation can generate the query to obtain resources referenced by the reverse chaining.

So you can add reverse chaining with the following code:

```
public class PatientQueryFactory
{
    public override PatientQuery AddValueFilter(string parameterName, ReferenceFromValue_
↪value)
    {
        if (parameterName == "subject" && value.Source == "Observation")
        {
            var obsQuery = value.CreateQuery(new BPQueryFactory(OnContext));
            var obsIds = obsQuery.Execute(OnContext).Select(bp => bp.PatientId);

            return PredicateQuery(p => obsIds.Contains(p.Id));
        }
        return base.AddValueFilter(parameterName, value);
    }
}
```

Note: The reverse chaining example above uses the `Patient` resource type as its base, so you will need to implement this in your `PatientQueryFactory`.

Now you can test if reverse chaining works: http://localhost:4080/Patient?_has:Observation:subject:_id=1

13.10.2 Get the goodies

At this point you get out of the box support for `_include` and `_revinclude` (:iterate as well), and combinations of search parameters. You can test the following scenarios:

1. `_include`: http://localhost:4080/Observation?_include=Observation:subject
2. `_revinclude`: http://localhost:4080/Patient?_revinclude=Observation:subject
3. combinations of the above

13.10.3 Adding a custom SearchParameter

Your Firely Server will load the standard parameters from the specification on first startup, so the `_id` SearchParameter from the exercise is already known to Firely Server, as well as any of the other standard search parameters for the resource types.

If you want to implement support for a custom search parameter, you will need to have the definition of that in the form of a SearchParameter resource, and add it to your Firely Server. The [Configure Search Parameters](#) section describes how to do that.

Of course you will also need to implement the correct `AddValueFilter` method in your `<resourcetype>QueryFactory` to handle the parameter correctly, as is done for the `_id` parameter in the exercise.

13.10.4 The end?

This concludes the exercise. An example [Github repository](#) contains the completed exercise.

Please feel free to try out more options, and [ask for help](#) if you get stuck!

The next topic will show you how to enable [Create, Update and Delete](#) interactions.

13.10.5 Postscript

If your resource is split across multiple tables in the database, you'll need to make use of `.Include()` to have EF load the dependent table. To do so in Firely Server, override the `GetEntitySet()` method in your `RelationalQuery` class, for example

```
protected override IQueryable<ViSiPatient> GetEntitySet(DbContext dbContext)
{
    // load the dependent Address table
    return dbContext.Set<ViSiPatient>().Include(p => p.Address).AsNoTracking();
}
```

13.11 Enable changes to the repository

In [Exercise: Build your first Facade](#) you have created read and search support for a Firely Server Facade on top of an existing database. The next part will walk you through enabling create, update and delete support. This will be done in three steps:

1. Map the FHIR resource to the database model;
2. Implement the `IResourceChangeRepository`;
3. Indicate support in the appsettings file

13.11.1 1. Map the FHIR data to the model

For both the Create and Update interactions, we need to map our incoming FHIR data to the ViSi model. To do that, we will add new mapping methods to our ResourceMapper class.

- Add a method called MapVisiPatient to the ResourceMapper class, that takes a FHIR Patient and returns a ViSiPatient.
- Implement the mapping from the FHIR object to the ViSiPatient model:

```
public ViSiPatient MapVisiPatient(IResource source)
{
    var fhirPatient = source.ToPoco<Patient>();
    var visiPatient = new ViSiPatient();

    if (source.Id != null)
    {
        if (!int.TryParse(source.Id, out int id))
            throw new VonkRepositoryException("Id needs to be integer to map_
↪resource");
        visiPatient.Id = id;
    }
    visiPatient.PatientNumber = fhirPatient.Identifier.Find(i =>
        (i.System == "http://mycompany.org/patientnumber")).
↪Value;

    // etc.

    return visiPatient;
}
```

- Where it says 'etc.', fill in the rest of the code to map the data to required fields of the database, and any other fields you have data for.

13.11.2 2. Implement the IResourceChangeRepository

You are going to implement a repository that handles changes to your database. The interface for this is called IResourceChangeRepository, which can be found in Vonk.Core.Repository.

- Add a new class ViSiChangeRepository to the project, that implements the IResourceChangeRepository:

```
public class ViSiChangeRepository : IResourceChangeRepository
```

- Choose to implement the interface, so the required methods are added to the class.
- Just like with the search repository, you will need your DbContext to query on, and the ResourceMapper to perform the mapping of the incoming data to your proprietary model.

So put all of that in the constructor:

```
private readonly ViSiContext _visiContext;
private readonly ResourceMapper _resourceMapper;

public ViSiChangeRepository(ViSiContext visiContext, ResourceMapper resourceMapper)
{
```

(continues on next page)

(continued from previous page)

```
_visiContext = visiContext;
_resourceMapper = resourceMapper;
}
```

Implementing Create

- Now implement the Create method with a switch on resource type, so you can add other resource types later:

```
public async Task<IResource> Create(IResource input)
{
    switch (input.Type)
    {
        case "Patient":
            return await CreatePatient(input);
        default:
            throw new NotImplementedException($"ResourceType {input.Type} is not_
↪supported.");
    }
}
```

- As you can see, we have deferred the work to a CreatePatient method, which we also need to implement. This method will add the new resource to the collection, and save the changes to the database:

```
private async Task<IResource> CreatePatient(IResource input)
{
    var visiPatient = _resourceMapper.MapViSiPatient(input);

    await _visiContext.Patient.AddAsync(visiPatient);
    await _visiContext.SaveChangesAsync();

    // return the new resource as it was stored by this server
    return _resourceMapper.MapPatient(_visiContext.Patient.Last());
}
```

- For the Create and Update methods, you will also need to implement the NewId and NewVersion methods, because Firely Server will call them. For the NewId method, we will return null, since our ViSi database does not allow us to create our own index value. Since our ViSi repository does not handle versions, we will let the NewVersion method return null as well:

```
public string NewId(string resourceType)
{
    return null;
}

public string NewVersion(string resourceType, string resourceId)
{
    return null;
}
```

Note: For the ViSi repository we're using a null value, but you can implement this method any way that's useful for

your own repository. The public Firely Server for example generates a GUID in these methods.

At this point you can skip ahead to [3. Configure the service and Firely Server](#), if you want to try and create a new patient in the ViSi database.

Tip: This is easiest to test if you retrieve an existing resource from the database first with your HTTP tool. Then change some of the data in the resulting JSON or XML, and send that back to your Facade.

Implementing Update

Implementing the Update method can be done like the Create, with a switch on resource type, and instead of adding a resource to the collection, you will update the collection:

```
private async Task<IResource> UpdatePatient(ResourceKey original, IResource update)
{
    try
    {
        var visiPatient = _resourceMapper.MapViSiPatient(update);

        var result = _visiContext.Patient.Update(visiPatient);
        await _visiContext.SaveChangesAsync();

        return _resourceMapper.MapPatient(result.Entity);
    }
    catch (Exception ex)
    {
        throw new VonkRepositoryException($"Error on update of {original} to {update.
↵Key()}", ex);
    }
}
```

Implementing Delete

Deleting a resource from the collection is done by first looking up the corresponding resource, and then removing it from the collection. Note that the database used for this exercise cannot process the deletion of the Patient when there are still related Observations in the BloodPressure table, so we need to remove them as well or choose to throw an error.

- First, create a switch on resource type in the main Delete method again.
- Implement the DeletePatient:

```
private async Task<IResource> DeletePatient(ResourceKey toDelete)
{
    int toDelete_id = int.Parse(toDelete.ResourceId);
    var visiPatient = _visiContext.Patient.Find(toDelete_id);

    var bpEntries = _visiContext.BloodPressure.Where(bp => bp.PatientId == toDelete_
↵id);

    var result = _resourceMapper.MapPatient(visiPatient);
}
```

(continues on next page)

(continued from previous page)

```

try
{
    _visiContext.BloodPressure.RemoveRange(bpEntries);
    _visiContext.Patient.Remove(visiPatient);
    await _visiContext.SaveChangesAsync();
}
catch (Exception ex)
{
    throw new VonkRepositoryException($"Error on deleting Patient with Id
→{toDelete_id}", ex);
}

return result;
}

```

13.11.3 3. Configure the service and Firely Server

Just like with the search repository, you will need to add your change repository as service to the pipeline. Also, you will need to indicate support for the CRUD interactions in your Firely Server appsettings.

- In your project, go to the ViSiConfiguration class, and add this line to add an IResourceChangeRepository to the pipeline:

```
services.TryAddScoped<IResourceChangeRepository, ViSiChangeRepository>();
```

- Add support for the interactions to the SupportedModel section of the Firely Server appsettings:

```

"SupportedInteractions": {
  "InstanceLevelInteractions": "read, update, delete",
  "TypeLevelInteractions": "search, create",
  "WholeSystemInteractions": "capabilities, search"
},

```

- Adjust `PipelineOptions.Branches.Include` from `Vonk.Core.Operations.Crud`. ReadConfiguration to `Vonk.Core.Operations` include all operations, including Create.

You can now build your project, copy the dll to the Firely Server plugins folder and run Firely Server to test the new interactions on your Facade.

13.11.4 The end?

This concludes the second exercise. Please feel free to try out more options, and *ask for help* if you get stuck!

The next topic will show you how to integrate *Access Control*.

FIRELY SERVER REFERENCE DOCUMENTATION

The reference documentation lists the available plugins for configuring the pipeline, and the public programming API of Firely Server for building Plugins and Facades.

14.1 Plugins available for Firely Server

14.1.1 Infrastructural plugins

Name

Scheduler

Configuration

`Vonk.Core.Quartz.QuartzConfiguration`

License token

<http://fire.ly/vonk/plugins/infra>

Order

10

Description

Registers a scheduler that can run jobs periodically. You can use this yourself, but with care (you don't want jobs slowing down the server):

- Implement a `Quartz.IJob`, let's say with class `MyJob {...}`
- Have the `Quartz.IScheduler` injected
- Call `IScheduler.StartJob<MyJob>(TimeSpan runInterval, CancellationToken cancellationToken)`

Name

Maintenance

Configuration

`Vonk.Core.Infra.MaintenanceConfiguration`

License token

<http://fire.ly/vonk/plugins/infra>

Order

20

Description

Periodically cleans the indexed values for deleted or superceded resources from the database.

Name

License

Configuration

Vonk.Core.Licensing.LicenseConfiguration

License token

<http://fire.ly/vonk/plugins/infra>

Order

120

Description

Registers the LicenseService that checks for a valid license. Without this plugin Firely Server does not work.

Name

Serialization

Configuration

Vonk.Core.Serialization.SerializationConfiguration

License token

<http://fire.ly/vonk/plugins/infra>

Order

130

Description

Registers an implementation for the ISerializationService and ISerializationSupport interfaces and actual serializers and parsers for JSON and XML.

Name

Pluggability

Configuration

Vonk.Core.Pluggability.PluggabilityConfiguration

License token

<http://fire.ly/vonk/plugins/infra>

Order

150

Description

Registers services to dynamically build the IModelService using registered IModelContributor implementations.

Name

Http to Vonk

Configuration

Vonk.Core.Context.Http.HttpToVonkConfiguration

License token

<http://fire.ly/vonk/plugins/http>

Order

1110

Description

Builds an *IVonkContext* out of the *HttpContext*. You can only access the *IVonkContext* in the pipeline from plugins that have a higher order.

