WHITE PAPER

The Annotation Exchange Format (AXF) and Hypergraph Text Encoding Protocol (HXTN)

Part I: Hypergraph Text Encoding Protocol

Overview

The Hypergraph Text Encoding Protocol is a framework for detailed encoding of text documents. HTXN is not a markup language, although the HTXN "Reference Implementation" includes a new markup format (called NGML, or "Next Generation Markup Language") which is one way to create HTXN documents. Instead, HTXN is a binary encoding that can represent the structure of several different markup formats, including XML, LaTeX, and RDF. HTXN allows document collections to have a common representation, even if the original documents are in different formats. HTXN also facilitates converting between formats; for example, one document can be represented in both LaTeX and XML.

The motivations for HTXN are both practical and technological. Practically speaking, mismatch between document formats can be a noticeable impediment to editing and fine-tuning publications, and integrating them into digital ecosystems. For example, converting from LaTeX to XML discards document anchors and auxiliary data that could be used to externally cite specific locations in a publication ("Micro-Citations" go hand-in-hand with fine-grained textual annotations, because PDF viewers can, given proper metadata, browse to the exact line where a person, data structure, or technical concept appears in a document). Technologically, different formats each have their own unique virtues. LaTeX, in many people's opinion, creates the most visually compelling documents, and also has sophisticated auxiliary-file and data-generation features (this auxiliary data is increasingly important in a publishing industry where text-based content, such as PDF files, are only one part of an integrated digital platform). LaTeX, however, has no established proofing and change-tracking protocol. XML is architecturally well-designed for multi-party editing and tracking, but typically yields mediocre (HTML-based) renderings. RDF, by supporting concurrent and overlapping markup, is more expressive than XML and LaTeX, but it lacks LaTeX's pre-processing (auxiliary data) capabilities as well as XML's post-processing document-interaction functionality.

Rather than accept the limitations of any one format, HTXN embraces a philosophy of encoding documents in its own internal system, then generating "views" onto the document in a range of structures, so as to benefit from different formats' features at different stages of finalizing a publication. An HTXN file may be converted to LaTeX to generate auxiliary data, then converted to XML for collaborative editing, and finally converted to RDF (or another graph representation) for searching and indexing.

HTXN Features

Apart from its overarching philosophy, HTXN's distinguishing features include:

1. A unified document-structure model which accommodates the idiosyncracies of different markup styles, such as attribute child nodes in XML, optional arguments in LaTeX, subject defaults in RDF, and concurrent/overlapping markup as in TeXMECS.
2. An encoding built from the ground up to support multi-party editing and collaboration. Internally, HTXN uses "stand-off" annotations which logically separate markup content from the underlying text. Multiple sources can thereby contribute distinct markup structure, which may be used, for example, to differentiate document changes made by authors from those made by editors.

3. A convenient architecture for "subordinate" documents which expose some restricted portion of their "parent" documents. This is also useful for editing. Starting with an overall HTXN parent, one can selectively create a subordinate document which (for example) includes only editable text, yielding a simplified version of the document suitable for textual revisions. After editing, changes made to the subordinate document are then folded into its parent.
4. A detailed system for pairing documents with external data and data sets. In particular, HTXN supports document anchors allowing structured data within a data set to reference locations in accompanying publications. These back-references could be used, for example, to identify points in a publication where there is a definition, explanation, review, or visualization of technical concepts and/or data aggregates introduced in a data set. Unlike LaTeX references or HTML anchors, these HTXN anchors are data-specific and logically separated from textual cross-references.
5. Several special features for data-based annotations, or ascribing metadata or interpretations to text segments which conform to structured data types (proper names, acronyms, dates, times, magnitudes, etc.). This includes numeric types typically unrepresented in most markup formats, such as range-delimited integer types and "Universal Numbers", or "posits", a recent alternative to floating-point decimals.
6. Functionality for creating data sets directly from the text itself. For instance, example sentences used within linguistic publications, to illustrate semantic or syntactic principles, can be compiled as a sentence-corpus to produce a distinct data set associated with the original publication. Larger corpora can then be compiled from collections of linguistic texts. Similar techniques can be employed in analogous "digital humanities" contexts; for example, extracting a corpus of discussions about cultural artifacts by extracting sentences or paragraphs pertaining to an externally identifiable object (such as an artwork held in a museum).
7. A framework for "complex microcitations", meaning individually citable locations in a document which involve multipart data structures. A case study would be linguistic examples that display (alongside or within a sentence) supplemental visual cues such as intonation, prosody, lexical category, morphology, or grammatical-structural annotations. Such material needs special typesetting instructions, but the visual format in turn depends on formal sentence models. For another example, graphics such as charts or plots may represent a specific statistical view on a data set; the construction of that distinct perspective marshals formal data, such as dependent and independent variables, axes, scales, scaling factors (e.g. scalar multipliers or logarithms), and ranges. In these examples, textual content is not only a presentation layer which merely exposits or reviews scientific frameworks; instead, the code describing textual structure and appearance reflects scientific paradigms and is therefore a theoretical artifact in its own right. In particular, textual anchors and typesetting reflects not only raw text but structured data whose format and provenance represents specific scientific models. Here it is valuable to cross-reference text with data sources: typesetting code may be generated directly from a data set, while the generated instructions (in LaTeX, say) can trigger anchor labels and page references to be "forward-referenced" into the data set via auxiliary files. Complete cross-references thereby integrate raw and textual data, yielding *bi-directional* complex microcitations.
8. A Qt-based Reference Implementation which conveniently documents the HTXN specifications. With no external dependencies apart from Qt Creator (a C++ Integrated Development Environment), authors can construct HTXN documents via NGML and, if desired, examine all parsing and encoding code (both statically and, dynamically, during document creation). The Reference Implementation also supports importing and conversions to HTXN from XML, using Qt's XML module.
9. Options for stashing compiled HTXN documents in a database, so that saving files containing text serialization (in NGML, for example) is not a prerequisite for maintaining persistent copies of HTXN documents. The Reference Implementation supports persistent HTXN documents via the WhiteDB database engine.
10. A flexible parsing engine which allows input languages for generating HTXN documents to be

customized and extended.

HTXN Architecture

Internally, HTXN uses a Hypergraph representation where nodes are ranges pointing toward character streams in a separate multi-layered buffer (zero-length ranges represent anchors and locations within documents). Ranges (and so therefore their corresponding nodes) can overlap or encompass one another in any combination. The data establishing nodes' ranges and markup specifications (such as tag/command names) is separated from the text layers.

Also external to the text layers are extra data that affects the interpretation of individual characters (for example, to accommodate special symbols). By factoring out this supplemental information, each HTXN character has a consistent bit-length (typically one or two bytes, or a 10-bit pack corresponding to two base-32 digits). In general, each HTXN character also has a visual representation (a "glyph"), because non-visual markup is modeled via standoff annotations rather than via content embedded in the text itself. In particular, HTXN does not internally use XML entity references, Unicode points, diacritics, or other variant-length constructions where multiple physical characters correspond to one single text character. Instead, any text character needing such supplemental detail is flagged via special codes, and an external table is used to retrieve that character's specifications. The rationale for this design is to fully separate the markup-related processes of defining and annotating *ranges*, from the text-encoding requirements concerning foreign letters, special symbols, and other nonstandard characters.

