Machine Rules for accessing FDOs

TSIG Task Force #2

(Specifications for informing machines how to recognize FDO implementations and FDO compatibility levels)

TSIG-TF 02
(February 28th 2024)

In a perfect world, machines would access all FDOs using the same technology, expect fully formed FDOs and be able to focus on the nature of the information right away. Currently too much effort is spent on simply determining how to get to the information. In our reality, FDOs will be implemented with a multitude of different implementations, some of which will not be interoperable, requiring clients to rely on a set of deterministic rules and procedures to determine the specific characteristics of each of the FDO implementations and determine dynamically how to best interact with each of them.

Within the FDO space, there will exist multiple different FDO specification implementations based on different underlying technologies providing discrete FDO compatibility levels. These different implementations and interoperability levels need to be described with respect to the FDO specification and to each other in what we propose to call an FDO interoperability matrix.

To facilitate interoperability across these different implementations, the FDO Forum will develop this interoperability matrix to help provide guidance, recommendations, and specifications and to assist services and clients navigate the implementation possibilities.

The goal of this Task Force is to identify and elaborate the sets of rules that software developers will be able to use to determine the nature of the various FDO implementations with which they are interacting, the various discrete steps necessary for carrying out requests and processing the responses, as well as a method for measuring their respective compatibility levels to help them deterministically traverse a diverse space of FDOs implemented using different technologies with different characteristics.

1. The FDO Specification

The simple specifications of the FDO Forum for this FDO configuration are listed here:

1. ***Data FAIR Digital Objects (FDO-D)*** are machine actionable units of information bundling all information that is needed to enable FAIR processing of any included bit-sequence.
2. A ***PID***, standing for a globally unique, persistent and resolvable identifier, is assumed to be at the base of FDOs.
3. A PID resolves to a structured ***FDO-Record*** compliant with a specified ***FDO-Profile,*** *the use of* which leads to predictive resolution results.

4. The FDO-Record needs to contain ***Mandatory FDO (kernel) Attributes***, may contain **Optional FDO attributes** and other attributes agreed upon and defined by recognized communities.
5. ***Mandatory-FDO-D Attributes*** are: (1) the ***FDO-Content-Type***, (2) the reference to the ***FDO-Profile***, (3) the reference to the **bit-sequence**(s) encoding data, (4) the references to the different **metadata** resources.
6. ***Metadata*** can themselves be FDOs.
7. A ***collection*** of FDOs can be also an FDO.
8. Each FDO identified by a PID can be accessed or operated on using an interface protocol by specifying the PID of a registered supported operation.

