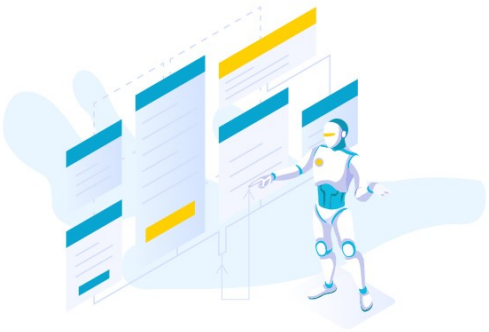
Firely Terminal

Firely

Mar 01, 2021

FIRELY PRODUCTS



Firely Terminal (formerly known as Torinox) is a cross platform FHIR command line tool that helps you working with files containing FHIR resources.

CHAPTER

ONE

INSTALLING FIRELY TERMINAL

Go to the Firely Terminal download page for instructions on how to install Firely Terminal.

CHAPTER

TWO

COMMON SCENARIOS

2.1 Search

In these examples, I will use vonk as the server to communicate with, but you can add and use any other server, like this:

```
> fhir server add myalias https://myserver/fhir
```

Search The basic search command requires a fhir server (alias), a resource type. This exmaple searches for any patient on the Vonk server.

```
> fhir search vonk Patient
```

The resulting resources will be pushed onto the stack. After that you can perform any of the other operations that Firely Terminal provides, like saving them to disk:

```
> fhir save --all
```

2.1.1 Count

The default result set maximum is 10 resources, but you can change that with the same parameter that you would use in a FHIR search URL:

```
> fhir search vonk Patient _count=20
```

2.1.2 Search parameters

With Firely Terminal you can use all search parameters that a FHIR server knows.

```
> fhir search vonk Patient name=Chalmers
```

Getting the core Patient structure definition:

```
> fhir search vonk StructureDefinition url=http://hl7.org/fhir/StructureDefinition/
↪Patient
```

2.2 How to use the resource stack

Firely Terminal was build with an internal stack, to allow you to combine multiple actions on one or more resources. All resource operations in Firely Terminal are performed on that stack.

We have provided as many as possible atomic operations, so that you can combine them yourself and give you much more flexibility. This way you can get multiple resources in one format from a fhir server and then bundling them before pushing them to a different server.

2.2.1 Using the stack

The resource stack of Firely Terminal is just like a stack in a your computer. You can add stuff to the top, and if you perform operations, they do that with the resource or resources on top of the stack.

You can basically group the types of operations you can do on a stack in three categories:

1. Adding resources to the stack

With commands like pusing a file from disk on top of the stack

```
> fhir push mypatient.xml
```

Or getting a file from a server:

```
> fhir read vonk Patient name=Chalmers
```

Duplicating the top resource on the stack:

```
> fhir dup
```

2. Taking resources from the stack:

Just throwing a resource away:

```
> fhir drop
```

Clearing the whole stack:

```
> fhir clear
```

But also saving files:

```
> fhir save mypatient.xml
```

3. Operations

In a classical (number) stack, you would use plus, minues or times. With Firely Terminal you get FHir operations like switching the top two resources:

```
> fhir swap
```

Bundling all resources

```
> fhir bundle
```

This will remove all resources from the stack and putting them back as a single FHIR Bundle resource. Another is to just show what is there:

```
> fhir show
```

2.3 Snapshot Generation

To generate a snapshot for a StructureDefinition, Firely Terminal needs to have two thing in place:

2.3.1 1. Get a StructureDefinition

A snapshot on the stack. You can do this by fetching a structure definition from a server or from disk;

```
> fhir read Vonk StructureDefinition/Patient
```

2.3.2 2. Create a scope

You also need a *scope* for resolving resources and data types. You can create a scope in any directory, by installing the required packages. For example:

```
> fhir install hl7.fhir.r3.core
```

2.3.3 Generate the snapshot

After this you can update the snapshot in the StructureDefinition, by one simple statement:

```
> fhir snapshot
```

After this the resource on the stack is replaced with a new StructureDefinition that contains the just generated snapshot.