Instead of statically modeling character encodings, HTXN provides an *interface-based* encoding protocol. This means that HTXN will recognize any encoding of a character, a character-set, a character-stream, and a character-stream offset range, so long as it is possible to obtain certain specific pieces of information with regard to the object in question. At the lowest level, this means that character-sets and character-streams are implemented via C++ classes which inherit from an abstract base class. Users can then build their own character sets by implementing their own C++ classes. At a higher level, one character-set class can provide multiple character-set instances by loading data from an external file. In either case, HTXN's character-encoding mechanism is orthogonal to encodings used for XML, LaTeX, Unicode, or any other markup/encoding format (though character sets can provide conversions to these formats as needed). This promotes HTXN's capabilities to convert between and to integrate multiple markup protocols, such as LaTeX and XML. More generally, it allows programmers to use Object-Oriented coding techniques to fine-tune the text-encoding process more aggressively than is possible with most markup and text-encoding formats.

Document Editing

One benefit of HTXN encoding concerns how HTXN may be integrated with those application which allow multiple users to edit the text at hand.

As already described, a parent HTXN document may be paired with a subordinate document providing restricted access to the parent. This schema applies in contexts where only *restricted editing* is appropriate. In contrast to *unrestricted editing*, restricted editing proceeds in the context of rules limiting which editing actions are possible. In particular, restrictions may stipulate that only certain document content — such as text that will be visible in the final publication — may be modified. Restrictions may also limit the degree to which a document's graph structure is altered. Added nodes, for example, may be required to fit inside other nodes (disabling interwoven markup that is normally accepted in HTXN). Added nodes may also be restricted to a subset of tags/commands, such as those providing visual markup (like italics and boldface) for published text, or metadata which permits annotations without changing the text or its appearance.

In order to enforce such restrictions, editors may be prevented from modifying HTXN documents as raw text files; this in turn demands software which presents views onto documents' contents other than through text serialization. For these use cases, HTXN natively supports integration with Qt

applications. In particular, subordinate HTXN documents can be rendered via the QTextDocument class, which allows restricted editing of visible textual content without directly changing the underlying HTXN data. With QTextDocument, editing actions can be initiated through a responsive native front end supporting context menus, secondary windows, undo/redo, and other convenient features associated with native desktop GUIs. Alternatively, subordinate documents may be converted to HTML and visualized through QWebEngineView, which offers similar interactive functionality.

Each edit initiated through QTextDocument or QWebEngineView may then generate a data structure through which the changes are registered in subordinate and parent HTXN documents. These data structures, describing individual edits and groups of edits, may then be shared over HTTP via QNetworkManager and related classes. This allows multiple users to synchronize their local document views in collaborative-editing contexts, and simultaneously allows publishers to track changes and maintain an official version for publication. Maintaining one canonical document — so that multiple users (e.g. authors, volume editors, series editors, and production editors) may update the document in real time — is obviously more efficient than users sending successive copies of their document back and forth. This, of course, is one reason why web-based collaborative editing is popular. The improvements introduced by HTXN are first that co-edited documents are actually subordinate versions of parent documents which have more complex typesetting and metadata capabilities. In so doing, the changes introduced in the subordinate document are thereby (as desired) reflected in the parent document. Second, the editing GUIs can be part of native-compiled applications with a full range of desktop-style functionality rather than web portals which have a more restricted interactive experience. In short, with the HTXN framework the publication (books and journals) production process would be accelerated by virtue of HTXN's native-application features. Other benefits of adopting HTXN are identified in the following subsection.

HTXN Benefits

HTXN can generate documents in a variety of different representations (such as LaTeX, XML, HTML, and RDF), and enables publication workflows to combine two or more existing formats (e.g., LaTeX and XML). A document with HTXN encoding will generate LaTeX or XML "views" which can each be constructed for use at different stages in the document-preparation pipeline. HTXN is also designed to work with publications that are embedded in a larger data or multi-media ecosystem — in particular, documents that are paired with data sets or multi-media content that need specialized software. These priorities yield several benefits, including:

Mix-and-Match Formats HTXN employs "stand-off" annotation, which allows multiple markup protocols to be defined simultaneously on the same document. In addition, HTXN uses a flexible character-encoding protocol that ensures compatibility with diverse text representations (such as XML, LaTeX, Unicode, and Qt/QString). Via secondary documents or "views" (generated from the primary HTXN document), HTXN can be used wherever XML or LaTeX (or other formats) are desired.

Fine-Grained Character Encoding The HTXN protocol includes more detailed document structure information compared to conventional markup. For example, HTXN identifies sentence boundaries and punctuation features, disambiguating logically distinct but often visually identical characters (such as dots/periods or dashes/hyphens, which may have different structural meanings in different context

Multi-Media and Data Set Integration HTXN is optimized for sophisticated publishing environments which combine conventional natural-language texts with interactive data sets and software. Contemporary publishing increasingly sees academic texts as part of a multi-faceted ecosystem, where publications, research data, and interactive software can be packaged into multi-media presentations. HTXN ensures that documents can be properly experienced within these augmented publishing platforms.

Complex Anchors and Interoperability

HTXN introduces a variation on micro-citations and conventional HTML or LaTeX anchors/labels, which we call "complex anchors", to facilitate interoperability with research data sets and external software. Complex anchors permit concepts, terms, and data structures to be tracked and cross-referenced between publications and their associated data sets or supplemental software. Consider a table-structured data set whose columns represent scientific concepts or statistical parameters that are discussed in an associated publication (e.g. article or book chapter), or where data from a specific table is presented graphically as a figure within the publication. Via complex anchors, these kinds of thematic relationships between data-set elements and corresponding points in the research text may be made explicit, so that data-set applications (including software expressly implemented for a newly published data set) can recognize references "pointing in" to the publication. In the context of tabular data, data-set software could provide context-menu actions associated with each column (or, in the case of figure illustrations, the overall table), linking to an anchor in the text where the relevant concepts, terminology, or visualizations are explained.

This level of interoperation can be achieved by implementing document viewers (such as a PDF viewer) embedded in scientific software or software expressly designed for a data set. With embedded viewers, the operation of opening associated publications to specific anchor-points (such as figure illustrations) can be integrated as an interactive extension to their host applications, triggered by menu items or other GUI components. To support this interoperability, HTXN provides anchors whose identifiers and embedded data can be customized to work with domain-specific host applications.

Complex Exolinks and Interoperability

HTXN "exolinks" refer to data or multi-media content *outside* the text (so they are effectively the converse of complex anchors, which support references pointing *inside* the text from external software components). Complex exolinks designate data or multi-media content — such as video, audio, or 3D (scenes or panoramic-photography) graphics — which are outside the text, but potentially part of a larger publication package wherein an article is accompanied by data sets and/or multimedia content. Traditional external/HTTP references in HTML or PDF are designed to reference resources on the World Wide Web, but not content downloaded in a package along with publications themselves. Therefore, special reference formats are required for document viewers to properly handle links in the text that point to non-web content (which is a technical limitation present in conventional document encoding formats). HTXN will ensure that, when accessing multi-media content, properly encoded signals will be sent between the document-viewer components and applications which support the relevant multimedia file types (potentially host applications where the document viewers themselves may be embedded).

