On this site you will find the documentation for various FHIR tools and APIs.

The currently documented projects are:
CHAPTER 1

Welcome to Simplifier's documentation!

Contents:

1.1 Introduction to Simplifier.net

Simplifier.net is a FHIR registry. Within this registry you can create, upload, download, find and view FHIR Conformance Resources. Simplifier.net offers functionality for management of FHIR Resources and collaboration in teams.

FHIR (Fast Health Interoperability Resources) is an HL7 standard for information exchange in health care. It is based on Internet technology. More information about FHIR can be found at HL7 FHIR.

FHIR Profiling

The FHIR Profiling tool Forge (https://fire.ly/forge) can be used off-line and independently from Simplifier.net.

Simplifier accounts

Simplifier.net distinguishes between free (personal) plans and paid plans. Anyone can find and download Resources in this Registry, with or without an account. Creating Resources requires an account. Paid plans are targeted at professional users and organizations.

Simplifier content

Simplifier content is organized at four levels: Users, Resources, Projects and Organizations.

Users have access to a personal environment as well as free or paid features depending on their account level.

Simplifier is a repository for FHIR resources. There are a multitude of resources that are available to the public including profiles, extensions, valuesets, dictionaries, mappings, examples and more.

All FHIR resources are organized in projects, which can be either public or private. Each user can add one public project for free (paid accounts allow for adding more or private projects), making him/her the owner of the project. Depending on the account level more projects can be added. The features available in the project depend on the project settings as well as the account level.

Organizations only exist for the highest account plan. One or more projects and users can be added to an organization.
Firely

Simplifier.net is created and managed by Firely, formerly known as Furore Health Informatics. We are a group of software engineers, support engineers and FHIR consultants, based in Amsterdam, the Netherlands. We have been involved in HL7 FHIR since day one and have been contributing to the standard ever since. For more information about our company, please visit fire.ly.

1.2 Browse Simplifier

Simplifier content is organized at four levels: Users, Resources, Projects and Organizations. Simplifier allows you to get in touch with other users as well as to browse organizations, projects and resources from the homepage.

1.2.1 Find other Simplifier users

When you visit the Members tab on a project page, you see which Simplifier users are involved in the project. By clicking on a project member, you will navigate to his or her user profile. Here you can find more information about this user, such as his or her activity on Simplifier and other projects this user is involved in. You can also get in touch with this user by using his or her contact details or sending a direct message.

1.2.2 Search existing content

To explore existing resources, projects and organizations simply type a search phrase into the Search field at the top of the Simplifier homepage. This search allows you to browse through publicly available organizations, resources and projects. Once you reach the page with your search results you will notice that you have a few options to further specify your results. In the left hand menu you will find options to limit the results to Organizations, Projects or Resources.

Your search results are RESTful, meaning that you can send your search results exactly as you have edited them to another using the URL. The person that you are sending the results to will receive the search results just as you have filtered them.

1.2.3 Search for FHIR resources

Simplifier is a repository for FHIR resources. There are a multitude of resources that are available to the public including profiles, extensions, valuesets, dictionaries, mappings, examples and more. If these resources are listed as public then you can find them here.

You can browse these resources easily from the Simplifier homepage or from within a specific project. To search for a specific resource navigate to the search box and type the term in the search field. When searching from the homepage, select Resources from the Search Type options, to limit your search results to resources. When searching from the Resources tab in a project, this option will be selected automatically. You can apply additional filters such as Resource Categories and Fhir Status to make your search more specific.
Once you have found the resource you are looking for select it to see further details. Within the resource page you can view which project that resource is a part of, the type, status (maturity level), versioning, and different resource views.

### 1.2.4 Search for projects

All content in Simplifier (e.g. resources and Implementation Guides) is organized in projects. Projects may be linked to an organization (depending on the account level) or stand alone. To search for projects, select the option Projects under Search Type. You can use additional filters to search for more specific projects. For example you can filter based on Project Scope. This allows you to make a distinction between Core, International, National, Local, and Regional projects. If you select the National Project Scope, the Nationality filter option will open. Here you can select one or more nationalities.

Once you have found the project you are interested in, you can select it to navigate to the Project page where all content of the project is organized. Here you will find a summary of the project, a list of the project’s resources, Implementation Guides, project members and log information.

### 1.2.5 Search for organizations

The ability to search by organization allows you to easily find an organization’s work and profile. Click Organizations under Search Type and type the name of the company you are interested in. Once you have found the organization you are interested in, you can select it to navigate to the Organization page. Here you will find additional information about the organization as well as a list of projects that are linked to this organization.

### 1.3 Users

Click on your avatar in the top right corner to access your personal menu. From here you can access your personal portal, invites page, user profile, snippets page, organization portal(s) and account settings.

#### 1.3.1 Personal Portal

All your personal content is accessible through your personal portal. The following tabs are available:
• **Projects**: On this tab you can create new projects and/or manage your existing projects.

• **Bookmarks**: Use this tab to quickly access your favourite projects and publications. You can bookmark projects or resources by using the *Bookmark* button at the top right corner of the Project or Resource page.

• **Connectors**: Here you can create and edit your *connectors* or shop for new connectors.

• **Organizations**: This tab shows an overview of the organizations you are part of. You can access your organization portals from here or create a new organization if you have the rights to do so.

• **Implementation guides**: In older versions of Simplifier, this was the place to create and edit implementation guides, but they are moved to projects. Old implementation guides that are not yet linked to a project can still be accessed here. If you want to have your guide moved to a project, please contact us by using the feedback button on the top right.

### 1.3.2 Invites Page

On the invites page you see an overview of the projects that you were invited to. You can either accept or decline the invite.

1.3.3 User Profile

This page shows your user profile as visible to other Simplifier users. It shows your personal details, user activity and a list of projects that you are member of. Other users that visit your profile may contact you by clicking the *Contact User* button on the right. This message will be sent to the email address that you provided during registration.
You can now add your usual social media handles, like Facebook, Twitter, LinkedIn etc. to your user profile. We provide this for users that want to have the option of having other users contact them about their work on Simplifier. You can add these in your account settings, and you can view this directly from your own profile page. Your profile page can be found in the User Menu of your Portal dropdown.

1.3.4 Snippets

Your Snippets page shows a list of all your personal Snippets. From here you can also make a new Snippet by clicking on the +New button on the right.
1.3.5 Account Settings

Here you can edit your account settings. You can also visit your User Profile from this page by clicking on Profile in the upper right corner.

- **Avatar**: Upload an image and save it as your avatar. Your avatar will be visible in your user profile.
- **Personal details**: Edit your personal details here: email, display name and a description, which will appear in your user profile.
- **Mail updates**: Choose your settings for mail updates. The digest mail is a mail that informs you of bookmarked projects and resources that have changed recently. You can choose one of the following frequencies: daily, weekly, monthly or never.
- **Change password**: This is where you can change your Simplifier password.
- **Social media**: Add social media like Twitter or LinkedIn. Your social media will appear in your user profile, so other Simplifier users can find you.
- **Licenses**: Lists your Simplifier licenses.
- **Features**: Shows which Simplifier features are active based on your account and license.

![Account Settings](image)

1.3.6 Reaching out to other users

Would you like to collaborate with other users or reach out to users to ask question about their profiles? You can contact a user directly by having Simplifier send a mail to that user on your behalf. Just visit the Simplifier member’s user page and click Contact User by Email. We will send your contact details along with a message so that the other user can respond to you directly by mail if they want to.

1.4 Resources

Simplifier is a repository for FHIR resources. There are a multitude of resources that are available to the public including profiles, extensions, valuesets, dictionaries, mappings, examples and more.
1.4.1 Resource page

You can visit the page of a resource by selecting the resource from your search results or the Resources tab in your project, or following the direct link to the resource. While viewing resources you can display information in a few different ways.

Depending on the type of resource, the different views include:

- **Overview** – This is either a preview (e.g. texts) or a Logical view (e.g. profiles) of the resource. The Logical view of a profile includes Element names in the leftmost column followed by Flags, Cardinality, Type, and Description & Constraints.

- **Details** – This is an easy-to-read list per element of all the details of a profile. The specification refers to this as the dictionary.

- **Mappings** - This is a list of all the mappings specified in a profile.

- **Table** – This is a simple table view of the resource.

- **XML & JSON** – Respective views of resources in either XML or JSON formatting.

- **History** – On this tab you can view the difference between two versions of the same profile. This is a great feature for comparing and tracking changes.

- **Issues** - On this tab users with a paid account can track issues. New issues can be created by clicking the New issue button. The issue list can be filtered on open, closed or your own issues. By clicking on an issue you can read the entire conversation and add a new comment.

1.4.2 Update Resources

When you want to update your resource, there are several ways to do so. Choose one of the following options from the Update menu at the top of the Resource page:

- **Upload**: Update by uploading a file (either XML or JSON)

- **Fetch**: Update by fetching from a different FHIR server (provide a GET request to the server where your resource is located)

- **Edit**: Update by editing the last version (opens a XML-editor in a small window where you can directly edit the XML code of your resource)

- **Editor**: Update by editing the last version (opens a stand-alone full screen XML-editor in a different tab where you can directly edit the XML code of your resource)

1.4.3 Download Resources

You may also choose to download the resource and save a local copy on your computer. You can either choose to download the resource as a XML or JSON file or directly copy the XML or JSON code of the resource to your clipboard, so you can easily copy-paste it to another location.

1.4.4 Add Resources

Go to your Project page to add new resources to your project.
1.4.5 Sharing resources with Snippet

The Snippet feature in Simplifier enables you to quickly share resources with another Simplifier user (e.g. for review purposes) without storing them in your project. Click on the Snippet button in the top right corner to use this feature.

Start by giving your Snippet a title, e.g. MyPatient. You can either upload a file or copy-paste your XML code in the editor. Select Add another file to upload more than one resource within the same Snippet. When you are finished uploading and editing your resources, click on the Create button on the right.

Your Snippet will now be available on Simplifier. The URL is displayed at the top right of the Snippet. You can quickly copy the URL to your clipboard by clicking on the Copy icon at the right.

When visiting the URL of a Snippet, you can choose to view either the XML code, JSON code or the rendered resource by selecting one of the available tabs. In the top right menu you can either select Edit to edit the resource, Clone to copy it in a new Snippet, New to create a new Snippet or Download to download the resource as a XML or JSON file. You can also validate the resource by using the green Validate button at the right.
**Personal Snippets**

Through your personal menu (click on your avatar at the top right and select Snippets) you can access a list of all your Snippets. From here you can also quickly add a new Snippet by clicking the green +New button.

---

**1.4.6 Metadata Expressions**

The metadata expressions editor enables you to define what Simplifier should display as Title, Description, UrlKey, Workflow or FilePath for your resources. To start editing the metadata expressions of your project, select Settings and click on Metadata Expressions.

When editing the metadata expressions you will have access to the default expressions used by Simplifier. If your FhirPath expressions are missing, are not correct or the value extracted is empty, Simplifier will fall back to the default expressions. If a default expression is not able to provide a value, a generic text based on the resource type will be used. Workflow and FilePath don’t have a default expression. For FilePath in this case, if the extracted value is empty or missing, the fallback will be the original/generated filepath of the uploaded file.

As a convention, you must specify the resource type followed by the property (Title, Description, UrlKey, Workflow or FilePath) with a semicolon and the FhirPath expression based on which we extract the value.

**Tip:** For more information on how to use FhirPath, visit the following link to the FhirPath specification: [http://hl7.org/fhirpath/](http://hl7.org/fhirpath/)

---

**Title and Description**

Using Title and Description in the expressions, you can influence how they are displayed in search results and on the resource page.

**Example:**

```
Patient.Title: identifier
Patient.Description: name.family
```

Note that in this example, identifier and name.family are both collections that could contain more than one item. By default Simplifier only takes the first item of a collection. We built a custom FhirPath function to concatenate multiple items called `glue()`. The items are separated by the argument that is passed to this function.

**Example:**

```
Patient.Description: name.family.glue(', ')
```
The description of the Patient resource will now show all family names of the patient, separated by a comma. It is also possible to take only the first name of the patient by adding [0] to the element. Another possibility is to merge collections from different elements by using the | character. So, to show the first given name and the first family name separated by a space, you would use:

```
Patient.Description: (name[0].given[0] | name[0].family).glue(' ')
```

Note that the | character usually functions as an OR operator, as Simplifier will only select the first item in a collection. For example, say you want to show the name of an organization, but if name is empty you want to show the (first) identifier. You could use the following expression to do so:

```
Organization.Title: name | identifier[0].value
```

In some cases, you may want to select the first item of a collection yourself. For example, when you want to add additional text as well. In this situation, you could also use the first() function that is available in FhirPath.

Example:

```
PractitionerRole.Title: 'Role(s) of ' + (practitioner.display | practitioner.reference).first()
```

This example will show the value in the display element in the Reference to practitioner, if available. If not available, it will show the value in the reference element. If none of these values are available, Simplifier will fall back to the default value. Example of a PractitionerRole that is defined in the core code.

There is a difference between using the + operator and the & operator to concatenate values. When the + operator is used and one of the values is empty, the complete result will be empty. When the & operator is used, it doesn’t matter if any of the values are empty. The remaining values will still be concatenated. So, if we would have used the & operator instead of the + operator in the example above, and both practitioner.display and practitioner.reference are empty, Simplifier will show the Title of the PractitionerRole resource as follows: Role(s) of, which is not what we wanted.

**FilePath**

Special attention must be paid to the FilePath property. Since the filepath of the file is used for matching files in Simplifier, uniqueness is necessary. The extracted value for FilePath using the metadata expression must be unique within the project. Otherwise, a default fallback will be used or a new filepath will be generated.

**Warning:** In case the project is linked to a Github repository and there is a FhirPath expression specified for FilePath in the metadata expressions screen, the resulting value must match the filepath of the file in Github. If the filepath doesn’t match, the link is broken and the file will not be synced anymore.

**Workflow**

The Workflow property is populated using a FhirPath expression for extracting a value from the extension of the resource. The extracted workflow key will be used to identify the corresponding workflow status from the custom workflow selected within the project in Simplifier.

Example:

```
```
Extras

When a property (Title, Description, UrlKey, Workflow or FilePath) should use the same FhirPath expression for all resource types, the generic Resource can be used.

Example:

```
```

The editor supports comments as well. Comments can be entered by using the # or // characters.

Example:

```
#Resource examples
Observation.Title: code.text
//Profiles
Profile.Description: <expression for your description>
StructureDefinition.FilePath: <expression for filepath>
```

1.5 Projects

Simplifier organizes all content (e.g. resources and Implementation Guides) in projects. A project can be used to share all your FHIR resources and documentation with the community as well as to collaborate with other project members.

1.5.1 Project page

Each project contains a couple of tabs depending on the settings of the project and your role in the project. The tabs described below are visible for each user in each Simplifier project.

Introduction

This section serves as an overview of your project. This is a good area to share information about your project with people that may be team members or viewing your project for the first time.

On the Introduction page of a project you can find:

- A summary text as added by the project owners
- A summary table describing the contents of the project:
  - Total number of resources per resource type
  - Total number of examples per resource type
  - The canonical base URLs supported in the project
  - The workflow statuses supported in the project

When clicking on a resource type in the summary table (e.g. profiles) you navigate to the Resources tab, where the resources will be filtered on the selected resource type.
Resources

On the Resources tab you can find all the Conformance and Example Resources for the project. This tab also offers a search and filter option. You can filter your results to include or exclude certain Resource categories, Core base types, Example Resources type, FHIR status, and Workflow status.

Guides

The Guides tab shows all Implementation Guides for this project built in Simplifier. Click on the Guide or hit the Browse button to go to the Implementation Guide. Use the IG-editor to create and edit Implementation Guides.

Members

On the Members tab you can find all project members and their role. This tab also offers a search option, allowing you to search for other members using their full name or username. Depending on your role in the project you can add project members here.

Log

On the Log tab you can see all the changes that have been made to this project in the past. This is a good way to stay in touch with what’s happening within your favorite projects.

Issues

On the Issues tab you can leave your issues regarding the project. Note that this tab is not visible in all projects. The Issue Tracker is a paid functionality that allows project members to collect feedback from other project members or (depending on the project settings) other Simplifier users.

Dependencies

Currently references to other profiles are resolved by latest version. By adding package dependencies, references will be resolved in those packages. This gives you much better versioning control over the resources you include.

Currently this only works for downloaded packages. In the next release it will work for snapshot generation, validation and link generation.

Packages

The Packages tab shows all released versions of a project. Project members may use this beta functionality to release project versions containing (a selection of) resources as a package. Packages are now public BETA. We encourage you to try them out. But you should not use them for production environments yet.

1.5.2 Bookmarks

To stay informed in real time click the Bookmark button in the top right. You do not have to be a member of a project to stay up to date on the latest developments.
1.5.3 Create a project

In the Projects tabs on your Portal page you can find the button labeled Create.

Clicking this button will allow you to create a new project by entering a Display Name, Description, and Scope. Once the project has been created you can then customize project information, add resources, add members, and follow changes that are occurring in that project.

**Title:**

My First Simplifier Project

Title is mandatory and should be longer than 3 characters.

**Description:**

This is just a test project for me to play around in Simplifier

A short description of your project that the title cannot convey.

**FHIR Version:**

STU3

What version of FHIR do you use for the resources in this project?

**Scope:**

Test

What is the target audience?

**Nationality:**

Netherlands

Select a country

**Status:**

Private projects can only be seen by you and other project members.

- Public
- Private

Create  Cancel
1.5.4 Project Settings

You can always change your project settings by clicking on the Settings button in the right upper corner. There are a couple of options in the Settings menu, which will be explained below.

Properties

Here you can edit the following properties:

- The title and subtitle of your project
- The FHIR version (DSTU2 or STU3)
- The scope of your project (core, international, national, institute, regional or test). As choosing the right scope will make it easier for others to find your project, please use test for all test projects and test projects only.
- Issue tracking by project members and other Simplifier users:
  - Turn issues on or off for this project (when activated the issues tab will be visible on the project page depending on the user’s role)
  - With the issues visibility setting you can chose whether issues are visible to all Simplifier users or project members only.
  - With the community issues setting you can chose whether all Simplifier users or only project members can create or respond to issues.
- Publishing project resources to the FHIR registry (registry.fhir.org). Note that this setting is only available in public projects. Private projects and test projects are excluded from the registry.

Project url

Here you can edit the URL key to your project on Simplifier, which is by default the name of your project. Be careful editing the URL key in a later stadium as it will break all existing links to your project.

Documentation url

If you have any external documentation on your project, you can add the link here.
Avatar

Choose this option to add your company logo or just any cool picture you like!

Workflow

Here you can select one of the custom workflows of your organization to use it in your project. The workflows are configured and mapped to the FHIR workflow at the organizational level.

Canonical base urls

Project owners can customize their base canonical URLs to brand their projects. Canonical URLs of resources will only be valid if they match the canonical base URL of their project. Make note that by choosing a canonical URL you are also certifying that it is within your rights to do so.

Import log

Use this option to retrieve a log with all uploads to your project.

Administration

This option is only available for project members with an admin role. Use this option if you want to delete your project or if you want to change its visibility to either public or private.

Score

This beta functionality scores your project based on some predefined quality criteria.

1.5.5 Add resources to your project

On the Resources tab you can find all the Conformance and Example Resources for the project.

If you have “Write” rights to a project you will see an option to Upload resources at the top of the Project Page. Here you can choose one of the following options:

1. Upload a local file

The following Upload options are available:

- You can upload .json or .xml
- You can upload a single resource or multiple in a bundle
- You can upload a .zip file containing multiple files

2. Fetch a resource from another FHIR server

If you choose to add resources from a FHIR server, you can do a simple GET or a FHIR search. The first will add a single resource, the latter will let you add multiple resources at once.

Examples:

- To add a Patient resource with id “example”: http://example.org/fhir/Patient/example
• To add all Patient resources that conform to the DAF profile: [http://example.org/fhir/Patient?profile=http://hl7.org/fhir/StringDefinition/daf-patient](http://example.org/fhir/Patient?profile=http://hl7.org/fhir/StringDefinition/daf-patient)

3. Copy/Paste json or xml code

By selecting the Copy/Paste option, you can add your own json or xml code to add a single resource or a bundle of resources. If your code contains a bundle, you can either upload it as a single resource or select the split bundle check box to upload all entries as separate resources.

Adding multiple resources at once

If you add a batch of resources (via a bundle, a zip, or a search query), you can choose how to publish the resources that are part of the batch. The following options are available:

- Do not publish these resources automatically
- Publish these resources when their status is “Active” (examples will always be published)
- Publish all of them

1.5.6 Add project members

The Members tab displays a list of all the members with rights to that project. In this section you can invite Simplifier and non-Simplifier members to your project by clicking the Invite User button and typing in an email address.

When adding new members to your project you have the option to assign “Admin”, “Writer”, or “Reader” rights to that user. This assigns their rights within that particular project and can be changed at any time should someone’s function change. Users have the following rights within each role:

- **Admin**- Has the rights to change anything within the project with the exception of project ownership.
- **Writer**- Has the rights to add, change, and delete resources within the project.
- **Reader**- Has the rights to view anything within the project but cannot make any additions or changes.

Along the top of the Members tab you will find a summary of User information for your project. The number of users, the max users allowed for this project (in accordance with the type of plan you have), and the number of invitations you have pending (the number of users who have a not yet accepted an invitation).

![Users 1 Max Users Allowed 10 Pending Invitations 0](image)

1.5.7 Track Project Changes

On the Log tab you will find event tracking of a project. This log keeps a list of all changes made to resources within the project, along with the name of the person that made changes and the time the changes were made.

At the top of the screen you will find the Atom feed button. This allows you to subscribe to stay informed about any changes being made within your projects. To utilize this feature, navigate to a project on Simplifier.net that you are interested in following. Once there click on the Subscribe button in the upper right hand corner and copy the link into a feed reader of your choice. You are then ready to start receiving updates.
1.5.8 Issue Tracker

Would you like to capture feedback about your resources from users? The Issue tracker option is a great way to do this. If you go to the Settings dropdown menu and then select Properties. You will see the option to Enable Issues at the bottom of your screen. By selecting the On option, you enable the issue tracking feature of your project. There are two additional options that display once you have turned the Issue tracking on. You have the option to limit Issue visibility to project members or make them publicly visible. The issues that are reported by the community can also be limited to be visible only to your projects member or visible to the public. These issues can either be reported at a resource level or at a project level. At the project level you will see issues that are project specific and issues from all the resources in that project on the Issues tab.

1.6 Organizations

For our members with an Enterprise plan there is the functionality to create and manage an organizational account within Simplifier. An Organization account allows you to create and share projects, resources, and Implementation Guides via your organization with others within and outside your organization. You can add members and adjust rights to the project.

1.6.1 Organization Settings

Once you have made your account you can create and organize organizational project lists, create custom workflows, manage members, and follow your organization’s newsfeed. These settings can be accessed via the Settings dropdown once you are on your organization’s page. The quickest way to visit your organization’s page is to select it from the Organizations tab in your personal portal.

1.6.2 Organization Portal

Your organization portal is a landing page for your organization that is visible to the outside world. On your portal you can add a logo, information about your organization and news feeds. Your organization portal also shows a list of your organization’s projects.
1.6.3 Custom Workflow

Custom workflows allows you to create statuses for your resources that fit your needs. You can use this to communicate to your organization team, in your own language, the progress of any given resource. This is in addition to the standard FHIR workflows of “Draft”, “Active”, and “Retired”. The FHIR workflows cover the following statues:

- Draft: This resource is still under development.
- Active: This resource is ready for normal use.
- Retired: This resource has been withdrawn or superseded and should no longer be used.

To create a Custom Workflow select Custom Workflow from the Settings dropdown on your organization’s page. The statuses that you make need to be mappable to the standard FHIR statuses. If you are creating your own workflows be sure to create a workflow for each associated level of the FHIR hierarchy. This is necessary because the FHIR statuses do not default into your custom list.
When you search for resources in Simplifier, the page with search results shows the Custom Workflow status next to the resource. You can click on the status to obtain an overview with the possible statuses and their explanations.

1.6.4 Organization Project Lists

Members with an Enterprise account have the ability to endorse others projects by creating Project Lists. This list is not necessarily a list of your own projects but of interesting projects that you would like to endorse within the Simplifier community. This list, once created, will show up on your organization’s page within Simplifier. To create this list, select Endorsed projects in the Settings menu on your organization’s page.

On the other hand your Organization projects are a compilation of the projects that are owned by the organization. This section of your project maintenance is where you need to create and maintain organization projects. A list of these projects will also display on your organizations page. You can manage your Organization projects by selecting Projects from the Settings menu.

1.6.5 Newsfeed

The News section of your Organization can be used to communicate to the outside world about new developments within the project. Once you have finalized resources or finished an Implementation guide this is a great place to create an article announcing the changes.
1.7 API

The endpoint of a project or resource can be used to either search for resources in Simplifier or to read, search, add and update resources in a FHIR client. System wide searches and history searches are also supported. To retrieve the endpoint of a project or resource in Simplifier click on API in the top right menu when visiting either the project or resource page. The below image shows the location.

1.7.1 Project FHIR API

The project FHIR API is available at both project and resource level. You will need the FHIR endpoint when you want to use a FHIR client to read or add and update resources to your Simplifier project. For example, the Vonk FHIR server supports the import of resources from Simplifier by supporting a (manual) import operation and specification of the project’s endpoint and authentication in the appsettings.

On the project page, you can retrieve the FHIR endpoint of your project.

On the resource page, you can retrieve the FHIR endpoint of your resource. You will need this endpoint to find your resource when using a FHIR client.
1.7.2 Project ZIP API

The project ZIP API is available at project level. You can use the ZIP endpoint for synchronization. With an HTTP tool you can use GET or PUT https://api.simplifier.net/<yourproject>/zip to retrieve or update your project in zipped form.

JWT authentication

The ZIP endpoint is available for Simplifier users based on JWT authentication.

First retrieve a JWT token from Simplifier. This works with a POST at https://api.simplifier.net/token with your account details in the message body in JSON format. Header should be Content-Type: application/json

```
POST https://api.simplifier.net/token
Header:
Content-Type: application/json
Body:
{
    "Email": "youremail@example.com",
    "Password": "your password"
}
```

Downloading or uploading your project works with a GET or a PUT at https://api.simplifier.net/<project>/zip with an authorization header that includes your retrieved token as shown below. The token is valid for 8 hours.

```
GET https://api.simplifier.net/<yourproject>/zip
Header:
Authorization: Bearer <access_token>
```

1.7.3 Simplifier FHIR API

Using the global Simplifier FHIR API, users can search for all resources in Simplifier. For example, the request GET https://stu3.simplifier.net/open/Patient can be used to retrieve all (STU3) Patient resources from
Simplifier. The global Simplifier endpoint of your resource is available at the resource page beneath the API icon. All resources have a globally unique GUID.

![Simplifier FHIR API](https://stu3.simplifier.net/open/StructureDefinition/3eafdf6d-db41-4dcc-b4fd-518de3695ff9)

*The global endpoint is where users can search for all resources in Simplifier. Resources have a globally unique guid*

**Search Parameters**

It is possible to use search parameters and search result parameter to filter the results from Simplifier. All parameters, with the exception of ‘description’, follow the STU3 FHIR specification. The following parameters are implemented:

Search parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>uri</td>
<td>The uri that identifies the structure definition</td>
<td>StructureDefinition.url</td>
</tr>
<tr>
<td>type</td>
<td>token</td>
<td>Type defined or constrained by this structure</td>
<td>StructureDefinition.url</td>
</tr>
<tr>
<td>status</td>
<td>token</td>
<td>The current status of the structure definition</td>
<td>StructureDefinition.url</td>
</tr>
<tr>
<td>publisher</td>
<td>string</td>
<td>Name of the publisher of the structure definition</td>
<td>StructureDefinition.publisher</td>
</tr>
<tr>
<td>jurisdiction</td>
<td>token</td>
<td>Intended jurisdiction for the structure definition</td>
<td>StructureDefinition.jurisdiction</td>
</tr>
<tr>
<td>kind</td>
<td>token</td>
<td>(primitive-type</td>
<td>complex-type</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Will look at the publication description used in Simplifier (set either manually by user or generated automatically using the FHIRpath metadata expressions written in project settings), not the description value inside the Confromance Resources.</td>
<td>StructureDefinition.description</td>
</tr>
</tbody>
</table>

Search result parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_sort</td>
<td>Only default “lastUpdated” is implemented.</td>
</tr>
<tr>
<td>_count</td>
<td>Default value is “false”. The parameter _count is defined as a hint to Simplifier regarding how many resources should be returned in a single page.</td>
</tr>
<tr>
<td>_summary</td>
<td>The _summary parameter requests the server to return a subset of the resource.</td>
</tr>
</tbody>
</table>

Examples
1.8 Package management

Simplifier.net offers functionality to handle packages and dependencies. This functionality allows you to publish packages based on your project resources, which can be immediately installed and used by people implementing your profiles. The Simplifier FHIR package server is NPM compatible. You may either connect to this server using a NPM client or use our cross platform FHIR command line tool called Torinox. On this page we will explain how to manage packages and dependencies in Simplifier, how to use Torinox to install packages for validation, how to manage packages and how to create your own packages in Torinox.