Name

Vonk to Http

Configuration`Vonk.Core.Context.Http.VonkToHttpConfiguration`**License token**<http://fire.ly/vonk/plugins/http>**Order**

1120

Description

Translates the response in the *IVonkContext* to a response on the *HttpContext*. It honors the value of the prefer header if present. It also adds the *VonkExceptionHandler* to the pipeline as a last resort for catching exceptions.

Name

Formatter

Configuration`Vonk.Core.Context.Format.FormatConfiguration`**License token**<http://fire.ly/vonk/plugins/infra>**Order**

1130

Description

Registers an implementation of *IFormatter* that can write the *IVonkContext.Response.Payload* to the response body in the requested format. Does not add a processor to the pipeline.

Name

Long running tasks

Configuration`Vonk.Core.Infra.LongRunning.LongRunningConfiguration`**License token**<http://fire.ly/vonk/plugins/infra>**Order**

1170

Description

If Vonk processes a task that could lead to inconsistent output, all other requests are rejected by this plugin. Long running tasks are e.g. the *Import of Conformance Resources* and *Re-indexing for new or changed SearchParameters*.

Name

Compartments

Configuration`Vonk.Core.Context.Features.CompartmentsConfiguration`**License token**<http://fire.ly/vonk/plugins/search>**Order**

1210

Description

Recognizes a compartment in a compartment search on system or type level (see [Search](#)). It is added as a feature of type `ICompartment` to the `IVonkContext.Features` collection, to be used by [Search](#) later on. This `ICompartment` feature will limit all queries to within the specified compartment.

Name

Supported Interactions

Configuration

`Vonk.Core.Context.Guards.SupportedInteractionsConfiguration`

License token

<http://fire.ly/vonk/plugins/infra>

Order

1220

Description

Blocks interactions that are not listed as supported.

Options

`SupportedInteractions`, see [Enable or disable interactions](#).

Name

Size Limits

Configuration

`Vonk.Core.Context.Guards.SizeLimitsConfiguration`

License token

<http://fire.ly/vonk/plugins/infra>

Order

1225

Description

Rejects bodies that are too large and bundles with too many entries.

Options

`SizeLimits`, see [Protect against large input](#)

Name

Url mapping

Configuration

`Vonk.Core.Context.UrlMapping.UrlMappingConfiguration`

License token

<http://fire.ly/vonk/plugins/infra>

Order

1235

Description

In a resource in the request, urls pointing to this instance of Firely Server are made relative. In a resource in the response, relative urls are made absolute, by adding the base url of the server. This way the server can be addressed in multiple ways (e.g. <http://intranet.acme.com/fhir> and <https://fhir.acme.com>) and still provide correct absolute urls.

Name

Default Shapes

Configuration

`Vonk.Core.Context.Guards.DefaultShapesConfiguration`

License token

<http://fire.ly/vonk/plugins/infra>

Order

4110

Description

If no sort order is given for a search, `_lastUpdated:asc` is added. If no count is given for a search, `_count=<default count>` is added.

Options

`BundleOptions.DefaultCount`, see *Search and History*.

14.1.2 Support for different FHIR versions

Name

FHIR R3

Configuration

`Vonk.Fhir.R3.FhirR3Configuration`

License token

<http://fire.ly/vonk/plugins/fhirr3>

Order

100

Description

Registers services to support FHIR STU3 (or R3).

Name

FHIR R3 Specification

Configuration

`Vonk.Fhir.R3.FhirR3SpecificationConfiguration`

License token

<http://fire.ly/vonk/plugins/fhirr3>

Order

112

Description

Registers an `Hl7.Fhir.Specification.IStructureDefinitionSummaryProvider` for FHIR STU3 (or R3).

Name

FHIR R4

Configuration

`Vonk.Fhir.R4.FhirR4Configuration`

License token

<http://fire.ly/vonk/plugins/fhirr4>

Order

101

Description

Registers services to support FHIR R4.

Name

FHIR R4 Specification

Configuration

`Vonk.Fhir.R4.FhirR4SpecificationConfiguration`

License token

<http://fire.ly/vonk/plugins/fhrr4>

Order

112

Description

Registers an `Hl7.Fhir.Specification.IStructureDefinitionSummaryProvider` for FHIR R4.

14.1.3 FHIR RESTful interactions

Name

Read

Configuration

`Vonk.Core.Operations.Crud.ReadConfiguration`

License token

<http://fire.ly/vonk/plugins/read>

Order

4410

Description

Implements FHIR instance read. It will return the Resource that matches the id *and* the FHIR version. If a Resource with matching id is found with another FHIR version you are notified.

Name

Create

Configuration

`Vonk.Core.Operations.Crud.CreateConfiguration`

License token

<http://fire.ly/vonk/plugins/create>

Order

4420

Description

Implements FHIR type create.

Name

Update

Configuration

`Vonk.Core.Operations.Crud.UpdateConfiguration`

License token

<http://fire.ly/vonk/plugins/update>

Order

4430

Description

Implements FHIR instance update, with support for 'upsert': creating a Resource with a pre-assigned id. Note that id's must be unique across FHIR versions.

Name

Patch

Configuration

```
Vonk.Core.Operations.Crud.FhirPatchConfiguration
```

License token
<http://fire.ly/vonk/plugins/update>
Order

4433

Description

Implements FHIR instance patch, as specified by [FHIR Patch](#).

Name

Delete

Configuration

```
Vonk.Core.Operations.Crud.DeleteConfiguration
```

License token
<http://fire.ly/vonk/plugins/delete>
Order

4440

Description

Implements FHIR instance delete. Since id's in Firely Server must be unique across FHIR versions, the delete is issued on the provided id, regardless of the FHIR version.

Name

Search

Configuration

```
Vonk.Core.Operations.Search.SearchConfiguration
```

License token
<http://fire.ly/vonk/plugins/search>
Description

Implements FHIR Search on system and type level. For data access it uses the registered implementation of `ISearchRepository`, which can be any of the implementations provided by Firely Server or an implementation provided by a Facade plugin. The implementations provided by Firely Server also require the Index plugin to extract searchparameter values from the resources.

Order

4220

Options

- `AdministrationImportOptions`, see *SearchParameters and other Conformance Resources*, for available Searchparameters
- `SupportedModel.RestrictToSearchParameters`, see *Restrict supported resources and SearchParameters* for available Searchparameters

- `BundleOptions`, see *Search and History*, for number of returned results

See *ISearchRepository* and *Firely Server Facade*.

Name

Search support

Configuration

`Vonk.Core.Repository.RepositorySearchSupportConfiguration`

License token

<http://fire.ly/vonk/plugins/search>

Order

140

Description

Registers services required for Search. It is automatically registered by Search.

Name

Index

Configuration

`Vonk.Core.Repository.RepositoryIndexSupportConfiguration`

License token

<http://fire.ly/vonk/plugins/index>

Order

141

Description

Extracts values matching Searchparameters from resources, so they can be searched on.

Name

Include

Configuration

`Vonk.Core.Operations.Search.IncludeConfiguration`

License token

<http://fire.ly/vonk/plugins/include>

Order

4210

Description

Implements `_include` and `_revinclude`. This acts on the result bundle of a search. Therefore it also works out of the box for Facade implementations, provided that the Facade implements support for the reference Searchparameters that are used in the `_(rev)include`.

Name

Elements

Configuration

`Vonk.Core.Context.Elements.ElementsConfiguration`

License token

<http://fire.ly/vonk/plugins/search>

Order

1240

Description

Applies the `_elements` parameter to the Resource that is in the response (single resource or bundle).

Name

Summary

Configuration

`Vonk.Core.Context.Elements.SummaryConfiguration`

License token

<http://fire.ly/vonk/plugins/search>

Order

1240

Description

Applies the `_summary` parameter to the Resource that is in the response (single resource or bundle).

Name

History

Configuration

`Vonk.Core.Operations.History.HistoryConfiguration`

License token

<http://fire.ly/vonk/plugins/history>

Order

4610

Description

Implements `_history` on system, type and instance level.

Options

BundleOptions, see *Search and History*

Name

Version Read

Configuration

`Vonk.Core.Operations.History.VersionReadConfiguration`

License token

<http://fire.ly/vonk/plugins/history>

Order

4620

Description

Implements reading a specific version of a resource (`<base>/Patient/123/_history/v3`).

Name

Capability

Configuration

`Vonk.Core.Operations.Capability.CapabilityConfiguration`

License token

<http://fire.ly/vonk/plugins/capability>

Order

4120

Description

Provides the CapabilityStatement on the <base>/metadata endpoint. The CapabilityStatement is tailored to the FHIR version of the request. The CapabilityStatement is built dynamically by visiting all the registered implementations of ICapabilityStatementContributor, see [Capabilities](#).

Name

Conditional Create

Configuration

Vonk.Core.Operations.ConditionalCrud.ConditionalCreateConfiguration

License token

<http://fire.ly/vonk/plugins/conditionalcreate>

Order

4510

Description

Implements FHIR conditional create.

Name

Conditional Update

Configuration

Vonk.Core.Operations.ConditionalCrud.ConditionalUpdateConfiguration

License token

<http://fire.ly/vonk/plugins/conditionalupdate>

Order

4520

Description

Implements FHIR conditional update.

Name

Conditional Delete

Configuration

Vonk.Core.Operations.ConditionalCrud.ConditionalDeleteConfiguration

License token

<http://fire.ly/vonk/plugins/conditionaldelete>

Order

4530

Description

Implements FHIR conditional delete.

Options

FhirCapabilities.ConditionalDeleteOptions, see [FHIR Capabilities](#)

Name

Validation

Configuration

Vonk.Core.Operations.Validation.ValidationConfiguration

License token

<http://fire.ly/vonk/plugins/validation>

Order

4000

Description

Implements [FHIR \\$validate](#) on type and instance level for POST: POST <base>/Patient/\$validate or POST <base>/Patient/123/\$validate.

Name

Instance Validation

Configuration

Vonk.Core.Operations.Validation.InstanceValidationConfiguration

License token

<http://fire.ly/vonk/plugins/validation>

Order

4840

Description

Implements [FHIR \\$validate](#) on instance level for GET: GET <base>/Patient/123/\$validate

Name

Structural Validation

Configuration

Vonk.Core.Operations.Validation.StructuralValidationConfiguration

License token

<http://fire.ly/vonk/plugins/validation>

Order

1227

Description

Validates the structure of resources sent to Firely Server (is it valid FHIR JSON or XML?).

Name

Prevalidation

Configuration

Vonk.Core.Operations.Validation.PreValidationConfiguration

License token

<http://fire.ly/vonk/plugins/validation>

Order

1228

Description

Validates resources sent to Firely Server against their stated profile compliance (in Resource.meta.profile). The strictness of the validation is controlled by the options.

Options

Validation, see [Validation](#)

Name

Profile filter

Configuration

Vonk.Core.Operations.Validation.ProfileFilterConfiguration

License token

<http://fire.ly/vonk/plugins/validation>

Order

4310

Description

Blocks resources that do not conform to a list of profiles.

Options

Validation.AllowedProfiles, see [Validation](#)

Name

Meta

Configuration

Vonk.Core.Operations.MetaOperation.MetaConfiguration

License token

<http://fire.ly/vonk/plugins/meta>

Order

5180

Description

Implements FHIR \$meta on instance level.