As a concrete example, consider a chemistry paper which is paired with 3D descriptions of a particular molecule. In order for readers to view this content, a signal needs to be sent to compatible molecular-visualization software (such as IQMOL — to cite a QT-based, open-source case-study). Analogously, when reading a paper concerning 3D radiology, signals may be sent to — or be read on components embedded within — 3D viewers such as MESHLAB (again QT/open-source). To support interoperability requirements as shown by these examples, the HTXN specification includes an HTXN-specific "Native Application Network" (HNAN) protocol to connect document viewers with external and/or host applications; HTXN exolinks encode data which ensures that the concordant HNAN signals are properly generated.

HTXN has numerous features for interoperation with software applications where document viewers may be embedded, or which support HNAN via plugins. In particular, HTXN parsers and generators are provided via a QT/C++ library that can be dropped in to the source code of native C/C++ applications, so it is straightforward to add HNAN to existing software. Based on QT (a C++ GUI and application-development framework), HTXN can leverage the full range of QT networking or front-end features (for instance, access to publishers' APIs can be introduced as a processing stage during document preparation). In addition, HTXN's graph-based parsing framework and hypergraph-based

annotation model can be customized and paired with project-specific GUI components for viewing and accessing data sets. Via such customizations, HTXN can be adapted to provide a domain-specific publication platform for specific research projects or environments (individual journals, book series, research labs and academic departments, etc.). HTXN's "reference implementation" includes a specially-designed "Next-Generation" markup language (NGML), as one way to build HTXN files. The NGML library is transparent and distributed in source-code form, so it can be modified to create domain-specific markup styles. A customized version of NGML can thereby be used to construct HTXN documents according to each individual lab's or journal's specifications.

Part II: Annotation Exchange Format for Images

Overview

Image annotation and segmentation is an important analytic process in many scientific, technical, and commercial fields. Nonetheless, there are few standard formats for describing and representing image annotations, and those which do exist tend to be used in specific, relatively narrow contexts.[1] This is not a new observation; Daniel L. Rubin *et. al.*, in 2007, note that:

> Images contain implicit knowledge about anatomy and abnormal structure that is deduced by the viewer of the pixel data, but this knowledge is generally not recorded in a structured manner nor directly linked to the image. [Moreover,] the *terminology* and *syntax* for describing images and what they contain varies, with no widely-adopted standards, resulting in limited interoperability. The contents of medical images are most frequently described and stored in free-text in an unstructured manner, limiting the ability of computers to analyze and access this information. There are no standard terminologies specifically for describing medical image contents — the imaging observations, the anatomy, and the pathology. [N]o comprehensive standard appropriate to medical imaging has yet been developed. A final challenge for medical imaging is that the particular information one wants to describe and annotate in medical images depends on the *context* — different types of images can be obtained for different purposes, and the types of annotations that should be created (the "annotation requirements" for images) depends on that context. For example, in images of the abdomen of a cancer patient (the context is "cancer" and "abdominal region"), we would want annotations to describe the liver (an organ in the abdominal region), and if there is a cancer in the liver, then there should be a description of the margins of the cancer (the appearance of the cancer on the image). (http://cedarweb.vsp.ucar.edu/wiki/images/d/d9/R_19.pdf, pp. 1-2).

These challenges inspired AIM (the "Annotation and Image Markup" project), which "provides a solution to the ... imaging challenges [of]: No agreed upon syntax for annotation and markup; No agreed upon semantics to describe annotations; No standard format ... for annotations and markup" (https://wiki.nci.nih.gov/display/AIM/Annotation+and+Image+Markup+-+AIM). However, AIM itself has not been widely adopted outside the specific field of cancer research and cancer-oriented image repositories.

One obstacle to formalizing image-annotation data is that annotations have a kind of intermediate status, neither intrinsic parts of an image nor merely visual cues supporting the presentation of the image within image-viewing software. Many applications exist which allow markup or comments to be introduced with respect to an image. From the application's point of view, these annotations are part of the application display, not part of the image — analogous to editing comments that might be added to a text document by a word processor or PDF viewer, which are records of user actions, not intrinsic to the document itself. Indeed, one mechanism for recording image annotations in DICOM (the "Digital Imaging and Communications in Medicine" format) is via "presentation state." The

[1] Current formats include AIM (Annotation and Image Markup), CVAT XML (CVAT is the Computer Vision Annotation Tool), DICOM-SR (Digital Imaging and Communications in Medicine Structured Reporting), PASCAL VOC XML (Pattern Analysis, Statistical Modeling and Computational Learning Visual Object Classes), and COCO JSON (Common Objects in Context).

presentation state includes all details about how the image currently appears to DICOM workstation users, such as Radiologists, which could include markings they have made to indicate diagnostically significant image regions or features. Insofar as image-annotations are considered to be artifacts of image-viewing software, rather than significant data structures in their own right, there is less motivation for imaging applications to support canonical annotation standards.

Nevertheless, in many scientific and technical areas image annotations *are* significant; they are intrinsic to the scientific value of a given image as an object of research or observation. Image regions, segments, and features have a semantic meaning outside the contexts of the applications that are used to view the corresponding images, which is why it is important to develop cross-application standards for describing and affixing data to image annotations.

It is also important for image-annotation models to be broadly applicable and multi-disciplinary. While image analysis serves different goals in different contexts (e.g., segmentation of microscope images to detect cancer cells serves different ends than segmentation of camera snapshots to study traffic patterns), there is always a possibility of analytic techniques developed in one subject area to be applicable for other image-processing problems, even if the practical outcomes desired of the analyses are very different. Furthermore, certain computational domains are similar enough to image analysis to warrant inclusion in a general-purpose image-annotation framework, even if the underlying data is not "images" in the conventional sense (not, for instance, captured via photographs or microscopy). For example, PDF document views, Flow Cytometry (FCM) data plots, and geospatial maps subject to Geographic Information Systems (GIS) annotations may all be considered images — by virtue of a semantic significance attributed to color and to geometric primitives as a way of characterizing phenomena observed or modeled through their data — even though such resources are not acquired by ordinary "image-producing" devices. The "Pantheon" project, characterized as a "platform dedicated to knowledge engineering for the development of image processing applications,"[2] offers one of the few attempts in imaging literature to rigorously define "imaging" and "image processing" in the first place. Pantheon (via its PANDORE component) also includes an image processing *objectives* ontology, which is partially incorporated into AXFi, so Pantheon's articulation of the general domain of imaging *per se* applies to AXFi as well. The problem of defining "images" as such, and therefore delineating the scope of image annotation, is addressed below (page 14). In brief, AXFi considers the realm of imaging to be more general than just graphics obtained by a direct recording of the optics of some physical scene via cameras, microscopes, or telescopes. That is to say, the image acquisition process is not necessarily one where data is generated by an instrument which produces a digital artifact by absorbing light, so that geometric and chromatic properties of the image are wholly due to the functioning of the acquisition device.

