Service Template Specification

STS id fertilisation\_advisor

Version 0.2.0

Status PUBLISHED

Author(s) M. Caballero (AgriCircle AG)

**The "fertilisation\_advisor" Service Template**

Abstract

This document describes the specifications for fertilisation\_advisor services whose purpose is to provide product application maps for selected nutrients. fertilisation\_advisor services are agnostic of tractors or other equipment; they typically use agronomical algorithms to provide advice on fertilization which would then be typically carried out by a fertilisation\_applicator service.

Table of Contents

[1 Introduction 3](#_Toc90042010)

[2 Terminology 3](#_Toc90042011)

[3 Fertilisation Advisor Usage Scenarios 3](#_Toc90042012)

[3.1 NPK Fertilisation 3](#_Toc90042013)

[4 Service Template Functions 4](#_Toc90042014)

[4.1 Fertilization Advice Functions 4](#_Toc90042015)

[4.1.1 Prepare Advice 4](#_Toc90042016)

[4.1.2 Get Advice Status 5](#_Toc90042017)

[4.1.3 Cancel Advice 5](#_Toc90042018)

[4.2 Product Application Functions 5](#_Toc90042019)

[4.2.1 Planned Applications 6](#_Toc90042020)

[4.2.2 Get Application Info 6](#_Toc90042021)

[5 Data Formats 7](#_Toc90042022)

[5.1 Product Application File Format 7](#_Toc90042023)

[6 Access and Authentication 10](#_Toc90042024)

[7 Dynamic Behaviour 10](#_Toc90042025)

[7.1 Fertilisation Lifecycle 10](#_Toc90042026)

[7.2 Advice Completion 11](#_Toc90042027)

[7.2.1 Polling 11](#_Toc90042028)

[7.2.2 Notification 12](#_Toc90042029)

# Introduction

Fertilisation is essential for crop yield. The fertilisation requirements depend on the nutrient requirements for a particular crop type and the availability of these nutrients in the soil.

fertilisation\_advisor services determine product application maps to optimally satisfy specific nutrients requirements which can later be applied in one or more applications by a fertilisation\_applicator (see fertilisation\_applicator Service Template) on a field.

Farmers may use different fertilisation\_advisor services depending on the specific set of nutrients they focus on, their budget, the accuracy and the granularity of information they require, etc.

# Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](https://datatracker.ietf.org/doc/html/rfc2119) [[RFC2119](https://datatracker.ietf.org/doc/html/rfc2119)] and indicate requirement levels for compliant implementations.

The notation "[xxx]" (xxx in square brackets) is equivalent to "array of xxx".

When used alone, the term "fertilisation\_advisor" refers to "fertilisation\_advisor service template". Instead, "fertilisation\_advisor service" is equivalent to "a service implementing the fertilisation\_advisor service template".

# Fertilisation Advisor Usage Scenarios

The following sections describe some aspects of the use of the fertilisation\_advisor service template. The examples were chosen to illustrate the basic functions of applications using fertilisation\_advisor, not to limit what fertilisation\_advisor may be used for.

## NPK Fertilisation

Most crops have nitrogen (N), phosphorous (P) and potassium (K) nutrient requirements.

A fertilisation service can be designed so as to focus on NPK nutrients and privilege the use of organic over chemical fertilisers.

The service receives a request to prepare an NPK fertilization for a corn plantation using slurry, and two other products, one with high P dosage and another with high K dosage. Based on the corn NPK requirements, the soil's properties and previously applied products, the fertilization service determines the optimal amounts of each product's needs. These application maps are delivered on-demand to another service that is responsible for converting the data to a format understandable by the equipment that will perform the tasks on the field.

# Service Template Functions

This section provides a very high-level summary of the fertilisation\_advisor service template functions:

Fertilization Advice Functions

Prepare Advice

Get Advice Status

Cancel Advice

Product Application

Planned Applications

Get Application Info

Within this section, functions are summarized with simple tables:

+--------------------------------------------+

| <logical function name> |

+-------------+------------------------------+

| Inputs | <URL parameters or |

| | request body attributes |

+-------------+------------------------------+

| Outputs | <response body attributes> |

+-------------+------------------------------+

Only the most meaningful parameters are discussed in this document. Please refer to the OpenAPI specifications for full details.

fertilisation\_advisor services are not required to handle intense traffic from a single client, especially for GET functions. Implementors MAY generate a 429 TOO MANY REQUESTS error response if the rate of calls exceed some pre-defined quota.