1.8.1 Packages

View packages

Visit the Packages tab of any Simplifier project to see which packages are available in this project. For more information about a package and its content, click on the name of the package.

The Introduction page will give you an overview of the package:

- Install instructions; shows the command you need to install the package, click on the blue copy icon to copy it to your clipboard. Click on Torinox or NPM to switch to your preferred tooling.
- Release notes; shows the release notes given by the author of the package.
- Dependencies; shows the dependencies to other packages.
- History; shows the previous versions of the package, click on a version name to see the details.
- Info; shows information about when the package was created, a link to the project it is part of and a download button to download the package.
Switch to the Files tab to see the content of the package.

**Publish packages**

Visit the Packages tab of your project and click on Create > Create new package to create a new package. Provide a name, version number, description and release notes for your package. Note that the name of your package should include at least one dot. Indicate if your package is a prelease package or not and click Create to publish your package. To create a new version of an existing package, click on Create and select Create new version for .. followed by the name of your package. Add the required information and click Create to publish the new version of your package.
1.8.2 Dependencies

View dependencies

Visit the Dependencies tab of any Simplifier project to see a list of its package dependencies as well as indirect dependencies. Click on the name of one of the listed packages to see the details of this package. This will show the information as explained in the View Packages section.
Add dependencies

Visit the Dependencies tab to add dependencies to your project. There are two ways to do so. One way is to browse Simplifier for existing packages and add them to your project. The other way is to directly edit the JSON code.

Click Manage to search for existing dependencies. Type a search string in the search box and select a package and its version from the search results. Click Add to add the package to your project. When you are finished adding packages click Save to save the changes to your project.

Click Edit to directly edit the JSON code and add the packages and their version to dependencies.
Remove dependencies

To remove dependencies from your project, you could either select Manage and click on the recycle bin icon next to the package you want to remove or select Edit to directly edit the JSON code.

1.8.3 Torinox

Torinox is our (free) command line tool for FHIR. Torinox allows you to communicate with any FHIR server. With simple commands you can easily download, upload, validate and transform resources, zip them, bundle them or split bundles. Torinox offers many features. One of them is to install, create and manage FHIR packages.

Install Torinox

Before you install Torinox, make sure you have dotnet core 2.1 SDK. installed. Next, run the following command in your command line window:

dotnet tool install -g torinox

That’s it, you’re all set! Run fhir –? to see which functions are available. Here below are a couple of examples in Torinox.
Install packages

Copy paste the install command from the install instructions in the package details on Simplifier. For example, try to install a demo package from the National Patient Care Package Demo project. Make sure to do a restore afterwards, so Torinox will calculate and install all dependencies as well.

```bash
fhir install Demo.National.PatientCare 1.0.0 fhir restore
```

Resource validation

When all required packages have been installed, you can start validating your resources. First, set up your FHIR server if you haven’t done this earlier.

```bash
fhir server add vonk https://vonk.fire.ly
```

Next, get an example Patient from your server.

```bash
fhir read vonk Patient/example
```

Finally, run the command to validate this resource.

```bash
fhir validate
```

Note that the validation will only work when you have all packages and dependencies installed.

Create package

Torinox allows you to create packages from your folder. For example, create a folder called ‘MyDemoProject’ on your machine and place one or more profiles in this folder. In addition, place a file called package.json in this folder to define your package:

```json
{
  "name": "MyDemoProject.core",
  "version": "0.0.1",
  "description": "My demo project",
}
```
Now you can create your package in Torinox: `fhir pack C:\MyDemoProject`

Run the following command to get the contents of your new package: `fhir contents MyDemoProject.core`

**Package management**

Torinox supports a couple of commands which allow you to manage your packages. Here below are a couple of examples:

Remove a package from your context: `fhir remove Demo.National.PatientCare`

See which versions are available of a specific package: `fhir versions Demo.National.PatientCare`

## 1.9 GitHub integration

The GitHub integration enables you to link Simplier projects to GitHub repositories. When linking the project with the repository, the resources from the repository are fetched and added to your Simplier project. The GitHub integration is part of the paid account plans of Simplier. This page will elaborate further on how to perform the linking.

### 1.9.1 GitHub Linking

In order to perform the initial setup link of a GitHub repository to a Simplier project, the user needs to have admin rights in GitHub and Simplier. You can link a repository in GitHub to your Simplier project by going to your project and selecting the *Link repository* button under the *Options* menu.

You will be redirected to the GitHub login or, if you are already logged in to GitHub, straight to the page where you will be asked to authorize that Simplier can access your repository.

Once this step is complete, you will be directed back to Simplier, where Simplier will ask you which repository you want to link and which branch within that repository.

To link an Organization GitHub repository after an already made link, you should edit your Organization settings in GitHub. Log in to GitHub and go to the Settings tab in your Organization. Press the *Remove Restrictions* button under third-party access. Now you should see the repositories from that organization when you try to link a GitHub Repository.

### 1.9.2 Settings

When connecting a GitHub repository you will be asked to choose a strategy about how to deal with resources that fail to import (most likely because they are not valid FHIR resources) and whether to remove resources from your project when they are removed from your repository.
1.9.3 Fetch Updates

After the linking is performed, any updates to your resources in the GitHub repository will be sent automatically to Simplifier. Based on the linking settings, resources from your Simplifier project will be updated to, deleted from, or added to GitHub.

1.9.4 GitHub Sync

GitHub Sync enables you to synchronize your Simplifier project with the linked GitHub repository. Images or css files that up until recently were not supported by Simplifier are now available. If you decided to skip the initial import after linking or you have old markdown, you are now able to access them in Simplifier by using the sync process.

1.9.5 GitHub Include/Exclude

The include/exclude feature allows users to specify the folders or file types that should be fetched from GitHub into Simplifier. This also applies to the folders or file types that you would like to be excluded from the import. You are able to specify these rules right after the initial link to the repository, before the initial import or later by accessing the Settings option from the Github dropdown menu on the project page. By default, everything is included. If include statements are added then everything else (not included in your include statement) will be excluded by default. Simplifier only imports xml, json, images, and markdown file types. Comments are also supported using the “#” character. The syntax is the following:

Examples:

# Include examples

FHIR/IG/**
*.xml

# Exclude examples

!FHIR/*.img
!*cs
!FHIR/examples/*

1.9.6 Gitwebhook per Branch

For our Team and Enterprise account users, we added the ability to filter files that are synced from GitHub. We followed the .gitignore logic, but expanded it slightly to work from a include perspective as well. After you’ve set up a GitHub link, you can specify rules to include or exclude certain files or folders from your GitHub repository.

# Include all files under examples folder: resources/examples/*

# Include all files and folders under staging: project/staging/**

# Include all xml files: *.xml

# Exclude all json files under temp folder: !temp/*.json

Note that if you only use exclude patterns (starting with a exclamation mark !), Simplifier will assume you want to include everything but those filters.

You can use the same globbing patterns you are used to in a .gitignore file.
1.9.7 GitHub multiple branch linking

We have created a way of allowing multiple branches from the same repository in GitHub to link to different projects in Simplifier. The GitHub API only allows one link per repository, so we created a multiplexer on our side to handle this.

1.9.8 GitHub webhook to manage Implementation Guides

The GitHub webhook allows you to edit your Implementation Guide without using the editor itself. Your GitHub repository can contain an IG resource which states the structure of the IG and refers to all the included pages (also in the same repository).

To get started, you create a project and establish a webhook to your GitHub repository.

- Create an ImplementationGuide resource, or download an ImplementationGuide resource from a Simplifier project together with all the belonging content (.md pages, images, etc.)
- Push the resources to your GitHub repository.
- Your Simplifier project will now contain all these files. Locate the ImplementationGuide resource.
- Click on Update followed by Edit: Update by editing in IG editor. This will convert the ImplementationGuide resource to a Simplifier IG.

From now on, you can edit the ImplementationGuide resource and its pages from within your GitHub repository. Your changes will be automatically pushed to Simplifier and your online Implementation Guide.

1.10 Backup and Synchronization

The Simplifier team has a solid backup process. We have continuous backup by our cloud storage provider. In addition, we perform a weekly manual backup. Please note that this is for emergency situations, and you should not require to depend or rely on that from a user perspective.

Customers may want to have a possibility to backup whatever content they have on Simplifier (or more generally in the cloud for that matter). Here are some ways to make sure your data is regularly backed up:
1.10.1 FHIR endpoint

Each Simplifier project has a FHIR endpoint. With this, you can get a specific resource from your project using any FHIR client. You can also get all resources from a specific resource type.

Future plans: We have plans to implement the FHIR “global search” endpoint where you can get all resources from all types.

1.10.2 Zip Endpoint

Simplifier has a ZIP API for every project. With an HTTP tool you can use GET or PUT https://simplifier.net/yourproject/api/zip to retrieve or update your project in zipped form.

1.10.3 Download

You can always download the current versions of all resources, including or excluding texts and images. Filepaths are preserved from GitHub and the regular upload. Resources that are initially uploaded through the fhir endpoint will have a persistent filename, but no absolute path.

1.10.4 Client tool

We can provide a client tool called Torinox that allows easy and automated synchronization and backup. Torinox uses the simplifier ZIP API and was built to assist CI/CD scenarios.

All you need is this command line syntax: fhir sync <projectname> -down
Download it free.

1.10.5 Atom feed

If you want to automate backup on any updated file, you can use the atom feed of the project log to trigger your client backup.

1.10.6 Webhook

This is not implemented yet, but will be put on our roadmap if there is enough demand.

1.11 Connectors

Simplifier allows users to create connector scripts: such scripts allow external web applications to open Simplifier’s resources or call an external service and show the result in Simplifier. Think about sending a StructureDefinition to an examples generator, a TestScript resource to a server test application, a ValueSet to a terminology server etc.

1.11.1 Subscribe to a connector

To make use of an already available connector, you need to personally “subscribe” to one. To do this, please follow these steps:

1. Go to https://simplifier.net.
2. Log in to your portal by clicking the Log in button in the upper right corner, or go to the following link.
3. Click on “Connectors” in the portal menu.
4. Click on “Shop” to see a list of available connectors.
5. Subscribe to the connector that’s right for you and click the “subscribe” button.

After you’ve subscribed to a connector, you can start using it - see below.

1.11.2 Use a connector

Each connector specifies one or multiple resources that it can be used with. You can view this in the shop or, if you are already subscribed, in your personal connector portal page. Once you have subscribed to a connector, you can use the connector for every resource type specified for that resource.

Example:

If you have subscribed to a resource which generates an example for StructureDefinitions, you can access that connector on each StructureDefinition. Once you click on it, the connector sends a request to open this specific resource from Simplifier.

1.11.3 Connector scripts

To create a connector your own personal use or for publication and use by others, follow these steps:

1. Go to https://simplifier.net.
2. Log in to your portal by clicking the Log in button in the upper right corner, or go to the following link.
3. Click on “Connectors” in the portal menu.
4. Click on “Create”.
5. Fill in the metadata for your connector under the “Properties” tab.
6. Use the “Script Editor” tab to write the actual script. **Note:** You have to write your script using Javascript.

Example:

```javascript
// trivial connector that redirects the webpage to example.com with a parameter to the resource
// the connector was used with
```

This script sends the FHIR endpoint of the resource to example.com. Example.com then would retrieve the profile and creates an example instance.

7. Once you are satisfied with your input, click on “Save”. The connector is then created and will be visible in the “Connectors” tab of your personal portal.

8. To submit your connector for publication, open the connector from your “Connectors” tab and click on “Request Connector Publication” in the upper right. Your connector will then be submitted for review by the Simplifier administrator. Once the connector is approved it will be available in the shop for use by others. **Note:** The Simplifier Administrator reviews all connectors submitted for publication to ensure the quality and security of all content.
1.11.4 Available libraries

There are several things that you will likely do when you write a connector script. And to make this as easy as possible, your script will have access to the following Javascript libraries:

- JQuery 2.1.1
- JQuery.Redirect 1.0.1
- fhir.js

1.11.5 Helpers

To perform actual FHIR requests, you have access to several methods and variables that will be set by Simplifier before your connector script is executed:

**fhirServer**  Wrapper of the fhir.js client with a jQuery adapter. You will instantiate a new client by providing the base url and you will have access to all the functionality provided by fhir.js with an embedded jQuery adapter (which is required by the plain fhir.js constructor). For example:

```javascript
var client = fhirServer("www.example.com");
```

**simplifier**  This is a ready to use fhir.js client that connects to Simplifier. When your connector is executed by a user, they will do that from the page that shows a resource. The FHIR endpoint of that resource can be accessed in the following manner:

```javascript
simplifier.ResourceEndpoint // contains the URL the resource is accessible at
```

This allows you to send the fhir endpoint to another server, so that that server can fetch the resource that is displayed on the current page.

You will also have access to the resource itself in JSON or XML in case of post calls to another server.

```javascript
simplifier.ResourceJson // contains the JSON of the resource
simplifier.ResourceXml // contains the XML of the resource
```

Since in essence it is a fhir.js client, the simplifier client also exposes all other methods from in fhir.js like:

```javascript
simplifier.read(…)
```

**returnUrl**  This placeholder will contain the url of the page from where your connector script is executed.

**$.get()**  JQuery’s AJAX function to fetch data from a server:

```javascript
$.get(url).success(function()
{


});
```

**$.post()**  JQuery’s AJAX function to post data to a server:

```javascript
$.post(url, body).success(function()
{


});
```

**$.redirect()**  Simplifier’s addition to JQuery to do a POST redirect to a webpage with a payload:
1.11.6 Read-only connectors

There are two categories of read-only connectors: ones that redirect and ones that run an AJAX call. Read-only connectors supply data to an external service but the service doesn’t write back to Simplifier, example:

```javascript
$.ajax({
  type: 'post',
  data: simplifier.ResourceJson,
  headers: {
    Accept: 'application/fhir+json',
    'Content-Type': 'application/fhir+json'
  },
  dataType: 'json',
  success: function (data) {
    $.redirect("/render/operationoutcome", data.responseText);
  },
  error: function (data){
    var output = data;
    if (output === undefined){
      console.log("Error! Action could not be performed");
      $.redirect("/render/operationoutcome", data.responseText);
    } else{
      console.log("Validation failed");
      $.redirect("/render/operationoutcome", data.responseText);
    }
  }
});
```

AJAX connectors run a query on an external server and can use Simplifier to render the result as an OperationOutcome:

```javascript
// sample connector that validates the patient resource
// and uses Simplifier to render the outcome
$.ajax({
  type: 'post',
  data: simplifier.ResourceJson,
  headers: {
    Accept: 'application/fhir+json',
    'Content-Type': 'application/fhir+json'
  },
  dataType: 'json',
  success: function (data) {
    $.redirect("/render/operationoutcome", data.responseText);
  },
  error: function (data){
    var output = data;
    if (output === undefined){
      console.log("Error! Action could not be performed");
      $.redirect("/render/operationoutcome", data.responseText);
    } else{
      console.log("Validation failed");
      $.redirect("/render/operationoutcome", data.responseText);
    }
  }
});
```

1.12 Embedded Rendering

Simplifier enables rendering of profiles as a tree, XML or table. Users with an enterprise account can also embed rendering in their own website by using the following URL and copying it into their own code. Note that this only works for public projects that are created by a user with an enterprise account.

https://www.simplifier.net/embed/render?id=[Name of your project]/[Name of your resource]

For example, https://www.simplifier.net/embed/render?id=nictizstu3-zib2017/nl-core-patient shows embedded rendering of the nl-core-patient defined by Nictiz. You can copy this URL in your browser to see what the rendering looks like.

To embed the rendering in your own website, you could use the following HTML code:
Below you see the result. You could adjust the height and width of the iframe as you like.

### 1.13 Implementation Guide Editor

The Implementation Guide Editor enables you to make implementation guides (IG) using the resources available in Simplifier. You can include your own project’s Resources or those from others in your IG. With the help of this Markdown based editor you can easily construct an organized and practical IG that is both easy to read and navigate. This IG editor is available in all Simplifier account plans. Visit [https://simplifier.net/pricing](https://simplifier.net/pricing) for more details.

This page will elaborate further on getting started and how you can use the IG editor.

#### 1.13.1 Create your first IG

You can access the IG editor via the **Guides** tab in your project. In earlier versions of Simplifier, Implementation Guides were created from a user’s portal, but this is no longer possible as Implementation Guides are linked to a specific project. Use the **Create** button to create a new Implementation Guide and provide a title for the IG.

Click on **Browse** or the Implementation Guide itself for a preview of the guide. Click on the **Edit** button to open the Implementation Guide in the IG editor.

#### Linking existing IG’s to your project

Existing Implementation Guides that are not yet linked to a project can be moved on request. Just contact us through the feedback button.

#### 1.13.2 IG Editor Settings

The IG editor opens on the page of the root element. The IG Editor consist of three sections. On the left is the IG’s tree table which is used to define the outline of your IG and navigate between the pages of the IG. The middle section is the actual editor. This is where you will add and edit content. The right section is where you will find a rendering of the selected page.

By way of dragging the section bars you can adjust the size of each section to customize your view.

To adjust the settings of your IG click on the Settings icon (the rightmost icon representing a gear wheel). This brings you to a section that allows you to adjust the title and privacy on the Settings tab, or select an IG rendering format and Stylesheet file on the Style tab.
**Formatting style**

An IG can be rendered in one of three formats: a Tree table, Two Level Menu or HL7 Ballot format.

A Tree table rendering will display your IG with the elements in a format similar to the tree table with the elements and their hierarchy along the left side of the page.
A Two Level Menu rendering will display your IG with the elements in tabs along the top of the page.
A HL7 Ballot rendering will display your IG with the elements in tabs along the top of the page similar to the Two Level Menu rendering, but in the style of a HL7 Ballot IG.

To add a new menu item to your Implementation Guide, use the + icons in the upper left corner of the IG editor. Click on either the **add Child guide item** icon to add a sub menu or on the **add Sibling guide item** icon to add a menu at the same level as the one you have selected. You can rename the items by either double clicking them or use the pencil icon in the icon bar. To delete a menu item, select it and click on the recycle bin icon.

### 1.13.3 CSS-editor

When you click on the icon on the left of the Settings icon in the IG-editor, the CSS-editor will be opened.

With this editor you can edit your Style Sheet to make overall changes in the overall look and feel of your IG. For example, you may change the color of the navigation bar to blue or add your own logo to it. It is also possible to reset your changes by going back to the original CSS or download the original CSS as a separate file, so you can compare the differences with your own code.

Here below are a couple of examples that you can use to configure the lay-out of your IG:

```css
/* Change menu bar font color (title, menu & submenu) */
#ig-viewer .ig-view-content #ig-view-twolevelmenu .header .navbar a {
   color: white;
}

/* Change menu bar background color (only main menu) */
#ig-viewer .ig-view-content #ig-view-twolevelmenu .header .navbar {
   background-color: red;
}

/* Change menu bar font color (only main menu) */
#ig-viewer .ig-view-content #ig-view-twolevelmenu .header .navbar-nav > li > a {
   color: green;
}
```
1.13.4 Markdown

In the middle section is a Markdown based editor used to compose your IG content. Markdown is a text-to-HTML conversion tool. It allows you to write using an easy-to-read, easy-to-write plain text format. The following link provides an overview of the Markdown features which can be used in this editor: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet.

A short summary of frequently used features are as follows:

- Header size edits using `#Header size 1 to ######Header size 6`
- Adding Emphasis, also know as italics, with `*asterisks*` or `_underscores_`
- Adding Strong emphasis, also known as bold, with `**asterisks**` or `_underscores_`
- Adding Combined emphasis with `**asterisks` and `_underscores_`
- Strikethrough uses two tildes. ~~Scratch this.~~

The IG editor has features which allow you to include Simplifier content in your IG. These features work by using the statements written below in the editor. After adding these statements in the editor refresh the page, by pressing Ctrl + Enter or clicking the Refresh button, to make them visible in the preview section.
• \{{{tree:ProjectName/ResourceName}}\} - renders a tree structure as seen in the resource overview tab
• \{{{table:ProjectName/ResourceName}}\} - renders a table as seen in the resource table tab
• \{{{structure:ProjectName/ResourceName}}\} - renders the resource in a newly defined tree (unfinished)
• \{{{link:ProjectName/ResourceName}}\} - provides a link to the specific resource page on Simplifier
• \{{{namingsystems:ProjectName}}\} - lists all namespaces of a project in a table

The statement is made up of 2 parts: the kind of function and the location of the content in Simplifier. The location is based on the name of the project and the name of the resource. They can be found in the Simplifier URL after the hostname (https://simplifier.net/). For example, if you want to include a profile on an Organization, as described at https://simplifier.net/DAF/daf-organization a correct statement would be \{{{tree:DAF/daf-organization}}\).

The following statements add an index within the IG.
• \{{{index:root}}\} - gives an index of the entire IG
• \{{{index:current}}\} - gives an index of the current selected element

### 1.13.5 IG Storage

Since release 16.5 IG’s are stored as separate mark down files in your project. This means that you can access them as separate resources, add issues to them and check version history. In addition, your IG’s will be available in your GitHub repository and can be downloaded in a ZIP file together with the other files of your project.

To illustrate how this works, see the screen picture of an example IG containing two chapters called ‘First part’ and ‘Second part’. The First part also contains a child called ‘Child of first part’. The different parts of the IG are now accessible from the Resources tab in your project as well as from the search engine. To search for IG parts, just check the Texts box.

![IG Storage Example](image)

To Save your IG as a Resource, click on the Save button in the left pane of the IG-editor. Note that it is the tree structure that is saved. Textual changes are save automatically.
1.13.6 Export your IG

To use your IG outside of Simplifier, click on the Export button next to your IG in the Guides section of your project.

1.13.7 Convert ImplementationGuide resource to a Simplifier webbased IG

An ImplementationGuide resource can be converted to a Simplifier webbased IG. This comes in handy if you for example accidently deleted your IG or if you want to duplicate your IG in another project.

- Make sure that the project contains the ImplementationGuide resource and all the belonging content (.md pages, images, etc.)
- Locate the an ImplementationGuide resource.
- Click on Update followed by Edit: Update by editing in IG editor. This will convert the ImplementationGuide resource to a Simplifier IG.
- Follow the configuration steps and locate the IG in the Guides tab.

1.13.8 Manage your IG using GitHub

The GitHub webhook allows managing your Implementation Guide, without using the editor itself. You can find more information on how to set this up in the GitHub integration documentation.
1.14 Use Vonk FHIR Server with your Simplifier artifacts

If you have defined some profiles, valuesets and examples in your Simplifier project, you may want to setup a FHIR Server for you or others to test against. We have tried to make that extremely simple for you, using Firely Vonk FHIR Server. Here is how:

Currently we only enabled this for Windows 10, using PowerShell. Later we will add support for Linux and/or MacOS.

1. **Install Docker for Windows**
2. **Start it.**
3. **Open its settings (right click on the docker icon in the system tray and click ‘Settings’)**
4. **Under shared drives, share your local drive.**

5. **Now in Simplifier, go to the project page of your project, drop down the Download button and click ‘demo FHIR Server’**
6. This takes you to the ‘Demo Project FHIR Server’ page.

7. Click the ‘Download’ button

8. Unzip the downloaded file. For this explanation we choose to unzip to `c:\programs\demo-VonkDockerServer`

9. Open a PowerShell window

10. Since we have not (yet) signed the Powershell script that will start the server, you need to temporarily allow
execution of downloaded scripts:

```
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope Process
```

11. Navigate to the directory where you unzipped the download.
12. Then run the script `.\start-vonk-server.ps1`

```
PS C:\> cd C:\programs\demo-VonkDockerServer
PS C:\programs\demo-VonkDockerServer> .\start-vonk-server.ps1
```

13. The script will:
   - pull the necessary images from the Docker hub (vonk-web and vonk-mongo-db)
   - start the containers
   - import conformance resources from the specification AND your Simplifier project
   - import the examples from your Simplifier project

This will take a couple of minutes. Subsequent startups will be a lot faster though.

**Attention:** Docker and PowerShell are not perfect friends, and Docker may show some ‘NativeCommandError’ messages. You can ignore those.

14. You can use the `.stop-vonk-server.ps1` script to stop Vonk again.

### 1.15 FHIR Registry

The FHIR registry enables you to search for published FHIR resources created for STU3 in International, National, Institute or Regional projects on Simplifier.net. Test projects are excluded. The aim of this registry is to encourage re-use of profiles and extensions by providing a centralised search location for all of them. The registry supports free-text search as well as the following parameters:

- **resourceType:** search by resource type, (e.g. `resourceType:Patient`)
- **jurisdiction:** search by jurisdiction code in a conformance resource (e.g. `jurisdiction:US`)

Below you can download a quick guide to the FHIR registry, which was made for the HL7 Working Group Meeting in New Orleans (Jan, 2018).

[Quick guide to the FHIR registry (pdf)]

### 1.16 Contact

There are two ways to contact Simplifier.net:

1. The Contact link in the lower left. This link offers an area to fill in more specific information about the user or company associated with the contact. Use this link for all inquiries about setting up new accounts and services.

2. The Dialog icon in the upper right. This icon is visible within all pages of Simplifier.net and can be used to provide quick comments to the Simplifier administrators about the site. If you are logged in your account details will automatically be associated with the comment. We welcome all feedback about usability or suggestions about content!
Chapter 1. Welcome to Simplifier’s documentation!
Welcome to Forge’s documentation!

Contents:

2.1 Introduction to Forge

2.1.1 Conformance in FHIR

Conformance Resources ("Profiles") are an important aspect of the FHIR standard. They allow you to tailor FHIR to your needs and define how exchanging partners use the FHIR specification. Profiles have an international, a national, regional or local scope, or are designed for specific use cases.

2.1.2 Profile Designer

Creating and maintaining these files by hand using an XML editor or an Excel sheet is error-prone and requires detailed knowledge of the Profile resource. Forge is the user-friendly editor for creating and editing profiles that enables modelers to create and manage profiles using a graphical user-interface.

2.2 Click once installer

2.2.1 Running on Windows

Forge is deployed using a ClickOnce installer. You can download and install the application without requiring administrator permissions (except .NET FW, see below). The application automatically detects and installs updates when you start the application. After installation of a new release, you will again receive automatic update notifications on application startup.

Forge is based on the Microsoft .NET Framework version 4.7.2 and WPF. The ClickOnce installer will download and install the .NET Framework 4.7.2 on demand, if not already available.
Warning! Installing or updating the .NET Framework requires administrator permissions. If you have insufficient permissions, please contact your system administrator.

### 2.2.2 Running on Linux

The .NET Framework and WPF support only the Windows operating system. However, it is possible to run Forge on Linux using Wine. While we do not officially support running Forge on Linux, here are the steps to get you started:

1. Download and install the latest version of Wine (4.0.1).
2. Install winetricks:
   ```
   wget https://raw.githubusercontent.com/Winetricks/winetricks/master/src/winetricks
   chmod +x winetricks
   sudo mv winetricks /usr/local/bin
   ```
3. Install .NET 4.7.2. This step may take 5-10 minutes to complete. Answer yes to installing Gecko and no to Mono.
   ```
   WINEPREFIX="$HOME/.forge" WINEARCH=win32 winetricks -q dotnet472
   ```
4. Install core fonts:
   ```
   WINEPREFIX="$HOME/.forge" WINEARCH=win32 winetricks corefonts
   ```
5. Download and run the Forge setup package:
   ```
   WINEPREFIX="$HOME/.forge" WINEARCH=win32 wine setup.exe
   ```
6. Ready!
   To run Forge again the next time, repeat the command:
   ```
   WINEPREFIX="$HOME/.forge" WINEARCH=win32 wine setup.exe
   ```

### 2.2.3 Running on macOS

The .NET Framework and WPF support only the Windows operating system. However, it is possible to run Forge on macOS using Wine. While we do not officially support running Forge on macOS, here are the steps to get you started:

1. Open the Terminal (you can do so via Launchpad)
2. Install Homebrew by copy/pasting the following into the Terminal:
   ```
   /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/ master/install)"
   ```
3. Install Wine itself. This may take some time:
   ```
   brew install wine cabextract winetricks
   ```
4. Install .NET 4.7.2. This step may take 5-10 minutes to complete.
   ```
   WINEPREFIX="$HOME/.forge" WINEARCH=win32 winetricks -q dotnet472
   ```
5. Improve font rendering. First, run the following:
6. Then in the **Wine** configuration window, go to the **Graphics** tab and set the **Screen resolution** to 150 dpi.