2.4 XML to JSON and vice versa

With Firely Terminal you can save a FHIR resource either as JSON or XML. This also allows you to convert from one format to the other. Here are some examples.

In order to allow you to perform multiple operations on a resource, Firely Terminal works with an internal stack where you push your resources to. In case of converting to another format, you just have to save those resources from the stack to the preferred format.

With this command you save the resource on the top of the stack as an XML file:

```
> fhir save --xml
```

2.4.1 Converting a file to JSON

Load a resource from disk:

```
> fhir push mypatient.xml
```

And save it as json:

```
> fhir save mypatient.json
```

Firely Terminal will automatically recognize from the target file extension that you want to transform the resource JSON.

2.4.2 Converting a file to XML

Of course the other way around works the same:

```
> fhir push mylabreport.json
> fhir save mylabreport.xml
```

2.4.3 Converting all files in a folder:

```
> fhir push *.xml
> fhir save --all --json
```

2.4.4 Getting JSON files out a of an XML server:

In the example here, use your own alias and of course a different URL:

```
> fhir server add myserver https://myxmlfhirserver.com/fhir/
> fhir search myserver Patient _count=20
> fhir save --all --json
```

2.5 Working with Bundles

With Firely Terminal you can bundle a set of resources or tear a bundle apart.

2.5.1 Bundle all resources in a folder

In this example, we assume, you have a set of xml resources. The first step is to load all the resources in the folder onto the stack.

```
> fhir push *.xml
```

After that the following command takes all resources on the stack, put them in a single bundle and set that Bundle resource on the stack.

```
> fhir bundle
```

The save commmand can then be used to get your bundle on disk

```
> fhir save mybundle
```

2.5.2 Splitting a bundle

The inverse scenario can be done with the same example: Load a bundle onto the stack; use the Split command and save the results:

```
> fhir push mybundle.xml
> fhir split
> fhir save --all
```

CHAPTER

THREE

COMMANDS

Firely Terminal has several groups of commands. Here is more information on them:

3.1 Simplifier Related Commands

Firely Terminal has several of commands to help you interact with Simplifier: from downloading and installing packages, which we discuss in a separate section, to listing your projects and getting a local project (folder) synchronized with an online (Simplifier) project.

3.1.1 Login with Firely Terminal into Simplifier

Several api endpoints in Simplifier require you to login first. By logging in, Firely Terminal gives you access to most of these endpoints:

- packages api
- zip upload and download api
- fhir endpoint
- writing (post and put) to private fhir endpoinds

In order to login with Firely Terminal use the `login` command:

```
> fhir login
```

After this you must provide your credentials: email address and password. Login uses standard JWT practices so that your credentials are sent safely over the wire only once and so that sequentual requests are done with a token, so that your credentials don't have to be stored locally either.

3.1.2 Project Synchronization

This documentation is for Firely Terminal version 0.9.6 or higher.

You can keep a local folder in sync with a project in Simplifier by using the `sync` command. For that it uses the zip api endpoint of Simplifier.

In order to sync with a project in Simplifier, you have to be *logged in*.

Synchronization has several options.

Download

To download the latest content of a Simplifier project to a local folder, use the `--down` flag:

```
> fhir sync --down
```

This will update all files in your current folder that are newer or missing.

Upload

To upload the contents of a local folder to a Simplifier use the `--up` command.

```
> fhir sync --up
```

Bidirectional

You can synchronize both ways using a combination of the `--up` and `--down` flag:

```
> fhir sync --up --down
```

or, using short flags:

```
> fhir sync -ud
```

Subfolders

If you want to include all subfolders of the current folder in synchronization, use the `--folders` flag. This works both for uploading and for downloading. For Files that are in folders that don't exist in the target location, the subfolders will be created.

Details

In order to get more detailed information during the sync process, use the flag `--verbose`. This will show each individual file that is being synchronized. This works both for syncing up and for syncing down.

Synchronizing deletes

It is possible to synchronize deletes when doing a down sync. With an up sync this is not yet possible. You can enable this by using the `--clean` flag.