Systematically identifying the scope of "image annotation" is important for AXFi because doing so clarifies the sorts of domains whose semantics could reasonably be incorporated into AXFi, as well as the sorts of applications which would be reasonable candidates for supporting AXFi annotations (i.e., the capability to parse AXFi data and represent it vis-à-vis the relevant images). For example, if immunoflourescent Flow Cytometry (FCM) data plots are classified as images, then the numerical properties of the "channel" axis, with notions of "decades" and a "log/linear" distinction, become relevant to the AXFi vocabulary for representing spatial dimensions and magnitudes. In general, AXFi uses paradigms and terminology from "Conceptual Space Theory" as part of the process of formalizing geometric and dimensional notions.[3]

When defining the scope of AXFi, it is also important to distinguish the *data models* encapsulated by AXFi resources from the file formats where AXFi data is encoded. The choice of one file type or another to represent a data structure — XML versus JSON, for instance — does not fundamentally affect the data thereby communicated. Therefore, it is important to formalize data models in such a way that numerous different serialization languages might actually be used to share/express the

[2] See https://hal.archives-ouvertes.fr/hal-00260065/document.
[3] See http://idwebhost-202-147.ethz.ch/Publications/RefConferences/ICSC_2009_AdamsRaubal_Camera-FINAL.pdf or https://arxiv.org/pdf/1801.03929.pdf.

data. However, in practice, format-specific standards, such as XML Schema Definitions, are often used as the basis for formalizing and enforcing compliance with data models. Therefore, AXFi cannot be completely unconcerned with the structure and requirements of files which convey AXFi data. This is particularly true because AXFi seeks to incorporate the data models of other specifications, such as AIM and GATING-ML, which are formalized via XML specifications. Although AXFi is not primarily XML-based, in short, it intends to be (in the relevant contexts) compatible with XML languages that rely on XML schematization. Further details on how AXFi manages the relation to XML and other serialization languages are provided below (page ??).

A further detail that should be clarified prior to expositing a formal outline of AXFi is that of how image-annotations originate. Sometimes, of course, annotations are manually introduced on images by human users of image-viewing software. On the other hand, automated image segmentation — or similar algorithmic or AI-driven image processing without human intervention — yields partitions of images into regions, or identification of semantically important locations in an image, therefore generating annotations computationally. In short, AXFi should support both human-generated and computer-generated annotations. This becomes complicated, however, insofar as image-processing may yield analyses which overlap with annotation objectives but may not intrinsically produce annotations in the conventional sense. For example, an algorithm to count the number of cars in a highway picture may rely on statistical analysis of some quantitative image feature — such as "zero-crossings" — without in fact producing determinate image segments.

It is important to remember that image processing operations do not always act directly on images themselves; sometimes algorithms are based instead on mathematical complexes derived from the image, but with their own quantitative properties. For instance, color-valued pixels may be replaced by matrices measuring the gradient of some image-feature field in eight directions around each point (an example would be "Sobel kernels" applied to the image intensity function). For a given "semantic" task — that is, an image-processing objective whose end-result is not just image-related data but some empirical observation — image segmentation, or other analyses yielding annotations, are a means to an end: one *way* to count cars is to delineate the edges of distinct cars in distinct segments, and then count the number of segments which result. However, statistical image-analysis may produce largely accurate results for such semantic tasks, given large image corpora (e.g., estimating traffic flows from highway cameras), without yielding artifacts such as human-visible segment representations. Or, in a different domain, AI-powered analysis of FCS (Flow Cytometry Standard) data could establish a largely accurate count of "events" (i.e., discrete FCS measurements of light-scattering and/or flourescent properties of cellular-scale entities) without manual "gating" (referring to the conventional practice of scientists using geometric annotations of FCS data-plots to isolate and thereby count different event-types). An AI-powered analysis of image features, or likewise of Flow Cytometry data, may yield calculations similar to those which for *human* users are achieved via image segmentation, manual gating, and similar operations which clearly yield annotation data. For AXFi, the complication arises when these AI mechanisms do not themselves yield results that would normally be considered annotations, but rather yield the desired empirical results for which the annotations would be a preliminary step — e.g., an approximate count of the number of cars in a highway photo, without a precise segmentation of the image marking the cars' respective borders.

AXFi ultimate considers these AI-related complications to be issues resolvable at the level of software, rather than standardized annotation models *per se*. An image annotation is, among other things, a visual (or viewable) record of some image-processing activity. If a radiologist manually clarifies a report to the effect that a given CT shows a tumor by circling the area where the tumor is visible, he or she is using the image annotation to communicate to others the thought-process which motivated the diagnostic conclusion. This is different than an AI-driven processor which would automatically demarcate an image segment outlining the tumor and use geometric properties of that segment to derive a pathological finding. In short, the data conveyed in an annotation — an image segment, rendered precisely, or rendered indirectly via a circle or polygon around the segment — may be *intrinsic* to an image-processing operation: it may be data acquired *at one stage* in an analytic

workflow. However, annotations may also be *retroactive*; if a radiologist circles a tumor, he or she has completed (at least mentally) the image analysis, and is using the analysis to summarize what occurred in the course of the analysis. Therefore, any image-processing task can be associated with *ex post facto* annotations which summarize the process even if they are not intrinsic to it.

To continue the example of counting cars from a highway photo, an **AI**-powered observation could be retroactively *justified* by providing an image segmentation where the number of car-segments matches the **AI** count. In lieu of precise segments, it may be simpler to provide location-points for the "geometric center" of each car, or the points furthest apart in the direction of each car's front-to-back — these may be the statistical signals used for the car-counting process. Analogously, facial recognition does not need to rely one segmenting out regions (eyes, nose, lips), but rather can be based on distances between individual points (such as the inner corners of each eye). But in any case, depending on the analytic algorithm used, it is often possible to identify some spatial/geometric feature or object that can be visualized in the image context, and that summarizes or legitimizes the analytic operation. This summarial data, then, can provide *retrospective* annotations which allow human viewers to understand and review the algorithmic process. In short, simply because image-processing tasks may not generate annotation data as part of their internal activity, it is still possible (and may be desirable) for the software operationalizing these tasks to implement annotation generators, where the resulting annotations document the operations for the scientific record and/or summarize them in **GUI** objects for the benefit of human viewers.

In general, then, **AXFi** distinguishes human-generated from computer-generated annotations, and moreover leaves open the possibility that some computer-generated annotations are *retrospective*: that instead of being internal to an imaging computation they are indirectly produced subsequent to such a computation, for purposes of documentation and validation. Annotations which are not *retrospective* are called *internal*, as in, internal to a given image-processing workflow.