## Fertilization Advice Functions

These functions relate to the creation of fertilization advices. An advice may be request to optimize fertilization for a set of nutrients to be applied on a field.

### Prepare Advice

This function allows for the preparation of the background information that may be necessary for the establishment of an advice. The preparation may be immediate in some case but could take from hours to days in the case where physical operations such as taking and analysing soil samples are required by a service implementation.

+--------------------------------------------+

| prepare\_advice |

+-------------+------------------------------+

| Inputs | field urn, nutrients, |

| | notification URL |

+-------------+------------------------------+

| Outputs | advice info |

+-------------+------------------------------+

fertilisation\_advisor services may retrieve any relevant information to preparing their advices from the field\_data service (e.g., boundaries, current crop, previous applications, etc.).

Clients may poll for the status of an advice (see Get Advice Status) or be notified of completion (successful or unsuccessful) if they supplied a notification URL.

### Get Advice Status

This function returns status information about an advice preparation.

+--------------------------------------------+

| get\_advice\_status |

+-------------+------------------------------+

| Inputs | advice id |

+-------------+------------------------------+

| Outputs | status |

+-------------+------------------------------+

The prescription status is one of "PREPARING", "READY", "CANCELED", "FAILED".

### Cancel Advice

+--------------------------------------------+

| cancel\_advice |

+-------------+------------------------------+

| Inputs | advice id |

+-------------+------------------------------+

| Outputs | - |

+-------------+------------------------------+

## Product Application Functions

Product application functions deal translating advices into applications that can be supplied on demand, typically by a fertilisation\_applicator service.

### Planned Applications

This function informs fertilisation\_advisor services about the applications that will be carried out (one or more) and their associated products, so that they may optimize advised application maps accordingly.

+--------------------------------------------+

| planned\_applications |

+-------------+------------------------------+

| Inputs | advice id, |

| | [application ref, |

| | product\_info] |

+-------------+------------------------------+

| Outputs | - |

+-------------+------------------------------+

"application ref" are client-supplied references that will serve to identify a specific product application in the get\_application\_info function.

planned\_applications must have been invoked before the first call to get\_application\_info. A service implementation MAY accept additional planned\_applications invocations with different products, even after previous get\_application\_info calls have been performed.

### Get Application Info

This function may only be performed on advices whose status is "READY". It returns a download URL to the application map and the total estimated amount of product needed carry out the application. All values are in the corresponding product units. Please refer to "5.1 Product Application File Format" for detailed specification of the downloaded product application map file format.

+--------------------------------------------+

| get\_application\_info |

+-------------+------------------------------+

| Inputs | advice id, application ref |

+-------------+------------------------------+

| Outputs | download URL, product amount |

+-------------+------------------------------+

fertilisation\_advisor services must return an error if no applications were planned or if an application reference that was not included in the planned\_applications function is requested.

In cases where multiple applications are planned without a mechanism for in-field cooperation, it is important that the client (e.g., FMIS) ensures that one application if fully completed by the fertilisation\_applicator service before querying the application map for the next application.

Also, for maximum accuracy, the application map should be requested in a "just-in-time" fashion by the fertilisation\_applicator service. fertilisation\_advisor services may choose to limit the validity of the download URL.

# Data Formats

This section focuses on the description of binary (file) data formats. Please refer to the fertilisation\_advisor OpenAPI specifications for details on all other payload and parameter descriptions.

## Product Application File Format

Nutrient Application data MUST be in GeoPackage (<https://www.geopackage.org/>) format version 1.2 or newer.

