Talk__BitstreamFormat+Renovation

Talk:BitstreamFormat Renovation

From DSpace Wiki

Jump to: navigation, search

Re: container formats... I understand the rationale, but what will we do about e.g. Apple Keynote and ODF files? Can we build in the option to open the file and check its innards for understandable pieces?Since there is only one `BitstreamFormat` object to describe the format of the Bitstream, there isn't any place to describe the inner formats. Unless you customize the archive to add formats for, e.g. "Zip file containing Word document", "Zip file containing PDF", etc.. which gets pretty unwieldy. I don't believe it's worth adding the complexity of nested formats when there is no use case for that extra information.

For a format like ODF, it's implied by the definition of the format that it *can* have text documents, spreadsheets, presentations, etc. within a document. In fact, PRONOM has subtypes for e.g. "ODF Drawing". So, I expect an ODF compound document would be identified as such, and if the Bitstream is an ODF document with just a drawing, it would be identified as the more precise subtype "ODF Drawing". It's up to the format identification method how aggressively to classify the Bitstream. LarryStone
Re section 4.2.6.2 Remove these methodsGranted that MIMEType is a general sort of category, I think it might be useful to retain the findByMIMEType() method. BrianF-S*Since many formats can (and do) share the same MIME type, it is not precise enough to identify a format.* Automatic format identification is better off ignoring whatever MIME type was attached to the Bitstream on ingest. The method existed before to assign a BitstreamFormat based on the MIME type accompanying a Bitstream on ingest; with good automatic identification we can do better. --LarryStone

Robert Tansley 31-Jul-07

You've incorrectly assumed the original intent of the 'internal' flag (you never asked?). It was originally intended to flag formats that were DSpace-specific and for DSpace's internal use. Hence it was intended to describe the format. That the UI used this as a hint was a side effect; further, others started to misuse the flag and/or use Bundles as the hint instead.I did post this question to a list some months ago. "License", in particular, does *not* describe the format in any useful way since it is erroneously applied to RDF/XML, plain text, and HTML files. I don't believe BitstreamFormat needs an "internal" bit at all; should there be two separate BitstreamFormat's for the "RDF/XML" format, one with the internal bit set and one without? If anything, that bit belogns on the Bitstream object. Then it could usefully identifiy e.g. Bitstreams used as logo images in the UIs. --LarryStone
You still haven't addressed the .avi use case, where the encodings of streams needs to be found (never responded to my comment on 'About Data Formats'.) Assuming 1-1 Bitstream - Bitstream Format is naive and not sufficient for numerous use cases (video, multimedia presos, XML, etc.) You can quite reasonably choose not to address this in the first pass but saying 'there are no use cases' is incorrect.I looked into AVI; it's a Microsoft-proprietary wrapper format that is specialized to contain mostly video and audio streams (of various formats) embedded in it. Logically it's similar to a restriction on a container format like Zip.
I've observed container formats can mostly be classified as one of these two:

1. General-purpose container like Unix "tar" or Zip, member files of any formats with no restrictions or expectations.
2. Specialized container for a particular application with structure and restrictions imposed on contents, e.g. JAR, WAR, IMSCP, AVI.

In the first case, it's easy to argue that the container itself is not significant and if you want the contents of the container *preserved* in the archive, unpack the container (and preserve its existence as logical relationships in the metadata of the individual Bitstreams). This is how we handled IMSCP for OCW. Alternately, treat the container as one Bitstream, it has a format.. but the contents of the container are invisible to the archive.

In the second case, the container is a logical unit; it is only meaningful to applications which expect and understand it. It should be treated as a single file by the archive. Any preservation tools that handle it will have to know how to unpack and repack it, anyway.

So what I mean by *use case* is not "are there files that *could* be described as a hierarchy of formats", but "is there a purpose to describing the format of a Bitstream as a hierarchy of formats" in the archive, what would DSpace do with that knowledge?

Even the GDFR format model (the most sophisticated one around) gives up on describing instances of container formats. It describes a container format as *possibly* containing or *necessarily* containing such-and-such other formats, but all instances of the container are described by that same format entry. --LarryStone
The set of 'quality' levels seems rather arbitrary. A real no. between 0 + 1 might work better, perhaps with 'user' as a special value. 'Confidence' would be a better term. That USER 'overrides' an automated identification seems problematic -- end users are not necessarily the experts on exact format identification.Is your reason behind a real number to allow later additions of constants in between the current ones? I agree that's worthwhile, but we can do that with an integer, just space out the initial values by 100 or so (remember BASIC line numbers?). I don't believe the level should be real, since that gives identification methods no basis for comparison. The level values are a small number of enumerated points. I'll consider renaming "Quality" to "Confidence", although "confidence" has an unfortunate alternate meaning wrt. establishing veracity.