Warning: this will delete all files in your folder that are not present in the project in Simplifier!

To make sure that files are not deleted by accident, you can only run a *clean* sync by explicitly providing the target folder from the root. For example:

```
> fhir sync myproject c:/fhirprojects/myproject --down --clean
```

3.1.3 Simplifier projects as a FHIR server

Firely Terminal understands that projects in Simplifier are a fhir server. You dont have to configure anything to make use of them. To get a fhir resource (in this example, the example patient) from the FHIR endpoint of your project, you can simply use:

```
> fhir read myproject Patient/example
```

3.2 Managing packages

Firely Terminal is a FHIR package manager. The following commands are available to manage them.

Packages are the core distribution mechanism for sets of profiles. Profiles and other conformance resources rarely live in isolation. Together they form a use case, and they usually need other resources from a national initiative, and they certainly need the FHIR core base profiles.

In FHIR these sets are distributed according to the FHIR package standard, which conforms largely with the NPM package standard. There are a few additions and a few restrictions.

One of the core problems when distributing profiles, is choosing the right version. And since profiles often reference eachother, you need all the ones that are of the same release. For that, packages are ideal.

3.2.1 Versions

Lists all available versions of the package on the server

```
> fhir versions simplifier.core.stu3
```

This shows you all the available versions of the package on the package server. What it does is that it fetches the *package listing* of that package from the server, which is a json document. To see that document, use the `--raw` flag. Example:

```
> fhir versions simplifier.core.stu3 --raw
```

3.2.2 Cache

The cache commands lists all the packages in the global package cache on your machine.

```
> fhir cache
```

If you want to know the folder location of the FHIR package cache, add the `--location` (or `-l`)parameter:

```
> fhir cache --location
```

3.2.3 Semver

The semver command allows you to test if a version range is what you intended. It will show you the version of a package the range actually resolves to. If the package `xyz.myprofiles` has three versions: 3.1.0, 3.2.0 and 4.0.0, the range `3.x` points to `3.2.0` Usage:

```
> fhir semver xyz.myprofiles 3.x
```

3.2.4 Install

The install command can be used to install any FHIR package from the package server. It will install the package on your local machine and add a reference from your current folder to that package, by adding it to the `package.json` file (also known as a package manifest). You can provide just the package name, to get the latest.

```
> fhir install xyz.myprofiles
```

Or you can specify a specific version or range.

```
> fhir install xyz.myprofiles 2.3
```

Folder install

By default a package is installed in the global package cache of your machine. But you can specify to install it in the current folder with the `--here` (`-r`) flag.

```
> fhir install simplifier.core.stu3 --here
```

This will install the latest version of package `simplifier.core.stu3' as a subfolder of your current folder. In most scenario's you should not need this option.

Install a file

If you have a package file that is not on the package server, for example you created it locally and want to test it, you cana install a package file (.tgz) from your current folder into the global package cache of your machine. For this you can use the `--file` (`-f`) flag.

```
> fhir install thisproject.tgz --file
```

3.2.5 Remove

If you have installed a package in your current project (folder), you can remove it. For a package called 'xyz.myprofiles' the command would be:

```
> fhir remove xyz.myprofiles
```

This will not remove your package from your global packages cache, but it is no longer part of your current project. The command will update the manifest (package.json) and your lock file `fhirpkg.lock.json`).

3.2.6 Pack

You can create a FHIR package file from the contents of the current folder, using:

```
> fhir pack
```

The folder must have a `package.json` file.

3.2.7 Contents

Displays the contents from a package file (.tgz)

3.2.8 Init

Optional parameters: package name and version. Generates a FHIR package manifest in the current folder, enabling this folder to use (consume) packages. It is also the start of making the current folder itself into a package.