Name

Meta Add

Configuration

Vonk.Core.Operations.MetaOperation.MetaAddConfiguration

License token

<http://fire.ly/vonk/plugins/meta>

Order

5190

Description

Implements FHIR \$meta-add on instance level.

Name

Meta Delete

Configuration

Vonk.Core.Operations.MetaOperation.MetaDeleteConfiguration

License token

<http://fire.ly/vonk/plugins/meta>

Order

5195

Description

Implements FHIR \$meta-delete on instance level.

Name

Snapshot Generation

Configuration

Vonk.Core.Operations.SnapshotGeneration.SnapshotGenerationConfiguration

License token

<http://fire.ly/vonk/plugins/snapshotgeneration>

Order

4850

Description

Implements [FHIR \\$snapshot](#) on a type level: POST <base>/administration/StructureDefinition/\$snapshot.

Name

Batch

Configuration

Vonk.Core.Operations.Transaction.FhirBatchConfiguration

License token

<http://fire.ly/vonk/plugins/batch>

Order

3110

Description

Processes a batch Bundle by sending each entry through the rest of the processing pipeline and gathering the results.

Options

SizeLimits, see *Protect against large input*

Name

Transaction

Configuration

Vonk.Core.Operations.Transaction.FhirTransactionConfiguration

License token

<http://fire.ly/vonk/plugins/transaction>

Order

3120

Description

Process a transaction Bundle by sending each entry through the rest of the processing pipeline and gathering the results. Different from Batch, Transaction succeeds or fails as a whole. Transaction requires an implementation of Vonk.Core.Repository.IRepoTransactionService for transaction support by the underlying repository. The SQL Server and SQLite implementations provides a real one, whereas the MongoDB provides a simulated implementation, to allow you to experiment with transactions on MongoDB.

Options

- SizeLimits, see *Validation*
- Repository, see *Repository*

14.1.4 Terminology

Name

CodeSystem Lookup

Configuration

Vonk.Plugins.Terminology.[R3|R4|R5].CodeSystemLookupConfiguration

License token

<http://fire.ly/vonk/plugins/terminology>

Order

5110

Description

Implements FHIR [\\$lookup](#) on type level requests: POST <base>/administration/CodeSystem/\$lookup or GET <base>/administration/CodeSystem/\$lookup?...

Name

CodeSystem FindMatches / Compose

Configuration

Vonk.Plugins.Terminology.CodeSystemFindMatchesConfiguration

License token

<http://fire.ly/vonk/plugins/terminology>

Order

5220

Description

Implements FHIR [\\$compose](#) on type level requests: POST <base>/administration/CodeSystem/\$find-matches`` and on instance level requests: ``POST <base>/administration/CodeSystem/[id]/\$find-matches or GET <base>/administration/CodeSystem/[id]/\$find-matches?...

Name

ValueSet Validate Code

Configuration

Vonk.Plugins.Terminology.ValueSetValidateCodeConfiguration

License token

<http://fire.ly/vonk/plugins/terminology>

Order

5120

Description

Implements FHIR [\\$validate-code](#) on type level requests: POST <base>/administration/ValueSet/\$validate-code and instance level requests: GET <base>/administration/ValueSet/[id]/\$validate-code?... and POST <base>/administration/ValueSet/[id]/\$validate-code

Name

ValueSet Expand

Configuration

Vonk.Plugins.Terminology.ValueSetExpandConfiguration

License token

<http://fire.ly/vonk/plugins/terminology>

Order

5140

Description

Implements FHIR [\\$expand](#) on instance level requests: GET <base>/administration/ValueSet/[id]/\$expand?... and POST <base>/administration/ValueSet/[id]/\$expand and on type level requests: POST <base>/administration/ValueSet/\$expand.

Name

ConceptMap Translate

Configuration

Vonk.Plugins.Terminology.ConceptMapTranslateConfiguration

License token

<http://fire.ly/vonk/plugins/terminology>

Order

5260

Description

Implements FHIR `$translate` on instance level requests: GET `<base>/administration/ConceptMap/[id]/$translate?...` and POST `<base>/administration/ValueSet/[id]/$translate` and on type level requests: POST `<base>/administration/ConceptMap/$translate`.

Name

CodeSystem Subsumes

Configuration

Vonk.Plugins.Terminology.CodeSystemSubsumesConfiguration

License token

<http://fire.ly/vonk/plugins/terminology>

Order

5280

Description

Implements FHIR `$subsumes` on instance level requests: GET `<base>/administration/CodeSystem/[id]/$subsumes?...` and on type level requests: POST `<base>/administration/CodeSystem/$subsumes` or GET `<base>/administration/CodeSystem/$subsumes?...`

Name

CodeSystem Closure

Configuration

Vonk.Plugins.Terminology.CodeSystemClosureConfiguration

License token

<http://fire.ly/vonk/plugins/terminology>

Order

5300

Description

Implements FHIR `$closure` on system level requests: POST `<base>/administration/$closure`

14.1.5 SMART on FHIR

Name

SMART on FHIR

Configuration

Vonk.Smart.SmartConfiguration.SmartConfiguration

License token

<http://fire.ly/vonk/plugins/smartonfhir>

Order

2000

Description

Implements SMART on FHIR authentication and authorization, see [Access control and SMART](#).

14.1.6 Subscriptions

Name

Subscriptions

Configuration

Vonk.Subscriptions.SubscriptionConfiguration.SubscriptionConfiguration

License token

<http://fire.ly/vonk/plugins/subscriptions>

Order

3200

Description

Implements the FHIR Subscriptions framework, see [Subscriptions](#).

14.1.7 Auditing

Name

Username log

Configuration

Vonk.Plugin.Audit.UsernameLoggingConfiguration

License token

<http://fire.ly/vonk/plugins/audit>

Order

2010

Description

Makes the user id and name from the JWT token (if present) available for logging. See [Auditing](#) for more info.

Name

Audit logging for transactions

Configuration

Vonk.Plugin.Audit.AuditTransactionConfiguration

License token

<http://fire.ly/vonk/plugins/audit>

Order

3100

Description

Logs requests and responses for transactions to a file. See [Auditing](#) for more info.

Name

Audit log

Configuration

Vonk.Plugin.Audit.AuditConfiguration

License token<http://fire.ly/vonk/plugins/audit>**Order**

3150

Description

Logs requests and responses to a file. See [Auditing](#) for more info.

Name

AuditEvent logging for transactions

Configuration

Vonk.Plugin.Audit.AuditEventTransactionConfiguration

License token<http://fire.ly/vonk/plugins/audit>**Order**

3160

Description

Logs requests and responses for transactions to a file. See [Auditing](#) for more info.

Name

AuditEvent logging

Configuration

Vonk.Plugin.Audit.AuditEventConfiguration

License token<http://fire.ly/vonk/plugins/audit>**Order**

3170

Description

Logs requests and responses to a file. See [Auditing](#) for more info.

14.1.8 Demo UI

Name

Demo UI

Configuration

Vonk.UI.Demo.DemoUIConfiguration.DemoUIConfiguration

License token<http://fire.ly/vonk/plugins/demoui>**Order**

800

Description

Provides the landing page that you see when you request the base url from a browser. If you want to provide your own landing page, replace this plugin with your own. There is an example of that, see [Firely Server Plugin example - Create a new landing page](#).

14.1.9 Documents

Name

Document generation

Configuration

Vonk.Plugins.DocumentOperation.DocumentOperationConfiguration

License token

<http://fire.ly/vonk/plugins/document>

Order

4900

Description

Implements FHIR *\$document* : POST <base>/Composition/\$document or GET <base>/Composition/[id]/\$document

Code

[GitHub](#)

Name

Document signing

Configuration

Vonk.Plugins.SignatureService.SignatureConfiguration

License token

<http://fire.ly/vonk/plugins/signature>

Order

4899

Description

Signs a document generated by *\$document*.

14.1.10 Conversion

Name

Format conversion

Configuration

Vonk.Plugins.ConvertOperation.ConvertOperationConfiguration

License token

<http://fire.ly/vonk/plugins/convert>

Order

4600

Description

Implements FHIR *\$convert* : POST <base>/\$convert to convert between JSON and XML representation.

14.1.11 Binary

Name

Binary wrapper (Encode)

Configuration

Vonk.Plugins.BinaryWrapper.BinaryEncodeConfiguration

License token

<http://fire.ly/vonk/plugins/binarywrapper>

Order

1112

Description

Wraps an incoming binary format in a Binary resource for further processing by the pipeline.

Settings

```
"Vonk.Plugin.BinaryWrapper":{
  "RestrictToMimeTypes": ["application/pdf", "text/plain", "image/png",
↪ "image/jpeg"]
},
```

Name

Binary wrapper (Decode)

Configuration

Vonk.Plugins.BinaryWrapper.BinaryDecodeConfiguration

License token

<http://fire.ly/vonk/plugins/binarywrapper>

Order

1122

Description

Implements GET <base>/Binary/<id>, retrieve back the Binary resource in its native format.

14.1.12 Transformation and mapping

Name

FHIR Mapper (Transform)

Configuration

Vonk.Plugins.Transform.TransformConfiguration

License token

<http://fire.ly/vonk/plugins/mapping>

Order

4560

Description

Implements FHIR `$transform` : POST <base>/administration/StructureMap/[id]/`$transform`. See `fhir_mapper_docs:mappingengine_index`.

Name

FHIR Mapper (Convert)

Configuration

Vonk.Plugin.MappingToStructureMap.MappingToStructureMapConfiguration

License token

<http://fire.ly/vonk/plugins/mapping>

Order

4550

Description

Implements FHIR `$convert` : POST `<base>/$convert` to convert between FHIR Mapping Language and its StructureMap representation.

14.1.13 Repository implementations

Name

Memory Repository

Configuration

Vonk.Repository.MemoryConfiguration

license token

<http://fire.ly/vonk/plugins/repository/memory>

Order

210

Description

Implements a repository in working memory that fully supports all of the capabilities of Firely Server. This implementation is mainly used for unittesting.

Name

Memory Administration Repository

Configuration

Vonk.Repository.MemoryAdministrationConfiguration

license token

<http://fire.ly/vonk/plugins/repository/memory>

Order

211

Description

Implements a repository in working memory for the Administration API. This implementation is mainly used for unittesting.

Name

MongoDb Repository

Configuration

Vonk.Repository.MongoDbConfiguration

license token

<http://fire.ly/vonk/plugins/repository/mongo-db>

Order

230

Description

Implements a repository in MongoDB that fully supports all of the capabilities of Firely Server, except Transactions.

Name

MongoDb Administration Repository

Configuration

`Vonk.Repository.MemoryAdministrationConfiguration`

license token

<http://fire.ly/vonk/plugins/repository/mongo-db>

Order

231

Description

Implements a repository in MongoDB for the Administration API.

Name

SQLite Repository

Configuration

`Vonk.Repository.SqliteConfiguration`

license token

<http://fire.ly/vonk/plugins/repository/sqlite>

Order

240

Description

Implements a repository in SQLite that fully supports all of the capabilities of Firely Server.

Name

SQLite Administration Repository

Configuration

`Vonk.Repository.SqliteAdministrationConfiguration`

license token

<http://fire.ly/vonk/plugins/repository/sqlite>

Order

241

Description

Implements a repository in SQLite for the Administration API.