7. Download the **Forge** setup package from Simplifier.

   Move `setup.exe` to your **Applications** folder and rename to `Forge-R4.exe` (or `Forge-STU3.exe`, depending on your version). Then execute the downloaded file to install the application:

   ```
   WINEPREFIX="$HOME/.forge" WINEARCH=win32 wine Forge-R4.exe
   ```

8. Ready!

   To run Forge again in the future, rerun the last command in the terminal:

   ```
   WINEPREFIX="$HOME/.forge" WINEARCH=win32 wine Forge-R4.exe
   ```

### 2.2.4 Known issues

1. On some systems, having an active secondary display will cause application rendering issues (e.g. gray window). Please try to detach/disable secondary screen before starting the application.

2. The application requires access to core fonts for rendering the UI. If Forge crashes during startup, try to install core fonts

   ```
   WINEPREFIX="$HOME/.forge" WINEARCH=win32 winetricks corefonts
   ```

### 2.3 Launching Forge

#### 2.3.1 Start Menu

The easiest way to launch Forge is via the start menu shortcut that the ClickOnce installer creates during the installation. Alternatively, open the start menu and search for “Forge”. Windows should locate and display the shortcut to the application. You can manually pin the shortcut to the start menu or the taskbar for your convenience.

#### 2.3.2 Command line

You can also start Forge from the command line. This allows you to launch Forge programmatically, from other applications. You can start the application from code in different ways:

1. Directly, via the main executable

   First, you need to determine the installation folder that contains the main executable `Forge.exe`. Unfortunately, the ClickOnce installer deploys the application to a personalized AppData subfolder that is hard to find. The Forge Options menu provides a command *Open application folder* that helps you find the location of the ClickOnce installation folder that contains the main executable.

2. Indirectly, via the start menu shortcut

   You can also launch the start menu `.appref-ms` shortcut created by the ClickOnce installer from code. The start menu shortcut is always created in a fixed location with the following path:
Alternatively, you can create or generate an .appref-ms shortcut in a well-known location. The shortcut is a single-line text file with the following contents:

```
http://downloads.simplifier.net/forge/stu3/Forge.application
#Forge.application, Culture=neutral, PublicKeyToken=d35f0fdbb3d5e195,
processorArchitecture=msil
```

### 2.3.3 Command line arguments

The main Forge executable accepts command line arguments. You can specify one or more documents to open:

```
Forge.exe [filePath] [filePath] [...]
```

The specified arguments must be fully qualified absolute file paths. After startup, Forge will try to load all the specified files, if they exist.

If you launch Forge indirectly via the .appref-ms shortcut, then you can also specify a single (!) command line argument. For example:

```
"%APPDATA%\Microsoft\Windows\Start Menu\Programs\Firely\Firely FHIR Tools\Forge for HL7 FHIR STU3.appref-ms" "C:\Profiles\MyPatient.xml"
```

Unlike the main executable, the .appref-ms shortcut does not accept multiple command line arguments. This is a limitation of the ClickOnce installer technology.

### 2.4 Features

Forge is the user-friendly editor for creating and editing profiles that enables modellers to create and manage profiles using a graphical user-interface.

With Forge you can:

- Open profile folders
- Create new Profiles
- Edit existing Profiles
- Define Constraints
- Define Bindings
- Define Extensions
- Define Slices
- Define Derived Profiles
- Define Logical Models

All these features will be described in more detail in the following sections.
2.5 Open a profile folder

When you want to start working in Forge, you will first need to open a profile folder (i.e. the working directory containing your profiles). When you select an item from your recent documents, Forge will automatically open its associated profile folder. In any other case click Open Profile Folder... and select your working directory.

Note that although Forge allows you to select any folder, it would be a very bad idea to select a general folder like "C:/Documents" as your profile folder. If you don’t have a specific folder for your project, please create one. For example, let’s open my profile folder called MyForgeProject. By default subdirectories are not included. Select this option to include all subdirectories of the selected folder. This may be convenient if you want to organize your profiles in separate folders, but still want to be able to edit them in the same project. On the other hand, if you have a subdirectory containing old versions of your profiles, it would be better to ignore the subdirectory.

Click Select Folder to open your folder. Now Forge will show you all profiles and extensions contained in your folder.
When there’s a lot of content in your folder, you may want to use one of the available filters to filter on structure type (e.g. StructureDefinition), category (e.g. profile or extension) or resource type (e.g. Patient).

You can also switch views. By default the **List** view is shown, showing all content in the folder and all its subdirectories (when these are included). The **Folders** view allows you to browse the subdirectories one by one as if you’re using your file explorer.

Select one of the structure definitions from the content list and choose **Open** to open it in Forge. Choose **New** to add new structure definitions to your project. Note that they will not be saved automatically. To add a newly created structure definition to your project, select **Save** and save it to your profile folder. Use the **Refresh** button to update the content list.

In the Session Explorer, you can choose **Open** to open another profile folder. Note that this will not automatically close your current profile folder. To close a profile folder, select the folder in the Session Explorer and select **Close**. Or select a single item in your project to close.

### 2.6 Create new Profiles

Forge enables you to create your own FHIR Profiles, based on one of the FHIR base resources. To open a base resource and start editing, you can select **New... > New Profile** (or Ctrl + N) from your profile folder.

This will open a window in which you can select one of the base Resources or DataTypes.

If you selected the base Resource you need, you can start to create your own Profile based on this Resource. The different options you can use in editing your Profile will be explained in the following sections.

### 2.7 Edit profiles

Within Forge you can differentiate between the five tabs which are shown in the screenshot below. This section describes these tabs in more detail.
2.7.1 Properties and Narrative

In Forge you can edit the properties of a Profile as well as the elements of the Profile. Properties of the Profile are, for example, the profile URL, name, and publishing date. These values can be edited under the ‘Properties’ tab. A profile’s narrative can be adjusted under the ‘Narrative’ tab. The narrative is a XHMTL fragment which is intended to contain a summary of the resource and is used to represent the content of the resource to a human. On default this is a Generated Narrative of the profile, meaning the content is entirely generated from the structured data in the profile.

2.7.2 Element Tree and Element Grid

Elements of the FHIR base Resource and extension can be edited to make your own profile. In Forge you can view and edit these elements in the Element Tree or in the Element Grid.

![Element Tree and Element Grid](image)

Via the Element Tree you can edit Element Properties, slice Elements, or add extensions.

2.7.3 XML viewer

The last tab is the XML viewer of the current profile. The XML viewer only shows the differential. A snapshot can be obtained by saving your work after the option ‘Save snapshot component’ is checked in the Options menu.

2.8 Define constraints

An important option for the editing of Element Properties is the option to add constraints. In Forge you can add one or more constraints to an element. This section shows how to place cardinality constraints, edit element types and valueset bindings. In addition, with Forge it is also possible to specify the Constraint Key, Requirements, Severity, Description, and XPath expression.
2.8.1 Cardinality

Adjusting the cardinality of an element might be the backbone in profiling FHIR. Forge offers the five most used cardinality options as a shortcut but it is also possible to enter the desired cardinality values directly in the ‘Element Properties’ as shown in the screenshot below. Note that profiles may only further restrict the base resource. For example, it is not allowed to make the maximum cardinality greater than the base Resource’s maximum cardinality. Forge shows a warning message when this happens.

![Cardinality Screenshot]

2.8.2 Element Type

Some elements in the FHIR specification contain multiple data types. For example, the element ‘value[x]’ in the Observation resource. In your profile, it is possible to constrain this to the desired data type(s). Forge allows you to place these constraints in the Element Properties by checking only the wanted datatypes. The screenshot below shows the unconstraint Type situation of the ‘value[x]’ element in the Observation resource. It is also possible to provide a custom datatype by entering the URI of the datatype profile after the checked Type. A Type can also be a reference to another Resource profile. Forge automatically gives the profile as a Type option if the profile is opened in the session explorer and the profile’s base Resource fits the element.

![Element Type Screenshot]

Observation.value[x] types  Observation.subject reference type

2.8.3 Valueset binding

Some elements can be constrained with a valueset. A valueset can be added or edited under ‘Valueset binding’ in the ‘Element Properties’. Here you can provide the valueset binding strength, description and the reference to the valueset. Note that you cannot relax the base Resource binding strength. The value must be equal to or stricter than the base value.

![Valueset Binding Screenshot]

2.8.4 New Feature - Inline constraints

FHIR allows you to constrain an element type by specifying a custom (external) datatype or resource profile. Forge has always allowed you to specify a profile url on an element type. But now, when you open a profile, Forge also tries to resolve the external type profile and merge it into your own profile.
Note: Forge does not (yet) dynamically resolve external profiles while you are editing an element type. For now, you have to manually (save and) reload the profile to enable to see and constrain (!) the expanded external type profiles.

This feature allows you to specify inline constraints that “walk into” elements defined by external type profiles.

This behavior also applies to Extension Definitions, as technically they are also external type profiles. Suppose you author a profile with an extension element. The profile extension element is mapped to a common Extension definition. Now when you open the profile in Forge 14.7, the application will try to resolve the target extension definition and merge it into your profile. Now you can define inline constraints on child elements of the extension in your profile to override constraints in the underlying extension definition.

2.8. Define constraints
2.9 Define extensions

Within Forge you can extend your profiles or elements with extensions created by yourself or others.

2.9.1 Extension registry

When using extensions, first consult the extension registry at http://hl7.org/fhir/extensibility-registry.html or Simplifier.net to find already defined extensions that may be suitable for your needs. For example, extending a Patient profile with the place of birth can be done with an already existing extension found in in the HL7 extension registry.

2.9.2 New extension

If you cannot find an already defined extension you can make your own. From your profile folder, go to New and click New Extension (or Ctrl + E) to create a new extension. Forge shows warning messages in the lower section of the your screen highlighting the need to provide context information for this new extension. Provide this information in Properties tab of the extension. Context Type indicates if the extension extends a Resource, Datatype,
Mapping or another Extension. Give the exact context by clicking the + symbol after Context. This brings you to a new screen where you can select the specific resource or datatype where the extension is allowed to be placed. If you extend on the resource level you can click Select Resource. If you extend an element first select that element and then click Select Element. Depending on where the extension may be used, you can add more context information. You can provide a canonical URL, name and other relevant information in the properties section as well.

2.9.3 Extension profile

A new extension starts with one element containing a Value\[X]. A Value\[X] can contain all datatypes, most likely this needs to be constrained to a more specific datatype. This can be done in the Element Properties tab by selecting the wanted datatype(s). An extension containing more elements is called a complex extension. Elements can be added and removed from the extension profile with the use of the Add and Remove buttons. Added elements will be placed a level lower than the selected element.

2.9.4 Add the extension to the profile

Note that to add extensions, they need to be available in your profile folder. So you will either need to create your own extension and save it to your profile folder or download an existing extension to your profile folder. Select the element in the Element Tree of your profile where you want to add the extension. Then, click the paperclip icon with the name Extend which is shown in the tool section above the element tree. Forge shows the extensions that are available in your profile folder.
Note that you can only select extensions that are compatible with your context (e.g. when extending an AdverseEvent resource you can only select an extension that has this resource as a context). By default, incompatible extensions are listed in grey, but you can also filter on compatible extensions by checking the check box.

### 2.10 Define slices

To further customize a base Resource you can slice available elements to capture a variety of the same element. This is particularly helpful when you want to capture multiple versions of an element of your base Resource. For instance multiple patient identifiers or multiple name parts. This allows you to define specifics such as CodeableConcepts for each part of the sliced object. This process entails slicing a structure element into sub items. The sub items that are created are then referred to as slices.

#### 2.10.1 Slicing in Forge

Select the element that you want to slice and click on the scissor icon. The element is now ‘Sliced’ and it is possible to add slices by clicking the scissor icon with a little plus. You can create as many slices as necessary by selecting the sliced element and clicking the Add Slice icon again. The added slices can be defined and constrained as any other element.

#### 2.10.2 Discriminator

After adding slices, Forge shows a warning message containing the message that there is no discriminator defined for the sliced elements. The discriminator is used to distinguish the sliced elements from one another. The discriminators in their sum should allow applications to easily determine which slice belong to each section. The discriminator information can be provided in the Element Properties of the sliced element under Slicing Details.
More information concerning slicing and discriminators can be found here: http://hl7.org/fhir/profiling.html.

### 2.10.3 New Feature - Type Slicing

Forge 14.7 provides improved support for so-called type slicing. This applies to so-called polymorphic “choice type” elements that support multiple data types. A choice type element has a name that ends with “[x]”, e.g. Observation.value[x]. Using type slicing, you can define constraints for specific element types. Per definition, the slicing discriminator of a type slice is equal to “@type”. Each slice constraint you define must have a unique type.

### 2.11 Derived Profiles

With the new release of Forge 14.7 you now have the ability to create a profile on top of another, existing profile, a.k.a. “derived” profile. This will enable you and your organization to benefit from the existing profiles and to further customize those profiles to your specific needs. For example, take an organization that would like to begin working with a national profile that is derived from a Core Resource. That organization would like to utilize the changes that were made to the Core Resource, the national profile, and then further customize that profile to reflect organization specific needs. With the new version of Forge you can now begin work directly on that national profile! This saves time and effort that you would spend recreating all the changes to the Core Resource to reflect the changes to the national profile and then further adding your organizational constraints.

It works like this; You have a Core Resource. These are data models that are created to fit most use cases (approximately 80% of all occurrences). A country takes that Core Resources and constrains it to fit the specific needs that reflect the situation in their country. This then becomes that countries version of that Core Resource, this is now our national profile. An organization in that country then realizes that they would like to use the national profile but make a few extra constraints to reflect the specific situation in their centers. The organization can now use that national profile.
and begin making changes to reflect their specific needs. This new organizational profile will have all the inherited changes from the national profile that were made to the original Core resource. This is what we refer to as a derived profile.

To do this in Forge you need to select the New Derived Profile command. Once you do you will be brought to your local drive to enable you to select your base profile. In the future we will have the ability to access your own and/or shared profiles, via your Simplifier account. For now, users have to visit http://simplifier.net/ to search and download relevant Core Resources.
Things to keep in mind are that you can only further constrain (or add extensions to) profiles. This means that each derived profile is more specific than its predecessor. Also important to point out is that the differential that you see for your derived profile will only show changes made to the profile which you started with. If the base profile as a Core Resource, then the differential will reflect all constraints with respect to that Core Resource. If the base profile is a derived profile, e.g. a national profile like in the example above, then the differential will reflect constraints on the national profile.

2.12 Logical Models

Forge also supports the authoring of Logical Models. A logical model represents an abstract data model that is not derived from one of the core FHIR resources or datatype.

Note: FHIR defines resources that allow you to define a mapping from a logical model to a set of FHIR resources. Forge does not support these mapping resources. This is of course very handy to facilitate the planning and for the eventual translation of elements from the logical model into new resources. To create Logical Models in Forge select the New Logical Model command. You can customize the meta properties for your model and add backbone elements and typed elements.

- A backbone element has no datatype. A backbone element may contain other child elements; either typed elements and/or other (nested) backbone elements.

- A typed element is mapped to a FHIR datatype or resource profile.
You cannot introduce new child elements of a typed element, because they are implicitly defined by the element datatype profile. You also have the ability to customize all backbone elements and sub elements to constrain their cardinality, specify their datatypes etc.

### 2.13 Contact

For questions or feedback on Forge, go to the Help menu in Forge and select ‘Feedback & Support’ or send an E-mail to forge@fire.ly.
Welcome to the Vonk server documentation

Vonk is Firely’s FHIR server and the successor to our Spark server. Vonk 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://vonk.fire.ly. This edition is free to use and intended for testing and educational purposes only.
On these pages we provide you with the documentation you need to get up and running with your own standard Vonk installation, as well as information on how to contact us when you have additional needs, such as a custom implementation or support contract.

3.1 Getting Started

If you want to start using the standard Vonk 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 Vonk on Docker section.

1. Download the Vonk binaries and the license file from Simplifier.net.
2. Extract the downloaded files to a location on your system, for example: C:\Vonk. 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 Vonk 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": "vonk-trial-license.json"
}
```

**Important:** the next step assumes you have a .Net Core environment installed. If not, please download and install .NET Core Runtime 2.2.x before you continue.

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

   Vonk will then run on port 4080 of the system.

7. If you want to check if Vonk is running correctly, open a browser and go to `localhost:4080`. You should see a homepage similar to this:

![Vonk FHIR Server](image)

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.
3.1.1 Configuration

The section Configuring the Vonk server explains how you can configure the Vonk server.

3.1.2 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 .\Vonk.Server.dll
```

3.2 Configuring the Vonk server

In this section we assume you have downloaded and installed the Vonk binaries, and have obtained a license file. If not, please see the previous section and follow the steps there first.

The steps you followed to get started will provide you with a basic Vonk 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 Vonk by adjusting the Vonk settings.

If you want to change the way Vonk logs its information, you can adjust the Log settings.

3.2.1 Vonk settings

Vonk 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 Vonk distribution directory, next to vonk.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 Vonk settings with Environment Variables.

Changes to the settings require a restart of Vonk.

Hierarchy of settings

Vonk reads its settings from these sources, in this order:

- **appsettings.default.json** Installed with Vonk, 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 Vonk distribution, it will not be overwritten by a next Vonk version.

- **environment variables** See Vonk settings with Environment Variables.

- **appsettings.instance.json** You can create this one to override settings for a specific instance of Vonk. It is not part of the Vonk 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 Vonk settings this is relevant for e.g. Validation.AllowedProfiles.

Even worse, if on a lower level the array has more items, you will still inherit those extra items. This is particularly a problem in the PipelineOptions. In that case, simply remove the whole array from appsettings.default.json (and repeat that when you install a new version of Vonk).

Settings after first install

After you installed Vonk (see Getting Started), either:

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

Adjust the new appsettings.json to your liking using the explanation below.

When running Vonk 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 Vonk, you can:

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

In both cases, check the Release notes Vonk 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.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 Vonk) accepts double slashes just fine:

```
"Administration": {
   "Repository": "SQLite", //Memory / SQL / MongoDb
   "SqlDbOptions": {
      "ConnectionString": "connectionstring to your Vonk Admin SQL Server database (SQL2012 or newer); Set MultipleActiveResultSets=True",
      "SchemaName": "vonkadmin",
      "AutoUpdateDatabase": true
      //"AutoUpdateDatabase": true
      //"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.

3.2. Configuring the Vonk server
Log of your configuration

Because the hierarchy of settings can be overwhelming, Vonk 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.

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 Vonk Admin SQL Server database (SQL2012 or newer); Set MultipleActiveResultSets=True",
        "SchemaName": "vonkadmin",
        "AutoUpdateDatabase": true
    },
    "SQLiteDbOptions": {
        "ConnectionString": "Data Source=./data/vonkadmin.db",
        "AutoUpdateDatabase": true
    },
    "Security": {
        "AllowedNetworks": [ "::1" ], // i.e.: ["127.0.0.1", ":1" (ipv6 localhost), "10.150.0/24", "10.5.3.0/24", "31.161.91.98"]
    }
},
```

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

License

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

The Getting Started explains how to obtain a licensefile for Vonk. 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 feature releases.
- WriteRequestInfoFileInterval sets the time interval (in minutes) to write aggregate information about processed requests to the RequestInfoFile.
Repository

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 Vonk SQL Server database (SQL2012 or newer); Set MultipleActiveResultSets=True",
   "SchemaName": "vonk",
   "AutoUpdateDatabase": true
   //"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.

**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.

**Validation**

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

Refer to *Validating incoming resources*.

**Search and History**

"BundleOptions": {
   "DefaultCount": 10,
   "MaxCount": 50
},

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 Vonk from being overloaded.
• DefaultCount should be less than or equal to MaxCount

Batch and transaction

```json
"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 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.

Protect against large input

```json
"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)://<vonk-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.

SearchParams and other Conformance Resources

```json
"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 **Controlling the Conformance Resources** and **Using Custom Search Parameters**.
Restrict supported resources and SearchParameters

```
"SupportedModel": {
  "RestrictToResources": [ "Patient", "Observation" ]
  "RestrictToSearchParameters": ["Patient.active", "Observation.patient"]
  "RestrictToCompartments": ["Patient"]
},
```

By default, Vonk supports all ResourceType, 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` cannot be disabled
- the Administration API requires support for the `url` SearchParameter on the conformance resourcetypes

Enable or disable interactions

By default, the value `SupportedInteractions` contains all the interactions that are implemented in Vonk. 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"
},
```

Subscriptions

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

See `Subscriptions`.

FHIR Capabilities

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

See `Create, read, update, delete`. 
Configuring the Vonk Pipeline

You can add your own components to the Vonk pipeline, or control which of the standard Vonk components are used for your Vonk server, by changing the `PipelineOptions`.

```json
"PipelineOptions": {
  "PluginDirectory": "./plugins",
  "Branches": [
    {
      "Path": "/",
      "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",
        // etc.
      ],
      "Exclude": []
    },
    {
      "Path": "/administration",
      "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",
        // etc.
      ],
      "Exclude": ["Vonk.Core.Operations"
      ]
    }
  ]
}
```

Please note the warning on merging arrays in *Hierarchy of settings*.

See *Vonk FHIR Components* for more information and an example custom component.

### 3.2.2 Vonk settings with Environment Variables

#### Environment Variables for appsettings

All the settings in *Vonk settings* can be overridden by environment variables on your OS. This can be useful if you want to deploy Vonk to several machines, each having their own settings for certain options. For *Using Vonk 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"
```

evironment variable:
To access an embedded value, using the ‘:’ separator:

appsettings.json:

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

evironment 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
```

**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",
```

Chapter 3. Welcome to the Vonk server documentation
environment variable:

VONKLOG_Serilog:MinimumLevel:Override:Vonk.Configuration=Information

### 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:
  ```powershell
  $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).

### 3.2.3 Configure the Administration API

This configuration is part of Vonk settings.

```json
"Administration": {
  "Repository": "SQLite", // Memory / SQL / MongoDb
  "MongoDbOptions": {
    "ConnectionString": "mongodb://localhost/vonkadmin",
    "EntryCollection": "vonkentries"
  },
  "SqlDbOptions": {
    "ConnectionString": "connectionstring to your Vonk Admin SQL Server database (SQL2012 or newer); Set MultipleActiveResultSets=True",
    "SchemaName": "vonkadmin",
    "AutoUpdateDatabase": true
  },
  "SQLiteDbOptions": {
    "ConnectionString": "Data Source=./data/vonkadmin.db",
    "AutoUpdateDatabase": true
  },
  "Security": {
    "AllowedNetworks": [ "::1" ], // i.e.: [ "127.0.0.1", "::1" (ipv6 localhost), "10.150.0.24", "10.5.3.0/24", "31.161.91.98"]
    "OperationsToBeSecured": [ "reindex", "reset", "preload" ]
  }
},
```

### 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.

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 a set of resources)
     - Read / write on the Conformance resources of type:
       - StructureDefinition
       - SearchParameter
       - ValueSet
       - CodeSystem
       - CompartmentDefinition
     - Subscription: all the read and write on /Subscription (see Subscriptions)

   • The AllowedNetworks have to be valid IP addresses, either IPv4 or IPv6, and masks are allowed.

3.2.4 Controlling the Conformance Resources

Vonk 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 operations.

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

1. With regular FHIR interactions (create, update, delete) on the Vonk 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 (<vonk-endpoint>/administration)

Attention: Please be aware that Conformance Resources have to have a unique canonical url, in their url element. Vonk 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.

---

### Import of Conformance Resources

The import process of conformance resources runs on every startup of Vonk, 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 Vonk process has write permission on the ImportedDirectory.

The process follows these steps:

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 Vonk 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.

---

### Default Conformance Resources

Vonk 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.
Load Conformance Resources from disk

Vonk can read SearchParameter and CompartmentDefinition resources from a directory on disk at startup. The AdministrationImportOptions in the Vonk 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.

**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.

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 Vonk read those. You configure this in the Vonk 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>"
    }]
},
```

**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.

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>

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 Vonk to actually load the profiles from the source(s) with a separate command:

```
POST http(s)://<vonk-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.

Manage Conformance Resources with the Administration API

The Vonk Administration API has a FHIR interface included, on the https://<vonk-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. https://<vonk-endpoint>/administration/StructureDefinition), Vonk will return statuscode 403.

Note: You can also do the same interactions on the same resourcetypes on the normal Vonk FHIR interface https://<vonk-endpoint>. This will only result in storing, updating or deleting the resource. But it will not have any effect on the way Vonk operates.

Example

To add a StructureDefinition to Vonk

```
POST <vonk-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 <vonk-endpoint>/administration/StructureDefinition/MyPatient
```

### 3.2.5 Validating incoming resources

You can have Vonk 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
  ]
},
```

**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.)

**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.
**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, Vonk 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, Vonk 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. Vonk 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.

**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"

### 3.2.6 Using the In-Memory storage

- Navigate to your Vonk working directory
- In a text editor open appsettings.json to find the **Repository** setting:

  ```json
  "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:

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

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

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

E.g.:
Warning: Using the In-Memory storage for the administration layer will cause Vonk to load the specification files on each startup. This takes several minutes, and Vonk 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 Vonk Administration for more information.

### 3.2.7 Using MongoDB

We assume you already have MongoDB installed. If not, please refer to the MongoDB download pages.

- Navigate to your Vonk working directory
- In a text editor open appsettings.json to find the Repository setting:

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

- Change the setting to `MongoDB`
- If you have your own database in MongoDB already, change the `MongoDbOptions` to reflect your settings:

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

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

  ```json
  "PipelineOptions" :
  {
    "Branches" : [
      "/" : {
        //...
      }
    }
  }
  ```

- 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.
Using MongoDB for the Administration API database

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

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

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

```json
"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 Vonk Administration:

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

3.2.8 Using SQL server

There are two ways to create the Vonk database on a SQL Server instance: Have Vonk create it for you entirely or create an empty database and users yourself and have Vonk 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.
- In a text editor open appsettings.json to find the Repository setting:

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

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

```json
"SqlDbOptions": {
  "ConnectionString": "connectionstring to your Vonk SQL Server database (SQL2012 or newer); Set MultipleActiveResultSets=True",
  "SchemaName": "vonk",
  "AutoUpdateDatabase": true
}
```
(continues on next page)
• Find the section called PipelineOptions. Make sure it contains the SQL repository in the root path for Vonk Data:

```json
"PipelineOptions": {
    "Branches": [
        "/": {
            "Include": [
                "Vonk.Repository.Sql.SqlVonkConfiguration"
            ]
        }
    ]
}
```

• 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 Vonk Administration, you can still use SQL Server for Vonk Administration as well:

```json
"Administration": {
    "Repository": "SQL",
    "SqlDbOptions": {
        "ConnectionString": "Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=VonkStu3;Data Source=Server\Instance;MultipleActiveResultSets=true",
        "SchemaName": "vonk",
        "AutoUpdateDatabase": true
    }
},
//...
"PipelineOptions": {
    "Branches": [
        "/administration": {
            "Include": [
                "Vonk.Repository.Sql.SqlAdministrationConfiguration"
            ]
        }
    ]
}
```

### Have Vonk 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:
• Set the SqlDbOptions for the Vonk database as follows (the values are example values for connecting with your own Windows login):

```json
"SqlDbOptions": {
    "ConnectionString": "Integrated Security=SSPI;Persist Security Info=False;
    Initial Catalog=VonkStu3;Data Source=Server\Instance;
    MultipleActiveResultSets=true",
    "SchemaName": "vonk",
    "AutoUpdateDatabase": true
},
```

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

```json
"Administration": {
    "Repository": "SQL",
    "SqlDbOptions": {
        "ConnectionString": "Integrated Security=SSPI;Persist Security Info=False;
        Initial Catalog=VonkStu3;Data Source=Server\Instance;
        MultipleActiveResultSets=true",
        "SchemaName": "vonk",
        "AutoUpdateDatabase": true
    }
}
```

• You don’t need to set AutoUpdateConnectionString since the ConnectionString will already have enough permissions.

• Start Vonk. It will display in its log that it applied pending migrations. After that the database is created and set up with the correct schema.

Create a database and users by script, and have Vonk create the schema

• Log into SQL Server as the Administrator user.
• From the working directory open `data\01-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 Vonk database and the Administration API database.
• From the working directory open `data\02-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 Vonk 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 Vonk database as follows:

```json
"SqlDbOptions": {
    "ConnectionString": "User Id=<dbUserName>;Password=<dbPassword>;Initial
    Catalog=<dbName>;Data Source=server\instance;MultipleActiveResultSets=True",
}
```
If you have set up a different user for running the AutoUpdateDatabase feature, you can provide that:

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

Set the SqlDbOptions under Administration for the Administration database likewise:

```json
"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:

```json
"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",
  }
}
```

**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 Vonk database and the Administration database)
- To read/write resources:
To execute the ResetDb feature:

- **db_ddladmin** only on the normal Vonk database for the user in the SqlDbOptions connectionString. (no extra permissions are required for the user on the Administration database).

