WHITE PAPER

ConceptsDB and the Scientific Research Data-Set Model: *Supplemental Archives/Structured Research Reporting/Plugins*

Overview

ConceptsDB is a hypergraph database engine being developed by Linguistic Technology Systems (LTS). **ConceptsDB** features a multi-paradigm data representation strategy which is informed by contemporary research into **AI** and information semantics, allowing theoretical frameworks such as Conceptual Spaces, Conceptual Role Semantics, and Petri Nets to be applied toward the computational modeling of information spaces. In addition, **ConceptsDB** is designed to prioritize application development, particularly naive, desktop-style applications: data about application/session/user state can be directly stored in the database, so that **ConceptsDB** provides a convenient scaffolding for implementing desktop software. Hypergraph-based data modeling ensures that information needed by the application can be conveniently marshaled between runtime formats and whatever persistent/serial/**GUI** representations which are necessary for application-level capabilites.

Preliminary to the commercial release of **ConceptsDB**, LTS is deploying a version of **ConceptsDB** as part of the "**ConceptsDB** Application Framework" (**CsAF**) which is oriented to scientific computing and software. **CsAF** database instances can be employed at several sites within a scientific-computing and/or data-sharing platform, included as an embedded component of published data sets, as a cloud service managing data and/or publication repositories, and as a tool for individual or institutional users to track data sets and publications. Our "**MOSAIC**" code libraries (which stands for "Multi-Paradigm Ontologies for Scientific and Technical Publications") encapsulate functionality applicable to using **ConceptsDB** in publishing contexts. A **MOSAIC** "Portal" is a data/publication repository where users can query and download resources following a protocol associated with **ConceptsDB**. The "**MOSAIC** Dataset Explorer" ($\mathbb{M}$**dsX**) libraries are designed for applications which interoperate with data/publication archives (including but not limited to **MOSAIC** Portals). Individual data sets may also embed a version of **ConceptsDB** — specifically, "DigammaDB" (ς**DB**), which is a lightweight, self-contained **ConceptsDB**-compatible engine suitable to be distributed as source code within a data set/code repository. LTS's "Dataset Creator" ($_{ds}C$) is a technology that can help researchers deploy data sets with customized desktop software, potentially using ς**DB**. In short, $\mathbb{M}$**dsX**, ς**DB**, and $_{ds}C$ form a trio of libraries helpful for the implementation of published data sets with special-purpose applications that are customized for viewing and examining information specific to the research methods and paradigms from which a data set originates; and for the implementation of tools to acquire and keep track of data sets and their associated publications.

Collectively, then $\mathbb{M}$**dsX**, ς**DB**, and $_{ds}C$ form the basis of a **CsAF** implementation — that is, an Application Framework centered on **ConceptsDB** — that prioritizes deploying and accessing scientific data sets. The information and assets included within a data set, and modeled via **CsAF** components, can span several different requirements, including:

1. Data sets, which may include raw data files and/or code for accessing this data as well as demonstrations/implementations of algorithms for analyzing or otherwise processing the data (or data having a similar format/structure or scientific background)

2. Full-text publications, potentially including both machine-readable and human-readable (e.g. **PDF**) formats, annotates so as to link parts of the text document with corresponding parts/elements within its associated data set(s);

3. Systematic descriptions of research methods, protocols, and (if applicable) equipment, such as may be formally expressed by standards like **MIBBI** (Minimum Information for Biological and Biomedical Investigations), **BIOCODER**, or **PANDORE** (a bioimaging library that contains an "Image Processing Objectives" Ontology);

4. Applications for viewing data sets, which may be pre-existing software or custom-built for an individual data set.

The $\mathbb{M}$**dsX** libraries include code for interoperating with data structures documenting each of these aspects of scientific resources, which may involve querying data sets themselves or querying of scientific repositories where data sets and publications are hosted. In order to work with a variety of different corpora, each with their own structures and protocol, $\mathbb{M}$**dsX** is built around a "Scientific Data Repository Model" (**SDRM**) which is concretized separataly for each corpus or repository connected with a given $\mathbb{M}$**dsX** application. **SDRM** is divided into distinct concretizations, or "modules," organized around the **API**s and data profiles of individual scientific portals/repositories. Therefore, LTS seeks to collaborate with organizations who maintain scientific portals so that open-access $\mathbb{M}$**dsX** modules may be made available to the general public targeted those specific scientific resources.