Name

SQL Server Repository

Configuration

`Vonk.Repository.SqlConfiguration`

license token

<http://fire.ly/vonk/plugins/repository/sql-server>

Order

220

Description

Implements a repository in SQL Server that fully supports all of the capabilities of Firely Server.

Name
SQL Server Administration Repository

Configuration
`Vonk.Repository.SqlAdministrationConfiguration`

license token
<http://fire.ly/vonk/plugins/repository/sql-server>

Order
221

Description
Implements a repository in SQL Server for the Administration API.

14.1.14 Administration API

Name
Administration API

Configuration
`Vonk.Administration.Api.AdministrationOperationConfiguration`

license token
<http://fire.ly/vonk/plugins/administration>

Order
1160

Description
Sets up a sequence of plugins for the Administration API. Administration API is different from general plugins since it branches off of the regular processing pipeline and sets up a second pipeline for the /administration endpoint.

Name
Fhir STU3 Administration services

Configuration
`Vonk.Administration.FhirR3.RepositoryConfigurationR3`

license token
<http://fire.ly/vonk/plugins/administration/fhrr3>

Order
4310

Description
Implements support services to work with FHIR STU3 conformance resources in the Administration API.

Name
Fhir R4 Administration services

Configuration
`Vonk.Administration.FhirR4.RepositoryConfigurationR4`

license token
<http://fire.ly/vonk/plugins/administration/fhrr4>

Order
4310

Description

Implements support services to work with FHIR R4 conformance resources in the Administration API.

14.1.15 Bulk Data**Name**

Bulk Data Export

Configuration

Vonk.Plugin.BulkDataExport

license token

<http://fire.ly/vonk/plugins/bulk-data-export>

Order

5005

Description

Request an export of bulk data sets. See *Bulk Data Export*.

Name

Patient everything

Configuration

Vonk.Plugin.PatientEverything

license token

<http://fire.ly/vonk/plugins/patient-everything>

Order

5006

Description

Request a Patient record. See *Patient \$everything*.

14.2 Important classes and interfaces

If you want to develop a plugin for Firely Server, there are a couple of classes that you will probably interact with. This page lists those classes, with an explanation of each.

14.2.1 IResource**namespace**

Vonk.Core.Common

purpose

IResource is the abstraction for all FHIR resources in Firely Server. It is used in the request and the response, and thereby all through the pipeline. It allows you to program against resources in different Fhir.NET API (the Resource class is defined in each version separately), as well as against resources that do not even have a POCO implementation.

```
/// <summary>
/// Abstraction of a resource in some format. Specifies the properties of a Resource
↳ that Firely Server needs to read and maintain.
```

(continues on next page)

(continued from previous page)

```

/// <para>Future: may be extended with Tags and Labels.</para>
/// </summary>
public interface IResource : ISourceNode
{
    /// <summary>
    /// Type of resource, e.g. Patient or AllergyIntolerance.
    /// </summary>
    string Type { get; }

    /// <summary>
    /// Logical identity of resource, e.g. 'example' or 'e3f5b0b8-4570-4e4c-b597-
    ↪ e6523aff3a19'. Does not contain the resourcetype.
    /// Refers to Resource.id
    /// IResource is immutable, so to update this, use resourceWithNewId = this.SetId(),
    ↪ from IResourceExtensions.
    /// In the context of a repository, consider IResourceChangeRepository.EnsureMeta().
    /// </summary>
    string Id { get; }

    /// <summary>
    /// Version of resource. Refers to Resource.meta.versionId.
    /// IResource is immutable, so to update this, use resourceWithNewVersion = this.
    ↪ SetVersion(), from IResourceExtensions.
    /// In the context of a repository, consider IResourceChangeRepository.EnsureMeta().
    /// </summary>
    string Version { get; }

    /// <summary>
    /// Model that the resource was defined in.
    /// Common models are the different versions of FHIR, defined in <see cref=
    ↪ "VonkConstants.Model"/>
    /// </summary>
    string InformationModel { get; }

    /// <summary>
    /// When was the resource last updated?
    /// Refers to Resource.meta.lastUpdated.
    /// IResource is immutable, so to update this, use resourceWithNewLastUpdated =
    ↪ this.SetLastUpdated(DateTimeOffset) from IResourceExtensions.
    /// In the context of a repository, consider IResourceChangeRepository.EnsureMeta().
    /// </summary>
    DateTimeOffset? LastUpdated { get; }

    /// <summary>
    /// Is this a contained resource, or a container resource?
    /// A resource is a container resource if it is not contained. Even if it has no
    ↪ contained resources embedded.
    /// </summary>
    ResourceContained Contained { get; }

    /// <summary>
    /// Direct access to contained resources, if any. Prefer to return an empty list.

```

(continues on next page)

(continued from previous page)

```

↪ otherwise.
    /// Refers to DomainResource.contained.
    /// </summary>
    IEnumerable<IResource> ContainedResources { get; }
}

```

If you work with a POCO, you can use the extension method `ToIResource()` from `Vonk.Fhir.R3` to adapt it to an `IResource`:

```

var patientPoco = new Patient(); //Requires Hl7.Fhir.Model
var resource = patientPoco.ToIResource();

```

`IResource` is immutable, so changes will always result in a new instance. Changes can usually be applied with extension methods on `ISourceNode`, found in `Vonk.Core.ElementModel.ISourceNodeExtensions`. There are also several extension methods specifically for `IResource` in `Vonk.Core.Common.IResourceExtensions`:

```

var updatedResource = oldResource.Add(SourceNode.Valued("someElement", "someValue");
//Continue with updatedResource, since oldResource will not have the change.

```

IResource extension methods

`IResource` has a whole list of extension methods for manipulating them, conversion between `ISourceNode` and `IResource` and caching objects within it. All these methods are in the namespace `Vonk.Core.Common.IResourceExtensions`. Please check the `///`-comments on the methods for more information.

14.2.2 ElementModel, manipulating IResource and ISourceNode

As stated in *IResource*, `IResource` is immutable, and so is `ISourceNode` that it is based on. Still, there are cases where you would want to manipulate a resource, e.g. add, change or remove elements. The namespace `Vonk.Core.ElementModel` has methods to do so. All of these methods do *NOT* change the original structure (the input `IResource` or `ISourceNode`), but instead *return* an updated structure.

ISourceNode manipulation

All the `ISourceNode` extension methods can be used on `IResource` as well. If you need an `IResource` as result, just turn the resulting `ISourceNode` to an `IResource` again. So if you added an element to an `ISourceNode`:

```

IResource result = input.AddIfNotExists(SourceNode.Valued("active", "false")).
↪ ToIResource(input.InformationModel);

```

For some of the more common extension methods we provide an overload on `IResource` that does this for you, like `IResource.Patch(...)`

All the methods below are in the namespace `Vonk.Core.ElementModel.ISourceNodeExtensions`:

Add(this *ISourceNode original*, *ISourceNode newChild*) → *ISourceNode*

Add the *newChild* as a child node to the *original*. It will be added at the end of the Children.

Add(this *ISourceNode original*, *ITypedElement newChild*) → *ISourceNode*

Overload of `Add(ISourceNode newChild)` that lets you add an `ITypedElement` as new child.

AddIf(*this ISourceNode original, ISourceNode newChild, Func<ISourceNode, bool> addIf*) → ISourceNode

Add the *newChild* as a child node to the *original* if the *addIf* predicate on *original* is met. It will be added at the end of the Children.

Add(*this ISourceNode original, TypedElement newChild, Func<ISourceNode, bool> addIf*) → ISourceNode

Similar to `AddIf(ISourceNode newChild, Func<ISourceNode, bool> addIf)` that lets you add an `ITypedElement` as new child.

AddIfNotExists(*this ISourceNode original, ISourceNode newChild*) → ISourceNode

Add the *newChild* as a child node to the *original* if there is no child with the same name yet. It will be added at the end of the Children.

AddIfNotExists(*this ISourceNode original, ISourceNode newChild, Func<ISourceNode, bool> exists*) → ISourceNode

Add the *newChild* as a child node to the *original* if the *exists* predicate on *original* is not satisfied. This is like `AddIfNotExist(ISourceNode newChild)`, but here you get to specify what 'exists' means. It will be added at the end of the Children.

AddIfNotExists(*this ISourceNode original, string location, ISourceNode newChild*) → ISourceNode

Navigate to *location*. Then add the *newChild* as a child node to the *original* if there is no child with the same name yet.

AddIfNotExists(*this ISourceNode original, ISourceNode newChild, string location, Func<ISourceNode, bool> exists*) → ISourceNode

Navigate to *location*. Then add the *newChild* as a child node if the *exists* predicate on the current node is not satisfied.

AnnotateWith<T>(*this ISourceNode original, T annotation, bool hideExisting = false*) → ISourceNode

Add an annotation of type *T* to the *original*. When *hideExisting* == true, any existing annotations of type *T* are not visible anymore on the returned *ISourceNode*.

GetBoundAnnotation<T>(*this ISourceNode original,) where T : class, IBoundAnnotation* → T

Retrieve an annotation that is bound directly to *original*, not to any of the nodes it may decorate. (*ISourceNode* is immutable, to changes are usually a pile of wrappers around the *original SourceNode*, and each of the wrappers can add / replace annotations.)

RemoveEmptyNodes(*this ISourceNode original, ISourceNode newChild*) → ISourceNode

Remove any nodes that have no value or children. This happens recursively: if a node has only children with empty values, it will be removed as well. This way the returned *ISourceNode* conforms to the invariant in the FHIR specification that an element either has a value or children.

RemoveEmptyNodes(*this ISourceNode original, ISourceNode newChild, string location*) → ISourceNode

Remove any nodes that have no value or children, from the specified *location* downwards. This happens recursively: if a node has only children with empty values, it will be removed as well.

Child(*this ISourceNode original, string name, int arrayIndex = 0*) → ISourceNode

Convenience method to get the child with name *name* at position *arrayIndex*. Usually used to get a child of which you know there is only one: `patientNode.Child("active")`

ChildString(*this ISourceNode original, string name, int arrayIndex = 0*) → ISourceNode

Convenience method to get the value of the child with name *name* at position *arrayIndex*. Usually used to get a child of which you know there is only one: `patientNode.ChildString("id")`

ForceAdd(*this ISourceNode original, string addAt, ISourceNode newChild*) → ISourceNode

Add the *newChild* at location *addAt*. Create the intermediate nodes if necessary.

AddOrReplace(*this ISourceNode original, Func<ISourceNode, bool> match, ISourceNode toAdd, Func<ISourceNode, ISourceNode> replace*) → ISourceNode

Find any child nodes of **original** that match the **match** predicate. Apply **replace** to them. If none are found, add **toAdd** as new child.

AddOrReplace(*this ISourceNode original, ISourceNode toAdd, Func<ISourceNode, ISourceNode> replace*) → ISourceNode

Optimized overload of the previous method for matching on the node name. It will perform **replace** on any child node of **original** with the same name as **toAdd**. If none are found it will add **toAdd** as new child node.

Remove(*this ISourceNode original, string location*) → ISourceNode

Remove the node at **location**, if any. If that results in parent nodes becoming empty (no Text, no Children), those are removed as well.