### 3.2.9 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, Vonk FHIR Server in this case.

For more background on SQLite please refer to the SQLite documentation.

SQLite is the default configuration of Vonk FHIR Server. For the Administration database there is little reason to change this. For the actual runtime data, (the ‘Vonk 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 Vonk Data and Vonk Administration.

**Prefer SQLite for Vonk Administration**

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

Vonk 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 Vonk Administration. And you can download the Vonk binaries and run them without any further configuration.

- Flexible on updates: Many of the features that we will add to Vonk 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 Vonk 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 Vonk FHIR Facade, the facade will not provide support for hosting conformance resources. With Vonk Administration on SQLite the facade has its own storage and you can use Vonk 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.

**Settings for using SQLite for Vonk Data**

- In a text editor open `appsettings.json` to find the `Repository` setting and set it to `SQLite` if it not already set to that:

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

- Find the section called `SQLiteDbOptions`. It has these values by default:
"SQLiteDbOptions": {
    "ConnectionString": "Data Source=./data/vonkdata.db",
    "AutoUpdateDatabase": true
},

Vonk 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"
                //...
            ],
            //...
        },
        //...
    ]
}

**Settings for using SQLite for Vonk 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"
    }
}

Vonk 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"
                //...
            ]
        }
    ]
}
Administration import history in SQLite

When Vonk imports Conformance resources, it keeps record of what is 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.

3.2.10 Using Microsoft Azure CosmosDB

You can connect Vonk 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 Vonk 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 Vonk working directory
5. In a text editor open appsettings.json to find the Repository setting:

   "Repository": "Sqlite",

6. Change the setting to CosmosDb
7. 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 databasesname 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 databasesname after the portnumber, like this:

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

8. If your CosmosDB account does not have a database or collection by this name, Vonk will create it for you.
9. You can set SimulateTransactions to “true” if you want to experiment with FHIR transactions. Vonk 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.

3.2. Configuring the Vonk server
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 Vonk 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.

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 Vonk:
   - Using the :not modifier
   - Using :missing=true

3.2.11 Configure http and https

You can enable http and/or https and adjust the port numbers for them in Vonk settings.

Changing the port number

By default Vonk will run on port 4080 of your system. You can change the port setting in the appsettings.json file that is part of the Vonk distribution:

- Navigate to your Vonk working directory
- In a text editor open appsettings.json to find this setting:

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

- Change the number to the port number you want

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 Vonk files.
- Open appsettings.json in a text editor and find these settings:


```json
"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 Vonk will no longer be available through unsecured http.

### 3.2.12 Log settings

Vonk 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. Vonk comes with default settings in logsettings.default.json. You can adjust the way Vonk logs its information by overriding these default settings in logsettings.json. You need to create this logsettings.json yourself.

Alternatively you can control Log settings with Environment Variables.

**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, Vonk uses Error as the minimum level of recording information.

To change the level of logging, follow these steps:

• Navigate to your Vonk working directory.
• Open logsettings.json in a text editor and find this settings:
"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
- Microsoft to log events from the Microsoft libraries
- 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"  
  }  
},

**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 Vonk you can use `Console, ColoredConsole, RollingFile and Seq.`
RollingFile

For the RollingFile sink, you can specify the location of the log files with the pathFormat argument. Please include the {Date}, {Hour} or {HalfHour} placeholder, so Serilog can add date and time information to your filename.

- Navigate to your Vonk working directory.
- Open logsettings.json in a text editor and find the WriteTo setting:

```json
"WriteTo": [
  { "Name": "ColoredConsole" },
  {
    "Name": "RollingFile",
    "Args": { "pathFormat": "c:/temp/vonk-{Date}.log" }
  },

  { "Name": "RollingFile",
    "Args": { "pathFormat": "c:/logfiles/vonk-{Hour}.log" }
  }
],
```

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

```json
{ "Name": "RollingFile",
  "Args": { "pathFormat": "c:/logfiles/vonk-{Hour}.log", "retainedFileCountLimit": "24" }
},
```

You can also limit the size of the log file, which is 1GB by default, with the fileSizeLimitBytes option. If you set the value for this argument to null, you remove the limit. Serilog retains a maximum of 31 files. This value can also be removed or changed, by providing the retainedFileCountLimit argument:

```json
{ "Name": "RollingFile",
  "Args": { "pathFormat": "c:/logfiles/vonk-{Hour}.log", "retainedFileCountLimit": "24" }
},
```

Seq

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

- Navigate to your Vonk working directory.
- Open logsettings.json in a text editor and find the Seq sink under the WriteTo setting:

```json
"WriteTo": [
  { "Name": "RollingFile",
    "Args": { "pathFormat": "c:/logfiles/vonk-{Hour}.log", "retainedFileCountLimit": "24" }
  },

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

- Change serverUrl to the URL of your Seq server

All sinks

For all sinks, you can set a restriction on the minimum log event level. This is not an override of the MinimumLevel setting we discussed earlier, but rather a filter on the events that are logged. With this extra sink argument, you can for example log only a small portion of the events to the console, but all of them to the log file.

3.2. Configuring the Vonk server
• Navigate to your Vonk working directory.
• Open logsettings.json in a text editor and find the sink you want to set the filter for under the WriteTo setting.
• Add the restrictedToMinimumLevel argument to the Args of the sink:

```
"WriteTo": [
    {
        "Name": "ColoredConsole",
        "Args": { "restrictedToMinimumLevel": "Warning" }
    },
],
```

### Database details

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

```
"MinimumLevel": {
    "Default": "Error",
    "Override": {
        "Vonk.Repository.Sql": "Verbose",
        "Vonk.Repository.MongoDb": "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 All sinks above.

### Azure Application Insights

Vonk can also log to Azure Application Insights. 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.
3.3 Vonk deployment options

Vonk can be deployed as:

- Binaries, see Getting Started
- As Docker image, see Using Vonk on Docker
- With Yellow Button - Vonk 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.

3.3.1 Using Vonk on Docker

We have created a Docker image for Vonk, 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.

Getting started

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

1. Open your favourite command line tool and execute this command:
   ```bash
   > docker pull simplifier/vonk
   ```

2. Go to the Simplifier website, login and download your evaluation license.

3. Create a working directory for Vonk and place the license file there.

Running a Docker Vonk in SQLite mode

The easiest and the default way to run a Docker Vonk container is to run Vonk 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:

<table>
<thead>
<tr>
<th>REPOSITORY</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>CREATED</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>simplifier/vonk</td>
<td>latest</td>
<td>50e1d33e32b7</td>
<td>8 minutes ago</td>
<td>323MB</td>
</tr>
</tbody>
</table>

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

- in cmd.exe:  
  ```bash
  docker run -d -p 8080:4080 --name vonk.server -v %CD%:/app/license -e "VONK_License:LicenseFile=./license/vonk-trial-license.json" simplifier/vonk
  ```

- in Powershell:  
  ```bash
  docker run -d -p 8080:4080 --name vonk.server -v ${PWD}:/app/license -e "VONK_License:LicenseFile=./license/vonk-trial-license.json" simplifier/vonk
  ```

If your license file has a different name, use that name instead of vonk-trial-license in the command above.
**Important:** It looks like the command wants to retrieve your license file from a subdirectory called `license`. This is a result of docker copying your file before spinning the image. You should **not** create the subdirectory. Just keep the license file in the root of your working directory.

This will spin up a Vonk 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 current directory (where the license file resides) from the host to the container. Also it passes an environment variable `VONK_License:LicenseFile` to the container with the switch `-e`. In this example the license file is called `vonk-trial-license.json`. At last 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`

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

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

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`

### Spinning with a docker-compose file

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

```yaml
version: '3'
services:
  vonk-web:
    image: simplifier/vonk
    ports:
      - "8080:4080"
    environment:
      - VONK_Repository=SQLite
      - VONK_Administration:Repository=SQLite
      - VONK_License:LicenseFile=./license/vonk-trial-license.json
    volumes:
      - ./:/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 `vonk-trial-license` in the text above - but make sure to keep `./license` as that maps to a Docker volume inside the container.

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.

**Running Docker with a SQL Server container**

Vonk can use also other repositories than Memory, for example SQL Server. This section describes how to spin up a Vonk 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.

```yaml
version: '3'

services:
  vonk-web:
    image: simplifier/vonk
    ports:
      - "8080:4080"
    depends_on:
      - vonk-server-db
    environment:
      - VONK_Repository=SQL
      - VONK_SqlDbOptions:ConnectionString=Initial Catalog=VonkStu3;Data Source=vonk-sqlserver-db,1433;User ID=vonk;Password=Tester01
      - VONK_SqlDbOptions:SchemaName=vonk
      - VONK_SqlDbOptions:AutoUpdateDatabase=true
      - VONK_SqlDbOptions:AutoUpdateConnectionString=Initial Catalog=VonkStu3;Data Source=vonk-sqlserver-db,1433;User ID=sa;Password=SQLServerStrong(!)Password*
      - VONK_Administration:Repository=SQL
      - VONK_Administration:SqlDbOptions:ConnectionString=Initial Catalog=VonkAdmin;
      - VONK_Administration:SqlDbOptions:SchemaName=VonkAdmin
      - VONK_Administration:SqlDbOptions:AutoUpdateDatabase=true
      - VONK_Administration:SqlDbOptions:AutoUpdateConnectionString=Initial Catalog=VonkAdmin;Data Source=vonk-sqlserver-db,1433;User ID=sa;Password=SQLServerStrong(!)Password*
      - VONK_License:LicenseFile=./license/vonk-trial-license.json
    volumes:
      - .:/app/license
      - script-volume:/app/data

  vonk-sqlserver-db:
    image: microsoft/mssql-server-linux
    ports:
      - "1433:1433"
    environment:
      - ACCEPT_EULA=Y
      - SA_PASSWORD=SQLServerStrong(!)Password*
      - dbName=VonkStu3
      - dbPath=/var/opt/mssql/data/
      - AdminDbName=VonkAdmin
      - AdminDbUsername=vonk
```

(continues on next page)
Save the text above to a file in your working directory with the name `docker-compose.mssqlserver.yaml`. Make sure your Vonk license file is named `vonk-trial-license.json` and is residing in your working directory (see Getting started on how to obtain the license). If your license file has a different name, use that name instead of `vonk-trial-license` in the text above.

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

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

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

### Running Docker with a SQL Server on host

Another possibility is to run a Vonk 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.

To run the Vonk container we will use the following docker-compose file:
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 line 9 the connection string to the database server is stated. Change the Data Source to your database server. In this example we are using a named instance `sql2016` on the host `my_host`.
- Also change the User ID and Password on line 9 to your credentials.
- Furthermore we have to tell Docker which IP address the host uses. This is done on line 17. 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 Vonk license file is named `vonk-trial-license.json` and is residing in your working directory (see Getting started on how to obtain the license). Or use the name of your license file instead of `vonk-trial-license` in the text above.

You can run the Vonk container as follows: `> docker-compose -f docker-compose.mssqlserver_host.yaml up -d`

We have to create the Vonk database on the host. For this we need two SQL scripts which are located in the Vonk container. Perform the following commands in the working directory to copy the scripts and execute them on your local SQL server:

```
> docker cp vonk_vonk-web_1:/app/data ./scripts
> cd scripts
> sqlcmd -S my_host\sql2016 -d master -v dbName = VonkSTU3 dbPath= "C:\Program Files\Microsoft SQL Server\MSSQL13.SQLO2016\MSSQL\DATA" -i 01-CreateDatabaseAndSchema.sql
> sqlcmd -S my_host\sql2016 -d master -v dbName = VonkSTU3 dbUsername = vonk -v dbPassword = Tester01 -i 02-CreateDBUser.sql
```

You might want to change the `dbPath` and provide the `sqlcmd` user (SA) credentials with the parameters `--U` and `--P`.

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

---

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

---

### Run Docker with a MongoDB container

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

```
version: '3'
```
3.3.2 Vonk deployment on Azure Web App Service

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

Getting started

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

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

Deployment

1. Go to Azure (https://portal.azure.com) and create a web app:
2. Choose a name for the webapp, we will use the placeholder <webapp>. Fill in an existing resource group or
create a new one and select Windows for the operation system (OS):

3. Add the trial license file (vonk-trial-license.json) to the vonk_distribution.zip by dragging the license file into the zipfile.

4. Open a webbrowser, navigate to https://<webapp>.scm.azurewebsites.net/ZipDeploy and drag vonk_distribution.zip into the browser window. This will install the Vonk server as a Web App in Azure. In our example the url is https://<webapp>.scm.azurewebsites.net/ZipDeploy This method of deployment does not work in Internet Explorer. It does work in Firefox, Chrome and Edge.

5. Open a browser and go to the site https://<webapp>.azurewebsites.net/. This will show the Vonk home page.

**Change database**

In this example Vonk 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:
3.3.3 Yellow Button - Vonk for your Simplifier project

Yellow Button is an easy way to run an instance of Vonk FHIR 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.

Prerequisites

Yellow Button provides you with a PowerShell script to start Vonk FHIR 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.

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.

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`.

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.

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`.
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:

```powershell
> 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`:

```powershell
> 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-VonkDockerServer\start-vonk-server.ps1?
```

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

As soon as Vonk is ready, the script will finish and it will open your browser on the endpoint of Vonk: `http://127.0.0.1:8080/`, showing the landing page. For further use of the Vonk 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.

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
```

Solution: This may happen at subsequent starts of the Vonk 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

```
→external connectivity on endpoint ...
```

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 Vonk 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 vonk@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.

Using the server

When your Vonk 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:

```
```

Please note that any Conformance resources that influence the behaviour of Vonk – such as the Validation – must be managed on the Administration API and not the regular FHIR endpoint. See Vonk 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>
```

Your project in progress

You created the Vonk FHIR 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. Vonk is connected to the Simplifier project. This means that you can update the conformance resource in Vonk from the contents of the project by:
- invoking importResources:
  
  ```
  POST http://127.0.0.1:8080/administration/importResources
  ```

- or restarting Vonk:
  
  ```
  > ./stop-vonk-server.ps1
  > ./start-vonk-server.ps1
  ```

Further steps

Yellow Button is an easy way to get started with Vonk FHIR Server. But there are many other deployment options for the server. Besides that you can add your own plugins with Vonk FHIR Components, or build a Vonk FHIR Facade with Vonk.
3.3.4 Deploy Vonk on a reverse proxy

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.

With IIS

A common option on Windows is using IIS. For a comparison of IIS and Kestrel features at the moment of this writing you can check https://stackify.com/kestrel-web-server-asp-net-core-kestrel-vs-iis/?utm_source=DNK-224416. For instructions on how to deploy Vonk on IIS see Deploy Vonk on IIS

With Nginx

A popular open source alternative is Nginx. For instruction on how to deploy Vonk on Nginx see Deploy Vonk on Nginx on Linux

Deploy Vonk 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. 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 Vonk using a setting in web.config, see Configuration.

4. 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).

5. Install the .NET Core Windows Server 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 Vonk (see Getting Started), or create your own solution by building a facade. If you are deploying a Vonk facade, 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.
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 Basic Settings... from the popup menu.
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 Vonk, so you can perform PUT and DELETE interactions, add this to the `web.config` file:

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

- You can configure the Vonk server using the `appsettings.json` file (see Configuring the Vonk server).

- If you are building a Vonk 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.

```csharp
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:
Deploy Vonk 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 Vonk server

Download the binaries for Vonk (see Getting Started), open a terminal console and start the Vonk 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:

```plaintext
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/ngx_http_core_module.html#location
    location / {
        # This should match the location where you deployed the Vonk 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.
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Now you can run the Vonk server.

Configuration

- To configure the Vonk server, you can use the appsettings.json file (see Configuring the Vonk 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.

3.3.5 Set up an Identity Provider

About Identity Providers and Vonk

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.
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](https://github.com/DominickT/IdentityServer4) 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 Vonk 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*.

**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
   You should see a page like this.

   ![IdentityServer4](https://example.com/identityserver4.png)

   Welcome to IdentityServer4

   IdentityServer publishes a discovery document where you can find metadata and links to all the endpoints, key material, etc.
   Click [here](https://example.com) to manage your stored grants.
   Here are links to the [source code repository](https://example.com) and ready to use samples.

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:5000/.well-known/openid-configuration`. You can see all the available scopes in this document.
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.

3.3.6 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 Vonk 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:

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. Click Get New Access Token, and in the popup window fill in the blanks:
10. You can alter the values in ‘Scope’ to get other claims in the token.

11. Click Request Token and you’ll be presented with the login screen of IdentityServer:

<table>
<thead>
<tr>
<th>GET NEW ACCESS TOKEN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token Name</strong></td>
</tr>
<tr>
<td><strong>Grant Type</strong></td>
</tr>
<tr>
<td><strong>Callback URL</strong></td>
</tr>
<tr>
<td><strong>Auth URL</strong></td>
</tr>
<tr>
<td><strong>Access Token URL</strong></td>
</tr>
<tr>
<td><strong>Client ID</strong></td>
</tr>
<tr>
<td><strong>Client Secret</strong></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
</tr>
<tr>
<td><strong>State</strong></td>
</tr>
<tr>
<td><strong>Client Authentication</strong></td>
</tr>
</tbody>
</table>

3.3. Vonk deployment options
12. Log in as Bob or Alice and you'll be presented with the grant screen of IdentityServer. It will ask you whether Postman may have the claims you requested in the ‘Scope’ field.
13. Click ‘Allow’ and you return to Postman with the newly retrieved token:

14. You can copy the value of the access token and paste it into JWT.io. It will show you the contents of the token.

15. Scroll down and click ‘Use Token’:

3.3. Vonk deployment options
16. The token will be added as Authorization header to the request.

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.

### 3.3.7 Performance of Vonk FHIR Server

#### About Performance

What is the performance of Vonk? 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.

#### Performance variables

**Vonk Configuration**

Vonk 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 unitests. 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 Vonk, 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. Vonk 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. Vonk manages the tables it needs, and the indexes on top of it are optimized for the way Vonk queries them.

4. MongoDB: Performance tuning of MongoDB is, as well, a topic on its own. Vonk manages the collections it needs, and the indexes on top of it are optimized for the way Vonk queries them. MongoDB is used in our own performance tests, see below.

Prevalidation

By default, a resource that is sent to Vonk 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 Vonk for storage, Vonk indexes the resource for all the search parameters that are applicable to the resource. The more search parameters are known to Vonk, 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 Controlling the Conformance Resources and Restrict supported resources and SearchParameters.

Pipeline

Vonk is made up of a pipeline of components. You can leave out any component that you don’t need - so if you don’t need conditional processing (create, update, delete), just exclude them from the pipeline. Excluded components are not loaded and thus never executed - see Configuring the Vonk Pipeline.

Platform

Vonk 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 Vonk is deployed, the way the database is deployed is an important factor. Vonk needs a very low latency connection between the Vonk process(es) and the database server. If you have configured Azure Application Insights, the calls to the database are recorded as separate dependencies so you can check whether this may be a bottleneck.

Vonk 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).

Vonk 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 Vonk.

Usage patterns

How will Vonk 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.

3.3. Vonk deployment options
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 Vonk is stateless, this has little influence. The total request rate is what counts.

Testing performance yourself

Because of all the variables mentioned above the best way to find out whether Vonk’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 Vonk 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 Vonk 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 Vonk, and a separate one for generating the requests. Testing Vonk on the same machine as you’re generating requests from will make Vonk 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 Vonk (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.

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.

Vonk 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 components 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 Vonk 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

3.3. Vonk deployment options
3. Page through all the CarePlan resources: GET \(\text{url}/\text{CarePlan?\_count=10}\), and follow next links.
4. Page through 1/5 of the Patient resources and delete them: DELETE \(\text{url}/\text{Patient/\{id\}}\)
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 Vonk a delete is essentially an update, since all old versions are retained.

3.4 Features

Vonk FHIR Server offers many features as defined in the FHIR Specification and beyond.

3.4.1 FHIR RESTful API

Vonk supports most of the features in the FHIR RESTful API.

Create, read, update, delete

These four operations to manage the contents of the Vonk FHIR Server, commonly referenced by the acronym CRUD, are implemented as per the specification. This includes version-read and the conditional variations. Only a few limitations apply.

Vonk 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.

Vonk 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.

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

**Limitations on CRUD**

1. `_summary` is not yet supported.

2. 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.

**Versioning**

Vonk keeps a full version history of every resource, including the resources on the Vonk Administration API.

**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 Vonk also allows Using 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.

**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. :recurse modifier on _include and _revinclude
11. * wildcard on _include and _revinclude

Furthermore:
1. _sort is only implemented for the parameter _lastUpdated in order to support History.
2. Paging is supported, but it is not isolated from intermediate changes to resources.

**History**

History is supported as described in the specification, on the system, type and instance level. The _since and _count parameters are also supported.

**Limitations on history**

1. _at parameter is not yet supported.
2. Paging is supported, but it is not isolated from intermediate changes to resources.

**Batch**

Batch is fully supported. Although the specification does not allow references between resources in a Batch, Vonk can handle them.

You can limit the number of entries accepted in a single Transaction. See *Batch and transaction*.

**Transaction**

Transactions are supported, with a single limitation:

1. Of the three storage implementations, only SQL Server truly supports 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.

You can limit the number of entries accepted in a single Transaction. See *Batch and transaction*.

**Capabilities**

On the Capabilities interaction (<vonk-endpoint>/metadata) Vonk 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.
Not supported interactions

These interactions are not yet supported by Vonk:

1. patch

3.4.2 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

Concepts

This explanation of access control and SMART in Vonk requires basic understanding of the architecture of Vonk, 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.

Identification and Authentication

Vonk 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 to authenticate, and receives an OAuth2 token back. Then, the application can do an http request to Vonk to send or receive resource(s), and provide the OAuth2 token in the http Authentication header, thereby acting on behalf of the user. Vonk can then read the OAuth2 token and validate it with the OAuth2 provider. This functionality is not FHIR specific.
Authorization

Authorization in Vonk 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 Vonk:

- **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 Vonk 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.

The assignment of these claims to users, systems or groups is managed in the OAuth2 provider and not in Vonk.

Access Control Engine

The Access Control Engine in Vonk evaluates two types of authorization:

1. **Type-Access**: Is the user allowed to read or write resource(s) of a specific resource type?
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 Vonk 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.

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 Vonk. 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.

Other forms of Authorization

In Access Control in Facade and Components you can find the interfaces relevant to authorization in Vonk. If your environment requires other authorization information than the standard SMART on FHIR claims, you can create your own implementations of these interfaces. All the standard interaction middleware components of Vonk can then use that implementation to enforce access control. Providing your own implementations is only possible with Vonk FHIR Facade or Components.
Configuration

You will need to add the Smart component to the Vonk pipeline. See Vonk FHIR Components for more information. In appsettings.json, locate the pipeline configuration in the PipelineOptions section, or copy that section from appsettings.default.json:

```
"PipelineOptions": {
   "PluginDirectory": "/plugins",
   "Branches": [
      {
         "Path": "/",
         "Include": [
            "Vonk.Core",
            "Vonk.Fhir.R3",
            ...
         ]
      }
   ]

Add Vonk.Smart to the list of included components. When you restart Vonk, 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 appsettings.json:

```
"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
      },
      {
         "FilterType": "Location", //Filter on a Location compartment if a 'location' launch scope is in the auth token
         "FilterArgument": "identifier=#location#" //... for the Location that has an identifier matching the value of that 'location' launch scope
      }
   ],
   "Authority": "url-to-your-identity-provider",
   "Audience": "name-of-your-fhir-server" //Default 'vonk'
   "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’), Vonk 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 *Compartments* 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. See *Set up an Identity Provider* for more background.

- **Audience**: Defines the name of this Vonk instance as it is known to the Identity Provider. Default is ‘vonk’.

- **RequireHttpsToProvider**: Token exchange with an Identity Provider 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.

### 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 Vonk is functionally not limited to these five, the specification does not allow you to define your own. Vonk will use a CompartmentDefinition if:

- the CompartmentDefinition is known to Vonk, see *Controlling the Conformance Resources* for options to provide them.
- the OAuth2 Token contains a claim with the same name as the CompartmentDefinition.code (but it may be lowercase).

So the launch contexts mentioned in SMART on FHIR – ‘patient’ and ‘encounter’ – map to the CompartmentDefinitions for Patient and Encounter. For the launch context ‘location’, the specification has no matching CompartmentDefinition.

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 Vonk:

- 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 Identity Provider. Let’s assume it does know one of the Identifiers of a Patient. The Filters in the *Configuration* allow you to configure Vonk 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 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 Vonk, 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 Vonk. You can simply configure the Filters according to the business rules in your organization.

**Tokens**

A client application that wants to access data in Vonk on behalf of its user, requests a token from the Identity Provider (configured as the Authority in the Configuration). The configuration of the Identity Provider 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 Identity Provider, 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 Vonk in the Authorization header of the request.

**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).
Testing

Testing the access control functionality is possible on a local instance of Vonk. It is not available for the publicly hosted test server.

You can test it using a dummy Identity Provider and Postman as a REST client. Please refer to these pages for instructions:

- Set up an Identity Provider
- Access Control Tokens with Postman

3.4.3 Custom Operations

Validation

Vonk 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 Vonk 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 Vonk must have access to all relevant StructureDefinitions.

Validation has one Limitation.

Note: The very first validation call will take a considerable amount of time, typically around 5 seconds. This is because Vonk 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 <vonk_endpoint>/$validate[?profile=<canonical-url-of-structuredefinition>]

There are two ways of calling $validate:

1. With a Resource 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, Vonk will validate the Resource against the base profile from the FHIR Specification.

If you call $validate on the system level, Vonk will make no assumptions about the ResourceType of the Resource to validate.
Validate on the ResourceType level

You can call $validate in the same two ways as with Validate on the system level.
If you call $validate on the ResourceType level, Vonk will check whether the Resource to validate is of the same <resourcetype> as provided in the url.

Validate an instance from the database

This time you can only use the (optional) profile parameter on the url to specify a StructureDefinition to validate against.

Precondition

Vonk must be aware of all the StructureDefinitions referenced directly or indirectly by a profile. Refer to the Controlling the Conformance Resources for more information.

Limitation

1. The mode parameter is not yet supported.

Snapshot generation

Vonk is capable of generating a snapshot for a StructureDefinition. This operation is not defined in the FHIR Specification.

You can invoke this operation with

Note: The very first call to $snapshot will take a considerable amount of time, typically around 5 seconds. This is because Vonk maintains a cache of StructureDefinition information, and on the first call that cache is still empty. Subsequent calls are much faster.
Precondition

Vonk must be aware of all the other StructureDefinitions that are referred to by the StructureDefinition in the body of the request. Refer to the Controlling the Conformance Resources for more information.

$meta

Vonk 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. Vonk offers two custom operations out of the box.

1. Validation
2. Snapshot generation

3.4.4 Custom Resources

Custom Resources are not formally defined in the FHIR Specification. To Vonk a Custom Resource is a resource with a definition that is a specialization of DomainResource, but that is not in the Core FHIR Specification. Vonk 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.

What to use them for?

Vonk 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 Vonk to store, search and version these structures. Note that this is only for internal use in the app.

Register the definition

Just like any resourcetype, the definition for a custom resource is formalized as a StructureDefinition. Vonk will recognize it as the definition of a custom resourcetype if and only if:

- base = DomainResource
- derivation = specialization
- kind = resource

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.

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 Vonk using any of the methods mentioned in Controlling the Conformance Resources. As an example we will issue an update interaction on the Administration API:

```
PUT <base-url>/administration/StructureDefinition/Foo
```

By using an update we can choose the id and hence the location of this StructureDefinition. Vonk does this by default for all the resourcetypes defined by the specification as well.

**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 Vonk: 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. There is one limitation however:

- Validation: Custom Resources can not be validated. This implies that Validating incoming resources cannot be used in conjunction with Custom Resources.

We expect to remove this limitation in a future version of Vonk.

### 3.4.5 Terminology

Vonk provides limited support for these Terminology operations:

- $validate-code
- $expand
- $lookup
- $compose

These operations are currently implemented on the Administration API and therefore only available through the administration endpoint on <vonk-endpoint>/administration.

Refer to Controlling the Conformance Resources on how to provide CodeSystem and ValueSet resources to the Administration API.

3.4. Features
Note: This is a first attempt on supporting Terminology operations. We welcome feedback on how you would like to use them. Contact us at vonk@fire.ly.

Note: This implementation does not support complex terminologies like Snomed CT or LOINC.

**ValueSet $validate-code**

| definition  | http://www.hl7.org/implement/standards/fhir/valueset-operations.html#validate-code |
| notes       | • Available on the type level <vonk-endpoint>/administration/ValueSet/$validate-code and the instance level <vonk-endpoint>/administration/ValueSet/<id>/$validate-code. |
|             | • It is only available with http method POST with a Parameters resource in the body. |
|             | • 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. |

**ValueSet $expand**

| definition  | http://www.hl7.org/implement/standards/fhir/valueset-operations.html#expand |
| notes       | • Available on the type level <vonk-endpoint>/administration/ValueSet/$expand and the instance level <vonk-endpoint>/administration/ValueSet/<id>/$expand. |
|             | • It is only available with http method POST with a Parameters resource in the body. |
|             | • Only the parameters url, valueSet and valueSetVersion 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. |

**CodeSystem $lookup**

| definition  | http://www.hl7.org/implement/standards/fhir/codesystem-operations.html#lookup |
| notes       | • Available on the type level <vonk-endpoint>/administration/CodeSystem/$lookup. |
|             | • It is only available with http method POST with a Parameters resource in the body. |
|             | • Only the parameters code, system, version, coding and date are supported. |
|             | • Code & system combination takes priority over the coding parameter. |
**CodeSystem $compose**

**definition**  [http://www.hl7.org/implement/standards/fhir/codesystem-operations.html#compose](http://www.hl7.org/implement/standards/fhir/codesystem-operations.html#compose)

**notes**

- It is only available with http method POST with a Parameters resource in the body.
- 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.

### 3.4.6 Using Custom Search Parameters

**Configure Search Parameters**

You can control which search parameters are known to Vonk. This is managed in the same way as all the conformance resources, see *Controlling the Conformance Resources*.

**Re-indexing for new or changed SearchParameters**

Vonk 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 Vonk.
- 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’.
Rebuild the whole search index

This is only needed if we changed something very significant to the way Vonk searches, like
- The way values are extracted for all or many searchparameters.
- The structure in which Vonk stores the search index.

To re-index all resources for all search parameters, use:

```plaintext
POST http(s)://<vonk-endpoint>/administration/reindex/all
```

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:

```plaintext
POST http(s)://<vonk-endpoint>administration/reindex/searchparameters
```

In the body of the POST, you put the parameters to actually re-index as form parameters:

```plaintext
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, Vonk will return statuscode 403.