Related to the distinction between *internal* and *retrospective* annotations, **AXFi** recognizes a contrast between "intrinsic" and "descriptive" annotation. An example of an *intrinsic* annotation might be an image segment, where calculating the boundary of the segment is intrinsic to a specific image-processing objective, whereas an *descriptive* annotation might be an arrow *pointing toward* that segment. Here the descriptive annotation is introduced primarily for the benefit of human viewers.

The intrinsic/descriptive distinction is not always clear-cut. Consider the following two cases: in one scenario, an image-segmentation routine precisely delineates a region of interest (e.g., an outline of a red bird against blue sky), notating the segmentation result via a two-color-depth transform of the original image. Image-viewing software then shows the segment indirectly by encircling it, producing, in effect, a secondary annotation intended to call attention to the primary (segment) annotation. Here, clearly, the segment itself (encoding by a separate two-toned image) is intrinsic to the original analysis, whereas the secondary annotation has a purely descriptive purpose.

However, consider an alternate scenario where the original segmentation is done with less precision (this may be the case where there is a more muted color differential between foreground and background). Imperfect segmentation may still be adequate for some semantic task (e.g., identifying a bird's species). An analysis might obtain a rough segmentation by marking certain points highlighted by an edge-detector, then protruding the convex hull of the region outward so as to be sure of encompassing the whole bird-segment within a polygon, albeit allowing some background pixels into the polygon as well. This approximate segment is only an indirect representation of the region of interest (which would be the avian sub-image clearly outlined with no background included), but for analytic purposes the rough polygon might be a reasonable substitute for the finer-grained segment, analogous to how a cubic or quartic polynomial may be an adequate approximation to a more complex curve. Depending on how it is used, the approximate segment may therefore be considered merely a visual cue connoting the region of interest, or a significant region in its own right. This contrast would, most likely, depend on whether the approximation is used itself as a basis for further analysis, or instead is mostly a presentation device. This usage-context would therefore

indicate whether an "imprecise" annotation (AXFi formally uses the term *approximative*) should be classified as *intrinsic* or *descriptive*.

All told then, annotations may be *internal* or *retrospective*, and *intrinsic* or *descriptive*. AXFi also contrast *computer-generated* from *human-generated* annotations (which are related to but not the same terms as *manual* and *automated*; see below, page 16). These distinctions are indepenedent of one another; retrospective annotations for instance could be either intrinsic or descriptive, and either computer-generated or human-generated.

With these preliminaries concerning the scope of AXFi clarified, the following sections will (1) outline the kind of data encoded via AXFi and (2) the relation between AXFi, data types, and applications/libraries that may use AXFi. For purposes of exposition, the following specifications will discuss images mostly in two (spatial) dimensions; however, AXFi does not preclude annotations applied to image-sources with other dimensional profiles, such as 3D graphics, or movies (2D plus time), or even audio files (consistent with DICOM-SR, which supports audio annotations because DICOM is sometimes used to share biomedical acoustical data, such as ultrasound).

Types of Image Annotations

The most fundamental kinds of image annotations are geometric primitives such as points, lines, and polygons. Many other forms of annotations are possible, however, mostly supplemental data which is associated with these primitives or, in some cases, with the image as a whole. In general, AXFi classifies annotations into the following groups:

Geometric Primitives These annotations delineate spatial/geometric regions in zero, one, or two dimensions (or potentially higher dimensions when working in non-2D contexts). At a minimum, AXFi data should support points, lines, polygons, polylines (considered a superkind of polygons where a polygon is a closed polyline), circles, and ellipses. Additionally, AXFi recognizes generic "closed" and "open" *curves*, which are nonlinear one-dimensional regions (or boundaries of two-dimensional regions) that are neither elliptical or circular arcs. Depending on context, AXFi can be extended to provide more detailed subkinds of curves generated by different sorts of mathematical equations, along with notations for the formulae (e.g., b-splines) that generate a particular curve.[4] More information about encoding ellipse data as well as other sorts of curves is outlined below (page 14).

A variation on the polygon/polygon-line alternative are polygon-lines demarcating spatial regions whose boundaries extend to one or more sides/corners of the image (e.g., a "quadrant" is defined via one central point with line segments parallel to and implicitly extended to the edges). These poly-lines may not explicitly represent the overall image boundary as part of their own boundary. Conceptually, treating the image-border itself as a different *sort* of boundary than those explicitly notated may be more accurate than eliding these distinctions. This applies also to segment boundaries. For instance, the sand/ocean border in a beach scene represents a physical discontinuity between two different material substances, and so it records an observable detail of the depicted scene. On the other hand, the edge of the sand-region at the bottom of the image is not a physical boundary, but an artifact of the camera position; we assume the beach itself extends further than what the camera captures. All told, then, in addition to polygons (or curves or segment-boundaries) being *open* or *closed*, AXFi recognizes a third form of closure dubbed "incomplete," meaning that a spatial region is geometrically closed by the edge of the image but that this closure has no observational or semantic significance. AXFi allows a notation that a spatial region is "closed by fiat" when the closure results from the intrusion of the global image boundary. A polygonal line may be closed by fiat when it does not explicitly include lines along the global boundary, but forms a

[4] In formal statements, this and other AXFi documentation will use the term "kind," as well as "superkind" and "subkind," to indicate groups of values/entities identified via a classification. Vocabulary based on "kind" is preferred to "type" or "class" because of the distinct meanings these latter terms have in computational contexts which are also discussed in reference to AXFi.

closed polygon when such lines are included in practice; the result is called a *fiat polygon* (similarly an annotation can be classified as a *fiat curve* or *fiat segment*).

Segments and Regions A *segment* is considered to be, canonically, an integral subimage with a semantically precise separation of "foreground" from "background." There may be vagueness or approximation in how the segment is precisely individuated from its surroundings, but these ambiguities are considered to be practical limitations due to limited computer power, limiting pixel resolution, and so forth. A *region* or *region of interest* is similar to a segment, but defined more loosely; regions can have vague descriptors, and spatially disconnected parts of an image could be treated as parts of one same region. In a photograph of a flock of birds, for example, there may be multiple segments, each outlining one single bird. However, the flock as a whole may be outlined by one *region*, which could (without being deemed approximative) include some of the background sky. A *segment* can be seen as subkind of *region* with stricter granular and topological requirements.

A *partition* of an image is a segmentation which exhaustively classifies every point into one or another segment. A *selective* segmentation, unlike a complete partition, only isolates certain segments or regions of interests. AXFi adopts these terms as parameters that can characterize segmentation processes, and by extension the resulting segments/regions.

Locations In AXFi, *locations* are considered to be designations of areas within an image (of varying dimensions) which are significant by virtue of their directional, morphological, or topological relations to the rest of the image, rather than by virtue of their intrinsic shape. Conceptually, a *location* is in many cases similar to a *point*, but AXFi does not require locations to be zero-dimensional. A location may be designated by a small disk, or even a region/segment. The distinguishing feature of locations is that their spatial shape or extent are not semantically significant; instead, what is important about locations is their position in the image and how this position relates to surrounding image content. For instance, a location might be the point/position where two roads intersect, or it could be the leftmost point on the segment-boundary of a car's fender or a bird's wings, or the geometric center of a car's body or a bird's torso.