SelectNodes(*this ISourceNode original, string fhirPath*) → IEnumerable<ISourceNode>

Run **fhirPath** over the **original**, but with the limitations of untyped nodes. It will return the matching nodes. Use **valueDateTime/valueBoolean** instead of just 'value' for choice types. Only use this method if you are familiar with the differences in the naming of nodes between **ISourceNode** and **ITypedElement**.

SelectText(*this ISourceNode original, string fhirPath*) → string

Run **fhirPath** over the **original**, but with the limitations of untyped nodes. Returns the **Text** of the first matching node. Use **valueDateTime/valueBoolean** instead of just 'value' for choice types. Only use this method if you are familiar with the differences in the naming of nodes between **ISourceNode** and **ITypedElement**.

Patch(*this ISourceNode original, string location, Func<ISourceNode, ISourceNode> patch*) → ISourceNode

Find any nodes at **location** and apply **patch** to them. For **patch** you can use other methods listed here like **Rename**, **Add** or **Revalue**. **location** is evaluated as a fhirpath statement, with the limitations of untyped nodes.

Patch(*this ISourceNode original, string[] locations, Func<ISourceNode, ISourceNode> patch*) → ISourceNode

Find any nodes having one of the **locations** as their **Location** and apply **patch** to them. If you don't know exact locations, use **original.Patch(location, patch)**, see above.

ForcePatch(*this ISourceNode original, string forcePath, Func<ISourceNode, ISourceNode> patch*) → ISourceNode

Enforce that **forcePath** exists. Then patch the resulting node(s) with **patch**.

ForcePatchAt(*this ISourceNode original, string fromLocation, string forcePath, Func<ISourceNode, ISourceNode> patch*) → ISourceNode

For each node matching the **fromLocation**: enforce that **fromLocation.forcePath** exists, then patch the resulting node(s) with **patch**. E.g. **someBundle.ForcePatchAt("entry", "request", node => node.Add(SourceNode.Valued("url", "someUrl"))** will add **request.url** with value "someUrl" to every entry.

Relocate(*this ISourceNode original, string newLocation*) → ISourceNode

Set **original.Location** to the **newLocation**, and update all its descendants' **Location** properties recursively.

Rename(*this ISourceNode original, string newName*) → ISourceNode

Set **original.Name** to the **newName**.

Revalue(*this ISourceNode original, string newValue*) → ISourceNode

Set **original.Text** to **newValue**.

Revalue(*this ISourceNode original, Dictionary<string, string> replacements*) → ISourceNode

replacements is a dictionary of location + **newValue**. On each matching location under **original**, the value will be set to the according **newValue** from **replacements**.

AnnotateWithSourceNode(*this ISourceNode original*) → ISourceNode

Add **original** as annotation to itself. Very specific use case.

ITypedElement manipulation

All the methods below are in the namespace `Vonk.Core.ElementModel.ITypedElementExtensions`:

Add(*this ITypedElement original, ITypedElement newChild, Func<ITypedElement, bool> addIf*) → `ISourceNode`

Add `newChild` as child to `original` if `addIf` on `original` evaluates to true. Convenience overload of `ISourceNodeExtensions.Add(ISourceNode, ITypedElement, Func<ISourceNode, bool>)`

Add(*this ITypedElement original, ITypedElement newChild*) → `ISourceNode`

Add `newChild` as child to `original`. Convenience overload of `ISourceNodeExtensions.Add(ISourceNode, ITypedElement)`

AddIfNotExists(*this ITypedElement original, ITypedElement newChild*) → `ISourceNode`

Add `newChild` as child to `original` if no child with the same name exists yet. Convenience overload of `ISourceNodeExtensions.AddIfNotExists(ISourceNode, ITypedElement)`

AddIf(*this ITypedElement original, ISourceNode newChild, Func<ITypedElement, bool> addIf*) → `ISourceNode`

Add `newChild` as child to `original` if `addIf` on `original` evaluates to true. Convenience overload of `ISourceNodeExtensions.AddIf(ISourceNode, ISourceNode, Func<ISourceNode, bool>)`

method

`Add(this ITypedElement original, ISourceNode newChild)` Add `newChild` as child to `original`.

method

`AddIfNotExists(this ITypedElement original, ISourceNode newChild)` Add `newChild` as child to `original` if no child with the same name exists yet. Convenience overload of `AddIfNotExists(ITypedElement, ITypedElement)`

Cache(*this ITypedElement original*) → `ITypedElement`

Prevent recalculation of the Children upon every access.

Child(*this ITypedElement element, string name, int arrayIndex = 0*) → `ITypedElement`

Returns n-th child with the specified name, if any.

ChildString(*this ITypedElement element, string name, int arrayIndex = 0*) → `string`

Returns the value of the n-th child with the specified name as string, if any.

DefinitionSummary(*this ITypedElement element, IStructureDefinitionSummaryProvider provider*) → `IStructureDefinitionSummary`

Returns the summary for the actual type of the element. Especially useful if the element is of a choicetype.

AddParent(*this ITypedElement element*) → `ITypedElement`

Add `Vonk.Core.ElementModel.IParentProvider` annotations to `element` and its descendants.

GetParent(*this ITypedElement element*) → `ITypedElement`

Get the parent of this element, through the `Vonk.Core.ElementModel.IParentProvider` annotation (if present).

AddTreePath(*this ITypedElement element*) → `ITypedElement`

Add the `Vonk.Core.ElementModel.ITreePathGenerator` annotation. `TreePath` is the Location without any indexes (no [n] at the end).

GetTreePath(*this ITypedElement element*) → `string`

Get the value of the `Vonk.Core.ElementModel.ITreePathGenerator` annotation, if present. `TreePath` is the Location without any indexes (no [n] at the end).

14.2.3 IVonkContext

namespace

Vonk.Core.Context

purpose

IVonkContext is the Vonk-specific counterpart to HttpContext from ASP.NET Core. It contains an IVonkRequest and IVonkResponse object that allow you to get information from the request and set results in the response, both in Firely Server terms.

Have IVonkContext injected in the method where you need it. Use a *configuration class* to call this method from the pipeline and have the actual context injected. A more complete template is found at *Template for a plugin*.

```
public class SomeService
{
    public async Task DoMyOperation(IVonkContext vonkContext)
    {
        //...
    }
}

public static class SomeServiceConfiguration
{
    public static IApplicationBuilder UseMyOperation(this IApplicationBuilder app)
    {
        return app.UseVonkInteractionAsync<SomeService>((svc, context) => svc.
        DoMyOperation(context));
    }
}
```

If you also need access to the raw HttpContext, you have two options:

1. The IVonkContext.HttpContext() extension method gives you the original HttpContext. Be aware though:
 - Situations may arise where there is no HttpContext, so be prepared for that.
 - When handling batches or transactions, an IVonkContext is created for each entry in the bundle. But they all refer to the same original HttpContext.
2. Create a normal ASP.NET Core Middleware class and access the IVonkContext with the extension method Vonk() on HttpRequest. A more complete template is found at *Returning non-FHIR content from a plugin*.

```
public class SomeMiddleware
{
    public SomeMiddleware(RequestDelegate next)
    {
        //...
    }

    public async Task Invoke(HttpContext httpContext)
    {
        var vonkContext = httpContext.Vonk();
        //...
    }
}
```

(continues on next page)

(continued from previous page)

```

public static class SomeMiddlewareConfiguration
{
    public static IApplicationBuilder UseSomeMiddleware(this IApplicationBuilder app)
    {
        return app.UseMiddleware<SomeMiddleware>(); //Just plain ASP.NET Core,
        ↪ nothing Firely Server specific here.
    }
}

```

IVonkContext has three major parts, that are explained below. The InformationModel tells you the FHIR version for which the request was made.

```

public interface IVonkContext
{
    IVonkRequest Request {get;}

    IArgumentCollection Arguments {get;}

    IVonkResponse Response {get;}

    string InformationModel {get;}
}

```

And because you frequently need the parts instead of the context itself, there is an extension method on IVonkContext:

```

public (IVonkRequest request, IArgumentCollection args, IVonkResponse respons)
    ↪ Parts(this IVonkContext vonkContext)

```

IVonkRequest

namespace

Vonk.Core.Context

purpose

Get information about the request made, in Firely Server / FHIR terms.

You can access the current IVonkRequest through the *IVonkContext*. Its properties are:

```

public interface IVonkRequest
{
    string Path { get; }
    string Method { get; }
    string CustomOperation { get; }
    VonkInteraction Interaction { get; }
    RequestPayload Payload { get; set; }
}

```

Path and Method relate directly to the equivalents on HttpContext. Interaction tells you which of the FHIR RESTful interactions was called. CustomOperation is only filled if one of the custom operations was invoked, like e.g. \$validate. All of these can be filtered by the *InteractionHandlerAttribute*, so you typically don't need to inspect them manually.

Payload indirectly contains the resource that was sent in the body of the request. You are advised to only use the extension methods to access it:

```
public static bool TryGetPayload(this IVonkRequest request, out IResource resource)
```

TryGetPayload is useful if your code wants to act on the payload *if it is present*, but does not care if it is not.

```
public void ThisMethodActsOnThePayloadIfPresent(IVonkContext vonkContext)
{
    var (request, args, response) = vonkContext.Parts();
    if (request.TryGetPayload(response, out var resource))
    {
        // do something with the resource.
    }
}
```

```
public static bool GetRequiredPayload(this IVonkRequest request, IVonkResponse response,
    ↪out IResource resource)
```

GetRequiredPayload is useful if your code expects the payload to be present. It will set the appropriate response code and OperationOutcome on the provided response if it is not present or could not be parsed. Then you can choose to end the pipeline and thus return the error to the user.

```
public void ThisMethodNeedsAPayload(IVonkContext vonkContext)
{
    var (request, args, response) = vonkContext.Parts();
    if (!request.GetRequiredPayload(response, out var resource))
    {
        return; //If you return with an error code in the response, Firely Server will end
    ↪the pipeline
    }
    // do something with the resource.
}
```

If you want to **change** the payload, assign a whole new one. Generally you would want to change something to the old payload. But IResource is immutable, so changes to it yield a new instance. That leads to this pattern

```
if (request.TryGetPayload(response, out var resource)
{
    //Explicit typing of variables for clarity, normally you would use 'var'.
    ISourceNode updatedNode = resource.Add(SourceNode.Valued("someElement", "someValue");
    IResource updatedResource = updatedNode.ToIResource();
    request.Payload = updatedResource.ToPayload();
}
```

IArgumentCollection, IArgument

namespace

Vonk.Core.Context

purpose

Access arguments provided in the request.

The `IVonkContext.Arguments` property contains all the arguments from the request, from the various places:

1. The path segments: `/Patient/123/_history/v1` will translate to three arguments, `_type`, `_id` and `_version`.
2. The query parameters: `?name=Fred&active=true` will translate to two arguments, `name` and `active`.
3. The headers:
 1. `If-None-Exists = identifier=abc&active=true` will translate to two arguments, `identifier` and `active`.
 2. `If-Modified-Since`, `If-None-Match`, `If-Match`: will each translate to one argument

An individual argument will tell you its name (`ArgumentName`), raw value (`ArgumentValue`) and where it came from (`Source`).

Handling arguments

An argument by default has a `Status` of `Unhandled`.

If an argument is of interest to the operation you implement in your plugin, you can handle the argument. It is important to mark arguments handled if:

- you handled them
- or the handling is not relevant anymore because of some error you encountered

In both cases you simply set the `Status` to `Handled`.