Re-index Configuration

Vonk 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 Vonk 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 Vonk will also be limited by the available computing resources.

```json
"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 Vonk
- and start a the reindex operation again.

**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.

### 3.4.7 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 Vonk 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 Vonk installation, in the file `errataR3.zip`, since they are corrections to the R3 / STU3 version of the specification. This file is imported automatically during startup, as are other conformance resources, see *Controlling the 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, Vonk 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.

### 3.4.8 Subscriptions

Subscriptions can be managed in the Vonk 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.

Vonk currently only supports Subscriptions with a Channel of type rest-hook.

If you are not permitted to access the `/Subscription` endpoint, Vonk will return statuscode 403.

See *Subscriptions in the specification* for more background on Subscriptions.
Configuration

Vonk 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.

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

- *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.

3.4.9 Reset the database

If you have set up Vonk 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 a set of resources*.

To reset the database, execute:

```
POST http(s)://<vonk-endpoint>/administration/reset
```

Vonk will return statuscode 200 if the operation succeeded.

If you are not permitted to perform the reset, Vonk 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 Vonk recreate it again.

- If you run on SQL Server, see *Using SQL server* for the AutoUpdateDatabase feature.
- If you run on MongoDB, Vonk 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.

3.4.10 Preloading a set of resources

If you have set up Vonk 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)://<vonk-endpoint>/administration/preload
Content-Type: application/octet-stream
Body: a zip file with resources, each resource in a separate file (xml or json).
```
Vonk will return statuscode 200 if the operation succeeded.

If you are *not permitted* to preload resources into the database, Vonk 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. Vonk currently has no special operation for bulk operations.

### 3.5 Vonk Administration API

Besides the regular FHIR endpoint, Vonk also exposes an Administration API. The endpoint for this is:

```
http(s)://<vonk-endpoint>/administration
```

#### 3.5.1 Functions

The following functions are available in the Administration API.

- Controlling the Conformance Resources
- Subscriptions
- Re-indexing for new or changed SearchParameters
- Reset the database
- Preloading a set of resources
- Terminology

#### 3.5.2 Configuration

You can Configure the Administration API, including restricting access to functions of the Administration API to specific ip addresses.

#### 3.5.3 Database

The Administration API uses a database separately from the main ‘Vonk Data’ database. Historically, SQL Server, MongoDB and Memory are supported as databases for the Administration API. As of 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.

### 3.6 Vonk FHIR Facade

Vonk FHIR Facade is a means to use the Vonk 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 Vonk FHIR Facade, and provides you with an exercise to get some hands-on experience with a Facade solution.

### 3.6.1 Facade setup options

There are two ways to set up a Vonk FHIR Facade. By creating a library with services and providing that as a plugin to Vonk, or by creating your own ASP.NET Core Web Application and utilizing Vonk NuGet packages. The first approach is the most widely used one, and also used in our exercise.

**Important:** We strongly recommend using the first approach because of its benefits described below, and because we may deprecate the second approach in the future.

**Provide a plugin to Vonk FHIR Server**

This leverages the capabilities of *Vonk FHIR Components*. With this setup you:

- create a new ASP.NET Core library
- include Vonk NuGet packages
- implement your own repository
- configure the PipelineOptions to use your library instead of Vonk’s own repository
- configure the PipelineOptions to limit the components to those that are supported by your repository implementation.

The benefit of using this approach is that you automatically get to use all of Vonk’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 Vonk.

**Create your own server**

In this setup you:

- create a new ASP.NET Core Web Application
- include Vonk NuGet packages
- set up the necessary Vonk services (usually a subset of the services that Vonk FHIR Server uses)
- implement your own repository

This also allows you to include the Vonk components into a web application of your own that may offer more interfaces than just the FHIR RESTful API.
3.6.2 Exercise: Build your first Facade

The best way to get experience in developing a Vonk FHIR 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 Vonk pipeline.

Using a Vonk 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 Vonk FHIR 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’.

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.

3.6.3 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
  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:
- Postman, Fiddler or a similar tool to issue http requests and inspect the responses.

Installing the Vonk server

Before you can start implementing your facade, you need to have the Vonk server installed. See Getting Started on how to download the binaries and license key.

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.
3.6.4 Starting your project

In this step, you will create and set up your project to start building the facade.

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

Add Vonk Components

1. Tools > NuGet Package Manager > Package Manager Console
   • Run Install-Package Vonk.Core
   • Run Install-Package Vonk.Fhir.R3

Note: You can install the latest beta release of the Vonk packages by adding -IncludePrerelease to the install command.

3.6.5 Mapping the database

In this step you will start mapping the existing database model to FHIR resources.

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.Core.Tools

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

(continues on next page)
The reverse engineer process created 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.

**Clean up generated code**

- To avoid naming confusion with the FHIR Resourcetype Patient, rename both files and classes:
  - Patient => ViSiPatient
  - BloodPressure => ViSiBloodPressure

- 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 Vonk Facade.
  - Rename the default Class1 class to DbOptions, and add this to interpret the setting:

    ```csharp
    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:

    ```csharp
    private readonly IOptions<DbOptions> _dbOptionsAccessor;
    public ViSiContext(IOptions<DbOptions> dbOptionsAccessor)
    {
        _dbOptionsAccessor = dbOptionsAccessor;
    }
    ```

  - Change the existing OnConfiguring method that contains the connectionstring to:

    ```csharp
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
            optionsBuilder.UseSqlServer(_dbOptionsAccessor.Value.ConnectionString);
        }
    }
    ```

**Create your first mapping**

1. Add a new class ResourceMapper to the project

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:

```csharp
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: Working with the model.

5. Then return the created Patient object as an IResource (you can use the extension method `ToIResource`):`

```
```IResource``` is an abstraction from actual Resource objects as they are known to specific versions of the Hl7.Fhir.Net API.
See :ref:`classes_iresource`.
```

3.6.6 Enable Search

Enabling search involves two major steps:

1. Creating a query 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.

1. Create a query

Vonk FHIR 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 Vonk 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, Vonk 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 Vonk 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-type per ResourceType. The Query object is used to capture the elements of the search that are provided to the QueryFactory. For each query-type you will need a 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 project
- Add using statements for Vonk.Facade.Relational and for <your project>.Models
- Above the actual PatientQueryFactory class insert the PatientQuery class:

```csharp
public class PatientQuery : RelationalQuery<ViSiPatient>
{
}
```

- Now flesh out the PatientQueryFactory:

```csharp
public class PatientQueryFactory : RelationalQueryFactory<ViSiPatient, _PatientQuery>
{
}
```

### Adding a constructor

You have to provide a constructor for the factory class. With this you tell Vonk for which resource type this QueryFactory is valid. The DbContext is used for retrieving DbSets for related entities, as we will see later:

```csharp
public PatientQueryFactory(DbContext onContext) : base("Patient", onContext) { }
```

### Handling the search request

Each of the search parameters 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 search parameter 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) 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:

```csharp
if (parameterName == "_id")
{
    if (!int.TryParse(value.Code, out int patientId))
    {
        throw new ArgumentException("Patient Id must be an integer value.");
    }
    else
    {
```

(continues on next page)
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 Vonk server will send a response with an error code and OperationOutcome. See the information about the IArgumentCollection and IArgument classes in Important classes and interfaces.

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 Vonk does not know the SearchParameter and hence not the type of it: RawValue.

This is the second step for enabling search.

3.6.7 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 project:

  ```csharp
  public class ViSiRepository : SearchRepository
  ```

- You have to provide a constructor that gets a QueryBuilderContext. 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:

  ```csharp
  private readonly ViSiContext _visiContext;
  private readonly ResourceMapper _resourceMapper;

  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 `<vonk-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:

```csharp
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 - a PatientQuery in this case.
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:

```csharp
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();
    }
}
```

(continues on next page)
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 Vonk response.

In the next paragraph you will configure your Vonk 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.

### 3.6.8 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 Vonk Server
3. Configure the Vonk Server to use your repository

1. **Add configuration class**

To add your repository service to the Vonk 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:

```csharp
[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;
    }
}
```
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 Vonk server

**Note:** If your Vonk 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.

3. Configure your Vonk Facade

- Create an appsettings.instance.json file in your Vonk working directory.

  **Tip:** See Vonk 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:

  ```json
  "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":

  ```json
  "SupportedInteractions": {
    "InstanceLevelInteractions": "read",
    "TypeLevelInteractions": "search",
    "WholeSystemInteractions": "capabilities, search"
  },
  ```

- Add the `SupportedModel` section to indicate which resource types and search parameters you support in your Facade implementation:

  ```json
  "SupportedModel": {
    "RestrictToResources": [ "Patient" ]
  },
  ```

- You will need to add your repository to the Vonk 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 Configuration of the pipeline with PipelineOptions for an example.

  - Copy the PipelineOptions section to your appsettings.instance.json file

  - To the Include part of the branch with "Path": "/" add your namespace, and remove the Vonk.Repository.* lines from it:
– Remove the PipelineOptions from appsettings.default.json, because of the warning mentioned on the Hierarchy of settings.

Test your work

Proceed to the next section to test your Facade, and for some helpful tips about debugging your code.

3.6.9 Debugging the Facade

• Start your Vonk server

  Note: If this is your first startup of Vonk, 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 components 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 Vonk Facade running!

Testing during implementation

Follow these steps if you want to test your work during the implementation phase without having to build, copy and start Vonk 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 Vonk 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-Vonk-working-directory>/Vonk.Server.dll.
• Set the Working directory to your Vonk working directory.

Now, whenever you click to start debugging, Vonk will start from your project and your project dll will be automatically built to the Vonk plugins directory.

Next part of the exercise

You can proceed to the next section to add support for Observations as well.

3.6.10 Finalizing your project

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 Vonk’s appsettings.instance.json:

   "RestrictToResources": [ "Patient", "Observation" ]

When you have completed these steps, build your project again and copy the dll to your Vonk plugins folder. After you (re)start Vonk, 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:

```csharp
public override BloodPressureQuery AddValueFilter(string parameterName, ReferenceToValue value)
{
    if (parameterName == "subject" && value.Targets.Contains("Patient"))
    {
        var patientQuery = value.CreateQuery(new RPatientQueryFactory(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 Vonk 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:

```csharp
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
Now you can test if reverse chaining works: 
http://localhost:4080/Patient?
_has:Observation:subject:_id=1

Get the goodies

At this point you get out of the box support for _include, _revinclude 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

Adding a custom SearchParameter

Your Vonk server will load the standard parameters from the specification on first startup, so the _id SearchParameter from the exercise is already known to Vonk, 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 Vonk 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.

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.

3.6.11 Enable changes to the repository

In Exercise: Build your first Facade you have created read and search support for a Vonk 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

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:

```csharp
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.

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:

```csharp
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:

```csharp
private readonly ViSiContext _visiContext;
private readonly ResourceMapper _resourceMapper;

public ViSiChangeRepository(ViSiContext visiContext, ResourceMapper resourceMapper)
{
    _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:

```csharp
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:

```csharp
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 Vonk 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:

```csharp
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 Vonk server for example generates a GUID in these methods.

At this point you can skip ahead to 3. Configure the service and Vonk 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:

```csharp
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:

```csharp
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);

    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;
}

### 3. Configure the service and Vonk 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 Vonk appsettings.

- In your project, go to the ViSiConfiguration class, and add this line to add an IResourceChangeRepository to the pipeline:

```csharp
services.TryAddScoped<IResourceChangeRepository, ViSiChangeRepository>();
```

- Add support for the interactions to the SupportedModel section of the Vonk appsettings:

```json
"SupportedInteractions": {
    "InstanceLevelInteractions": "read, update, delete",
    "TypeLevelInteractions": "search, create",
    "WholeSystemInteractions": "capabilities, search"
},
```

You can now build your project, copy the dll to the Vonk plugins folder and run Vonk to test the new interactions on your Facade.

**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*.

### 3.6.12 Access Control in Facade and Components

The *Access Control feature* is also available to users of a Vonk FHIR Facade or Vonk FHIR Components. 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

<table>
<thead>
<tr>
<th>Interface / Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAuthorization</td>
<td>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</td>
</tr>
<tr>
<td>ICompartment</td>
<td>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</td>
</tr>
<tr>
<td>IReadAuthorizer</td>
<td>Calculates access control for a type of resource given an instance of IAuthorization and/or ICompartment</td>
</tr>
<tr>
<td>IWriteAuthorizer</td>
<td>Calculates access control for writing a new (version of a) resource given an instance of IAuthorization and/or ICompartment</td>
</tr>
<tr>
<td>AuthorizationResult</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

### 3.7 Vonk FHIR Components

Vonk FHIR Components is the means to adjust a Vonk FHIR Server to your own special needs, beyond the configuration. Vonk FHIR Server is built as a pipeline of middleware, see Architecture for more background on that. Vonk FHIR Components allows you to put components of your own into that pipeline, and remove components that you do not need.
Warning: Vonk FHIR Components and the PipelineOptions described below are in their early stages of development. This means you may encounter rough edges, parts of the programming interface may still change and the PipelineOptions themselves may still change. Especially the use of Vonk FHIR Components for building a Facade in combination with the Administration API will get a lot of attention in upcoming releases. All changes will be announced in the Release notes Vonk. You are encouraged to use Vonk FHIR Components, but we may not be able to avoid breaking changes.

3.7.1 Configuration of the pipeline with PipelineOptions

Configuration of the pipeline in Vonk FHIR Server is done with the PipelineOptions. A default setup is installed with Vonk in appsettings.default.json, and it looks like this:

```json
"PipelineOptions": {
   "PluginDirectory": "../plugins",
   "Branches": [
     { 
      "Path": "/",
      "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",
        "Vonk.Repository.Sql",
        "Vonk.Repository.MongoDb",
        "Vonk.Repository.Memory",
        "Vonk.Subscription",
        "Vonk.UI.Demo"
      ],
      "Exclude": [
      ]
    },
    { 
      "Path": "/administration",
      "Include": [
        "Vonk.Core.Context",
        "Vonk.Core.Infra",
        "Vonk.Repository.Sql",
        "Vonk.Repository.MongoDb",
        "Vonk.Repository.Memory",
        "Vonk.Administration"
      ],
      "Exclude": [
      ]
    }
  ]
}
```

**PluginDirectory:** You can put plugins of your own (or third party) into this directory for Vonk to pick them up, without polluting the Vonk binaries directory itself. 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.

**Branches:** A web application can branch into different paths, and Vonk has two by default:

- `/`: the root branch, where the main FHIR RESTful API is hosted;
- `/administration`: where the Vonk Administration API is hosted.

Branches contains a subdocument for each of the defined paths:

3.7. Vonk FHIR Components
Path  The path for this branch. This is the part after the base URL that Vonk is hosted on.

Include  (Prefixes of) Configuration classes that add services and middleware to Vonk.

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.

3.7.2 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 Vonk FHIR Server dependency injection system, and add middleware to the pipeline.

```
[VonkConfiguration (order: xyz)] //xyz is an integer
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 Vonk (package Vonk.Core, namespace Vonk.Core.Pluggability). It tells Vonk 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 components in the log at startup.

MyVonkConfiguration  You can give the class any name you want, it will be recognized by Vonk 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 components, so you allow yourself to configure your components 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 Vonk.

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 Vonk.

We provided an example of this: creating your own landing page.

3.7.3 Detailed logging of loading components

If your component or any of the Vonk components 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, Vonk 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.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, Vonk 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 Vonk FHIR Server.

Configuration:
/
  FhirR3Configuration [100] | 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

(continues on next page)
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: V
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
/security
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: X
FormatConfiguration [1130] | Services: V | Pipeline: V
SecurityConfiguration [1150] | Services: V | Pipeline: V
AdministrationOperationConfiguration [1160] | 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
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, Vonk 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 [Vonk] [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 [Vonk] [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 [Vonk] [Verbose] [Machine: XYZ] [ReqId: ] "Vonk.Core.Serialization.SerializationConfiguration" was not included on "/administration" because it did not match any include
```

### 3.7.4 Important classes and interfaces

If you want to develop a plugin for Vonk FHIR Server, there are a couple of classes that you will probably interact with. This page lists those classes, with an explanation of each.

**IResource**

```csharp
namespace Vonk.Core.Common

purpose IResource is the abstraction for all FHIR resources in Vonk. 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 Vonk needs to read and maintain.
/// <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-e6523af3a19'. 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.
    /// </summary>
    string Id { get; }

    (continues on next page)
/// 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.
/// </summary>
string Version { get; }

/// 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.
/// </summary>
DateTimeOffset? LastUpdated { get; }

/// 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; }

/// Direct access to contained resources, if any. Prefer to return an empty list
/// 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:

```csharp
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 ex-
tension methods on ISourceNode, found in Vonk.Core.ElementModel.ISourceNodeExtensions. There are also several
extension methods specifically for IResource in Vonk.Core.Common.IResourceExtensions:

```csharp
var updatedResource = oldResource.Add(SourceNode.Valued("someElement", "someValue");
//Continue with updatedResource, since oldResource will not have the change.
```

IVonkContext

```csharp
namespace Vonk.Core.Context.Features

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 Vonk 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.
```
```csharp
public class SomeService
{
    public async Task DoMyOperation(IVonkContext vonkContext)
    {
        //...
    }
}

public static class SomeServiceConfiguration
{
    public static IApplicationBuilder UseMyOperation(IApplicationBuilder app)
    {
        return app.UseVonkInteractionAsync<SomeService>((svc, context) => svc.
        DoMyOperation(context));
    }
}

If you also need access to the raw HttpContext, you can 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.

```csharp
public class SomeMiddleware
{
    public SomeMiddleware(RequestDelegate next)
    {
        //...
    }

    public async Task Invoke(HttpContext httpContext)
    {
        var vonkContext = httpContext.Vonk();
        //...
    }
}

public static class SomeMiddlewareConfiguration
{
    public static IApplicationBuilder UseSomeMiddleware(IApplicationBuilder app)
    {
        return app.UseMiddleware<SomeMiddleware>(); //Just plain ASP.NET Core, nothing...
        Vonk specific here.
    }
}

IVonkContext has three major parts, that are explained below:

```csharp
public interface IVonkContext
{
    IVonkRequest Request {get;}
    IArgumentCollection: Arguments{get;}
    IVonkResponse Response {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 response) → Parts(this IVonkContext vonkContext)

**IVonkRequest**

```csharp
namespace Vonk.Core.Context.Features

purpose Get information about the request made, in Vonk / FHIR terms.
```

You can access the current `IVonkRequest` through the `IVonkContext`. Its properties are:

```csharp
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:

```csharp
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.
```

```csharp
public void ThisMethodActsOnThePayloadIfPresent(IVonkContext vonkContext)
{
    var (request, args, response) = vonkContext.Parts();
    if (request.TryGetPayload(response, out var resource))
    {
        // do something with the resource.
    }
}
```

```csharp
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.
```

```csharp
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, Vonk will end the pipeline
    }
}
```

(continues on next page)
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

```csharp
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();
}
```

**namespace** Vonk.Core.Context.Features

**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, identifier 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 Vonk automatically.

Any argument that is not handled will automatically be reported as such in an **OperationOutcome**.

Useful extension methods:

**IVonkResponse**

```csharp
namespace Vonk.Core.Context.Features

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.

**InteractionHandlerAttribute**

```csharp
namespace Vonk.Core.Pluggability

public class InteractionHandlerAttribute
{
    [InteractionHandler]
    public async Task DoMyOperation(IVonkContext vonkContext)
    {
        // Implementation
    }
}
```

You can specify different filters, and combine them at will:

- **Specific interaction(s):** `[InteractionHandler(Interaction = VonkInteraction.type_create | VonkInteraction.instance_update)]`
- **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 })]`
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`).

```csharp
public static class MyOperationConfiguration
{
    public static IApplicationBuilder UseMyOperation(this IApplicationBuilder app)
    {
        return app
            .OnInteraction(VonkInteraction.type_create | VonkInteraction.instance_update)
            .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.

```csharp
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.

### 3.7.5 Template for a plugin

**Attention:** A complete template for a Vonk plugin can be found on Github. It covers all details on how to create a custom operation and how to use Vonk services internally.

A regular Vonk 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 `InteractionHandlerAttribute` for details on that.

You can use the following code as a template for a plugin:

```csharp
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;
        }

        public static IApplicationBuilder UseMyPlugin(IApplicationBuilder app)
        {
            app.OnCustomInteraction(VonkInteraction.system_custom, "myOperation").
                HandleAsyncWith<MyPluginService>((svc, context) => svc.Act(context));
            return app;
        }
    }
}

3.7.6 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. The component with order 1110 makes the IVonkContext accessible. The component with order 1120 turns the IVonkContext.Response to the HttpContext.Response. That means if you pick an order between 1110 and 1120 you can read the IVonkContext.Request and Arguments, and write directly to the HttpContext.Response. The steps to take are:

1. Configure your plugin with an order between 1110 and 1120
2. Write the HttpContext.Response.Body directly

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;
(continues on next page)
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;
        }

        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 need 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.GetBytes(message).ToString();
                httpContext.Response.Headers.Add("Content-Type", "text/plain; charset=utf-8");
                httpContext.Response.Headers.Add("Content-Length", contentLength);
                await httpContext.Response.WriteAsync("This is a response that is not a FHIR resource");
            }
            else
            {
                await _next(httpContext);
            }
        }
    }
}

[VonkConfiguration(order: 1115)] // note the order: between 1110 and 1120
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;
    }
}
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 may put the middleware anywhere in the pipeline. If you want to return e.g. a custom json object, you should put it between 1110 and 1120 as described above. Do not forget to set the Content-Type (to application/json for a custom json object).

3.7.7 Vonk FHIR Components example - Create a new landing page

As a minimal example of how to use Vonk FHIR Components 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 Vonk. The landing page is the webpage you see when you access Vonk’s endpoint with a browser. By default it looks like this:

![Landing Page](image)

Create a new ASP.NET Core web application

In Visual Studio create a new project of type ASP .NET Core Web Application:
Choose a name for your project and solution. Press OK to continue.
Choose ASP.NET Core 2.0 and select Web Application (Model-View-Controller). Press OK.

Add Vonk Package

Add Vonk.Core via the Nuget Package Manager:
This will give you access to all the core components of Vonk, including the `Vonk.Core.Pluggability.VonkConfiguration` attribute.

### Adjust project file

Add `wwwroot` and `Views` as an Embedded resource in the project file (that is necessary for Vonk to pick them up from a library dll). To edit the project file, right click on the project file and select Edit `<projectname>.cproj`:

```xml
<ItemGroup>
  <EmbeddedResource Include="wwwroot\**\*" />
  <EmbeddedResource Include="Views\**" />
</ItemGroup>
```

And set `Microsoft.AspNetCore.All` to 2.0.0:

```xml
<PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
```

The project file will look like this:

```xml
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
    <PackageReference Include="Vonk.Core" Version="0.7.0-alpha843" />
  </ItemGroup>

  <ItemGroup>
  </ItemGroup>

  <ItemGroup>
    <EmbeddedResource Include="wwwroot\**\*" />
    <EmbeddedResource Include="Views\**" />
  </ItemGroup>

</Project>
```

Save the project file.

### Create the configuration class

Next, add a new file for the configuration class, as described in *Configuration classes*. Annotate it with `[VonkConfiguration(order: 802)]`. Then add the static methods as prescribed:

```csharp
public static IServiceCollection AddUIServices(IServiceCollection services)
{
  var thisAssembly = typeof(UIConfiguration).GetTypeInfo().Assembly;
  services
    .AddMvc()
    .AddApplicationPart(thisAssembly)
    ...

  return services;
}
```

(continues on next page)
.AddControllersAsServices();

var embeddedFileProvider = new EmbeddedFileProvider(
    thisAssembly,
    thisAssembly.GetName().Name
);

services.Configure<RazorViewEngineOptions>(options =>
{
    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:
Deploy and Configure

Build this project in Release mode and copy the produced dll (located in <src>\bin\Release\netcoreapp2.0) to the plugin directory of Vonk, as configured in the PipelineOptions:PluginDirectory.

Go to the Vonk settings of Vonk, and replace the namespace of the landingpage (Vonk.DemoUI) in the include of the PipelineOptions:

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

(continues on next page)
Run and admire

Now run Vonk from the commandline or Powershell window with

```bash
> dotnet .\Vonk.Server.dll
```

Open a browser and visit the homepage of Vonk (http://localhost:4080) to admire your own landingpage.

### 3.7.8 Vonk FHIR Components 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.

### 3.8 Architecture

#### 3.8.1 Pipeline and middleware components

Vonk 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.

Vonk FHIR Components are pieces of middleware that handle 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 translates 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 Vonk FHIR 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. Vonk 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.

### 3.8.2 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 Vonk, 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.

