***Best Practice 26: Use an API***

*Offer an API to serve data*

**Why**

An “Application Programming Interface”, or API simplifies the way that computer programs can access and subset the data. An API is particularly helpful for speeding up the development of programs that use the data and also protects the data source from queries that can be resource intensive. If your dataset is large, frequently updated, or highly complex, an API is likely to be helpful. APIs can provide the means to deliver data in varied formats (e.g. CSV, JSON, Turtle) and can be designed to be accessed using HTTP (using REST or HATEOS principles)

**Intended Outcome**

Developers will have simplified programmatic access to the data that protects the dataset from resource-intensive subsetting. APIs can be designed to be accessed using HTTP but other transport protocols are also possible.

**Possible Approach to Implementation**

Relational databases and other similar data stores can be provided with a simple REST-ful API using elementary programming (e.g. <https://gist.github.com/pwin/8391942> or https://developers.apigee.com/get-started ). RDF triplestores can be provided with more sophisticated APIs using readily and freely available software such as PublishMyData (<https://github.com/Swirrl/publish_my_data> ) or Elda (<https://github.com/epimorphics/elda>) . If you use a data management platform, such as CKAN, you may be able to simply enable an existing API. Many web development frameworks include support for APIs, and there are also frameworks written specifically for building custom APIs. Examples include Swagger, Apigility, Apache CXF, and Restify

Example 26

To be done

**How to Test**

Use Service Virtualization to simulate calls and responses, make sure that the performance is acceptable.

Issue 10

To review the BP "Use an API" and possibly rewrite *Possible Approach to Impelmentation* section.