If an argument is incorrect, you can set its status to `Error` and set the `Issue` to report to the client what the problem was. These issues will be accumulated in the response by Firely Server automatically.

Any argument that is not handled will automatically be reported as such in an `OperationOutcome`.

Useful extension methods:

```
IArgument.Handled()
IArgument.Warning(string message, Issue issue)
IArgument.Error(string message, Issue issue)
```

Firely Server has a lot of issues predefined in `Vonk.Core.Support.VonkIssues`.

IVonkResponse

namespace

Vonk.Core.Context

purpose

Inspect response values set by other middleware, or set it yourself.

```
public interface IVonkResponse
{
    Dictionary<VonkResultHeader, string> Headers { get; }
    int HttpResult { get; set; }
    OperationOutcome Outcome { get; }
    IResource Payload { get; set; }
}
```

If your operation provides a response, you should:

1. Set the response code `HttpResult`.
2. Provide a resource in the `Payload`, if applicable.
3. Add an issue if something is wrong.

If you just listen in on the pipeline, you can check the values of the response. Besides that, the *InteractionHandlerAttribute* allows you to filter on the `HttpStatus` of the response.

IFormatter

namespace

Vonk.Core.Context.Format

purpose

Serialize response resource in requested format to the body of the `HttpContext.Response`. Although this interface is public, you should never need it yourself, since the *VonkToHttp plugin* takes care of this for you.

14.2.4 Interaction Handling

In the configuration of a plugin you specify on which interaction(s) the plugin should act. That can be done with an attribute on the main method of the service in the plugin, or with a fluent interface on `IApplicationBuilder`.

InteractionHandlerAttribute

namespace

Vonk.Core.Pluggability

purpose

Add an `[InteractionHandler]` attribute to a method to specify when the method has to be called. You specify this by providing values that the `IVonkContext` should match.

Without any arguments, the method will be called for every possible interaction.

```
[InteractionHandler()]
public async Task DoMyOperation(IVonkContext vonkContext)
```

You can specify different filters, and combine them at will:

- Specific interaction(s): `[InteractionHandler(Interaction = VonkInteraction.type_create | VonkInteraction.instance_update)]`
- Specific FHIR version(s) of the request: `[InteractionHandler(InformationModel = VonkConstants.Model.FhirR4)]`

- Specific resource type(s): `[InteractionHandler(AcceptedTypes = new["Patient", "Observation"])]`
- Specific custom operation: `[InteractionHandler(Interaction = VonkInteraction.all_custom, CustomOperation = "myCustomOperation")]`. Note that the \$ that is used on the url is not included in the name of the custom operation here.
- Specific http method: `[InteractionHandler(Method = "POST")]`
- Specific statuscode(s) on the response: `[InteractionHandler(StatusCode = new[]{200, 201})]`

Now to configure your service to be a processor in the Firely Server pipeline, you use `UseVonkInteraction[Async]()`:

```
public static class MyOperationConfiguration
{
    public static IApplicationBuilder UseMyOperation(this IApplicationBuilder app)
    {
        return app.UseVonkInteractionAsync<MyService>((svc, ctx) => svc.
        ↪ DoMyOperation(ctx));
    }
}
```

InteractionHandler fluent interface

Because `InteractionHandler` is an attribute, you can only use constant values. If that is not what you want, you can use the fluent interface in the `configuration class` instead. The code below shows the same filters as above, although you typically would not use all of them together (e.g. the PUT excludes `type_create`).

```
public static class MyOperationConfiguration
{
    public static IApplicationBuilder UseMyOperation(this IApplicationBuilder app)
    {
        return app
            .OnInteraction(VonkInteraction.type_create | VonkInteraction.instance_update)
            .AndInformationModel(VonkConstants.Model.FhirR4)
            .AndResourceTypes(new[] { "Patient", "Observation" })
            .AndStatusCodes(new[] { 200, 201 })
            .AndMethod("PUT")
            .HandleAsyncWith<MyService>((svc, ctx) => svc.DoMyOperation(ctx));
    }
}
```

Other `Handle...` methods allow you to define a pre-handler (that checks or alters the request before the actual operation) or a post-handler (that checks or alters the response after the actual operation), either synchronously or asynchronously.

If you have a very specific filter that is not covered by these methods, you can specify it directly with a function on the `IVonkContext` that returns a boolean whether or not to call your operation.

```
app
    .On(ctx => MyVerySpecificFilter(ctx))
    .Handle...
```


Attention: The filter you specify is called for **every** request. So make sure you don't do any heavy calculations or I/O.

IApplicationBuilder extension methods

UseVonkInteraction`<TService>(this IApplicationBuilder app, Expression<Action<<TService, IVonkContext>> handler, OperationType operationType = OperationType.Handler) -> IApplicationBuilder`

Handle the request with the handler method when the request matches the `InteractionHandler` attribute on the handler method. The `OperationType` may also specify `PreHandler` or `PostHandler`. If you need to do anything lengthy (I/O, computation), use the `Async` variant of this method.

UseVonkInteractionAsync`<TService>(this IApplicationBuilder app, Expression<Func<TService, IVonkContext, T.Task>> handler, OperationType operationType = OperationType.Handler) -> IApplicationBuilder`

Handle the request with the asynchronous handler method when the request matches the `InteractionHandler` attribute on the handler method. The `OperationType` may also specify `PreHandler` or `PostHandler`.

OnInteraction`(this IApplicationBuilder app, VonkInteraction interaction) → VonkApplicationBuilder`

Used for fluent configuration of middleware. This is one of two methods to enter the `VonkApplicationBuilder`, see [VonkApplicationBuilder extension methods](#). It requires you to choose an interaction to act on. If you need your services to act on every interaction, choose `VonkInteraction.all`.

OnCustomInteraction`(this IApplicationBuilder app, VonkInteraction interaction, string custom) → VonkApplicationBuilder`

Used for fluent configuration of middleware. This is one of two methods to enter the `VonkApplicationBuilder`, see [VonkApplicationBuilder extension methods](#). It requires you to choose an interaction to act on. This should be one of the `VonkInteraction.all_custom` interactions. `custom` is the name of the custom interaction to act on, without the preceding '\$'.

VonkApplicationBuilder extension methods

`VonkApplicationBuilder` is used to fluently configure your middleware. It has methods to filter the requests that your middleware should respond to. Then it has a couple of `*Handle...` methods to transform your service into middleware for the pipeline, and return to the `IApplicationBuilder` interface.

AndInteraction`(this VonkApplicationBuilder app, VonkInteraction interaction) → VonkApplicationBuilder`

Specify an interaction to act on.

AndResourceTypes`(this VonkApplicationBuilder app, params string[] resourceTypes) → VonkApplicationBuilder`

Specify the resourcetypes to act on.

AndStatusCodes`(this VonkApplicationBuilder app, params int[] statusCodes) → VonkApplicationBuilder`

Specify the statuscode(s) of the response to act on. This is mainly useful for posthandlers.

AndMethod`(this VonkApplicationBuilder app, string method) → VonkApplicationBuilder`

Specify the http method (GET, PUT, etc) to act on.

AndInformationModel`(this VonkApplicationBuilder app, string model) → VonkApplicationBuilder`

If your service can only act on one FHIR version, specify it with this method. Common values for `model` are `VonkConstants.Model.FhirR3` and `VonkConstants.Model.FhirR4`.

```
PreHandleAsyncWith<TService>(this VonkApplicationBuilder app, Expression<Func<TService, IVonkContext, T.Task>> preHandler) -> IApplicationBuilder
```

Mark the preHandler method as a prehandler, so it will act on the IVonkContext and send it further down the pipeline.

```
PreHandleWith<TService>(this VonkApplicationBuilder app, Expression<Action<TService, IVonkContext>> preHandler) -> IApplicationBuilder
```

Synchronous version of PreHandleAsyncWith for synchronous preHandler methods.

```
HandleAsyncWith<TService>(this VonkApplicationBuilder app, Expression<Func<TService, IVonkContext, T.Task>> handler) -> IApplicationBuilder
```

Mark the handler method as a handler, so it will act on the IVonkContext, provide a response and end the pipeline for the request.

```
HandleWith<TService>(this VonkApplicationBuilder app, Expression<Action<TService, IVonkContext>> handler)
```

Synchronous version of HandleAsyncWith for synchronous handler methods.

```
PostHandleAsyncWith<TService>(this VonkApplicationBuilder app, Expression<Func<TService, IVonkContext, T.Task>> postHandler) -> IApplicationBuilder
```

Mark the postHandler method as a posthandler, so it will pass on the IVonkContext to the rest of the pipeline, and on the way back through the pipeline inspect or modify the response. Make sure that the VonkConfiguration order you have for this is lower than whatever action you need to post-handle.

```
PostHandleWith<TService>(this VonkApplicationBuilder app, Expression<Action<TService, IVonkContext>> postHandler) -> IApplicationBuilder
```

Synchronous version of PostHandleAsyncWith for synchronous postHandler methods.

14.2.5 Pipeline configuration

VonkConfigurationAttribute

namespace

Vonk.Core.Pluggability

purpose

This attribute is used on a static class to make Firely Server recognize it as part of the configuration of the processing pipeline. See [Configuration classes](#).

properties

- Order: Determines the place in the pipeline. See [The order of plugins](#) for background.

14.2.6 Repository interfaces

ISearchRepository

Vonk.Core.Repository.ISearchRepository is central in the functioning of Firely Server. It defines all read-access to the underlying repository, being one of Firely Server's own database implementations or a Facade implementation.

It has a single method:

```
Task<SearchResult> Search(IArgumentCollection arguments, SearchOptions options);
```

Using ISearchRepository

1. Have it injected by the dependency injection into the class where you need it:

```
public MyService(ISearchRepository searchRepository)
{
    _searchRepository = searchRepository;
}
```

Note: Implementations of ISearchRepository have a *Scoped* lifetime, so MyService should also be registered as Scoped:

```
public IServiceCollection AddMyServices(this IServiceCollection services)
{
    services.TryAddScoped<MyService>();
    return services;
}
```

2. Prepare search arguments that express what you are looking for. Search arguments are effectively key-value pairs as if they came from the querystring on the request. So the key must be the code of a supported Searchparameter.

```
var args = new ArgumentCollection
{
    new Argument(ArgumentSource.Internal, ArgumentNames.resourceType, "Patient") {
        ↳ MustHandle = true //optional },
    new Argument(ArgumentSource.Internal, "name", "Fred") { MustHandle = true //
        ↳ optional },
}
.AddCount(20);
```

Note: The Search implementation will in general update the arguments, especially their Status property and the Issue if something went wrong. So be careful with reuse of arguments. Use IArgumentCollection.Clone() if necessary.

3. Prepare search options that guide the search. Usually you can use one of the predefined options on the SearchOptions class.

```
var options = SearchOptions.Latest(vonkContext.ServerBase, vonkContext.Request.
    ↳ Interaction, vonkContext.InformationModel);
```

4. Execute the search.

```
var searchResult = await _searchRepository.Search(args, options);
```

5. Check the status of the arguments, especially if they could not be ignored (MustHandle = true). Because this is a common pattern, there is an extension method CheckHandled that throws a VonkParameterException if MustHandle arguments are not handled.

```
try
{
    args.CheckHandled("Arguments must all be handled in MyService");
```

(continues on next page)

(continued from previous page)

```

}
catch (VonkParameterException vpe)
{
    //report it in the vonkContext.Response.Outcome
}

```

6. Inspect the number of the results to check whether anything was found. If so, you can enumerate the results or process the set as a whole, since `SearchResult` implements `IEnumerable<IResource>`.

```

if (searchResult.TotalCount > 0)
{
    foreach(var resource in searchResult)
    { ... }
}

```

Implement `ISearchRepository`

Implementing `ISearchRepository` is only needed in a Facade.