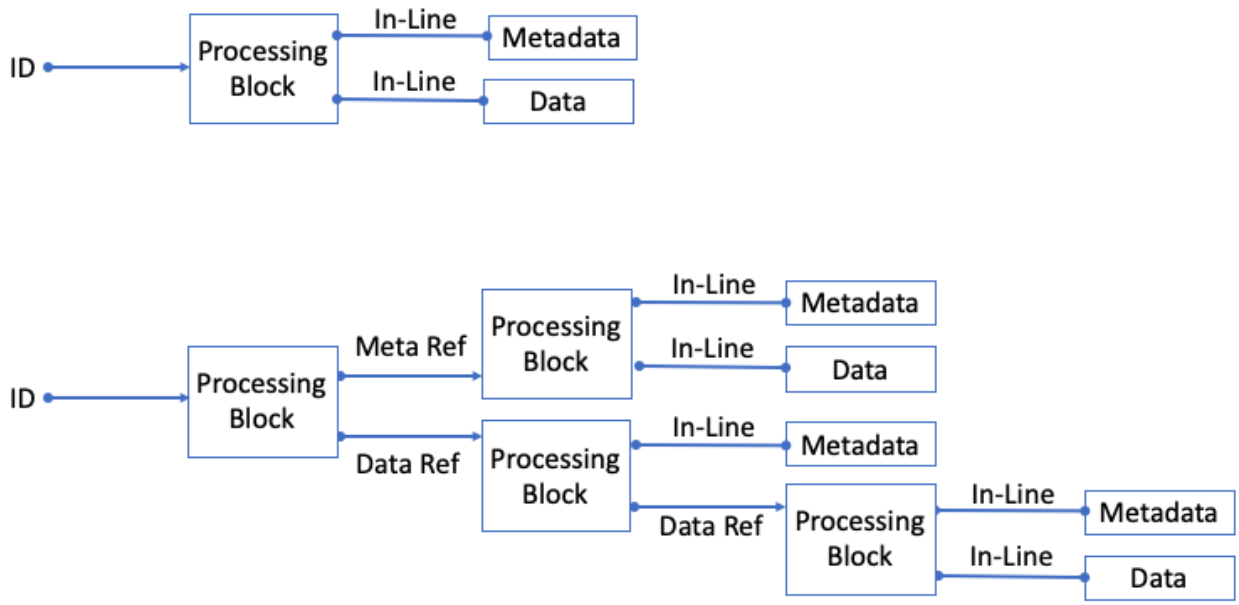
2. The FDO Core Processing Block

At the most fundamental level, regardless of the technology used, all machines will have to process FDO references and FDOs in a similar manner. We call this the FDO Core Processing Block and it consists of a basic set of operations that can be combined and / or recursed on to process any FDO of any complexity. We refer to this processing loop as the core processing loop.

The FDO Core Processing Block consists of the following steps:

1. Resolve an ID / URIs into an ID Record (an ID record can be an FDO record).
2. Parse the ID Record into a set of type value pairs
3. Identify all the type value pairs of relevance (some template).
4. Parse the values of the type value pairs of relevance to the FDO.
5. Process the values of interest.
6. Results from the processing:
    6.1. If the values of interest are found, the FDO core processing block terminates.
    6.2. If the values of potential interest are referenced by IDs / URIs, the machine starts a new FDO Core Processing Loop to process the references.
    6.3. If the values of potential are a specific data service access point where the data is accessible knowing a specified protocol
    6.4. If no values of interest could be found, the FDO core processing block terminates

These steps can be represented in the following illustration. The first illustration shows the most basic use of the processing block and the second shows how it can be combined when the results of the processing block are references to other FDOs.



This illustration shows that the results of the Processing Block are either in-line or by-references. In the case of in-line results, the client has access to what it was looking for and the FDO processing comes to an end. In the case the results are by-references, the references are evaluated by using additional Processing Blocks and the results determine whether the processing needs to continue or has reached its completion.

The FDO Core Processing Block is conceptually simple but it can be combined to access complex FDOs. In addition, each processing block can be more or less complex depending on the technology used to implement the FDO. The following section goes into more detail as to some of the variations that can be expected.

3.Deeper dive in the individual Machine Processing steps

When a machine is asked to access a FAIR Digital Object (DO) given a PID or a URI reference, the FDO specification provides a high level template of what an FDO may look like but there are many details that need to be specified in order to make this work consistently across a wide range of FDOs using diverse underlying technology.

To enable developers to implement software that can automatically access and process any FDOs, the FDO specifications need to be expanded into a more complete set of technical steps and rules specific to each choice of implementation technology.
Furthermore, additional FDO specifications may need to describe how to implement solutions to interact with FDOs implemented using different underlying technologies.

It is expected that these technical specifications will be instrumental in delineating the levels of FDO compatibility within a specific and across different implementation technologies.

The steps below are meant to be a starting point from which to start identifying some of the core compatibility issues and the needed specifications that will help machines process FDOs of different compatibility levels.

The following set of steps start with a client being given a PID. This does not happen in a vacuum: The client may have no information as to what to expect or the client is maybe looking for metadata or data from a previous loop cycle.