```csharp
IChangeRepository    //for create, update and delete
ISearchRepository    //for all types of search, read and history
```

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 Vonk FHIR 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. Vonk 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 Vonk to interpret the search, you can use the QueryBuilderFactory.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 Vonk also supports **Using Custom Search Parameters**.

The ResultShape method is called when an argument is recognized as a ‘Search result parameter’, like _sort or _include.

---

**3.8. Architecture** 187
3.8.3 Capabilities

A FHIR server has to express its capabilities in a CapabilityStatement, available on the /metadata endpoint. Vonk’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 IConformanceContributor, in which it gets access to the IConformanceBuilder. 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.

3.8.4 Current status

Currently we have implemented three different generic repository implementations and two different facades to existing systems. This has given us a fairly good idea of the strengths and weaknesses of this architecture and its interfaces. Based on that experience we released Vonk FHIR Facade in September 2017.

However, as we continue to expand on generic implementations (PostgreSQL, CosmosDB or MySQL are just some of the ideas) and guide more facade implementations we will further strengthen the interfaces and support methods. So they will still change over time. Especially the interfaces for the generic implementations. And we don’t want to break your implementation on every update. Therefore the Vonk FHIR Components are not yet made publicly available on NuGet.

If you want to use the Vonk FHIR Components already, please contact us. We can then provide you with the necessary NuGet packages along with personal guidance to help you succeed with your implementation. Also, we can then make sure that your implementation gets updated upon changes to the architecture or the interfaces. And we will learn from your requirements and adapt Vonk where needed.

When the Vonk FHIR Components are made publicly available on NuGet we will also extend this documentation.

3.9 Release notes Vonk

3.9.1 Older Vonk Release notes

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 you Simplifier project! See Use Vonk FHIR 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 ResourceType, 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 0.95.0 (DSTU2, STU3 and R4) (released 20180412)).

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 neccessary 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).
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**, 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) resource types 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 *Controlling the Conformance Resources* for details.

2. Feature: Experimental support for **Terminology** 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 AllowedProfiles 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 resource type 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/`)

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.

**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 Vonk 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 Vonk settings.

4. Feature: Improved logging, including Dependency Tracking on Azure Application Insights, see Azure 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 Vonk settings.

7. Feature: Added documentation on running Vonk behind IIS or NGINX: Deploy Vonk 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 (\(\text{Observation.code=somesystem}\))

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 Controlling the Conformance Resources.

18. Fix: If a Compartment filter is used on a parameter that is not implemented, Vonk will return an error, see Compartments.

**Release 0.6.1.0**

Name change from Furore to Firely

**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 Controlling the Conformance Resources for instructions.

3. Feature: ValueSet and CodeSystem resources can be loaded into the administration endpoint, and loaded from Simplifier. See Controlling the 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 *Controlling the Conformance Resources*.

**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, refering 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.

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.

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 nog 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.

**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 a set of 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.


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).


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.

---

**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’:

```json
"Hosting": {
  "HttpPort": 4080,
  "HttpsPort": 4081, // Enable this to use https
  "CertificateFile": "<your-certificate-file>.pfx", // Relevant whenHttpsPort 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.

Release 0.3.2.0

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

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.

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

3.9. Release notes Vonk
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/updated 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. _revinclude

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.
   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

5. DevOps:
   1. New Vonk 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.

## 3.9.2 Security notifications for Vonk

### 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 up to secure versions. No action is required by the administrator of Vonk.

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.

3.9.3 Release 2.0.1 hotfix

Fix

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

3.9.4 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 its 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.

3.9.5 Release 2.0.0-beta2

Fix

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

3.9.6 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](http://simplifier.com/downloads), 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](https://github.com/FhirNETApi/Fhir.NET/issues/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 its [release notes](https://github.com/FhirNETApi/Fhir.NET/releases/tag/v1.1.2).
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.
3.9.7 Release 1.1.0

Attention: New security issues have been identified by Microsoft. See the Security notifications for Vonk 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 HI7.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 Vonk Pipeline) may now contain subfolders. Plugins will be read from all underlying folders.
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 Vonk FHIR Server.
8. Upgrade of the HI7.Fhir.API library to 1.1. See the API releasenotes for 1.1.0 (DSTU2, STU3) (beta - final version to be released 20190128).

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.

3.9.8 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 - Vonk 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.

3.9.9 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\times\text{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.
3.9.10 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.

3.9.11 Release 0.7.2.1

Fix

1. Delete on /administration was broken.

3.9.12 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 release notes for the FHIR API.
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, 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).

**3.9.13 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.

**3.9.14 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 Vonk 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.

### 3.9.15 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.


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 Components* has been released. You can now add libraries with your own components through configuration.

2. Feature: Through *Vonk FHIR Components* 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 their 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.


### 3.10 Contact us

For questions or feedback on Vonk, please send an e-mail to vonk@fire.ly.
Welcome to the VonkLoader documentation

4.1 Installing

VonkLoader is a free, multithreaded tool for Windows / macOS / Linux to upload resources in bulk to any FHIR server. To install, go to simplifier.net/downloads and log in.

4.2 Running

Available parameters are:

- **-file:<filename>**
  - zip file with xml and/or json files with resources, mandatory
- **-server:<url>**
  - FHIR server base address, mandatory
- **-limit:x**
  - load no more than x resources, default is all the resources in the zip
- **-parallel:**
  - how many requests are sent in parallel, default is 10
- **-collectionHandling:** AsIs / Split / Transaction
  - AsIs: send a collection Bundle as is (to /Bundle)
  - Split: send each of the Bundle.resource's from a collection Bundle separately
  - (STU3 only)
  - Transaction: convert the collection Bundle to a transaction Bundle and send it to the server base address (STU3 only)
- **-forcePutInTransaction:** resources in collections that have no id get one and are turned into a PUT entry in the transaction.
  - By default they are turned into a POST. Only useful in combination with -
  - -collectionHandling:Transaction

See also vonkloader --help on running the tool.

Example:
This is documentation site for the support API for working with HL7 FHIR on the Microsoft .NET platform. The library provides:

- Class models for working with the FHIR data model using POCO’s
- A REST client for working with FHIR-compliant servers
- Xml and Json parsers and serializers
- Helper classes to work with the specification metadata, and generation of differentials
- Validator to validate instances against profiles
- A lightweight in-memory terminology server

On these pages we provide you with the documentation you need to get up and running with the API. We’ll first explain how the FHIR model is represented in the API and give you code examples to work with the model. The FhirClient and its methods will also be demonstrated. Within an hour you can create your own simple FHIR client!

After those topics to get you started, we have added some pages that delve deeper into nice API features, such as parsing and serializing FHIR data, working with transactions, and using the ResourcIdentity functionality.

**Note:** All code examples on these pages are for the STU3 version of the library. Some of them can still be used with the DSTU2 library, but you might need to tweak the code a bit.

Please look at the Contact us page for ways to ask questions, contribute to the API, or reach out to other .Net developers in the FHIR community.

### 5.1 Getting Started

It’s easy to start using the FHIR .Net API in your solution: start with the right NuGet package.
5.1.1 Install via .nuspec

dotnet add package Hl7.Fhir.R4

5.1.2 Install via Visual Studio

Open your project, or start a new one and go to the NuGet Package Manager. The current official version of the FHIR specification is R4, but you might be developing for one of the previous versions. For each version of the specification, there is a corresponding library.

For R4, you will need the Hl7.Fhir.R4 package:

1. Choose Tools NuGet Package Manager Manage NuGet Packages for Solution...
2. Click on Browse, and type fhir into the search field.
4. Check the box next to the name of your project and click on Install.

The next section explains how the API works with the FHIR model.

5.2 Working with the model

Hl7.Fhir.Model contains model classes that correspond to the FHIR Resources and data types, like Patient and HumanName. The code is generated by the FHIR publication tool, which is used to build a new version of the FHIR website.
In this chapter, we will explain how to work with the model, and give you some code examples. We also include a complete code example that sets up an instance of the Patient resource, filling in values for several of the fields. We conclude the chapter with a discussion of the Bundle resource type, plus some code examples of how to work with Bundles.

Add this using directive to your code:

```csharp
using Hl7.Fhir.Model;
```

### 5.2.1 Model classes

For each Resource type and data type in FHIR, the API contains a class in the form of a public partial class. If you need extra code to be performed when using these classes, you can just implement your own partial class for it. That way you can keep your code separate from the API code.

Creating a new Patient resource instance, and an instance of the Identifier data type:

```csharp
var pat = new Patient();
var id = new Identifier();
```

**Important:** When you are creating an instance of a resource or data type, lookup the definition in the FHIR specification to see which elements are mandatory for that particular type.

### Class fields

The API classes have a field for each of the elements in the Resource or data type model. For example, the Patient resource has an `active` element:

<table>
<thead>
<tr>
<th>Name</th>
<th>Flags</th>
<th>Card.</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient</td>
<td></td>
<td></td>
<td>DomainResource</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>identifier</td>
<td>Σ</td>
<td>0..*</td>
<td>Identifier</td>
</tr>
<tr>
<td>active</td>
<td>Ψ Σ</td>
<td>0..1</td>
<td>boolean</td>
</tr>
<tr>
<td>name</td>
<td>Σ</td>
<td>0..*</td>
<td>HumanName</td>
</tr>
</tbody>
</table>

The Patient class in the API has a field called `Active` that corresponds with this element:

```csharp
namespace Hl7.Fhir.Model
{
    /// <summary>
    /// Information about an individual or animal receiving healthcare services
    /// </summary>
    /// [FhirType("Patient", IsResource=true)]
    {
        ..
        /// Whether this patient’s record is in active use
        /// <remarks>This uses the native .NET datatype, rather than the FHIR equivalent</remarks>
        [NotMapped]
        [IgnoreDataMember]
        public bool? Active
        {
            ..
        }
    }
}
```

Likewise, the Identifier data type has an element called `use`:

5.2. Working with the model
And the Identifier class in the API has a field called `Use` that corresponds with this element:

```
{
    ... // (summary)
    // An identifier intended for computation
    // </summary>
    [Hl7.Fhir.Model.Identifier]
    public Identifier use : Identifier Use
    {
        ... // (summary)
        // usual | official | temp | secondary (if known)
        // </summary>
        // (remarks)This uses the native .NET datatype, rather than the FHIR equivalent</remarks>
        // [NoMapped]
        // [IgnoreDataMemberAttribute]
        // [reference]
        public Identifier.Use
        {
        }
    }
}
```

As you can see, the classes and fields all have inline documentation describing them.

### 5.2.2 Primitive data types

In FHIR, the data types are divided into ‘primitive’ and ‘complex’ data types. The primitive data types are types like `string`, `integer`, `boolean`, etc. that can take a single value. The complex types consist of multiple values grouped together.

**Important:** Primitives are not really primitive in FHIR!

Because you can extend resources and data types in FHIR, the API has provided FHIR data types for the primitive types. Where the name of the FHIR data type would conflict with existing .Net data types, the word ‘Fhir’ is added to the type, e.g. `FhirString`.

For each of the fields that take a primitive data type, the API provides you with two fields in the class. One of the fields has the same name as the element it corresponds with in the FHIR resource, e.g. `Active` in the `Patient` class. This field is of the standard .Net data type.

You can fill this field just the way you would expect:

```csharp
var pat = new Patient();
pat.Active = true;
```

The other field has got the name of the element, with ‘Element’ added to it, for example `ActiveElement` in the `Patient` class. You fill this field with the FHIR data type that is in the API:

```csharp
pat.ActiveElement = new FhirBoolean(true);
```

**Note:** Both of the statements set the same private data member of the class.
5.2.3 Complex data types

Complex data types in FHIR are data types that group certain values together, such as `Address`, `Identifier` and `Quantity`. The FHIR specification describes which elements are part of these data types.

The API has created classes for each of the data types, with fields for each of the elements. Most of the elements will be of a primitive data type, but you can also encounter complex types within a complex data type.

Filling in the fields for the primitive types is explained *in the previous paragraph*. However, if you need to fill in a field that is of a complex data type, you will need to create an instance of that type first.

For example, if we want to fill in the data for a field of type `Identifier`, we can use this code:

```csharp
var id = new Identifier();
id.System = "http://hl7.org/fhir/sid/us-ssn";
id.Value = "000-12-3456";
```

5.2.4 Lists

For elements with a maximum cardinality > 1, the API has a list of the type for that element.

To work with data in a list, you can use the standard C# List methods.

So for example, if we want to add the `Identifier` we created in the previous paragraph to the `Identifier` field of the instance of `Patient` we created earlier, we can do this:

```csharp
pat.Identifier.Add(id);
```

*Note:* If you did not initialize a field before adding to the list, the API will create the List for you, and will not generate a `NullReferenceException`.

5.2.5 Components

Resources can have elements with a subgroup of elements in them. These are called ‘BackboneElements’ or ‘components’. For example, the Patient resource type has a component called `contact`.

5.2. Working with the model
In the API, a component block is represented by a class within the resource type class. This subclass has the name of the field, followed by ‘Component’, for example `ContactComponent` in the `Patient` class:

```csharp
namespace HL7.Fhir.Model
{
    {
        ...
        [PhishType("ContactComponent")]
        [DataContract]
        {
            ...
        }
    }
}
```

Code example, adding contact details to our Patient:

```csharp
var contact = new Patient.ContactComponent();
contact.Name = new HumanName();
contact.Name.Family = "Parks";
// setup other contact details
pat.Contact.Add(contact);
```

### 5.2.6 Enumerations

For coded types in FHIR, the elements are bound to a ValueSet. When the specification states that the ValueSet is ‘Required’, this means it is a fixed list of codes. The API provides an enumeration for each fixed ValueSet. You can use these enumerations to fill in the correct value.

The `Patient` resource has a fixed ValueSet for the `gender` element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Flags</th>
<th>Card.</th>
<th>Type</th>
<th>Description &amp; Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient</td>
<td></td>
<td></td>
<td>DomainResource</td>
<td>Information about an individual or animal receiving health care services</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Elements defined in Ancestors: id, meta, implicitRules, language, text, cr</td>
</tr>
<tr>
<td>gender</td>
<td>Σ</td>
<td>0..1</td>
<td>code</td>
<td>male</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AdministrativeGender (Required)</td>
</tr>
</tbody>
</table>
Code example, adding a gender to our Patient:

```csharp
pat.Gender = AdministrativeGender.Male;
```

### 5.2.7 Choice properties

In the FHIR specification, you will encounter ‘choice properties’ for some of the resource’s elements. This means that you can choose the type you fill in for that element, from the possible types listed.

For the Patient resource type for example, we have a choice for the `deceased` element:

<table>
<thead>
<tr>
<th>Name</th>
<th>Flags</th>
<th>Card. Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient</td>
<td></td>
<td></td>
<td>DomainResource</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deceased[0]</td>
<td>?!</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>deceasedBoolean</td>
<td></td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>deceasedDateTime</td>
<td></td>
<td>dateTime</td>
<td></td>
</tr>
</tbody>
</table>

In the API, you will see that the corresponding field is of type `Element`, which is the base for all data types.

```csharp
/// <summary>
/// Indicates if the individual is deceased or not
/// </summary>
[Element("deceased", InSummary=true, Order=150, Choice=ChoiceType.DateTimeFieldChoice)]
public null

This means that in your code, you will first have to create an instance of the data type of your choice, before you can fill in the field. For example, if we choose to use a date for the `Deceased` field of our Patient, we could implement that like this:

```csharp
var deceased_date = new DateTime("2015-04-23");
pat.Deceased = deceased_date;
```
Or, if we choose to fill in a boolean value:

```csharp
pat.Deceased = new FhirBoolean(true);
```

### 5.2.8 Special initializers

As you can see from the example in the previous paragraph with the `FhirDateTime` or `FhirBoolean`, for several data types, the API provides you with extra initialization methods. Visual Studio’s IntelliSense will help you to view the possibilities while you type, or you can take a look at `Hl7.Fhir.Model` with the Object Browser to view the methods, plus their attributes as well.

For the `HumanName` data type, the API has added some methods to make it easier to construct a name in one go, using fluent notation:

```csharp
pat.Name.Add(new HumanName().WithGiven("Christopher").WithGiven("C.H.").AndFamily("Parks"));
```

If you need to fill in more than the `Given` and `Family` fields, you could first construct a `HumanName` instance in this manner, and add to the fields later on. Or you could choose not to use this notation, but instead fill in all the fields the way it was explained in the other paragraphs.

### 5.2.9 Extensions

In the `Primitive data types` paragraph, we have mentioned that both resource and data types can be extended in FHIR. To add an extension to your data, you will have to fill in both a URL identifying the extension and a value that is valid according to the definition of the extension.

The following code example adds a place and time of birth to our Patient instance defined by standard extensions in the FHIR specification. The URL and the type of value you can fill in are listed in the definition of the extension. The definition will also tell you on what field the extension can be added.

```csharp
var birthplace = new Extension();
birthplace.Url = "http://hl7.org/fhir/StructureDefinition/birthPlace";
birthplace.Value = new Address() { City = "Seattle" };
pat.Extension.Add(birthplace);

                             new FhirDateTime(1983,4,23,7,44));
pat.BirthDateElement.Extension.Add(birthtime);
```

The extension for birthPlace is pretty straightforward to add. The URL is taken from the extension definition. The value is of type `Address`, and the extension can be added to the top-level of the Patient instance. The birthTime extension is a little more complex. This extension takes a `dateTime` value, and has to be added to the `BirthDate` field. For this field the API provides you with the easy way to fill it, by allowing you to set the value of `BirthDate` as a `string`—internally converting this to the `Date` type. This means you will have to use the `[fieldname]Element` construction to add extensions to the field.

### 5.2.10 Code example for Patient

With the code examples from the previous paragraphs, plus some additions, we have constructed a code example that sets up an instance of the Patient resource, with some information covering all of the topics of this section. We have tried to include different ways to fill in the fields, so you can see the possibilities and choose what suits your programming style best.
```javascript
// example Patient setup, fictional data only
var pat = new Patient();

var id = new Identifier();
id.System = "http://hl7.org/fhir/sid/us-ssn";
id.Value = "000-12-3456";
pat.Identifier.Add(id);

var name = new HumanName().WithGiven("Christopher").WithGiven("C.H.").AndFamily("Parks");
name.Prefix = new string[] { "Mr." };
name.Use = HumanName.NameUse.Official;

var nickname = new HumanName();
nickname.Use = HumanName.NameUse.Nickname;
nickname.GivenElement.Add(new FhirString("Chris"));
pat.Name.Add(name);
pat.Name.Add(nickname);

pat.Gender = AdministrativeGender.Male;
pat.BirthDate = "1983-04-23";

var birthplace = new Extension();
birthplace.Url = "http://hl7.org/fhir/StructureDefinition/birthPlace";
birthplace.Value = new Address() { City = "Seattle" };
pat.Extension.Add(birthplace);

    new FhirDateTime(1983, 4, 23, 7, 44));
pat.BirthDateElement.Extension.Add(birthtime);

var address = new Address()
{
    Line = new string[] { "3300 Washtenaw Avenue, Suite 227" },
    City = "Ann Arbor",
    State = "MI",
    PostalCode = "48104",
    Country = "USA"
};
pat.Address.Add(address);

var contact = new Patient.ContactComponent();
contact.Name = new HumanName();
contact.Name.Given = new string[] { "Susan" };
contact.Name.Family = "Parks";
contact.Gender = AdministrativeGender.Female;
contact.Relationship.Add(new CodeableConcept("http://hl7.org/fhir/v2/0131", "N"));
contact.Telecom.Add(new ContactPoint(ContactPoint.ContactPointSystem.Phone, null, ""));
pat.Contact.Add(contact);

pat.Deceased = new FhirBoolean(false);
```

5.2. Working with the model 219
5.2.11 Bundles

Although Bundle is just another resource type in FHIR, and you can fill in the values for the fields in the way that has been described in this chapter, a Bundle is still a bit special. Bundles are used to communicate sets of resources. Usually you will first encounter them when you’re performing a search interaction. In that case, the server will send you a Bundle, and you will browse through the contents to process the data. Sometimes you will need to construct a Bundle instance and fill in the details, for example when you are going to setup a FHIR document, or want to perform a transaction, or if you’re implementing the server side response to a search request.

Looking at the content

A Bundle resource has got some fields that contain metadata about the Bundle, such as the type of Bundle, and a total if the Bundle contains a result from a search or history interaction. The resources that are put in a Bundle, are located in the entry element of the Bundle resource. Since entry is a 0..* element and a component block, the API provides you with a list of EntryComponent in the Bundle.Entry field. You can loop through that list, or use any of the standard C# List methods to work with the list.

The fully qualified URL identifying the resource that is in the entry, is stored in the FullUrl field of the entry. The API doesn’t know the type of resource that is in the entry, so the data type for the Resource field in the entry is the base type Resource. You will need to cast to the actual resource type if you want to have access to the fields for that type.

Tip: You can check the Resource.ResourceType field first, if you don’t know the type of resource for an entry.

Suppose we have performed a search interaction on Patient resources, and have stored the results in a variable called result. We can then loop through the resources in the Entry list like this:

```csharp
foreach (var e in result.Entry)
{
    // Let's write the fully qualified url for the resource to the console:
    Console.WriteLine("Full url for this resource: " + e.FullUrl);

    var pat_entry = (Patient)e.Resource;

    // Do something with this patient, for example write the family name that's in the first
    // element of the name list to the console:
    // Patient's last name: " + pat_entry.Name[0].Family);
}
```

Filling a Bundle

When constructing a Bundle, you will need to look at the definition for the Bundle resource to see which elements are mandatory, just as you would do for other resource types.

Tip: If you want to create a Bundle for a batch or transaction, you can use the helper methods in the API to construct the Bundle correctly, described in the transactions paragraph.

Then, for each resource you want to include in the Bundle, you will add an EntryComponent to the Entry list. This can be done by creating an instance of type EntryComponent, which you fill with the fully qualified URL
for the resource, and the resource. Or, you could use the `AddResourceEntry` method of the `Bundle` class. This second option creates cleaner code if you only need to fill in the URL and resource. However, if you need to fill in more fields for the `EntryComponent` class, the first option can be useful.

This example shows both ways:

```javascript
var collection = new Bundle();
collection.Type = Bundle.BundleType.Collection;

var first_entry = new Bundle.EntryComponent();
first_entry.Resource = res1;
collection.Entry.Add(first_entry);

// adding a second entry
collection.AddResourceEntry(res2, "urn:uuid:01d04293-ed74-4f93-aa0a-2f096a693fb1");
```

In this example we create a Bundle with a general collection of resources, and have set the type accordingly. The first resource we want to add, `res1`, is a resource that already has a technical id. For now, we have constructed the `FullUrl` with parts of the information that’s in the resource instance, but we could also have used the helper methods for `ResourceIdentity` which are in the `Hl7.Fhir.Rest` namespace. See About resource identity for more information. The second resource we add to this collection, `res2`, is a new resource that has not been stored, and doesn’t have a technical identifier assigned to it yet. We still have to fill in the `FullUrl` field, as demanded by the `bd1-7 constraint` for `Bundle` in the specification. This is done by creating a temporary UUID, and representing that as a fully qualified URL with the `urn:uuid:` notation.

### 5.2.12 Code example for Bundle

In this paragraph, we provide a code example that sets up a Bundle that could be the response to a search request. Next to adding the resources to the `Entry` field, we have also added some extra data such as the total number of results. At the end, we have included the loop to walk through the Bundle entries.

```javascript
// example searchset Bundle setup, fictional data only
var search_response = new Bundle();
search_response.Type = Bundle.BundleType.Searchset;

// adding some metadata
search_response.Id = "[insert temporary uuid here, or real id if you store this Bundle];";
search_response.Meta = new Meta()
{  
    VersionId = "1",  
    LastUpdatedElement = Instant.Now()
};

// we assume the search has already taken place on your database, and the resulting resources are available to us in a list called 'dataset'
search_response.Total = dataset.Count;

// for searches, we need to fill in the 'self' link, that represents the search request
// as understood by the server, e.g. "http://myserver.org/fhir/Patient?name=steve"
// if you are paging the response, also fill in the other relevant links, like 'next', 'last', etc.
search_response.SelfLink = new Uri("[search request]";
```

(continues on next page)
foreach (var r in dataset)
{
    var full_url = r.ResourceBase.ToString() + r.ResourceType.ToString() + r.Id;

    // instead of using AddResourceEntry, we use the search variant to also
    include
    // the search mode
    search_response.AddSearchEntry(r, full_url, Bundle.SearchEntryMode.Match);
}

// the Bundle is now ready to be sent to the requesters

// walking through the entries in the Bundle:
foreach (var e in search_response.Entry)
{
    // Let’s write the fully qualified url for the resource to the console:
    Console.WriteLine("Full url for this resource: " + e.FullUrl);

    // Do something with this resource, for example write human readable text of
    // the resource to the console:
    var resource = (DomainResource)e.Resource;
    Console.WriteLine("Human readable text of this resource: " + resource.Text);
}

5.3 Working with REST

In this section we explain the methods of the FhirClient that are in the Hl7.Fhir.Rest part of the API package. Add this using directive to your code:

using Hl7.Fhir.Rest;

The first topics in this chapter cover the settings of the FhirClient and the CRUD interactions you can perform with it. We will then give some examples of history and search interactions, and explain how to perform operations and transactions. There’s also a section on helper methods for resource identities, and we end the chapter with other miscellaneous methods and helpers in the Hl7.Fhir.Rest namespace.

5.3.1 Creating a FhirClient

Before we can do any of the interactions explained in the next part, we have to create a new FhirClient instance. This is done by passing the url of the FHIR server’s endpoint as a parameter to the constructor:

var client = new FhirClient("http://vonk.fire.ly");

The constructor method is overloaded, to enable you to use a URI instead of a string. As second parameter to the constructor, you can specify whether the client should perform a conformance check to see if the server has got a compatible FHIR version. The default setting is false.

You’ll create an instance of a client for every server you want to work with. Every call we’ll do on this client will be for interactions with this server. Since resources may reference other resources on a different FHIR server, you’ll have to inspect any references and direct them to the right FhirClient. Of course, if you’re dealing with a single server within your organization or a single cloud-based FHIR server, you don’t have to worry about this.
There’s a list of publically available test servers you can use.

**FhirClient communication options**

To specify the preferred format – JSON or XML – of the content to be used when communicating with the FHIR server, you can use the `PreferredFormat` attribute:

```csharp
```

The FHIR client will send all requests in the specified format. Servers are asked to return responses in the same format, but may choose to ignore that request. The default setting for this field is XML.

When communicating the preferred format to the server, this can either be done by appending `_format=[format]` to the URL, or setting the `Accept HTTP header`. The client uses the latter by default, but if you want, you can use the `_format` parameter instead:

```csharp
client.UseFormatParam = true;
```

If you perform a `Create`, `Update` or `Transaction` interaction, you can request the server to either send back the complete representation of the interaction, or a minimal data set, as described in the Managing Return Content section of the specification. The default setting is to ask for the complete representation. If you want to change that request, you can set the `ReturnFullResource` attribute:

```csharp
client.ReturnFullResource = false;
```

This sets the `Prefer HTTP header` in the request to `minimal`, asking the server to return no body in the response.

You can set the timeout to be used when making calls to the server with the `Timeout` attribute:

```csharp
client.Timeout = 120000; // The timeout is set in milliseconds, with a default of 100000
```

### 5.3.2 CRUD interactions

A `FhirClient` named `client` has been setup in the previous topic, now let’s do something with it.

**Create a new resource**

Assume we want to create a new resource instance and want to ask the server to store it for us. This is done using `Create`.

```csharp
var pat = new Patient() { /* set up data */ }; var created_pat = client.Create(pat);
```

**Tip:** See *Working with the model* for examples on how to fill a resource with values.

As you’d probably expect, this interaction will throw an Exception when things go wrong, in most cases a `FhirOperationException`. This exception has an `Outcome` property that contains an `OperationOutcome` resource, and which you may inspect to find out more information about why the interaction failed. Most FHIR servers will return a human-readable error description in the `OperationOutcome` to help you out.

If the interaction was successful, the server will return an instance of the resource that was created, containing the id, metadata and a copy of the data you just posted to the server as it was stored. Depending on the server implementation,
this could differ from what you’ve sent. For instance, the server could have filled in fields with default values, if the
values for those fields were not set in your request.

If you’ve set the ReturnFullResource property of the FhirClient to false, the server will return the technical
id and version number of the newly created resource in the headers of the response and the Create method will
return null. See Looking at LastResult for an example of how to retrieve the information from the returned headers.

For the conditional version of this interaction, see Conditional interactions.

Reading an existing resource

To read the data for a given resource instance from a server, you’ll need its technical id. You may have previously
stored this after a Create, or you have found its address in a ResourceReference (e.g. Observation.Subject.
Reference).

The Read interaction on the FhirClient has two overloads to cover both cases. Furthermore, it accepts both relative
paths and absolute paths (as long as they are within the endpoint passed to the constructor of the FhirClient).