The general pattern for implementing `ISearchRepository` is:

1. For each of the `IArguments` in the `IArgumentCollection`:
 1. If you support the argument, translate it to a ‘where’ clause on your repository. If your backend is another Web API, this could have the form of a querystring.
 2. Call `IArgument.Handled()` to update its status. There is also `.Warning()` and `.Error()` when something is wrong with the argument. If you simply don’t support the argument, you can leave the status to ‘Unhandled’.
 3. Pay special attention to the `_count` and `_skip` arguments for proper paging.
2. ‘AND’ all the arguments together, e.g. forming a database query or complete querystring.
3. Issue the query to your repository and await the results (await used intentionally: this should be done asynchronously).
4. For each of the resulting records or objects: map them to matching FHIR resources, either by:
 1. Creating POCO’s:

```

var result = new Patient() { /*fill in from the source object*/ };
return result.ToIResource(); //InformationModel implied by the assembly of
↳class Patient

```

2. or by crafting `SourceNodes`:

```

var result = SourceNode.Resource("Patient", SourceNode.Valued("id", /* id from
↳source */), SourceNode.Node("meta", SourceNode.Valued("versionId", "v1"), ....
↳), ...))
return result.ToIResource(VonkConstants.Model.FhirR3 /* or FhirR4 */);

```

5. Combine the mapped resources into a `SearchResult`:

```

return new SearchResult(resources, pagesize, totalCount, skip);

```

- `pagesize`: should be the value of the `_count` argument, unless you changed it for some reason.

- `totalCount`: total number of results, if there are more than you are returning right now.
- `skip`: number of results skipped in this set (if you are serving page x of y).

For a Facade on a relational database we provide a starting point with `Vonk.Facade.Relational.SearchRepository`. Follow the exercise in *Exercise: Build your first Facade* to see how that is done.

IResourceChangeRepository

`IResourceChangeRepository` defines methods to change resources in the repository:

```
public interface IResourceChangeRepository
{
    Task<IResource> Create(IResource input);
    Task<IResource> Update(ResourceKey original, IResource update);
    Task<IResource> Delete(ResourceKey toDelete, string informationModel);
    string NewId(string resourceType);
    string NewVersion(string resourceType, string resourceId);
}
```

`ResourceKey` is a simple struct to identify a resource by Type, Id and optionally VersionId.

Using IResourceChangeRepository

You should hardly ever need to use the `IResourceChangeRepository`. It is used by the *Create*, *Update*, *Delete* and the conditional variations thereof.

Should you need to use it, the methods are fairly straightforward.

method

Create

description

Provide an `IResource` having an id and versionId. These can be obtained by calling `NewId` and `NewVersion`. The return value will contain a possibly updated `IResource`, if the implementation changed or added elements. To fill in all the metadata there is a convenient extension method on `IResourceChangeRepository`:

```
var withMetaInfo = changeRepository.EnsureMeta(resource); //Will keep
↳existing id, version and lastUpdated and fill in if missing.
//EnsureMeta also exists as extension method on IResource - that uses
↳Guids for id and version.
var createdResource = await changeRepository.Create(withMetaInfo);
```

method

Update

description

Assert that a resource exists that can be updated. If not, use `Create`, otherwise go for `Update`.

```
var existingKey = new ResourceKey(resourceType, resourceId);
var args = existingKey.ToArguments(true);
var args = args.AddCount(0); //We don't need the actual result - just want
↳to know whether it is there.
var options = SearchOptions.Latest(vonkContext.ServerBase, VonkInteraction.
```

(continues on next page)

(continued from previous page)

```

↪ type_search, InformationModel: null); //search across informationmodels,
↪ we expect ids to be unique.
var exists = (await searchRepository.Search(args, options)).TotalCount = 1;
↪ //Take care of < 1 or > 1 matches

var withMetaInfo = changeRepository.EnsureMeta(resource, KeepExisting.Id);
↪ //Will keep existing id and provide fresh version and lastUpdated.
var updatedResource = await changeRepository.Update(existingKey,
↪ withMetaInfo);

```

method

Delete

description

Delete the resource that matches the provided key and informationModel. Returns the resource that was deleted.

```

var existingKey = new ResourceKey(resourceType, resourceId);
var deletedResource = await changeRepository(existingKey, vonkContext.
↪ InformationModel);

```

method

NewId

description

Get a new Id value generated by the repository (e.g. when the repository wants to use a sequence generator or ids in a specific format). Generally used through the extension method `IResourceChangeRepository.EnsureMeta(IResource resource, KeepExisting keepExisting)`, see `Create` above.

method

NewVersion

description

Get a new Version value generated by the repository (e.g. when the repository wants to use a sequence generator or ids in a specific format). The repository may want to base the version on the id, therefore the Id is passed as an argument. Generally used through the extension method `IResourceChangeRepository.EnsureMeta(IResource resource, KeepExisting keepExisting)`, see `Create` above.

Implement IResourceChangeRepository

Implementing `IResourceChangeRepository` is only needed in a Facade that wants to provide write-access to the underlying repository.

For all three methods, you will have to map data from FHIR resources to your internal data structures and back.

Note that you also need to implement `ISearchRepository` to support the *Create* and *Update* plugins and of course the conditional variants of those.

14.2.7 Constructing bundles

In a Plugin or Facade you may need to construct a bundle from a set of resources, e.g. a SearchResult (see [here](#)). There are two ways of doing this: with the Bundle POCO or with SourceNodes.

Bundle POCO

This is fairly straightforward. Create a new Bundle object, and fill its properties, iterating over the set of resources that you have. The code looks like this:

```
using Hl7.Fhir.Model; //Either from Hl7.Fhir.Core.Stu3 or Hl7.Fhir.Core.R4

...

var searchResult = await searchRepository.Search(args, options)
var bundle = new Bundle() { Type = Bundle.BundleType.Searchset }; // Type is required

foreach (var resource in searchResult)
{
    bundle.Entry.Add
    {
        new BundleEntryComponent
        {
            Resource = resource.ToPoco<Resource>();
            // fill in any other details like Request or Response.
        }
    }
}

//fill in details of the bundle as a whole, like Meta or Identifier.
```

The limitation of this is that you are bound to either STU3 or R4, and that also implies that you cannot include *Custom Resources* in the bundle.

Bundle from SourceNodes

ISourceNode is from Hl7.Fhir.ElementModel and not tied to a specific FHIR version, and Firely Server can serialize ISourceNode provided that the right StructureDefinition is available in the Administration API - which is the case for Bundle by default.

You start by creating the Bundle itself using the class SourceNode that allows for construction of ISourceNode nodes.

```
using Hl7.Fhir.ElementModel;

...

var bundleNode = SourceNode.Resource("Bundle", "Bundle", SourceNode.Valued("type",
↪ "document"));

//choose type as one of the bundle types from the spec, see http://hl7.org/fhir/R4/
↪ bundle-definitions.html#Bundle.type
```

Then you can add elements to the bundle itself that are not in the entries of the bundle. Like an identifier:

```
var identifier = SourceNode.Node("identifier");
identifier.Add(SourceNode.Valued("system", "urn:ietf:rfc:3986"));
identifier.Add(SourceNode.Valued("value", Guid.NewGuid().ToString()));
bundleNode.Add(identifier);
```

Then you can turn this into the helper class `GenericBundle` that provides several helper methods on an `ISourceNode` that is known to be a `Bundle`.

```
var documentBundle = GenericBundle.FromBundle(bundleNode);
documentBundle = documentBundle.Meta(Guid.NewGuid().ToString(), DateTimeOffset.Now);
```

Maybe you already saw an alternative way of adding the identifier in the intellisense by now:

```
documentBundle = documentBundle.Identifier("urn:ietf:rfc:3986", Guid.NewGuid().
↳ ToString());
```

Note that you always have to continue with the *result* of the modifying function. All these functions act on `ISourceNode` and that is immutable, so you get a new instance with the changes applied as a return value.

Now you have the skeleton of the `Bundle`, it is ready to add entries with resources to it.

```
IResource resourceForDocument = ... ; //Get or construct a resource that is one of the
↳ entries of the Bundle.
documentBundle = documentBundle.AddEntry(resourceForDocument, resourceForDocument.Key().
↳ ToRelativeUri());
```

Other extensions methods available on `GenericBundle`:

```
public static GenericBundle Total(this GenericBundle bundle, int total)
public static GenericBundle AddLink(this GenericBundle bundle, string relation, string
↳ uri)
public static GenericBundle Links(this GenericBundle bundle, Dictionary<string, string>
↳ links)
```

Search result bundles

Usually you don't need to construct a searchset bundle yourself, since the `SearchService` takes care of that when a search is issued on the FHIR endpoint. But should you want to do it in a custom operation, then the methods for doing so are at your disposal.

To help construct a bundle of type 'searchset', there is a special kind of bundle class `SearchBundle`. Create the sourcenode for the bundle as above. Then instead of creating a `GenericBundle`, turn it into a `SearchBundle`:

```
var searchBundle = bundleNode.ToSearchBundle();
```

Now you can use various methods to add entries for matches, includes or an `OperationOutcome`:

```
//SearchBundle methods
public SearchBundle AddMatch(ISourceNode resource, string fullUrl, string score = null)
public SearchBundle AddInclude(ISourceNode resource, string fullUrl, string score = null)
public SearchBundle AddOutcome(ISourceNode outcome, string fullUrl, string score = null)

//Extension methods
```

(continues on next page)

(continued from previous page)

```

public static SearchBundle ToSearchBundle(this IEnumerable<SearchInfo> searchInfos,
    ↪ string informationModel)
public static SearchBundle ToSearchBundle(this IEnumerable<ISourceNode> resources,
    ↪ string searchMode, string informationModel)
public static SearchBundle ToSearchBundle(this IEnumerable<ITypedElement> resources,
    ↪ string searchMode, string informationModel)

```

The SearchInfo struct essentially captures all the information that goes into an entry of a searchset bundle:

```

public struct SearchInfo
{
    public SearchInfo(ISourceNode resource, string mode = SearchMode.match, string
    ↪ fullUrl = null, string score = null)

    public string Mode { get; }
    public ISourceNode Resource { get; }
    public string FullUrl { get; }
    public string Score { get; }
}

```

Using all this to turn the SearchResult returned from the ISearchRepository.Search() method into a bundle looks like this (using the second extension method above):

```

var bundle = searchResult
    .ToSearchBundle(SearchMode.match, vonkContext.InformationModel)
    //informationModel is needed because bundle has slight differences between STU3
    ↪ and R4
    .Total(searchResult.Page.TotalCount)
    //Total is defined on GenericBundle
    .Links(searchResult.Page.PagingLinks(vonkContext));
    //Links is defined on GenericBundle
return bundle.ToIResource(vonkContext.InformationModel).EnsureMeta();

```

14.2.8 Access Control in Plugins and Facades

The *Access Control feature* is also available to users of a Firely Server Facade or Firely Server Plugins. You can use the default implementation based on SMART on FHIR, or provide an implementation of your own.