1. Determine the nature of the PID

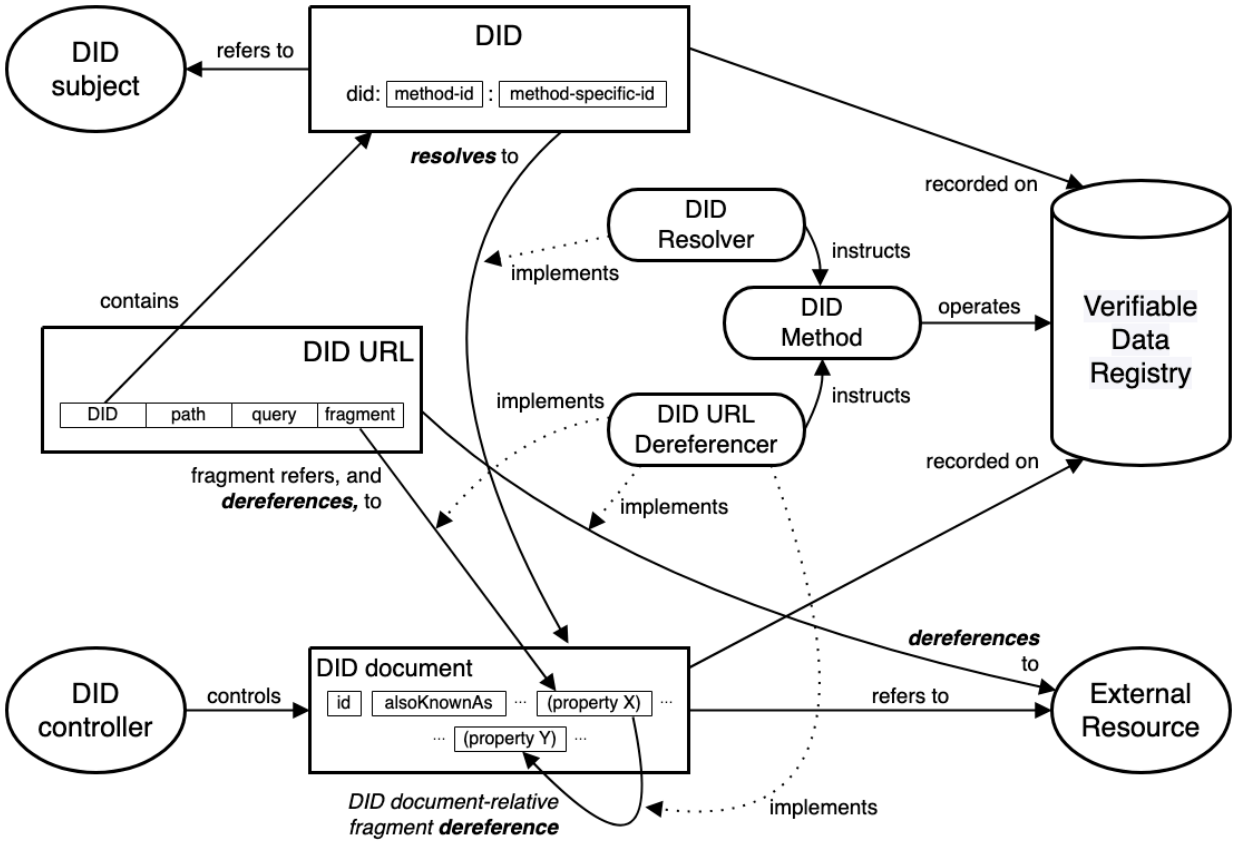
    1.1. Is the PID a URI?

        1.1.1. Is it a proxy handle or DOI? Goto 2.1.1
        1.1.2. From the DNS name can I assume something about the PID??…
        1.1.3. If the URI starts with HTTP will try HEAD/GET.
        1.1.4. If the URI starts with SFTP:…
        1.1.5. If the URI starts with SCP:…
        1.1.6. If the URI starts with URN:...
        1.1.7. Other…
        1.1.8. Cannot figure out the nature of the specific URI.
            1.1.8.1. Try to see if anything comes up.
            1.1.8.2. Abort

    1.2. Does the PID require a PID specific resolver?
        1.2.1. I can determine which one.
        1.2.2. Is there a resource I can use to resolve PIDs I am not familiar with?
            1.2.2.1. If yes, use the resource
            1.2.2.2. If not Abort

Cannot figure it out. Abort.

    1.3. Is it a DID? (Decentralized IDs) (Drop for now)
        The resolution process is DID is complicated….



1.4. Is it a DOIP reference?

1.5. Other systems that have their own service access points.

2. Select PID resolution mechanism

2.1. For Handles / DOI (Level1 interoperabiliy)

2.1.1. Use Handle Web Proxy using http interface
2.1.2. Using handle protocol library

2.2. For HTTP/HTTPS URI - Determine the request.

2.2.1. Are there some indications in the structure of the URI of the type of information I will get? (URI ends in .json or .xml).
2.2.1.1. Yes: Use GET.
2.2.1.2. No: Use HEAD
2.2.2. Are there some indications in the syntax of the URI of the type of information I will get?

2.3. For URN

2.3.1. Use namespace / domain specific protocol or Web Proxy using http interface

2.4. For DID (strike that out) (Level 3)

2.4.1. Acquire proper operation.
2.4.2. Does it return something I know how to process.

2.5. For DOIP
2.5.1. Determine the supported protocol implementation

2.6. Others
2.6.1. SFTP, SCP, …

2.7. Cannot find a resolution mechanism: Abort

3. Issue ID resolution request to a record as per the resolution mechanism selected above
3.1. Resolution success - continue
3.2. Resolution requests authentication
3.2.1. The authentication approach is known (this is an entire other section)
3.2.1.1. Authenticate using one of the approaches known to the client.
3.2.1.1.1. Success, re-issue 3.
3.2.1.1.2. Failure,
3.2.1.1.2.1. If there is another authentication approach use it and goto 3.2.1.1.
3.2.1.1.2.2. Failure to negotiate access.
3.3. Resolution failure
3.3.1. If there is an alternative resolution option, select it and goto 3.
3.3.2. There are no other solutions - Abort.