```csharp
// Read the current version of a Patient resource with technical id '31'
var location_A = new Uri("http://vonk.fire.ly/Patient/31");
var pat_A = client.Read<Patient>(location_A);
// or
var pat_A = client.Read<Patient>("Patient/31");

// Read a specific version of a Patient resource with technical id '32' and version id '4'
var location_B = new Uri("http://vonk.fire.ly/Patient/32/_history/4");
var pat_B = client.Read<Patient>(location_B);
// or
var pat_B = client.Read<Patient>("Patient/32/_history/4");
```

Tip: See the paragraph about ResourceIdentity for methods to construct URIs from separate values and other neat
helper methods.

Note that Read can be used to get the most recent version of a resource as well as a specific version, and thus covers
the two ‘logical’ REST interactions read and vread.

Updating a resource

Once you have retrieved a resource, you may edit its contents and send it back to the server. This is done using the
Update interaction. It takes the resource instance previously retrieved as a parameter:

```csharp
// Add a name to the patient, and update
pat_A.Name.Add(new HumanName().WithGiven("Christopher").AndFamily("Brown");
var updated_pat = client.Update(pat_A);
```

There’s always a chance that between retrieving the resource and sending an update, someone else has updated the re-
source as well. Servers supporting version-aware updates may refuse your update in this case and return a HTTP status
code 409 (Conflict), which causes the Update interaction to throw a FhirOperationException with the same
status code. Clients that are version-aware can indicate this using the optional second parameter versionAware set
to true. This will result in a conditional call of the interaction.
Deleting a Resource

The `Delete` interaction on the FhirClient deletes a resource from the server. It is up to the server to decide whether the resource is actually removed from storage, or whether previous versions are still available for retrieval. The `Delete` interaction has multiple overloads to allow you to delete based on a url or a resource instance:

```csharp
// Delete based on a url or resource location
var location = new Uri("http://vonk.fire.ly/Patient/33");
client.Delete(location);

// or
client.Delete("Patient/33");

// You may also delete based on an existing resource instance
client.Delete(pat_A);
```

The `Delete` interaction will fail and throw a `FhirOperationException` if the resource was already deleted or if the resource did not exist before deletion, and the server returned an error indicating that.

Note that sending an update to a resource after it has been deleted is not considered an error and may effectively “undelete” it.

5.3.3 Conditional interactions

The API provides support for the conditional versions of the `Create`, `Update` and `Delete` interactions. Not all servers will support conditional interactions and can return an HTTP 412 error with an `OperationOutcome` to indicate that.

All of the conditional interactions make use of search parameters. See the page of the resource type you want to work with in the HL7 FHIR specification to check which search parameters are available for that type. Then, setup the conditions.

For example, if we want to base the interaction on the `identifier` element of a resource, we can setup that search parameter with a value:

```csharp
var conditions = newSearchParams();
conditions.Add("identifier", "http://ids.acme.org|123456");
```

**Tip:** See search for more explanation about `SearchParams` and example search syntax.

For the `Create` interaction you can have the server check if an equivalent resource already exists, based on the search parameters:

```csharp
var created_pat_A = client.Create<Patient>(pat, conditions);
```

If no matches are found, the resource will be created. If one match is found, the server will not create the resource and will return an HTTP 200 (OK). In both cases `created_pat_A` will contain the resource that was sent back by the server, unless you set the FhirClient to ask for the `minimal representation`. When multiple resources match the conditions, the server will return an error.

To perform a conditional `Update`, the code is similar to that of the `Create` interaction above. Again, setup a `SearchParams` object and add it to your request:

```csharp
// using the same conditions as in the previous example
var updated_pat_A = client.Update<Patient>(pat, conditions);
```
If a match is found, the update is performed on that match. If no matches are found, the server will perform the interaction as if it were a Create. When multiple resources match, the server will return an error.

The conditional Delete takes a string as first argument, indicating the resource type. The search parameters are passed as second argument:

```csharp
client.Delete("Patient", conditions);
```

When no match is found, the server will return an error. If one match is found, that resource will be deleted. The server may choose to delete all resources if multiple instances match, or it may return an error.

---

### 5.3.4 Refreshing data

Whenever you have held a resource for some time, its data may have changed on the server because of changes made by others. At any time, you can refresh your local copy of the data by using the Refresh call, passing it the resource instance as returned by a previous Read, Create, or Update:

```csharp
var refreshed_pat = client.Refresh(pat_A);
```

This call will go to the server and fetch the latest version and metadata of the resource as pointed to by the Id property in the resource instance passed as the parameter.

---

### 5.3.5 Looking at LastResult

After the FhirClient has received a response from the server, you will usually work with the resource instance that was returned. But if you have set the ReturnFullResource option to false, the server will not return a resource on Create and Update interactions. If you still need to check the response from the server, for example to lookup the technical id or version id the server has assigned to your resource instance, you can do this by looking at the LastResult property of the FhirClient:

```csharp
client.Create(pat);
if (client.LastResult.Status == "201")
{
    Console.WriteLine("The location for the resource is: " + client.LastResult.
    +Location);
}
```

---

### 5.3.6 FhirClient event handlers

The FhirClient provides you with two event handlers, which you can use to hook into the request/response cycle. With these handlers you can implement extra code to be executed right before a request is sent, or directly after a response has been received.

**Adding extra headers**

It could be necessary to add extra headers to the requests your FhirClient sends out, for example when you want to send an authorization token with your request. Or perhaps you need other code to be executed each time the FhirClient sends a request. You can use the OnBeforeRequest event in these cases:
Extra code after response

The `OnAfterResponse` event can be used to add extra code that needs to be executed every time a response is received by the FhirClient:

```csharp
client.OnAfterResponse += (object sender, AfterResponseEventArgs e) =>
{
    Console.WriteLine("Received response with status: " + e.RawResponse.StatusCode);
};
```

5.3.7 Retrieving resource history

There are several ways to retrieve version history for resources with the FhirClient.

**Note:** Servers are not required to support version retrieval. If the `history` interaction is supported, the server can choose on which level it is supported. You can check the `CapabilityStatement` of the server to see what it supports.

**History of a specific resource**

The version history of a specific resource can be retrieved with the `History` method of the FhirClient. It is possible to specify a date, to include only the changes that were made after the given date, and a count to specify the maximum number of results returned.

The method returns a Bundle resource with the history for the resource instance, for example:

```csharp
var pat_31_hist = client.History("Patient/31");
// or
var pat_31_hist = client.History("Patient/31", new FhirDateTime("2016-11-29").ToDateTimeOffset());
// or
var pat_31_hist = client.History("Patient/31", new FhirDateTime("2016-11-29").ToDateTimeOffset(), 5);
```

**Note:** The Bundle may contain entries without a resource, when the version of the instance was the result of a `delete` interaction.

**History for a resource type**

Sometimes you may want to retrieve the history for a `type` of resource instead of an instance (e.g. the versions of all Patients). In this case you can use the `TypeHistory` method.
As with the method on the instance level, a date and page size can optionally be specified.

**System wide history**

When a system wide history is needed, retrieving all versions of all resources, the FhirClient's `WholeSystemHistory` method is used. Again, it is possible to specify a date and a page size.

```csharp
var lastMonth = DateTime.Today.AddMonths(-1);
var last_month_hist = client.WholeSystemHistory(since: lastMonth, pageSize: 10);
```

In this case the function retrieves all changes to all resources that have been done since the last month and limits the results to a maximum of 10. See *Paged results* for an example on how to page through the resulting Bundle.

### 5.3.8 Paged results

Normally, any FHIR server will limit the number of results returned for the `history` and `search` interactions. For these interactions you can also specify the maximum number of results you would want to receive client side.

The FhirClient has a `Continue` method to browse a search or history result Bundle, after the first page has been received. `Continue` supports a second parameter that allows you to set the direction in which you want to page: forward, backward, or directly to the first or last page of the result. The standard direction is to retrieve the next page. The method will return `null` when there is no link for the chosen direction in the Bundle you provide.

```csharp
while (result != null)
{
    // Do something with the entries in the result Bundle
    // retrieve the next page of results
    result = client.Continue(result);
}
```

```csharp
// go to the last page with the direction filled in:
var last_page = client.Continue(result, PageDirection.Last);
```

### 5.3.9 Searching for resources

FHIR has extensive support for searching resources through the use of the REST interface. Describing all the possibilities is outside the scope of this document, but much more details can be found online in the specification.

The FHIR client has a few operations to do basic search.

**Searching within a specific type of resource**

The most basic search is the client’s `Search<T>(string[] criteria = null, string[] includes = null, int? pageSize = null)` function. It searches all resources of a specific type based on zero or more criteria. Criteria must conform to the parameters as they would be specified on the search URL in the REST interface, so for example searching for all patients named ‘Eve’ would look like this:

```csharp
Bundle results = client.Search<Patient>(new string[] { "family:exact=Eve" });
```
The search will return a Bundle containing entries for each resource found. It is even possible to leave out all criteria, effectively resulting in a search that returns all resources of the given type. Additionally, there is a `Search()` overload that does not use the generic `T` argument, you can pass the type of resource as a string in the first parameter instead.

**Searching for a resource with a specific id**

In some cases you may already have the id of a specific resource (e.g. an Observation with logical id 123, corresponding to the url `Observation/123`). In this case you can use `SearchById<T>(string id, string[] includes = null, int? pageSize = null)`.

Note that this function still returns a Bundle. The operation differs from a `Read<T>()` operation because it can return included resources as well. E.g. given an id 123 for an Observation, you can ask a FHIR server to not only look for the indicated Observation but to return the associated subject as well:

```csharp
var incl = new string[] { "Observation.subject" };
Bundle results = client.SearchById<Observation>("123", incl);
```

**System wide search**

Some servers allow you to execute searches across all resource types. This would use FhirClient’s `WholeSystemSearch(string[] criteria = null, string[] includes = null, int? pageSize = null)`.

Doing this search:

```csharp
Bundle results = client.WholeSystemSearch(new string[] { "name=foo" });
```

would then not only return Patients with “foo” in their name, but Devices named “foo” as well.

**Complex searches**

An alternative way to specify a query is by creating a `Query` resource and pass this to the client’s `Search(Query q)` overload. The `Query` resource has a set of fluent calls to allow you to easily construct more complex queries:

```csharp
var q = new Query()
    .For("Patient")
    .Where("name:exact=ewout")
    .OrderBy("birthDate", SortOrder.Descending)
    .SummaryOnly().Include("Patient.managingOrganization")
    .LimitTo(20);

Bundle result = client.Search(q);
```

Note that unlike the search options shown before, you can specify search ordering and the use of a summary result. As well, this syntax avoids the need to create arrays of strings as parameters and tends to be more readable.
Paged Results

Normally, any FHIR server will limit the number of search results returned. In the previous example, we explicitly limited the number of results per page to 20.

The FhirClient has a `Continue` function to browse a search result after the first page has been received using a `Search`:

```csharp
var result = client.Search(q);
while (result != null)
{
    // Do something useful
    result = client.Continue(result);
}
```

Note that `Continue` supports a second parameter that allows you to browse forward, backward, or go immediately to the first or last page of the search result.

5.3.10 About resource identity

Read only takes urls as parameters, so if you have the Resource type and its Id as distinct data variables, use `ResourceIdentity`:

```csharp
var patResultC = client.Read<Patient>(ResourceIdentity.Build("Patient","33"));
```

5.4 Parsing and serialization

The .NET API makes it easy to work with XML and Json-based FHIR data. There are two approaches for getting data in and out of your application:

- Work with the POCO classes (as described in Working with the model). These are .NET classes that represent the FHIR resources mostly one-on-one.
- Work with the `ElementModel` classes, which is an abstract memory model representing the FHIR data as an in-memory tree of data.

The first approach is the simplest and is most applicable if you prefer working with strongly typed classes that align with the FHIR resources. E.g. there is a class `Patient` with a property `name`, as you would expect from looking at the FHIR documentation. For most users, this is all they need.

However, there are several reasons why the POCO-based approach may not work for you:

- The generated POCO classes are based on a specific version of FHIR, so if you need to deal with FHIR data independent of versions, POCO’s are cumbersome to work with.
- The parsers for POCO classes cannot deal with incorrect data - there is no way to express invalid FHIR data as a POCO, so if you will get parser errors, there is no way to recover or correct the data.
- You may only be working with FHIR data in the sense that you need to be able to parse, persist and retrieve bits of data, without needing the full overhead of creating POCOs in memory.
- You need to be able to customize data when parsing or serializing data.

The second approach, using the `ElementModel` abstraction, has been designed to work with these usecases. In fact, the POCO parsers are built on top of the more low-level `ElementModel` classes.
5.4.1 Parsing with POCOs

Start by add this `using` directive to your code:

```csharp
using Hl7.Fhir.Serialization;
```

You will now have access to the parsers and serializers for XML and Json:

- For XML: `FhirXmlParser` and `FhirXmlSerializer`
- For Json: `FhirJsonParser` and `FhirJsonSerializer`.

The way you would work with these does not differ much between XML or Json, so this section will just show you how to work with XML.

First, let us parse a bit of XML representing a FHIR Patient into the corresponding `Patient` class:

```csharp
var xml = "<Patient xmlns='http://hl7.org/fhir'><active value='true'></Patient>";
var parser = new FhirXmlParser();
try {
    var parsedPatient = parser.Parse<Patient>(xml);
    Console.WriteLine(parsedPatient.Active);
} catch (FormatException fe) {
    // the boring stuff
}
```

In the example above, we knew the data contained a patient, but it is perfectly alright to be less specific and work with the `Resource` base class instead:

```csharp
Resource parsedResource = parser.Parse<Resource>(xml);
```

You can then use C# constructions like the `is` operator to make your code behave differently depending on the type of resource parsed.

The `Parse` method has a few overloads, one of which allows you to pass in an `XmlReader` instead of a string, which makes sense if you have a stream of data that you don’t want to read into a string first.

### POCO’s and parsing incorrect data

The POCO parsers are pretty strict about what data they will accept: since the data read and parsed must fit the POCO structure there is little room in allowing incorrect FHIR data. It is possible to allow a bit of flexibility however, which is controlled by passing a `ParserSettings` instance to the constructor of the xml or json parser:

```csharp
var parser = new FhirXmlParser(new ParserSettings { AcceptUnknownMembers = true, AllowUnrecognizedEnumerations = true });
```

`AcceptUnknownMembers` will ensure the parser does not throw an exception when an instance of FHIR data contains an unknown (or incorrectly spelled) property. This is particularly useful when you want to make sure that your software will be able to read data from newer FHIR versions: for normative parts of the FHIR specification, all existing properties will remain unchanged, but newer versions of FHIR might add new members. By settings this property to `true`, you can make sure your older software can still read newer data. There is, however, no way to access the unrecognized data.

The same is true for `AllowUnrecognizedEnumerations`. When the parser turns a coded value (say `Patient.gender`) into an `enum`, the parser will allow values that are not part of the enumeration, and can therefore not be turned into an...
This means that the property will return a null value - you can, however, get to the ‘unrecognized’ value using the `ObjectValue` backing field, as demonstrated in the code below:

```csharp
var parser = new FhirXmlParser(new ParserSettings {
    AllowUnrecognizedEnums = true });
p = parser.Parse<Patient>(xml2); // xml2 contains a Patient.gender of 'superman'
Assert.IsNull(p.Gender); // p.Gender will now be null
Assert.AreEqual("superman", p.GenderElement.ObjectValue); // but you can query the
backing value
```

### 5.4.2 Serialization with POCOs

Serialization, unsurprisingly, turns a given POCO back into Json or XML, and is handled by either `FhirXmlSerializer` or `FhirJsonSerializer`. Both classes have several methods to serialize the POCO into different forms:

- `SerializeToString`, `SerializeToBytes` - will turn the POCO into an XML/Json string or directly into a UTF-8 encoded byte representation.
- `Serialize` - writes the POCO to an XmlWriter
- `SerializeToDocument` - turns the POCO into an `XDocument` or `JObject`

Continuing the previous example, we can change some value in the parsed Patient and then serialize it back out:

```csharp
parsedPatient.active = false;
var serializer = new FhirXmlSerializer();
var xmlText = serializer.SerializeToString(parsedPatient);
```

Note that creating a new `FhirXmlSerializer` (or `FhirXmlParser`) is cheap. The constructor for the `FhirXmlSerializer` and `FhirJsonSerializer` take a single parameter to change settings, most notably to ‘pretty print’ your output:

```csharp
var serializer = new FhirJsonSerializer(new SerializerSettings() {
    Pretty = true
});
```

### Summaries

The FHIR specification introduces several summary versions of resources. You can serialize a POCO into one of these summary forms by passing the `summary` parameter to any of the serialization methods described above:

```csharp
var xml = serializer.SerializeToString(b, summary: Fhir.Rest.SummaryType.Text);
```

### Convenience methods

**Caution:** This documentation describes features in a prelease of version 1.0 of the API. The documentation may be outdated and code examples may become incorrect.

Although the code examples above are simple enough, there is also a set of extension methods available on POCOs to make serialization even easier, without the need of explicitly creating a serializer:
## 5.4.3 Introduction to ElementModel

While most developers will be most comfortable with using .NET POCOs to work with FHIR data, the .NET API itself largely uses the ElementModel classes to read and manipulate data. These classes enable the API to work independently of FHIR versions (e.g. DSTU2, STU3, R4, etc), and can even work with incorrect data. In addition, these classes make it simple to traverse data and obtain type information about the data without the need for .NET reflection.

### The ElementModel interfaces

The ElementModel namespace contains two interfaces that represent FHIR data as a tree of nodes, each node containing children, primitive data, or both. They are `ISourceNode` and `ITypedElement`. The former is an abstraction on top of the serialized formats (currently XML, Json and RDF), whereas the second represents the strongly typed logical FHIR data model. The parsing API has a low-level `ISourceNode` implementation for each serialization format. The API then allows you to turn an untyped, low-level tree represented by the `ISourceNode` interface into an `ITypedElement` based typed tree by adding type information to it. Both interfaces can be found in the `Hl7.Fhir.ElementModel` assembly and namespace.

The next sections will details how to use the `ISourceNode` based parsers in the API, how to add type information and serialize data back out using the `ITypedElement` based serializers.

## 5.4.4 Parsing with ISourceNode

This interface exposes serialization-level, untyped instance data of a single resource at a level that abstracts away the specific details of the underlying representation (e.g. xml and json) and is shown below:

```csharp
interface ISourceNode
{
  string Name { get; }
  string Text { get; }
  string Location { get; }
  IEnumerable<ISourceNode> Children(string name = null);
}
```

The interface represents a single node in the tree. Each node has a name, and a location (as a dot separated list of paths leading to the current node). The property `Text` contains the node’s primitive data (if any).

Note that the name of the node is often, but not always, the same as the name of the element in the FHIR specification. The two may differ if the element has a choice of types (e.g. `Observation.value`). In this case, the name of the source node is suffixed by the type, exactly as it would be in the serialized form. For the root of the tree, the name is the same as the type of the resource present in the instance.
Navigation through the tree is done by enumerating the children using the `Children()` method, optionally filtering on the name of the children.

Below is a tree which shows the instance data for an example Observation:

```json
{
    "resourceType": "Observation",
    "code": {
        "coding": [
            { "code": "7113001", "system": "http://snomed.info/sct"
            }, { "et" },
            { "coding": [ "et" ]
        ]
    }
    "valueQuantity": {
        "value": "185"
    }
}
```

The tree represented by the `ISourceNode` can be represented like this:

Some of the more subtle points illustrated here are:

- The root of the tree is named after the type of the instance (visible in Json as “resourceType”, and a root element in Xml)
- There is no explicit representation of an array of repeated elements, arrays are flattened into sibling nodes with the same name, just like repeating elements in Xml.
- The location ensures each node is named uniquely by suffixing each path with an array index ([0], [1], etc), even if the given element does not repeat according to the FHIR specification (remember, at this level we do not have type information, nor are we aware of the differences between different versions of FHIR, so this information is simply not available).
- The choice element value, has its instance type (Quantity) appended to it: the name of the node in the tree agrees with the name of the element in the Json (or xml) serialization.

The API offers a set of extension methods on top of `ISourceNode` (like `Visit()` and `Descendants()`) to make it easier to select subtrees and process the data in the tree.
Parsing

The FHIR parsers available (currently for the FHIR Xml and Json formats) implement the ISourceNode interface and can be found in the Hl7.Fhir.Serialization assembly. The parsers are not created directly, instead there are two sets of factory methods, one for each serialization format: FhrXmlNode and FhirJsonNode. The factory methods are:

- **Read()**, to read data directly from an XmlReader or JsonReader.
- **Parse()**, to parse data from a string.
- **Create()**, to turn XElements, XDocuments or JOObjects into an ISourceNode

All methods optionally take a settings parameter and return an ISourceNode, which represents the root of the data read.

Here is an example parsing a string of xml and then querying some of its data:

```csharp
var xml = "<Patient xmlns="http://hl7.org/fhir">" +
    "<identifier>
        <use value="official" />
    </identifier></Patient>";
var patientNode = FhirXmlNode.Parse(xml);
var use = patientNode.Children("identifier").Children("use").First();
Assert.AreEqual("official", use.Text);
Assert.AreEqual("Patient.identifier[0].use[0]", use.Location);
```

By swapping out the FhirXmlNode for an FhirJsonNode you can make this example to read Json data - there would not be any change to the rest of the code.

Constructing a tree in memory

It is also possible to construct an in-memory tree with data “by hand”, using the SourceNode class. Since SourceNode implements ISourceNode, there would not be any difference from data read from a file or other source:

```csharp
patient = SourceNode.Node("Patient",
    SourceNode.Valued("active", "true"),
    annotatedNode,
    SourceNode.Valued("id", "myId2"),
    SourceNode.Node("extension",
        SourceNode.Valued("value", "4")),
    SourceNode.Node("extension",
        SourceNode.Valued("value", "world!")));
```

Note that by using the C# using static Hl7.Fhir.ElementModel.SourceNode; this example could be make quite a bit shorter.

Handling parse errors

By default, parsing errors thrown as exceptions, but all parsers implement IExceptionSource to alter this behaviour. See errorhandling for more information.

The parsers try to parse the source lazily, so in order to detect all parse errors, one would have to do a complete visit of the tree, including forcing a read of the primitive data by getting the Text property. There is a convenience method VisitAll() that does exactly this. Additionally, there is a method VisitAndCatch() that will traverse the whole tree, returning a list of parsing errors and warnings.
5.4.5 Working with ITypedElement

The main difference between ISourceNode (see Parsing with ISourceNode) and ITypedElement is the presence of type information: metadata coming from the FHIR specification about which elements exist for a given version of FHIR, whether they repeat, what the type of each element is, etcetera. Type information is necessary for many operations on data, most notably serialization, running FhirPath statements and doing validation. As such, it is more common to work with ITypedElement than it is to work with ISourceNode. However, as one could imagine, in cases where type information is not necessary, ISourceNode is more performant and has a smaller memory footprint.

This is what ITypedElement looks like:

```csharp
public interface ITypedElement
{
    string Name { get; }
    object Value { get; }
    string Location { get; }

    string InstanceType { get; }
    IElementDefinitionSummary Definition { get; }

    IEnumerable<ITypedElement> Children(string name=null);
}
```

Just like ISourceNode, the interface represents a single node in the tree, has a name and a location and you can enumerate its children using Children().

Unlike ISourceNode however, type information is available by looking at the InstanceType and Definition properties. As a consequence, the Name property now returns the actually defined name of the element in the specification, so for choice types the element would not include the type suffix. For example, Observation.valueQuantity in an ISourceNode would turn into Observation.value in ITypedElement.

Similarly, the primitive value of the node (if any) is now of type object, and returns the value of the primitive, represented as a native .NET value. The following table lists the mapping between the encountered FHIR primitive and the .NET type used to represent the value:

<table>
<thead>
<tr>
<th>FHIR Type</th>
<th>.NET type</th>
</tr>
</thead>
<tbody>
<tr>
<td>instant</td>
<td>Hl7.Fhir.Model.Primitive.PartialDateTime</td>
</tr>
<tr>
<td>date, dateTime</td>
<td>Hl7.Fhir.Model.Primitive.PartialDateTime</td>
</tr>
<tr>
<td>decimal</td>
<td>decimal</td>
</tr>
<tr>
<td>boolean</td>
<td>bool</td>
</tr>
<tr>
<td>integer</td>
<td>long</td>
</tr>
<tr>
<td>unsignedInt</td>
<td>long</td>
</tr>
<tr>
<td>positiveInt</td>
<td>long</td>
</tr>
<tr>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td>code</td>
<td>string</td>
</tr>
<tr>
<td>id</td>
<td>string</td>
</tr>
<tr>
<td>uri, oid, uuid, canonical, url</td>
<td>string</td>
</tr>
<tr>
<td>markdown</td>
<td>string</td>
</tr>
<tr>
<td>base64Binary</td>
<td>string (uuencoded)</td>
</tr>
<tr>
<td>xhtml</td>
<td>string</td>
</tr>
</tbody>
</table>

Note that Location is exactly the same on both interfaces - every part of the path still has an array index,
whether the element repeats or not. If you would like to have a shortened path, where non-repeating elements have their array index removed, you can check whether the underlying implementation of `ITypedElement` implements `IShortPathGenerator` (which the implementations from the API do), and get its `ShortPath` property.

**Important:** The `IElementDefinitionSummary` interface returned by the `Definition` property is very likely still to change. You are welcome to experiment with it and provide feedback, but the next release of the API will most likely add (incompatible) capabilities.

The API offers a set of extension methods on top of `ITypedElement` (like `Visit()` and `Descendants()`) to make it easier to select subtrees and process the data in the tree.

### Obtaining an `ITypedElement`

The API enables you to turn data in POCO or `ISourceNode` form into an `ITypedElement` by calling the `ToTypedElement()` extension method.

In the first case, the POCO has all additional type information available (being based on a strongly-typed object model), and simply surface this through the `ITypedElement` interface. In the second case, the API needs an external source of type information to associate type information to the untyped nodes in the `ISourceNode` tree. The `ToTypedElement` method on `ISourceNode` looks like this:

```csharp
public static ITypedElement ToTypedElement(ISourceNode node,
                                          IStructureDefinitionSummaryProvider provider,
                                          string type = null,
                                          TypedElementSettings settings = null);
```

Notice that the `provider` parameter is used to pass in type information structured by the `IStructureDefinitionSummaryProvider` interface. Currently, the API supplies two implementations of this interface:

- The `PocoStructureDefinitionSummaryProvider`, which obtains type information from pre-compiled POCO classes. This is very similar to calling `ToTypedElement()` on a POCO, but this method does not require the caller to have data present in POCOs.

- The `StructureDefinitionSummaryProvider`, which obtains type information from `StructureDefinitions` provided with the core specification and additional Implementation Guides and packages. The constructor for this provider needs a reference to an `IResourceResolver`, which is the subsystem used to get access to FHIR’s metadata resources (like `StructureDefinition`). See `specification-sources` for more information about `IResourceResolver`.

This is a complete example showing how to turn the `patientNode` from the last section into a `ITypedElement` by using external metadata providers:

```csharp
ISourceNode patientNode = ...
IResourceResolver zipSource = ZipSource.CreateValidationSource();
ITypedElement patientRootElement = patientNode.ToTypedElement(zipSource);
ITypedElement activeElement = patientRootElement.Children("active").First();
Assert.AreEqual("boolean", activeElement.Type);
```

### Compatibility with `IElemnetNavigator`

Previous versions of the API defined and used the precursor to `ITypedElement`, called `IElemnetNavigator`. Though functionally the same, `ITypedElement` is stateless, whereas `IElemnetNavigator` was not. To aid in parallelization, we have chosen to obsolete the stateful `IElemnetNavigator` in favor of `ITypedElement`. At this moment, not all parts of the API have been rewritten (yet) to use the new `ITypedElement` and we expect
the same is true for current users of the API. To aid in migration from one concept to the other, the API provides a set of adapters to turn `IElementNavigators` into `ITypedElements` and vice versa. These can be constructed by simply calling `ToElementNavigator()` on a `ITypedElement` or `ToTypedElement()` on an `IElementNavigator`. The compiler will emit messages about this interface being obsolete to stimulate migration to the new paradigm.

**Handling structural type errors**

While traversing the `ITypedElement` tree, the implementations will try to associate type information from the specification with the data encountered. If this fails, errors are by default thrown as exceptions, but the all underlying implementations of `ITypedElement` implement `IExceptionSource` to alter this behaviour. See errorhandling for more information.

Detecting type errors is done *lazily*, so in order to detect all errors, one would have to do a complete visit of the tree, including forcing a read of the primitive data by getting the `Value` property. There is a convenience method `VisitAll()` that does exactly this. Additionally, there is a method `VisitAndCatch()` that will traverse the whole tree, returning a list of errors and warnings.

### 5.4.6 Serializing `ITypedElement` data

The API provides functionality to turn `ITypedElement` data into XML and JSON formats using the following set of extension methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToJson()</td>
<td>string</td>
</tr>
<tr>
<td>JsonBytes()</td>
<td>byte[]</td>
</tr>
<tr>
<td>JObject()</td>
<td>JObject</td>
</tr>
<tr>
<td>WriteTo()</td>
<td>JsonWriter</td>
</tr>
<tr>
<td>ToXml()</td>
<td>string</td>
</tr>
<tr>
<td>XmlBytes()</td>
<td>byte[]</td>
</tr>
<tr>
<td>XDocument()</td>
<td>XDocument</td>
</tr>
<tr>
<td>WriteTo()</td>
<td>XmlWriter</td>
</tr>
<tr>
<td>ToPoco()</td>
<td>Base</td>
</tr>
<tr>
<td>ToPoco&lt;T&gt;()</td>
<td>T (where T:Base)</td>
</tr>
</tbody>
</table>

The last two methods deserve a bit of explanation: from the standpoint of the serializers in the API, POCOs are also an output format. This means that in addition to the XML and JSON serializers, there are methods to turn `ITypedElement` data directly into POCO’s. Continuing with last section’s example:

```csharp
ITypedElement patientRootElement = patientNode.ToTypedElement(zipSource);
string xml = patientRootElement.ToXml();
Patient p = patientRootElement.ToPoco<Patient>();
```

It will come as no surprise that the higher level serializers and parsers described in poco-parsing are thin convenience methods, simply calling the more primitive methods described in the last sections.

### Working with subtrees

It is possible to traverse into a tree, away from the Resource root and then call one of the serialization methods above, with a caveat: the FHIR specification does not specify a “standard” serialization for subtrees, so there will not always be a “natural” way to serialize a subtree. More specifically, there is no way to serialize a subtree to JSON that
represents a primitive, but this is quite possible in XML. The same is true for parsing into a subtree. The API will raise an error at runtime if the subtree cannot be represented in the chosen serialization.

**Round-trip serialization**

The FHIR serialization formats need type information to work correctly. For example, repeating elements require the use of an array in JSON, while narrative in XML uses a different (xhtml) namespace. This is the reason that under most circumstances, serialization needs to be done based on the type-aware `ITypedElement` interface. However, when working with a single format (or doing a direct round-trip), the xml and json parsers annotate the (untyped) `ISourceNode` tree with enough information to allow for a serialization without type information, resulting in a cheaper round-trip than would be possible if the user has to retrieve and add type information first. As such, the serializers mentioned in this section have overloads on their methods to take a `ISourceNode` as well. This only works for round-trips however, so you will get a runtime exception when trying to serialize an `ISourceNode` to XML when it was created from JSON.

### 5.4.7 ElementModel Summary

The interfaces and methods described in this section are summarized in the picture below. Note that there are more conversions possible than shown (e.g. reading from an `XmlReader` is not explicitly depicted), and the API may be extended to support formats other than XML and JSON by the time you read this.

![Diagram of ElementModel Summary](image)

### 5.4.8 Writing your own `ITypedElement` implementation

The components that make up the parsing and serialization stack of the .NET framework are designed to be composed to be provide flexibility for different usecases and make it easy to add additional behaviour. You could build a pipeline converting XML to JSON by starting out with the `FhirXmlNode` (implementing `ISourceNode`), calling `ToTypedElement()` on it (effectively wrapping it in an internal implementation of `ITypedElement` called `TypeElement`). This is then again wrapped by the `FhirJsonBuilder` class, which turns the output into JSON.

---

**5.4. Parsing and serialization**
This pipeline can be extended by custom implementations of both interfaces, and the API already contains a few extra’s out of the box:

- The **MaskingNode**, which wraps another ITypeElement and which represents a tree pruned to just the nodes marked with isSummary. It could also be extended to mask out data based on the user’s authorization.

- The **ScopedNode**, which tracks parent/child relationships and keeps track of the nearest “parent resource” (amongst other things) while traversing the tree. This information is used by both the validator and the FhirPath evaluation engine.

The [MaskingNode source code](#) is an excellent place to start exploring the possibilities provided by the framework.

### 5.5 Contact us

We actively monitor the issues coming in through the GitHub repository. You are welcome to register your bugs and feature suggestions there!

We are also present on chat.fhir.org on Zulip, on the dotnet stream, or for more general implementation questions, the implementers stream.

For broader discussions, there’s a “FHIR on .NET” [Google group](#).

You can also ask questions by sending an e-mail to fhirapi@fire.ly.

If you’re interested in a FHIR training, or want to know what other FHIR tools are available, please check [here](#).

### 5.6 Release notes

#### 5.6.1 1.2.1 (STU3, R4) (released 20190416)

The fix for issue #889 should have been in version 1.2.0, but it was not because of a merge error. In this hotfix (1.2.1) we corrected this mistake and added the fix for issue #889.

**Bugfixes**

- #889 Additional rules will get confused by hybrid typed trees

#### 5.6.2 1.2.0 (DSTU2, STU3, R4) (released 20190329)

**New Functionality**

- Support for R4 final (4.0.0)
- #748 You can now set the ParserSettings used by the DirectorySource
- #756 Type-aware validations can now also be run by the POCO parser
- #924 Performance improvements
Bugfixes

- #907 Transaction code documentation was wrong
- #833 FhirClient.Read would return null when server not responding
- #896 StructureDefinitionElementDefinitionSummaryGenerator would not always derive IsRequired correctly
- #889 Additional rules will get confused by hybrid typed trees
- #888 TypedElementToSourceNodeAdapter should return ISourceNode annotation
- #872 Quantity now binds to <code>, not <unit>

Note: The parsing subsystem now catches way more error in the FHIR xml/json syntax. Since this would raise errors in instances that previously parsed without problems, this feature is turned off by default. Turn it on by setting ParserSettings.PermissiveParsing to “false”.

5.6.3 1.2.0-beta2 (DSTU2, STU3, R4) (released 20190228)

Because of a mistake in creating the release of 1.2.0-beta1 for STU3, this beta1 release was not correct. The other 1.2.0-beta1 releases (DSTU2 and R4) were properly released. In STU3 you got a System.MissingFieldException during parsing a resource from xml to poco. This has been fixed now.

- Fix: Parsing of the primitive type decimal is done better now.

5.6.4 1.2.0-beta1 (DSTU2, STU3, R4) (released 20190220)

This first beta of 1.2 includes support for FHIR version R4! In the coming period we will investigate the performance of this release and take the last bugs out (if there are any :) ). We will probably release a second beta, or if we are real confident about this beta1, we will create a final release 1.2.0.

- Fix: #750 The WebResolver should allow clients to configure custom ParserSettings for de-serialization
- Fix: #748 The DirectorySource should allow clients to configure ParserSettings for de-serialization
- Fix: #863 ParserSettings.Clone() does not copy PermissiveParsing
- #846 Enable cross-platform development
- Fix: #814 Empty contained block leads to NullReferenceException
- Fix: #824 Type-slice shortcut for choices in R4 does not yet work
- Added: #853 R4 Snapshot Generator
- Fix: #854 ElementDefinition.Base component #854
- Fix: #864 Validation of References to contained resources where there are multiple potential types, and the contained resource is not the first one targeted.
- Fix: #827 [Snapshot Generator] Child constraints are not included in snapshot when constraining a .value element
- Fix: #820 Fixed IgnoreUnknownElements naming inconsistency
- Improvement: #821 Better error reporting by always mentioning the context of the error
- Fix: #725 Debugger display doesn’t work as the property has been renamed
- Fix: #817 Setting an empty Meta object breaks Json Serialization
- Fix: #756 Validation issue with text.div
• Fix: #807 FhirPath indexOf returns -1 instead of empty.

5.6.5 1.1.3 (DSTU2, STU3) (released 20190213)

Hotfix release

• Fix: ParserSettings.PermissiveParsing was not copied in Clone() and constructor

5.6.6 1.1.2 (DSTU2, STU3) (released 20190131)

Hotfix release

• Fix: v2-tables.xml of specification.zip contained an invalid codesystem (id =v2-0550)

5.6.7 1.1.1 (DSTU2, STU3) (released 20190130)

Hotfix release

• #817 Setting an empty Meta object breaks Json Serialization
• Fix: Added PermissiveParsing setting to ParserSettings
• Fix: Setting primitives to null AND using extensions would crash ITypedElement.Value

Note: Since 1.0, the parsers are more strict than before in what they accept as input. To disable this behaviour, set PermissiveParsing to ‘true’ in the ParserSettings which can be passed to the Json/Xml POCO parsers.

5.6.8 1.1.0 (DSTU2, STU3) (beta - final version to be released 20190128)

This is a minor release.

### New functionality

• #180 We added a method to retrieve a contained resource on the parent resource by url.
• #460 Add support for _elements on the POCO serializer
• #731 FhirDateTime.ToDateTimeOffset() now asks you to pass in a TimeSpan - previously UTC was assumed.
• #578 DirectorySource will now resolve a specific version of a conformance resource if the canonical is versioned.
• #726 Add type-less overload for the POCO Parse() methods to mean “expect any resource type”.
• #773 Added new FhirPath functions to support new invariants in R4.

### Bugfixes

• #339 The Patient search parameter for some resources had an incorrect target list in ModelInfo
• #553 The namespace of the internal Sprache parser in the FhirPath assembly has been moved to a new namespace to avoid conflicts when also using the external assembly.
• #559 The LocalTerminologyService no longer throws an exception when there is no code at all passed to it.
• #561 The FhirPath compiler keeps a cache of recently encountered expressions. Multi-threaded access to it has been sped up.
• #624 Validator will now give a warning instead of an error if the display in a code is different from the one in the codesystem.
• #718 Corrected ‘|’ or “or” in invariants in DSTU2/STU3.
• #732 Json serialization no preserves significant digits.
• #746 ModelInfo.IsCoreModelTypeUri did not handle relative urls well.
• #754 Roundtripping patient.ToTypedElement().ToPoco<Patient>() failed.
• #755 Arguments to FhirClient.Search() with a key/value pair without value would throw an exception.
• #793 Element and Backbone element were handled differently in the ClassMappings that feed the parsers.
• #794 GetResourceFormatFromContentType would throw a null reference exception when the content-type had non-alphanumeric characters.

5.6.9 1.0.0 (DSTU2, STU3) (20181217)

This large release fixes about 80 issues - but more importantly introduces a completely new parsing/serialization subsystem that allows you to work without POCOs and also is more strictly following the serialization rules for XML and Json. This means you will get parse errors on instances that where (incorrectly) accepted as correct by the older versions of the API. More information on the new parsing subsystem can be found in the documentation. Please note that we have strived to keep the existing top-level POCO-parsing API intact - your projects should still compile without problems.

• #248 Json output can now be formatted
• #356 Parsing/serialization subsystem replaced to support working without using the generated POCO classes.
• #400 TransactionBuilder.Delete would cause “Invalid resource URL” in some circumstances.
• #433 Made the interface of all settings-related classes consistent across the whole API surface.
• #483 Introduction of IErrorSource to facilitate forwarding of errors and warnings between components of the API.
• #538 Summary=true still let some non-issummary fields through
• #569 Prefer header was not set on PUT
• #593 Fix .ttinclude file to prevent generating errors in some build environments.
• #619 Snapshot Generator ignores multiple codings with only display value
• #627 ToFhirDateTime() produced dateTimes without timezones when input DateTime.Kind was Unspecified
• #639 Target platforms are now 4.5, Netstandard 1.1, Netstandard 2.0
• #642 SnapshotGenerator does not expand custom element profile on Reference
• #657 Json Serializer was losing accuracy on serializing DateTimeOffset (last 4 digits in ticks)
• #663 Faster generation of property getters/setters with reflection emit
• #670 DifferentialTreeConstructor can now be used publicly to turn sparse differentials into full trees
• #676 Speed-up of serializers when running in Debug mode
• #684 DirectorySource can now retrieve summary data given a specific filename
• #696 SummaryGenerator now also extracts the extension context
• #704 Replaced uses of .NET DateTime with DateTimeOffset everywhere in the public API surface.
• Build scripts changed because of migration to Azure DevOps from AppVeyor
5.6.10 0.96.1 (Just R4) (released 20180925)

- R4-only release with all changes to the spec included up to the September ballot release.
  Note: There are no new packages for other versions, since we did not add any new functionality.

5.6.11 0.96.0 (DSTU2, STU3 and R4) (released 20180606)

- #595 Added capability to harvest metadata directly from a stream
- #524 Search paths no longer use [x] suffix in DSTU2
- #556 Fixed threading issues in value set expansion and snapshot generation when using the CachedResolver
- #577 FhirBoolean.value no longer has incorrect [BooleanPattern] in R4
- #591 Added ignoreCase option for parsing to enums
- #599 Fixed threading issue in FhirPath engine
- #601 Made the DifferentialTreeConstructor class public
- #606 FhirPath evaluator now support Resource and DomainResource as path roots
- #612 Bugfix for snapshot generator
- #614 Improved encoding for search parameters when doing POST-based search

5.6.12 0.95.0 (DSTU2, STU3 and R4) (released 20180412)

- Added support for R4 (warning: early alpha - client connectivity and parsing/serialization only)
- Added a SnapshotSource resource resolver that creates snapshots on the fly
- Added functionality to quickly harvest metadata from conformance resources on a file system
- #423 Internal references within contained resources are not validated properly
- #474 Validation fails when start date is 0001-01-01
- #477 ZipSource is extracting the content in a temp directory without a discriminator for the spec version
- #479 Use search using POST
- #454 Invoking an operation using GET (i.e. $everything) does not work with primitive type as parameters.
- #494 Directory Source - single threaded by default
- #461 Check support for both types of extensions for regexes on primitive values (tracker GF#12665)
- #510 Missing diff annotation on ElementDefinition.TypeRefComponent
- #536 FhirClient.Transaction() method incorrectly POSTs to FHIR Base URI with trailing slash
- #544 DateTime.ToDateTime failed to convert “1976-12-12”
- #557 workaround for slice validation when discriminator is missing in slice
- #571 Serialize to XDocument and JObject directly
5.6.13 0.94.0 (DSTU2 and STU3) (released 20171207)

- #448, the FhirXmlSerializer/FhirJsonSerializer should now be instantiated, use of the static FhirSerializer class is obsoleted
- #434, the API is no longer creating empty <meta> tags in the serialization
- #420, the json parser no longer returns -1,-1 for positions on errors
- #412, added support for read-through and cache invalidation to CachedArtifactSource
- #355, the POCO parser is now using IElememtNavigator as a source
- #474, fixed a bug where the parser would not accept ‘01-01-01’ as a date
- #371, the validator will now complain when it encounters unsupported discriminator types
- #426, when you tell the validator to not follow external references, it will no longer produce warnings that it cannot locate the external references.
- #489, the validator would sometimes report incorrect indices in paths with repeating elements
- #477, the location where the specification.zip is unpacked now includes the version in the name, thus avoiding conflicts when switchin branches between dstu2/stu3 at development time
- #419, calling Severything with no parameters will no longer result in an incorrect http request.

5.6.14 0.92.5 (DSTU2) / 0.93.5 (STU3) (released 20171017)

Changes to both versions:

- Changed the IElememtNavigator interface to enable skipping directly to a child with a given name, thus increasing navigation performance
- Improved performance of validation and fhirpath for POCOs
- Split off IFhirClient interface from the FhirClient implementation (primarily for testing/mocking)
- Many smaller bugfixes
- Improved error messages produced by the validator based on input from the NHS UK
- The validator will now let you put a constraint on children of Resource.contained, Bundle.entry. resource and similar nested resources.
- SerializationUtil.XmlReaderFromString() will no longer try to seek the stream passed in and rewind it.
- TransactionBuilder now has a (default) argument to specify the type of Bundle to build. Thanks mbaltus!
- DirectorySource now has Include/Exclude patterns (with globs) to have more control over directory scans for resource files.
- DirectorySource now supports processing conformance resources in json
- FhirClient now has async support
- You can now have List<> properties (like Extensions and other repeating elements) with null elements - these will simply be ignored and not serialized. Thanks wdebeau1!
- Made date to string and string to date conversion more consistent, fixing problems with locales using something else than ‘:’ for time separators.
- Fixed an error where the If-None-Exists header included the base url of the server. Thanks angus-miller+tstolker!
• All `Search()` overloads on `FhirClient` now also have a `reverseInclude` parameter

• Update with a conditional would not set the `If-Match` header when doing a version-aware update. Thanks tstolker!

• `DeepCopy()` did not actually deep-copy collections - if you changed the original collection before you iterated over the clone, you would see the changes. This has been fixed. Thanks mattiasflodin!

• Client would not pass on 1xx and 3xx errors to client, instead throwing a generic `Unsupported` exception, making it harder to handle these errors by the client. Thanks tstolker!

• Added a fall-back terminology service so the validator can now invoke an external terminology service if the local in-memory service (provided with the API) fails.

• You can now specify a binding on an Extension, which translates to a binding on `Extension.value[x]`

• Fixed a bug where -if the definition of `element[x]` had a binding and a choice of bindeable and non-bindeable types- the validator would complain if the instance was actually a non-bindeable type.

• **BREAKING:** `FhirClientOperation.Operation` has been renamed to `RestOperation`

• **BREAKING:** Revision of calls to terminology services to support all parameters and overloads

• Validation across references will now include the path of the referring resource in errors about the referred resource to make interpretation of the outcomes easier.

• `FhirPath's resolve()` now actually works, and will resolve contained/bundled resources in the instance under evaluation. This also means the `FhirPath` evaluator will now take an `EvaluationContext` in which you can pass your resolver over to the evaluator.

• The enums in the generated code now also have an attribute on them with information about the codesystem, which can be retrieved using `GetSystem()` on any enum. Thanks brianpos!

• Added a few specific `[Serializable]` attributes to make the POCOs serializable with the Microsoft Orleans serializer. Thanks alexmarchis!

• Several improvements & bug fixes on the `SnapshotGenerator`

• Fixed handling of non-fhir json files in the conformance directory.

• Fixed `eld-16` constraint, which used an invalid regex escape (\_)

• Now using the new NuGet 3.3 `<contentFiles>` tag to replace the (failing) `install.ps1`, so a) you'll get the new `specification.zip` transitively in dependent projects and b) the build action will be correctly set.

**DSTU2:**

• Fixed small errors in the generated `ConstraintComponent` properties, giving more correct validation results

**DSTU3:**

• Fixes to the snapshot generator to create better `ElementDefinition` ids

• `_sort` parameter now uses STU3 format (`_sort=a,-b,c`) instead of modifier

• You can now set the preferred return to `OperationOutcome`. Thanks cknaap!

• You can now request the server to notify the client about unsupported search parameters. Thanks tstolker!

Changes to the DSTU2 version:

• Fixed small errors in the generated `ConstraintComponent` properties, giving more correct validation results

Changes to the STU3 version:

• Fixes to the snapshot generator to create better `ElementDefinition` ids
• `_sort` parameter now uses STU3 format `(_sort=a,-b,c)` instead of modifier
• You can now set the preferred return to `OperationOutcome`. Thanks cknaap!
• You can now request the server to notify the client about unsupported search parameters. Thanks tstolker!

### 5.6.15 0.90.6 (released 20160915)

- Fix: FhirClient will no longer always add `summary=false` to search queries
- Fix: FhirClient will not throw parse errors anymore if the server indicated a non-success status (i.e. a 406)

### 5.6.16 0.90.5 (released 20160804)

- Enhancement: Portable45 target includes support for validation, and no longer depends on Silverlight 5 SDK. Thanks Tilo!
- Enhancement: Support for serialization where `summary=data` (and automatically adds the Subsetted flag - temporarily adds the Tag then removes after serialization, if it wasn’t there already)
- Enhancement: Added Debugger Displays for commonly used types
- Enhancement: Debugger Display for BundleEntries to show the HttpMethod and FullURL
- Enhancement: Additional method `public static bool IsKnownResource(FhirDefinedType type)` in ModelInfo (Thanks Marten)
- Enhancement: You can (and should) now create an instance of a `FhirXmlParser` or `FhirJsonParser` instead of using the static methods on `FhirParser`, so you can set error policies per instance.
- Enhancement: Introduced `ParserSettings` to configure parser on a per-instance basis:

```
FhirXmlParser parser = new FhirXmlParser(new ParserSettings { AcceptUnknownMembers = true });
var patient = parser.Parse<Patient>(xmlWithPatientData);
```

- Enhancement: Introduced a setting to allow parser to parse (and serialize) unrecognized enumeration values. Use `Code<T>.ObjectValue` to get to get/set the string as it was encountered in the stream. The FhirClient now has a `ParserSettings` property to manage the parser used by the `FhirClient`.
- Enhancement: By popular demand: re-introduced `FhirClient.Refresh()`
- Enhancement: Snapshot generator now supports all DSTU2 features (re-slicing limited to extensions)

```
ArtifactResolver source = ArtifactResolver.CreateCachedDefault();
var settings = new SnapshotGeneratorSettings { IgnoreMissingTypeProfiles = true }; StructureDefinition profile;
var generator = new SnapshotGenerator(source, _settings);
generator.Generate(profile);
```

- Fix: Status 500 from a FHIR server with an HTML error message results in a `FhirOperationException`, not a `FormatException`. Thanks Tilo!
- Fix: `Code<T>` did not correctly implement `IsExactly()` and `Matches()`
- Fix: Now parses enumeration values with a member called “Equals” correctly.
- Fix: `Base.TypeName` would return incorrect name “Element” for Primitives and `Code<T>` (codes with enumerated values)
• And of course numerous bugfixes and code cleanups.

5.6.17 0.90.4 (released 20160105)

• Enhancement: Additional Extension methods for converting native types to/from FHIR types

```java
public static DateTime? ToDateTime(this Model.FhirDateTime me)
public static DateTime? ToDateTime(this Model.Date me)
public static stringToFhirDate(this System.DateTime me)
public static stringToFhirDateTime(this System.DateTime me)
public static stringToFhirId(this System.Guid me)
```

• Enhancement: Added the SnapshotGenerator class to turn differential representations of a StructureDefinition into a snapshot. Note: we’re still working with the Java and HAPI people to get the snapshots 100% compatible.

• Breaking change: All BackboneElement derived classes are now named as found on BackboneElement page in the specification, under the specializations heading. Usual fix for this will be removing the resource typename prefix from the classname, e.g. Bundle.BundleEntryComponent -> Bundle.EntryComponent

• Fix: Elements are not serialized correctly in summary mode

• Fix: Validate Operation does not work

• Fix: DeepCopy does not work on Careplan f201

• Fix: SearchParameters in ModelInfo are missing/have invalid Target values

From this version on, the model is now code generated using T4 templates within the build from the specification profile files (profiles-resources.xml, profiles-types.xml, search-parameters.xml and expansions.xml)

5.6.18 0.90.3 (released 20151201)

• Enhancement: IConformanceResource now also exposes the xxxElement members. Thanks, wmrutten!

• Enhancement: Parameters.GetSingleValue<> now accepts non-primitives as generic param. Thanks, yunwang!

• Enhancement: ContentType.GetResourceFormatFromContentType now supports charset information. Thanks, CorinaCiocanea!

• Enhancement: Operations can now be invoked using GET

• Fix: Small code analysis fixes. Thanks, bnantz!

• Fix: SearchParams now supports _sort without modifiers. Thanks, sunvenu!

• Fix: FhirClient: The “Prefer” header was never set. Thanks, CorinaCiocanea!

• Fix: FhirClient could not handle spurious OperationOutcome results on successful POST/PUT when Prefer=minimal. Thanks, CorinaCiocanea!

• Fix: Json serializer serialized decimal value “6” to “6.0”. Thanks, CorinaCiocanea!

• Fix: Json serializer now retains full precision of decimal on roundtrip.

• Fix: ETag header was not correctly parsed. Thanks, CorinaCiocanea!

• Fix: Parameters with an “=” in the value (like pre-DSTU2 <=) would become garbled when doing FhirClient.Continue(). Thanks rtaixghealth!

• Fix: FhirClient.Meta() operations will use GET and return Meta (not Parameters)
5.6.19 0.90.2

- Added support for $translate operations on ConceptMap
- Added support for the changed _summary parameter
- ArtifactResolver can now resolve ValueSets based on system
- The CachedArtifactSource is now thread-safe

5.6.20 0.90.0

- Updated the model to be compatible with DSTU2 (1.0.1)
- Added support for comments in Json
- Fixed a bug where elements called ‘value’ in Json could not have extensions or comments
- FhirClient now returns the status code in an OperationException
- Bugfixes

5.6.21 0.50.2

- Many bug and stability fixes
- ReturnFullResource will not only set the Prefer header, but will do a subsequent read if the server ignores the Prefer header.
- Client will accept 4xx and 5xx responses when the server does not return an OperationOutcome
- Client gives clearer errors when the server returns HTML instead of xml/json
- Call signatures for OnBeforeRequest and OnAfterResponse have been changed to give low-level access to body and native .NET objects. OnAfterResponse will now be called even if request failed or if response has parsing errors.
- The FhirClient has a full set of new LastXXX properties which return the last received status/resource/body.
- Serializers now correctly serialize the contents of a Bundle, even if summary=true

5.6.22 0.20.2

- FhirClient updated to handle conditional create/read/update, Preference header
- Introduction of TransactionBuilder class to easily compose Bundles containing transactions
- Model classes updated to the latest DSTU2 changes
- Serialization of extensions back to “DSTU1” style (as agreed in San Antonio)

5.6.23 0.20.1

- Added support for async
5.6.24 0.20.0

• This is the new DSTU2 release
• Supports the new DSTU2 resources and DSTU2 serialization
• Uses the new DSTU2 class hierarchy with Base, Resource, DomainResource and Bundle
• Further alignment between the Java RM and HAPI
• Support for using the DSTU2 Operation framework
• Many API improvements, including:
  • deep compare (IsExactly) and deep copy (DeepCopy)
• Collections will be created on-demand, so you can just do patient.Name.Add() without having to set pa-tient.Name to a collection first
• Note: support for .NET 4.0 has been dropped, we support .NET 4.5 and PCL 4.5

5.6.25 0.11.1

• Project now contains two assemblies: a “lightweight” core assembly (available across all platforms) and an additional library with profile and validation support.
• Added an XmlNs class with constants for all relevant xml namespaces used in FHIR
• Added JsonXPathNavigator to execute XPath statements over a FHIR-Json based document
• Added a new Hl7.Fhir.Specification.Source namespace that contains an ArtifactResolver class to obtain schema files, profiles and valuesets by uri or id. This class will read the provided validation.zip for the core artifacts. For more info see [here](artifacts.html).
• Changed FhirUri to use string internally, rather than the Uri class to guarantee round-trips and avoid url nor-malization issues
• All Resources and datatypes now support deep-copying using the DeepCopy() and CopyTo() methods.
• FhirClient supports OnBeforeRequest and OnAfterRequest hooks to enable the developer to plug in authentica-tion.
• All primitives support IsValidValue() to check input against the constraints for FHIR primitives
• Models are up-to-date with FHIR 0.82
• And of course we fixed numerous bugs brought forward by the community

5.6.26 0.10.0

• There’s a new FhirParser.ParseQueryFromUriParameters() function to parse URL parameters into a FHIR Query resource
• The Model classes now implements INotifyPropertyChanged
• FhirSerializer supports writing just the summary view of resources
• Model elements of type ResourceReference now have an additional ReferencesAttribute (metadata) that indi-cates the resource names a reference can point to
• ModelInfo now has information telling you which FHIR primitive types map to which .NET Model types (this only used to work for complex datatypes and resources before)
• We now support both .NET 4.0, .NET 4.5 and Portable Class Libraries 4.5
• For .NET 4.5, the FhirClient supports methods with the async signature
• All assemblies now have their associated xml documentation files bundled in the NuGet package
• Models are up-to-date with FHIR 0.80, DSTU build 2408

5.6.27 0.9.5

This release brings the .NET FHIR library up-to-date with the FHIR DSTU (0.8) version. Additionally, some major changes have been carried out:

• There is now some documentation
• The FhirClient calls have been changed after feedback of the early users. The most important changes are:
  • The Read() call now accepts relative and absolute uri’s as a parameter, so you can now do, say, a Read(obs.subject.Reference). This means however that the old calling syntax like Read("4") cannot be used anymore, you need to pass at least a correct relative path like Read("Patient/4").
  • Since the FHIR create and update operations don’t return a body anymore, by default the return value of Create() and Update() will be an empty ResourceEntry. If you specify the refresh parameter however, the FHIR client will immediately issue a read, to get the latest updated version from the server.
  • The Search() signature has been simplified. You can now either use a very basic syntax (like Search(new string[] {"name=john"}), or switch to using the Query resource, which Search() now accepts as a (single) parameter as well.
  • The validator has been renamed to FhirValidator and now behaves like the standard .NET validators: it validates one level deep only. To validate an object and it’s children (e.g. a Bundle and all its entries and all its nested components and contained resources), specify the new recursive parameter.
  • The validator will now validate the XHtml according to the restricted FHIR schema, so active content is disallowed.
  • The library now incorporates the 0.8 version of the Resources. This means that developers using the API’s source distribution need only to compile the project to have all necessary parts, there is no longer a dependency on the Model assembly compiled as part of publication. Note too that the distribution contains the 0.8 resources only (so, no more Appointment resources, etc.).
  • The library no longer uses the .NET portable class libraries and is based on the normal .NET 4.0 profile. The portable class libraries proved still too unfinished to use comfortably. We’ve fallen back on conditional compiles for Windows Phone support. Cross-platform compilation has not been rigorously tested.
  • After being updated continuously over the past two years, the FHIR client needed a big refactoring. The code should be readable again.

5.6.28 Before

Is history. If you really want, you can read the SVN and Git logs.