3.2.9 Restore

The restore command is used to make sure that all packages in the `package.json` file are installed on your machine, and creates a lock file, to make sure your project uses the correct dependencies.

```
> fhir restore
```

3.2.10 Scope

Lists all the packages that are in scope for this folder context.

```
> fhir scope
```

If you only want to know your direct dependencies, use:

```
> fhir dependencies
```

or the short form:

```
> fhir deps
```

3.2.11 Canonicals

Lists canonicals from resources in a package.

```
> fhir canonicals
```

3.2.12 Find

The find command allows you to find packages. You can either provide a partial package name or a canonical. All matches will be listed.

```
> fhir find http://hl7.org/fhir/StructureDefinition/Patient
```

3.3 FHIR Client commands

Using Firely Terminal as a command line FHIR client.

3.3.1 Post

You can post a resource to a FHIR server, using the `post` command:

```
> fhir post <server>
```

What this does is that it takes the resource from the top of the stack and posts that using the FHIR protocol to FHIR server you choose. Below is a full scenario, to mak that clear:

Scenario

Let's add a server alias to the known server list of Firely Terminal. This can be any FHIR server.

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

We can now use a server called 'vonk'. Next, let's get a resource from that server: a patient, with id `example`:

```
> fhir read vonk Patient/example
```

The patient record has been fetched from the server and pushed on the local stack.

Now we can post this resource to another server, and let's take a project on Simplifier that you own, because every STU3 FHIR project in Simplifier has it's own FHIR server. Let's imagine it to be 'myproject'.

```
> fhir post myproject
```

The resource is removed from the stack and posted as a new resource to that project.

3.4 Quality Control

Commands that help you with quality control.

3.4.1 Expand

Important: This feature is available from the Professional plan and up.

Expand takes the ValueSet on top of the stack, and uses the current *Scope* to expand all the codes in the ValueSet The resulting ValueSet is then placed back on top of the stack.

Usage

```
> fhir expand
```

License

The expand command can only be used by licensed customers of Simplifier.

Limits

The current expand command is limited to regular ValueSets, so special cases like SNOMETD need to be done on a FHIR server. The expand command is cucrrent limited to 1.000.000 codes. That should be enough for most ValueSets. Beware that it can take a long time though.

3.4.2 Snapshot

The snapshot command takes a StructureDefinition on top of the stack, and builds the snapshot element of that StructureDefinition. The resulting StructureDefinition will be placed back on top of the stack.

Usage

```
> fhir snapshot
```

Scope

The command uses the project *Scope* (current folder) to resolve all structures needed to build the snapshot.

3.4.3 Validate

The validate command validates the resource on the top of the stack, using the current *Scope*.

Scope

Validation needs a lot of data: from StructureDefinitions on the resource itself, to data types, Extensions and ValueSets. It will look into your project (current folder) and any packages, wether dependencies or depenencies of dependencies (etc.) to find all these assets.

Output

Validation will report any errors and warnings and information messages to the command line output.

```
> fhir validate
```

Parameters

You can use your own terminology server or specifiy an additional profile to validate against. For more details on these paremters, see:

```
> fhir ? validate
```

CHAPTER

FOUR

USING NPM INSTEAD

If you can't or don't want to use the fhir command line tool, but you do need to install packages from Simplifier, you can use *npm*.

4.1 Install npm

To get *npm*, you can install nodejs. The npm command line tools is part of that. You can download nodejs here: https://nodejs.org/en/download/.

4.2 Installing a FHIR package with npm

To install anything with npm, you need to have a `package.json` file. You can do that with either `npm create` or create a `package.json` file yourself. It's sufficient to add two brackets in that file to make it work: `{ }`.

To install a FHIR package with all dependencies, this gets you there. In this example we'll install package `hl7.fhir.r4.core`.

```
> npm create
> npm --registry https://packages.simplifier.net install hl7.fhir.r4.core
> npm --registry https://packages.simplifier.net install
```

4.3 Simplifier as default npm package server

If you want to set Simplifier as your default package server, you can do that with NPM. We recommend you only do that if you don't need npm for .js packages.

```
> npm config set registry https://packages.simplifier.net/
> npm install hl7.fhir.r4.core
> npm install
```

CHAPTER

FIVE

QUICK OVERVIEW

For questions or feedback on Firely Terminal, please send an email to simplifier@fire.ly.