+=========================================================+

| TABLE gpkg\_contents |

+=====+=============+=====================================+

| ROW | COLUMN | VALUE |

+-----+-------------+-------------------------------------+

| | table\_name | "atlas" |

| +-------------+-------------------------------------+

| | data\_type | "attributes" |

+-----+-------------+-------------------------------------+

All Atlas GeoPackage files MUST contain an attributes table named "atlas" with a single row and the following structure:

+============================+

| TABLE atlas |

+================+===========+

| COLUMN | TYPE |

+----------------+-----------+

| id | INTEGER |

| type | TEXT |

| participant | TEXT |

| format\_version | TEXT |

+----------------+-----------+

and a single row:

+=========================================================+

| TABLE atlas |

+=====+================+==================================+

| ROW | COLUMN | VALUE |

+-----+----------------+----------------------------------+

| | type | "fertilisation\_application" |

| +----------------+----------------------------------+

| | participant | "<atlas participant id>" |

| 1 +----------------+----------------------------------+

| | application | "…" |

| +----------------+----------------------------------+

| | format\_version | "MAJOR.MINOR" |

+-----+----------------+----------------------------------+

Services SHALL validate that the "atlas.participant" field matches the information attached to the authentication context in which the file is uploaded.

If the GeoPackage was generated and uploaded by an Atlas service, then "atlas.application" SHOULD be "<service name>-<service version>", otherwise it MAY contain "<application name>-<application version>". In either case, no validation will be performed.

The "atlas.format\_version" field MUST be the version of the fertilisation\_advisor template that is targeted by the client uploading the file. GeoPackage files of a given type are guaranteed to be compatible (no breaking changes) for a same MAJOR version of "atlas.format\_version".

A fertilization application GeoPackage MUST contain the following additional tables:

+=========================================================+

| TABLE gpkg\_contents |

+=====+=============+=====================================+

| ROW | COLUMN | VALUE |

+-----+-------------+-------------------------------------+

| … |

+-----+-------------+-------------------------------------+

| | table\_name | "product" |

| +-------------+-------------------------------------+

| | data\_type | "2d-gridded-coverage|features" |

+-----+-------------+-------------------------------------+

| | table\_name | "application\_info" |

| +-------------+-------------------------------------+

| | data\_type | "attributes" |

+-----+-------------+-------------------------------------+

If the data\_type of the product table is "features", then the columns must be as follows:

+=========================================================+

| TABLE product |

+=====+=================+=================================+

| ROW | COLUMN | VALUE |

+-----+-----------------+---------------------------------+

| | id | "INTEGER" |

| +-----------------+---------------------------------+

| | geometry | "GeoPackage Geometry" |

| +-----------------+---------------------------------+

| | amount | "amount in product units (REAL)"|

|-----+-----------------+---------------------------------+

Instead, if the data\_type of the product table is "2d-gridded-coverage" then it must be defined as follows:

+=========================================================+

| TABLE product |

+=====+=================+=================================+

| ROW | COLUMN | VALUE |

+-----+-----------------+---------------------------------+

| | id | "INTEGER" |

| +-----------------+---------------------------------+

| | zoom\_level | "INTEGER" |

| +-----------------+---------------------------------+

| | tile\_column | "INTEGER" |

| +-----------------+---------------------------------+

| | tile\_row | "INTEGER" |

| +-----------------+---------------------------------+

| | tile\_data | "BLOB (TIFF)" |

|-----+-----------------+---------------------------------+

Where tile\_data is a tiff blob holding Float32 values as per GeoPackage specifications ( <http://docs.opengeospatial.org/is/17-066r1/17-066r1.html#_storage_formats_and_grid_cell_values>).

+============================+

| TABLE application\_info |

+================+===========+

| COLUMN | TYPE |

+----------------+-----------+

| id | INTEGER |

| type | TEXT |

| date | DATETIME |

| info | TEXT |

+----------------+-----------+

and a single row:

+=========================================================+

| TABLE application\_info |

+=====+=============+=====================================+

| ROW | COLUMN | VALUE |

+-----+-------------+-------------------------------------+

| | type | "fertilization" |

| 1 +-------------+-------------------------------------+

| | date | "yyyy-MM-ddTHH:mm:ssZ" |

| +-------------+-------------------------------------+

| | info | "<application\_info\_json>" |

+-----+-------------+-------------------------------------+

Fertilization "application\_info.info" is a JSON object serialized as a string, such as:

+-----------------------------------------------+

| { |

| "product": { |

| "name": "optional name", |

| "unit": "kg|l", |

| "amount": <amount\_in\_product\_unit>, |

| "nutrients": { |

| "N": <amount\_in\_grams\_per\_unit>, |

| "P2O5": <amount\_in\_grams\_per\_unit>, |

| "K2O": <amount\_in\_grams\_per\_unit>, |

| … |

| } |

| } |

| } |

+-----------------------------------------------+

The units of the "product" data table are in application\_info.info.product.unit" units.