Re "User" as a value, it is given the highest value not because it is necessarily the best quality, but to reflect that gets to override all other identification methods. Maybe it can be renamed "USER_OVERRIDE".

The mix of externally-sourced metadata from registries and internal metadata (and overrides) is concerning, and contradicts one of the design goals (registry as cache) and your point about how the registry is not the place for info that 'have nothing to do with describing the format of the data'. Also, if a DSpace instance overrides values, how will you know whether to update when you next sync with the central repo?There is an override bit for each metadata field that can have an override (name, description, mime-type, and canonical extension). It's not the greatest idea from a purity point of view, but I can anticipate archive administrators asking for it -- already happened in the case of the MIME type, and canonical extension. I can forsee cases where the format registry has an overly technical or cluttered name for a format that the archive administrator wants to simplify so as not to scare end-users.
External format registries have different specificities. How are you planning to address this? It seems a little concerning that some curators might think that BSF X in the DSpace registry 'is' format Y from GDFR but actually has different specificity (e.g. bitstreams tagged with this would more appropriately be tagged as more specific GDFR format Z).That's quite true, and it's why (a) the Bitstream includes a quality metric that indicates how precise the identified format is, and (b) more specific formats rate as higher quality. It is the duty of the FormatIdentifier to award the quality as it makes the identification, so presumably it has a good notion of the specificity of the format it has identified.
I considered putting a ratign of specificity in the BSF, but gave up on it because the different registries have different concepts of specificity - does the "superclass" format just describe a family of related formats, or an actual superclass that matches all of them, etc.. GDFR has an extensive relationship model that is probably a superset of other current registries, but that's only for now. Rather than try representing relationships in DSpace BSFs, I think it makes more sense to tell anyone who cares deeply about format relationships to look at the namespaced external identifiers and go back to the source registry with its own data model. That's why you can get the configured "contact" URI for a registry, to hand it to the registry client code to use the same file or network resource.

Also, the BSF isn't a registry in itself, it's a vehicle to apply formats in external registries, so as soon as a curator needs any real information about the format she'll haev to follow the external identifier to the GDFR or whatever.

The conformsTo() method is the one exception, because there is a clear use for it in matching up bitstreams with configured applications. Additionally, it seems likely that identification tools will differ in opinion on some formats. So, why limit the Bitstream-Format relationship to 1-1? e.g. a Bitstream might have several formats with different confidences. This could be used as the basis of container formats later on. To simplify the UI etc you can simply use the format with the highest confidence. Perhaps this is overkill for a first pass, however.It's definitely overkill for what I need, what what I believe DSpace 1.6 needs.

Richard Rodgers 9-Aug-07

I also wonder about the value of 'quality' as it is structured - what does 'user' imply vis-a-vis quality? Rather than confidence, I'd maybe consider a 'source' or 'assigned by' field (which would also allow 'user' as a value). You already have a typing system for these, and then you could rather easily find all the DROID/XYZ assignments that were erronenous. It would also in a funny way actually really be a 'quality' or 'confidence' code over time, since the better FormatIdentifiers would replace the poorer ones.In a live discussion after this, I think the conclusion was that both *confidence* (quality) and *source* are useful in Bitstream. The confidence metric is useful to drive policy decisions, e.g. "re-evaluate all format identifications at lower than *heuristic* confidence."
In the FormatRegistryManager API, I wonder why the registry names are hard-coded constants - shouldn't all this be extensible without new code? You already use PLug-ins, etc for the implementation classes - so why restrict the name space in this way?The registry names *are* extensible, the constants are just there to indicate the default set of well-known registries. I have an aversion to "magic strings" in code.
Point taken, though, that this is misleading. I'll move the constant for each registry to its own plugin class.
Finally, although I think the infrastructure you describe for automatic discovery is flexible & useful, I'm a little concerned that legislating it as APIs constrains the solution space needlessly. There could well be local policy issues that override the 'highest-confidence' match algorithm, etc Why not just classify all that as a reference implementation, and confine the API to BitstreamFormat, FormatRegistry, and maybe FormatIdentifier? Also clarified a bit in discussion.. Richard was concerned that the overall algorithm for detecting formats is fixed by the framework that calls identification methods and integrates their results. Actually, each identification plugin is responsible for modifying the list to add its results, so it has ultimate power; the confidence-based algorithm shown is just an example. A plugin could replace the whole list with its results, or delete everything with too low a confidence, etc. You can substitute your own identification mechanism by configuring one plugin to call it, so in effect it *is* already fully extensible.
Since each identification method contributes its results to the constantly-changing result stack, the FormatHit structure is still needed to package those results since they don't get assigned to the Bitstream until all methods have been called. I will change the identification method signature so it *returns* a List, allowing complete replacement of the list or a remote procedure call.

I'll clarify this in the documentation, too. --LarryStone