Access control implementation

The access control engine is programmed using interfaces for which you can provide your own implementation. Because we think the model behind SMART on FHIR covers many cases, these interfaces are loosely modelled after it. The important interfaces and class are:

Table 1: Access Control interfaces

Interface / Class	Description
IAuthorization	Defines whether your are allowed to read or write a type of resource. Abstraction of the concept of a scope (like user/Observation.read) in SMART
ICompartment	Confines the user to a compartment, expressed as a combination of a CompartmentDefinition and a search argument. Abstraction of the concept of a launch context (like patient=123) in SMART
IReadAuthorizer	Calculates access control for a type of resource given an instance of IAuthorization and/or ICompartment
IWriteAuthorizer	Calculates access control for writing a new (version of a) resource given an instance of IAuthorization and/or ICompartment
AuthorizationResult	Return value of IReadAuthorizer and IWriteAuthorizer methods. It expresses whether you are authorized at all, and if so - under which conditions. These conditions are expressed as search arguments.

IReadAuthorizer

Provides two methods to check authorization for reading types of resources.

- AuthorizeRead
- AuthorizeReadAnyType

The latter is only used if a system wide search is performed, without a `_type` parameter. In that case it is not efficient to call the first method for every supported resourcetype.

The input of these operations is an IAuthorization and an ICompartment. The result is an AuthorizationResult. With this class you can return:

- simply true or false
- extra search arguments to add to the search query in order to confine the search to those resources the user is allowed to read.

The AuthorizationResult Filters member is a collection of IArgumentCollections. Arguments within a collection will be AND'ed together. Multiple collections will be OR'ed together.

IWriteAuthorizer

Provides one method to assess whether the user is allowed to write a resource. Input is again IAuthorization and ICompartment, but also IResource - the resource that is to be written - and an Uri called 'serverBase'. The 'serverBase' parameter is primarily provided because it is required to perform a search on the ISearchRepository interface. The IAuthorization instance can be used to decide whether the user is allowed to write resources of the given resourcetype at all. The ICompartment can be used to search in the database whether the to-be-written resource is linked to the current compartment.

14.3 Architecture

14.3.1 Pipeline and middleware components

Firely Server is built upon ASP.NET Core and its modular setup of a pipeline of [middleware](#). It also leverages the [dependency injection](#) pattern that is central to ASP.NET Core. If you are not familiar with these concepts, please review them first.

Firely Server's core plugins provide middleware components that handles the interactions of the [FHIR RESTful API](#). These components are placed in the pipeline. The pipeline starts with a few general components that interpret the incoming http request and determine which interaction is being called. This information is fed to the rest of the pipeline as the Request property of an `IVonkContext` object, analogous to the `HttpContext` and its Request property of ASP.NET Core itself. The `IVonkContext` also provides the Response property that is used to communicate the status, resource and/or `OperationOutcome` resulting from the interaction. On the way back through the pipeline the counterparts of the interpreting middleware translate the `IVonkContext` Response back to an http response, conforming to the FHIR specification. The rest of the pipeline consists of middleware components that each fulfill one of the interactions. So there is one for read, one for create, for search et cetera. These components are just regular ASP.Net Core Middleware components, except that they have access to the `IVonkContext`, and act on that.

Adding custom middleware

Using Firely Server Components you can add your own middleware anywhere in the pipeline. It can be standard ASP.NET Core middleware - having nothing to do with FHIR - or middleware acting on the `IVonkContext`, e.g. to implement a custom operation. Firely Server also provides convenience methods to register your middleware as one that handles a FHIR interaction, including attributes to declare for which interaction and which resource types your middleware should be invoked. This is explained in [Firely Server Plugins](#).

14.3.2 Repository interfaces

Many of the FHIR interactions require access to a repository of resources. E.g. `create` must be able to store a resource, whereas `search` must be able to query resources and retrieve the results. In Firely Server, the middleware components that implement these interactions access the repository through interfaces. There are two different interfaces for different parts of the [FHIR RESTful API](#).

<code>IChangeRepository</code>	<i>//for create, update and delete</i>
<code>ISearchRepository</code>	<i>//for all types of search, read and history</i>

In many scenarios, read-only access is sufficient, and you only need to implement the `ISearchRepository`. In that implementation you can choose which of the search parameters you want to support, and whether to expose versions and deleted resources.

These interfaces enable you to implement a Firely Server Facade. And they enable us to support database engines as diverse as MongoDB, SQL Server and in-memory.

Search

The [FHIR RESTful Search](#) is the most complex part of the [FHIR RESTful API](#). Firely Server is capable of interpreting the search and translating it to small query-bits irrespective of the actual repository implementation. When implementing the `ISearchRepository` you get full control over which parameters you support and how you support them. On the method `ISearchRepository.Search()`, you just get the list of arguments that make up the search, as an `IArgumentCollection`. If you decide to act on these raw arguments directly, you can. But if you want Firely Server to interpret the search, you can use the `QueryBuilderContext.CreateQuery` method that will decompose the `IArgumentCollection`, interpret and validate every argument in it and then call into the `IRepoQueryFactory.Filter` or `IRepoQueryFactory.ResultShape` method for each of the arguments.

So the `Filter` method is called with interpreted search arguments. E.g. `identifier=abc` will be provided as (in pseudocode) `Filter("identifier", TokenValue{code = "abc", noSystem = true})`. If no search parameter is defined for an argument, you still get the change to handle it. E.g. `myspecialparameter=qyz` will be provided as `Filter("myspecialparameter", RawValue{ value = "qyz" })`. This allows for easy extensibility without defining your own `SearchParameter` resources, and is suitable for adding parameters that map to non-FHIR structures in your backend system. Note however that Firely Server also supports [Custom Search Parameters](#).

The `ResultShape` method is called when an argument is recognized as a ‘[Search result parameter](#)’, like `_sort` or `_include`.

14.3.3 Capabilities

A FHIR server has to express its capabilities in a `CapabilityStatement`, available on the `/metadata` endpoint. Firely Server’s capabilities are defined by the middleware components that make up its pipeline. Every component knows what interaction it adds to the capabilities. Therefore, we keep that information close to the component itself. Typically, every component has an implementation of `ICapabilityStatementContributor`, in which it gets access to the `ICapabilityStatementBuilder`. The latter provides methods to add information to the `CapabilityStatement` without having to worry about what is already added by other components or the order of execution.

14.4 Dependencies of Firely Server and their licenses

Firely Server is mainly built using libraries from Microsoft .Net Core and ASP.NET Core, along with a limited list of other libraries. This is the full list of direct dependencies that Firely Server has on other libraries, along with their licenses.

This list uses the NuGet package names (or prefixes of them) so you can easily lookup further details of those packages on [NuGet.org](#) if needed.

1. Microsoft.AspNetCore.* - Apache 2.0
2. Microsoft.ApplicationInsights.* - MIT
3. Microsoft.Bcl.AsyncInterfaces - MIT
4. Microsoft.EntityFrameworkCore - Apache 2.0
5. Microsoft.Extensions.* - MIT
6. Microsoft.AspNetCore.WebApi.Client - [MS-.NET-Library License](#)
7. System.Interactive.Async - MIT
8. Microsoft.Data.SqlClient - MIT
9. Microsoft.CSharp - MIT
10. System.Interactive.Async - MIT

11. System.Text.Json - MIT
12. System.* - [MS-.NET-Library License](#)
13. Newtonsoft.Json - MIT
14. IdentityServer4.AccessTokenValidation - Apache 2.0
15. IPNetwork2 - BSD 2-Clause “Simplified” License
16. Quartz - Apache 2.0
17. Serilog(*) - Apache-2.0
18. LinqKit.Microsoft.EntityFrameworkCore - MIT
19. Hl7.Fhir.* - Firely OSS license (see below)
20. Fhir.Metrics - as Hl7.Fhir
21. Simplifier.Licensing - as Hl7.Fhir
22. Dapper - Apache 2.0
23. SqlKata.* - MIT

MongoDB / CosmosDB:

1. MongoDB.Driver - Apache 2.0

For unittesting:

1. XUnit - Apache 2.0
2. Moq - BSD 3
3. FluentAssertions - Apache 2.0
4. Microsoft.NET.Test.Sdk - [MS-.NET-Library License](#)
5. System.Reactive - MIT
6. coverlet.collector - MIT
7. WireMock.Net - Apache 2.0

14.4.1 Firely OSS License

Firely Server relies on the reference .NET FHIR library: Hl7.Fhir.*, also created and maintained by Firely. The license is this (as stated in the [LICENSE file](#)):

Copyright (c) 2013-2020, HL7, Firely (info@fire.ly), Microsoft Open Technologies and contributors. See the file [CONTRIBUTORS](#) for details

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Firely nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CONTACT US

For questions, feedback, or consultancy for Firely Server, please send an e-mail to server@fire.ly. We look forward to hearing from you!

A

Add()
 built-in function, 249, 250, 252
 AddIf()
 built-in function, 249, 252
 AddIfNotExists()
 built-in function, 250, 252
 AddOrReplace()
 built-in function, 250, 251
 AddParent()
 built-in function, 252
 AddTreePath()
 built-in function, 252
 AndInformationModel()
 built-in function, 259
 AndInteraction()
 built-in function, 259
 AndMethod()
 built-in function, 259
 AndResourceTypes()
 built-in function, 259
 AndStatusCodes()
 built-in function, 259
 AnnotateWithSourceNode()
 built-in function, 251

B

built-in function
 Add(), 249, 250, 252
 AddIf(), 249, 252
 AddIfNotExists(), 250, 252
 AddOrReplace(), 250, 251
 AddParent(), 252
 AddTreePath(), 252
 AndInformationModel(), 259
 AndInteraction(), 259
 AndMethod(), 259
 AndResourceTypes(), 259
 AndStatusCodes(), 259
 AnnotateWithSourceNode(), 251
 Cache(), 252
 Child(), 250, 252

ChildString(), 250, 252
 DefinitionSummary(), 252
 ForceAdd(), 250
 ForcePatch(), 251
 ForcePatchAt(), 251
 GetParent(), 252
 GetTreePath(), 252
 OnCustomInteraction(), 259
 OnInteraction(), 259
 Patch(), 251
 Relocate(), 251
 Remove(), 251
 RemoveEmptyNodes(), 250
 Rename(), 251
 Revalue(), 251
 SelectNodes(), 251
 SelectText(), 251

C

Cache()
 built-in function, 252
 Child()
 built-in function, 250, 252
 ChildString()
 built-in function, 250, 252

D

DefinitionSummary()
 built-in function, 252

F

ForceAdd()
 built-in function, 250
 ForcePatch()
 built-in function, 251
 ForcePatchAt()
 built-in function, 251

G

GetParent()
 built-in function, 252
 GetTreePath()

built-in function, [252](#)

O

OnCustomInteraction()

built-in function, [259](#)

OnInteraction()

built-in function, [259](#)

P

Patch()

built-in function, [251](#)

R

Relocate()

built-in function, [251](#)

Remove()

built-in function, [251](#)

RemoveEmptyNodes()

built-in function, [250](#)

Rename()

built-in function, [251](#)

Revalue()

built-in function, [251](#)

S

SelectNodes()

built-in function, [251](#)

SelectText()

built-in function, [251](#)