# Access and Authentication

Farmers MUST have an account setup on a fertilisation\_advisor service in order to authenticate and access API endpoints. The service implementor is responsible for the creation of accounts; it is not covered in the service template specifications.

Unless specifically documented in the OpenAPI specifications, all API calls must include credentials in form of Bearer authentication (also called token authentication). Clients can obtain an access token on behalf of their user from the service's authorization server (see ATLAS service pairing).

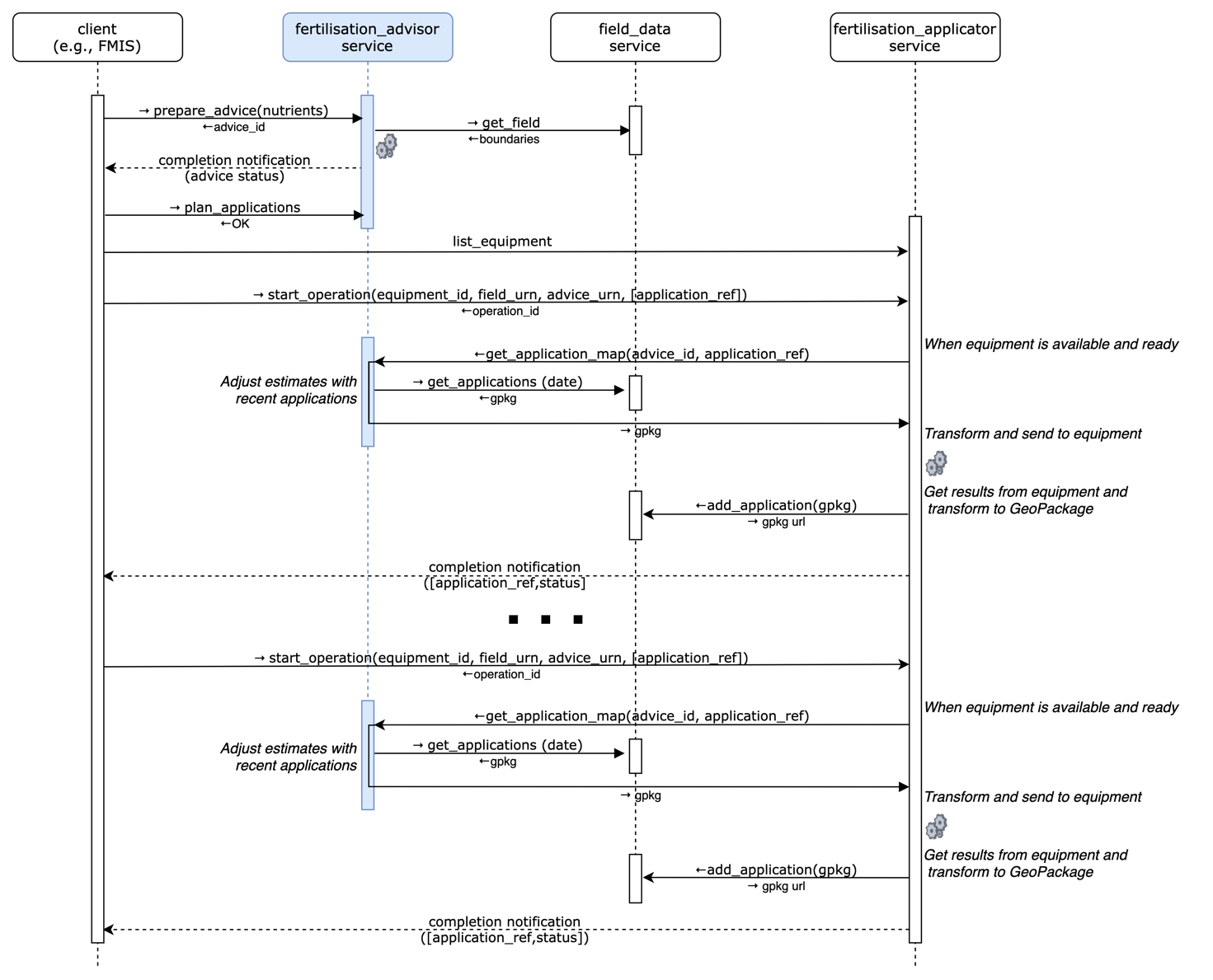
Some of the information held by fertilisation\_advisor services may be considered to be sensitive from a GDPR perspective. The service's authorization server SHOULD request the client's end-user consent at service pairing time in order to deliver an access token.