AXFi distinguishes location-annotations from secondary annotations used to identify locations (insofar as these may be visually distinct). For instance, a position may be modeled as a single point, but visually conveyed by an arrow, or by a circle hovering above the relevant point.

Focal Points The concept of *focal points* integrates locations and geometric primitives for certain analytic tasks. In general, focal points are important for positional rather than morphological regions, similar to locations. However, focal points may function more like segments or geometric objects when used as part of an analytic objective. For instance, points embodying the center of a car's body or a bird's torso could be used to count the number of birds or cars appearing in a photograph. In this case the concept of "geometric center" has no meaningful morphological properties, so it is analogous to a location. On the other hand, the center may be treated as a proxy for, or a most significant component of, a segment or region; in this sense the point is *part of* a segment. This mereological aspect makes focal points act conceptually more like geometric primitives than like locations. In short, the classification *focal point* is available for spatial objects which behave somewhere between locations and regions/points, and particularly when they are used in some proxying or indicative relation to other regions (e.g. for counting).

Secondary Images In some contexts, such as segmentation, image-transforms are used to convey image-processing operations, in contrast to geometric-style annotations that can be defined via vertex coordinates. A crisp segment can be defined via a two-toned image with the same dimensions as the annotated image, but with only two logical colors (colors are called "logical" insofar as the relevant detail is how the colors compare to one another, irregardless of the optical colors used to render them[5]). A "fuzzy" segment can likewise be defined via a greyscale image (anything which

[5] In Qt, for instance, the **QColorConstants::Color0** and **QColorConstants::Color1** color values are considered to be special

is pure-background becoming either pure white, or pure transparent, depending on context). Secondary images can be used to denote annotations which are too granular to be summarized by any mathematical expression. In these cases, the actual annotation data should identify the secondary image (e.g., via a file path) and explain how it relates to the primary (or "ground") image, whereas the secondary file itself fills in the annotation details.

Secondary images may be derived from ground images merely to present annotations, but they may also be intermediate analysands which are themselves annotated. In the latter case, annotations on secondary images are usually meaningful also as annotations on ground images, so the interrelations between both images should be notated in the annotation data.

Proscriptive Annotations AXFi allows for the designation of certain annotations as "proscriptive" when they do not formally delineate a geometric object, but indirectly express such an object's shape or how it may be derived. A canonical example is the use of color — perhaps color-enhanced secondary images — to designate segments or regions of interest. For instance, one common segmentation technique uses color simplification; smoothing out color patches can facilitate image partitioning by reinforcing the boundaries between different regions. With sufficient color manipulation, image-segments can potentially be described chromatically: a region may for instance be identified as "the area all of whose pixels display a red channel above" some threshold. AXFi data should therefore allow such indirect designations of segments (and spatial/geometric regions in general) to be encoded as annotations.

The concept of proscriptive annotations is not only chromatic — one can imagine other scenarios where morphological features, such as symmetries, may be employed to similar effect. In describing a chess board, for instance, a set of image-regions (each square on the board) can be derived via translational transforms of an initial two-segment kernel (one black square and one white). This representation is possible because of translational symmetries in the underlying image. Such geometric transforms and symmetries can sometimes be used to "generate" meaningful image segments, so they should be notated in AXFi data when appropriate.

Semantic Labels Some annotations supply data about other annotations (what AIM calls "Annotation on Annotation" as opposed to "Annotation on Image"). In general, we can supply a label, description, or semantic classification indicating what an image segment or region is "about", i.e., what it "depicts" (a bird, a flock of birds, a car, a traffic jam, and so forth). Such meta-annotation may be seen as a semantic postlude to an imaging process, but it may also be seen as providing further detail about the process itself. For instance, suppose segmentation isolates a bird from the sky: the epilogue to this operation is an ability (for a human user or a computer algorithm) to classify the segment as "bird." However, we can also say that the given fact of the segment capturing the optics of a bird (with its apparent anatomical outline and coloration) is what allowed the segmentation to be possible in the first place. Therefore, the label "bird" serves both to utilize the segmentation for further analytic/classificatory purposes and also, potentially, to characterize the inner workings or effectiveness of achieving the desired processing objective. It should be kept in mind, in short, that label annotations are not exclusively intended to be used for practical labeling in the contexts of tasks such as subject-marking images in corpora; labeling could also be used to notate how semantic properties of the image make segmentation (and other processing objectives) more or less feasible, or computationally intensive/error-prone.

The semantics of labels themselves is outside the scope of AXFi. AXFi makes no effort to proscribe what sorts of terms are useful as labels ("bird," "car," "ocean," "tumor," etc.), or whether labels are simple strings/words or have internal structure of their own. AXFi does, however, make the following minimal stipulations about labels. First, AXFi distinguishes "mass," "count," and "plural" label targets — e.g., *ocean*, *one bird*, *two birds*. These distinctions are directly relevant to how segments are bounded and counted; for instance, labeling a body of water as *a lake* (e.g. on a satellite image where the lake's full shore is visible) versus *ocean* (continuing to the

"colors" (i.e., **QColor** instances) which define the foreground and background of a two-toned image; they are not formally assigned to a visual color, like black or white.

horizon) implies different desiderata for how the labeled region spatially extends in relation to the surrounding image. Secondly, with respect to semantic labels, AXFi recommends that applications and libraries enable labels to be typed values — instances of application-specific data types — as well as simple character strings (like "bird.") This is consistent with AXFi's general approach to application integration and type systems, to be discussed further in Part II.

In addition to labels being second-order annotations — assertions attached to an image segment, for example, or any other image feature deemed semantically relevant in some context — labels may be applied to the image as a whole. In this context labels would be an example of *metadata* annotations (discussed next), which apply to the image itself rather than any proper part.

Image Metadata This category of annotation concerns any information "about" an image which is apart from the specific image content. Such details could encompass computational/encoding details such as color depth, pixel dimensions, and image file format; or acquisition details such as the make and settings of the camera whose photograph yields the current image. In theory, any information prerequisite to displaying an image may potentially be relevant to how annotations are used and interpreted. Therefore, AXFi should support representations of image metadata where such information is needed to fully specify annotation data; the metadata would then, typically, be considered an annotation on the entire image. It is outside the scope of AXFi to present image-related data which is applicable to the sharing or rendering of images but not, in a particular context, important to annotations themselves — unlike DICOM, for instance, which is designed as a standard for ensuring that images are correctly rendered and therefore includes detailed specifications of image formats and prerequisites, AXFi introduces image metadata only where such information can be considered intrinsic to the specific objectives and operations of image annotation.

Acquisition Context Among the class of annotations which apply to the image as a whole, AXFi recognizes a distinct kind of annotation covering what AIM calls the "context," or the purpose and concrete goal which compels users to initiate image-processing operations. In general, some of this contextual data should be preserved as a supplement to the overall information generated by image-processing operations. Often image processing itself is only one stage in a multi-faceted scientific workflow, and contextual information which exists prior to the image-annotation step may still be important for subsequent steps. For example, clinical data which motivates a radiographic study should remain linked to images and annotations which result from that study; this is why DICOM-SR includes clinical as well as imaging data.