4. Process Return

4.1.1. Record returned as expected.
4.1.2. Record returned is not as expected
4.1.2.1. If http request, look for MIME headers, could it be a different format?
4.1.2.2. Abort
4.1.3. The resolution returned nothing. Maybe the wrong request? Try other request. If no other possible request abort.
4.1.4. The resolution request resulted in a error. Maybe the wrong request? Try other request. If no other possible request abort.

5. Parse the record into type value pairs
5.1. If error parsing record - Abort

6. Look for the expected types using the type template. The set of types found can be used to determine how well formed the FDO is. All of the types need to be FDO PID.

FDO-Type	Explicitly specifies itself as an FDO
FDO-Metadata	Specifies that there is FDO Metadata in-line. Note that this requires information about the specific type of metadata and format (DC in JSON for instance)

FDO-Metadata-REF	Specifies that the metadata is accessible by reference.
FDO-Data	Specifies that there is in-line data. As with the metadata, there is a need for the type of this data.
FDO-Data-REF	Specifies that the metadata is accessible by reference

If the client does not find these types, it could try to look for other types that it knows which could include other known types defined as FDOs, MIME types, magic numbers etc. If the client does not have the ability to find anything it will give up.
As far as access restriction types, it is not clear what those would look like. Harsh words are not enough to keep people at bay. Access control will need negotiations.

7. Type Processing:

    7.1. FDO-Type
        7.1.1. FDO Type is what I am looking for
        7.1.2. I resolve the FDO type to get more information
            7.1.2.1. This type is related to the type I am looking for.
            7.1.2.2. ?

    7.2. FDO-Metadata
        7.2.1. If by reference, recurse on the loop.
        7.2.2. If by instance, process it to determine whether this is the FDO we are looking for.

    7.3. FDO Data
        7.3.1. If by reference, recurse on the loop
        7.3.2. If by instance read it.

    7.4. Other types

Need to integrate Operations within this model.

FDO = GetFDO(a)
FDO.ListOps() => null;

FDO.listOps()={100/GetMetadata, 100/GetData}
FDO.runOps(100/GetMetadata)

========== original document ends here ==========

Potential associated discussion topics

- (from Maggie: I would like to add something related to how a script should deal with the situation that yes, we have found a properly described FDO fulfilling the tech specification, and it contains a (pointer to) a bitstream of data. However, to actually access and operate on that bitstream requires authorisation of some sort - for example obtaining an access token via some identification & authentication step. Should these kinds of conditions/limits be explicitly encoded at the level of the FDO kernel information profile, or should there just be some kind of flag (indicating "you will need human input and/or extensions to your code to proceed further")? Should the act of "unlocking the gate" be treated as a (regular) operation on the FDO (bitstream)? (Apologies in advance if my questions don't belong in this document…)
- …
- …

Some coding notes Stian & Christophe 2024-06-26

To describe the processing of the FDO we could write some ABNF. That may be too structural in nature and not reflect the conditional processing that would take place.
Write some ABNF
Class fdo

    getFDOType
    getOperations
    getTuple

A programmable / pseudo code approach would to more justice to the complexity that we are trying to describe.

```
T fdo2 = fdo("20.123123/12398231").data[0]
m = fdo2.metadata.get()
print(m)
```

Stian sketched out some Java classes in
https://github.com/stain/fdobenchmark/tree/main/lib/src/main/java/org/fairdo/benchmark/api
(inspired by
https://github.com/apache/commons-rdf/blob/master/commons-rdf-api/src/main/java/org/apache/commons/rdf/api/RDF.java)

Web IDL https://webidl.spec.whatwg.org/ see https://webidl.spec.whatwg.org/#example-25f60a7c

The topic of reference within an FDO (or serialized in some other form) needs to be revisited.
FDO
FDO Reference:

    -URI

-type
-processingtype
-title
-housecleaning (hash, signature timestamp)
-etc

fdo:URI:type:processingtype:ref:housecleaning
fdo:Uri:ref

Metadata-Ref: URI or FDO Reference
Metadata: type, titled, housecleaning,etc metadata
Data-ref: FDO Reference
Data: type, titled, housecleaning,etc metadata
Operation
GetMetadataRef
GetMetadataInfo
GetMetadata(json, DC)

Use the code to build the ingestion and reading of FDO. Include “drivers” for each solution.