# Dynamic Behaviour

The purpose of the diagrams in this section is to illustrate communication patterns, more complex than plain request/response API calls, that involve several interactions and/or asynchronous behaviour. Even though a sequence diagram representation is used, the diagrams are by no means to be interpreted as UML Sequence Diagrams. Specifically, in the spirit of focusing on functional behaviour and readability, error handling is deliberately not covered in the diagrams.

## Fertilisation Lifecycle

The "fertilisation lifecycle" in ATLAS typically operates in a larger context involving three services and an end-user application such as an FMIS which, in collaboration, provide an end-to-end solution from algorithmically computed advices to actual operations performed on a field.

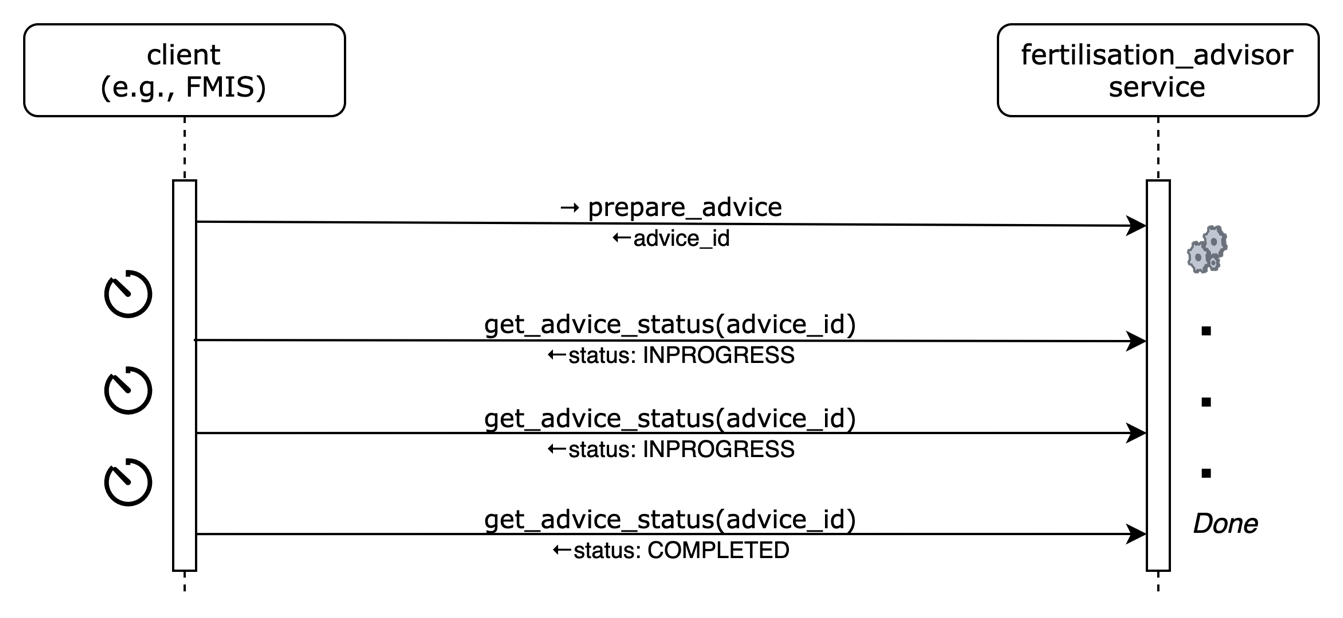


## Advice Completion

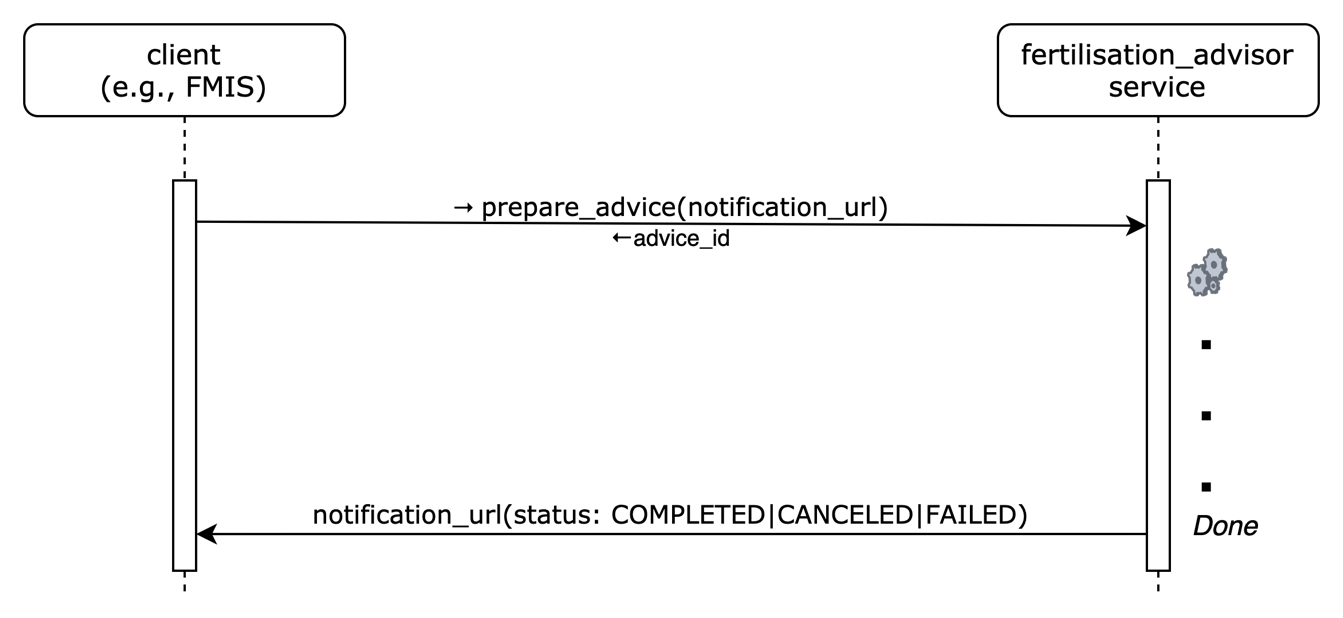
There are two methods for determining whether an advice preparation has completed (successfully or unsuccessfully): by polling the get\_advice\_status or by notification.

### Polling

After requesting an advice preparation, the client polls get\_advice\_status at regular intervals until the returned status is "READY", "CANCELED", or "FAILED".



### Notification



If a notification URL was supplied on the prepare\_advice function, it will be invoked by the fertilisation\_advisor when the preparation status changes.

Service MUST invoke the notification URL supplied by the client with an HTTPS POST command.

The payload will be identical to the one that would be returned by a get\_advice\_status request.

Services must provide best efforts to deliver notifications. A notification is considered successful if the target returns an http result code 2XX.

Errors may occur during notification delivery. Depending on the type of error, services must react in different ways:

i) Network error - the connection to the client's host (from notification URL) cannot be established. The service MUST retry a certain number of times. The number of retries and possible backoff strategy is left at the discretion of the service implementer.

ii) Server errors (5XX result code) – these errors are potentially transient. The same strategy as for Network errors SHOULD be applied.

iii) Client errors (4XX result code) – typically when the notification URL is invalid or the authentication is invalid/expired. 4XX errors should never be sent for transient client-side conditions and therefore services SHOULD NOT attempt retries.

Upon an excessive number of errors, services MAY give up further notification attempts. In that case, clients can only retrieve completion information via polling.