This outline covers the different kinds of annotation data recognized by AXFi. The following section will fill in some missing details by addressing other sorts of information, apart from annotations themselves, which might be included in an AXFi data package.

Non-Annotation Data in AXFi

This section covers background information which may be addressed by AXFi-modeled objects to ensure that annotations are properly understood. The discussion will consider some details of non-annotation data and also clarify some details in how annotations are constructed.

Dimensions and Coordinate Systems Many different coordinate systems may come into play during an overall image-processing task. Even if attention is restricted only to 2D graphics — where each pixel is mathematically locatable with just two numbers — different algorithms or code libraries define 2D coordinates in different ways, so that representations of individual points need to be mapped across several different transformations. Coordinate systems differ in terms of whether locations are specified with integer or real-valued (via binary approximation) magnitudes, and whether designations of shapes and geometric structures are oriented to bounding-box points or centers; whether magnitudes increase left-to-right/top-to-bottom or vice-versa; and so forth. The application library with which an image is rendered may have a different system then the analytic

library with which it is processed. These differences become significant insofar as, for example, annotation data should be modeled so that applications can display visualizations of annotations and respond to GUI events so that users can examine or manipulate them.

During the course of image processing, certain operations moreover generate new sorts of coordinates; for example, segmentation may result in one part of an image being isolated in such a way that a separate coordinate system can be used for the interior of that region. The "Insight Toolkit" (ITK), for example, distinguishes "object space" (internal to individual geometric objects) from "world space."

Furthermore, additional coordinate systems may be relevant to the context or empirical situation wherein an image is acquired. Geospatial images, for example, can associate image-space locations with geographical coordinates. In some contexts, the coordinate frame applicable to some image can be contextualized with reference to parameters of the acquisition equipment; consider microscope configuration. The ground image may also be the result of intermediate processing subsequent to data acquisition; consider Flow Cytometry transformations. In these cases, the coordinates and dimensions considered "internal" to the image can be associated with other dimensions pertaining to the real-world context and/or image-acquisition protocol.

To make these connections explicit, AXFi employs a distinct category of *dimension* objects, which can ground multiple parameters related to dimensions/axes, their scales, units of measurement, kinds of magnitude, transformations, and so forth. One or more dimension objects can then be merged into an object representing a specific *coordinate system*.

Of course, an image's coordinate system is in part a product of the equipment used to acquire the image. Therefore, metadata about coordinates may include descriptions of the acquisition process and the instruments/settings used. In general, the domain of imaging overall can be characterized by stipulating that geometric meaning in the sense of spatial orientation/topological relations (left, right, above, intersecting, around, inside) and/or chromatic value (color as an indicator of light hue/intensity) are semantically and observationally meaningful in the image data — they reflect the process whereby empirical data becomes digitally manifest in the image. The actual image data may be a computational transformation of the empirical details (as in the wavelength alterations involved in visualizing infrared camera pictures).

Vertices A vertex is an image-position which can be expressed in one or more coordinate systems. Because image-processing environments typically utilize multiple coordinate systems, an intrinsic facet of vertex data is how a single position will map between different coordinate systems. For this reason, AXFi utilize a notion of vertex objects which are distinct from any specific coordinates; a concrete representation of one position within an image then derives from pairing a vertex object with a coordinate-system object.

When integrating with applications and/or code libraries, AXFi vertex objects will typically be connected to context-specific data types, and translations between coordinate systems may be provided via object methods of these context-specific types. Examples in this case would be **mapFcsToScreen** in FACSanadu (a Flow Cytometry application); **mapToGlobal/mapFromGlobal** in Qt (where "global" in this context refers to screen coordinates); and **convertWindowToPDFCoords** in XPDF (recall that AXFi considers PDF page views to be annotatable images).

Geometric or spatial primitives such as points, lines, poly-lines/polygons, and image-locations are defined in terms of one or more vertices. This means that the objects representing such primitives are built on top of vertex-objects, which in turn are built on top of dimension and coordinate-system objects. Any geometric object in AXFi should therefore be understood to have a coordinate-system context which specifies how the numeric values defining the primitive are to be interpreted, even if the surface-level representation of the object does not explicitly designate this context.

Circles, Ellipses, and Curves AXFi does not preclude representing linear or curved geometric shapes in different ways. AXFi *recommends*, however, that representations build off of the dimension/coordinate/vertex stack as organically as possible. This has several consequences for

representing curves of different varieties.

In the case of ellipses (with circles as a special case) a vertex-centric representation is based on elliptical foci (which reduce to circle-centers in the circle context; i.e., a circle is an ellipse whose foci occupy the same position). The full characterization of the curve requires one additional scaling parameter, such as minor-axis length (for ellipses, collapsing to diameter for a circle). AXFi object libraries should convert ellipse-representations from this foci-oriented approach to others which are common in scientific computing, such as covariance matrices and bounding rectangles.

More general curves can sometimes be constructed out of poly-lines, as is the case with B-SPLINEs. Although B-SPLINE "control points" are not part of the curve, they are geometric locations which determine the shape of the curve, so that representing B-SPLINEs does not require any descriptive capacity apart from that of poly-lines. Other kinds of curves, however, may call for more elaborate mathematical descriptions. Therefore, AXFi does not restrict the sorts of formulae which may be encoded in annotation descriptions.

This does not mean, however, that any and every mathematical expression should be representable via AXFi data specifically. For an analogous scenario, GATING-ML (which is used to demarcate gates or, in effect, "segments" within Flow Cytometry data/data-plots) "Originally [reused] MathML to universally describe any kind of mathematical transformation. While this would be a very flexible solution, it is generally impossible to evaluate expressions describable by MathML ... Transform[ing] gate vertices into the data space and describ[ing] (using mathematical formulas) how edges 'have been curved'" was avoided because the "description (formulas) would be significantly difficult to create."[6] That is, translating arbitrary formulae that may be concretized by application-level code into a serialization/markup format can be very difficult. Essentially the same problem will arise with any attempt to define arbitrary geometric overlays on a two-dimensional image-like space, as required for AXFi annotations.

The solution favored by AXFi is simply to allow procedural code to serve as a proxy for describing the relevant curve, coordinate transformation, or geometric object. An AXFi annotation context can provide a unique identifier for a procedure which evaluates any given mathematical equation, and then use this procedural reference as an indirect way of denoting the parameters of an annotation. This makes the proper use of the annotation dependent on users having access to the code in question, and integrating that code (or functionally equivalent code) into the libraries which process AXFi objects. However, as discussed below, application-level integration is considered an intrinsic part of deploying AXFi, so the reliance on application context to fully disambiguate AXFi semantics is not regarded as a limitation.

Application-Level Algorithms, Procedures, and Data Types AXFi explicitly assumes that any annotation data is created and consumed by applications which display and/or analyze images. In this sense there is no conclusive reason to exclude application-specific data from a collection of AXFi objects. Any application-level procedure or data structure could potentially be used to define annotation contents and/or targets, or to supply contextual information about annotations and image-processing objectives.

Of course, the usefulness of AXFi data will be limited if applications introduce idiosyncratic information which cannot be understood in other AXFi environments. For this reason, it is recommended that applications and code libraries which use AXFi construct mappings between AXFi data and application/library-specific types and procedures in a manner which prioritizes transparency and reusability. This goal shapes the architecture and terminology used by AXFi, as will be reviewed in Part II. In general, an application/library utilizing AXFi should provide an "AXFi Object Library", i.e., a collection of data types whose instances (objects) are serialized and deserialized via AXFi. As much as possible, this Object Library should be autonomous from the application/library in whose context it is developed.

Notwithstanding this criterion of autonomy, AXFi Object Libraries can include designations of

[6] See http://flowcyt.sourceforge.net/gating/latest.pdf, pp. 12 and 15-16.

application/library-specific data types and procedures. As much as possible, these computational artifacts should also be autonomous from the overall application context. It is recommended, that is, that types and procedures directly referenced by AXFi Object Libraries be factored into a separate "AXFi Host Library" which wraps the Object Library and bridges it with the main application. The Host Library should then be annotated rigorously enough that code-annotations in the Host Library provide as basis for image-annotations in AXFi data, insofar as such annotations rely on application-specific objects or procedures.

Annotation Body and Targets Although AXFi is specifically concerned with image annotation, it is desirable to model the image-annotation process as much as possible within the more generic environment of resource-annotation as a whole, such as the "OpenAnnotation" model and the Linguistic Annotation Framework (LAF).[7] To conform with these precedents, AXFi vertices — as well as additional measures which delineate geometric shapes together with vertices, such as circle diameters — are called "anchors." A geometrically-described subimage or geometric primitive (or also a secondary image used to demarcate primary-image regions, as when a two-toned bitmap is used as a pixel bitmask) is called the "target" of an annotation. The *body* of an annotation is designated via image-processing objectives terminology to clarify the analytic significance of a delineated region/primitive.

As a concrete example, suppose an annotation environment uses disks to notate image locations. A location-annotaton would then have, as *target*, the description of the desired disk as a geometric object (i.e., via a circle with center-position and radius). The *body* of the annotation would then be a label or descriptor indicating that the geometric object (the circle) is demarcated so as to define a location annotation (and not, say, an image-segment).

Note that this notion of annotation body only covers semantics intrinsic to the base-level annotation process. In AXFi, the "body" of an annotation applied directly to an image would not include higher-level semantic or contextual information; instead, those assertions are provided by further annotations attached to an image-specific annotation. For example, suppose a location-annotation is constructed so as to assert that a particular image-location represents the main entrance to a photographed building. The architectural concept "main entrance" (or "door," etc.) would be asserted via a semantic annotation whose *target* is the preliminary (purely "geometric") annotation which fixes the location. These paradigms imply that certain descriptions which in other contexts (such as linguistics) would be represented via a single annotation are instead, in AXFi, constructed via a network of two or more distinct annotations.

The body of an annotation may also contain metadata about the annotation, such as a clarification of whether it was generated by a human user or an algorithm (or some combination). A "manual" annotation is produced solely by a human user, whereas an annotation is called "human-generated" if it is defined by a user non-programmatically (i.e., through a GUI), but with the aid of some computational/algorithmic process.

Annotation Graphs In AXFi, as just alluded to, complete information about a specific annotation often spreads across multiple annotations. Associated with each annotation is an annotation *object*, i.e., a computational artifact conveying information and parameters specifying the annotation. Each annotation object is moreover a *node* within an overall annotation graph. Between any two annotation objects, one can via AXFi assert a relation or connection, establishing a graph-edge between the nodes. Moreover, AXFi uses a "hypergraph" data-modeling framework wherein nodes can be grouped together into different sorts of aggregates. Specifically, AXFi serializations can use the "Hypergraph Exchange Format" (HGXF) to be developed by Linguistic Technology Systems, or else some functionally similar hypergraph modeling paradigm. HGXF is outside the scope of this outline, but hypergraphs models are discussed further in the context of graph-serialization in

[7] See https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4485125/ on the connection between bioimaging annotation and the OpenAnnotation framework; and https://dash.harvard.edu/bitstream/handle/1/11879601/3889917.pdf?sequence=1, https://www.cs.vassar.edu/~ide/papers/LREC.pdf, or https://www.cs.vassar.edu/~ide/papers/ide-romary-clergerie.pdf on LAF.

For more information please contact:
Amy Neustein, Ph.D., Founder and CEO
Linguistic Technology Systems
amy.neustein@verizon.net • (917) 817-2184

Part II.

Similar in spirit to **DICOM-SR**, using a node-based and graph-oriented data model allows **AXFi** to flexibly connect image-annotations to whatever contextual, semantic, and "real-world" data is important for the larger image-processing objectives. For instance, if the annotation upon a **CT** scan circles or outlines (what is diagnosed to be) a tumor — visually calling attention to the malignancy and perhaps providing a measurement — the raw annotation has obvious conceptual/empirical links with a large body of contextual and follow-up information, such as the patient's clinical data, the prescribed Radiation Treatment Plan, and other kinds of data about the cancer (e.g., genomic profiles of the cancer cells) obtained by other sorts of biomedical investigations. It would be impossible to formalize all such representations of contextual data for all image-annotation environments: the kinds of supplemental data relevant in one biomedical context (such as oncology) will be different in other contexts (such as immunology or osteoporosis) — let alone imaging contexts beyond bioimaging.

These considerations suggest that **AXFi** should provides objects as affixation points for open-ended contextual data, where the inner structure of such supplemental data is outside **AXFi**'s scope. In the above oncology case, for instance, the tumor-annotation would be annotated with a labeling to the effect that the image Region of Interest evinces a diagnostically significant finding. That is, the annotation is situated within the processing objectives that influence the overall imaging process, namely, in this case, to diagnose a pathology or malady of some kind. The tumor-image — i.e. an image segment identified by a person or a computer as suggesting the presence of that tumor — represents the culmination of this image-processing activity, insofar as that specific segment (having been outlined or otherwise pointed to, e.g. via a circle or arrow) summarizes the entirety of the image-processing in relation to its objectives. An annotation isolating that segment, and then labeling it as diagnostically meaningful, then serves as a "pivot" connecting the image-processing activity to any other clinical or course-of-treatment information spaces. In short, the secondary annotation object (the node labeling the annotation itself as a diagnostic finding) can be connected to any other object, whose type and semantics is at the discretion of the relevant **AXFi** object library.

While allowing for these open-ended interrelationships between image-annotations and other data models, however, **AXFi** would be most useful in contexts where non-imaging data is managed according to paradigms consistent with **AXFi** itself. That is, the overall architecture of **AXFi** systems — in terms of **AXFi** Object Libraries, Reference Implementations, and "grounded" serialization, all to be discussed in Part II — can be applied to other contexts outside of imaging proper. Code libraries which employ these paradigms will be able to integrate with **AXFi** more easily.