The following sections with discuss $\mathbb{M}$**dsX**, $_\text{ds}$**C**, and ς**DB** in greater detail as well as provide more information about $\mathbb{M}$**dsX** modules.

Part I: MOSAIC Components

In recent years — as publishing has become more "digitized" — many online platforms have emerged for scientists to share and publish their research work, including both raw data and scientific papers. These portals generally provide an index/search feature where readers can locate research based on the name of the author, title of the publication, keywords, or subject matter, along with a document viewer where readers can view the abstract of each publication (and sometimes full text of the article) and other publication details (such as co-authors, date of publication, bibliographic references, etc.). However, these portals offer only limited features for interactively exploring publications, particularly when it comes to examining data sets and/or browsing multimedia assets associated with publications, such as audio, video, interactive diagrams, or **3D** graphics.

The MOSAIC system is envisioned, over all, as a suite of code libraries developed by LTS allowing institutions to host publication repositories — and for individual users to access publication repositories — free of the limitations of existing scientific document portals. In the immediate future, LTS is focused on the more modest goal of implementing software to access existing portals (via $\mathbb{M}$**dsX**) and to create individual data sets (via $_\text{ds}$**C**). With MOSAIC, each publication can be linked to a supplemental archive that contains information about the author's research methods, data sets, and focal topics. If desired, these archives can include machine-readable representations of full publication text, to support advanced text-mining techniques across the repository. The supplemental archive can be explisitly linked to publications within their host repository, or it may be maintained in a decentralized manner external to the hosting platform.

Using MOSAIC, developers can implement a hosting platform and/or a client platform for this supplemental material, in addition to the publications themselves, where the platform provides software enabling readers to browse and access supplemental archives and their data sets and/or methodological descriptions.[1] MOSAIC can be customized for different subject areas by incorporating domain-specific ontologies into document-search features as well as machine-readable document-text annotations.

MOSAIC is designed so that the software to access publications and publications' supplemental archives can be embedded in scientific-computing applications via MOSAIC *plugins*. This ensures that publications and data sets can be examined interactively with the same software that scien-

---

[1] A **MOSAIC** publication repository or "portal" is a structured collection of data and documents which can be hosted via web servers (including fully encapsulated and containerizable cloud services) using technology related to **ConceptsDB**. **MOSAIC** repositories can serve as general-purpose portals, hosting academic papers covering a broad range of topics. Alternatively, **MOSAIC** repositories can serve as targeted portals hosting papers focused on more narrowly focused subject domains. Organizations can utilize **MOSAIC** internally as the basis of a private document management system (**DMS**), or to provide (public) open access to complete publications, including human-readable and machine-readable full text and all supplemental content, with no restrictions. Excepting sensitive/private information which might be provided via a dedicated component within the portal with specific features for authors, editors, and administrators. Any project which *does* restrict access to some part of any document requires a commercial **MOSAIC** license.

tists employ to conduct research. MOSAIC also introduces a *structured reporting system* (MOSAIC-SR) for describing research/experimental/lab methods/protocols. Supplemental archives, plugins, and the structured reporting system are outlined below.

Supplemental Archives

MOSAIC supplemental archives are additional resources paired with MOSAIC publications. In general, supplemental archives may include raw data, descriptions of research methods, or annotated data linked to publication texts. Each supplemental archive could have any of the following resources:

1. Interactive versions of the publication, with annotations indicating important concepts and phrases, perhaps aggregated into a "glossary" defining technical terms;

2. Machine-readable representations of document texts, with special-purpose character encodings designed to facilitate Text and Data Mining (TDM);

3. Structured files containing raw data discussed in the publication, along with interactive software allowing scientists to access and reuse the data;

4. Detailed reports of the author's research methods and experimental setup and/or protocols, conforming to the relevant standards with respect to the publication's subject classification — for instance, the "Minimum Information for Biological and Biomedical Investigations" (MIBBI) includes about 40 specific standards for different branches of biology and medicine;

5. Representations of analytic methods and algorithms underlying the research findings, which are provided directly via computer code or indirectly via formal descriptions of computational workflows;

6. Self-contained computer software which demonstrates code that the author developed and/or used for analyzing/curating research data; and

7. Multi-media assets such as audio or video files, annotated images, 3D graphics, interactive 2D/3D plots and diagrams, or other kinds of non-textual content which needs to be viewed with special multi-media software.

The contents of a supplemental archive will be different for different publications, depending on whether the archive contains specific raw data or just a summary of the methods used to obtain the raw data. In the former case, a typical archive will include a *data-set application*, or a semi-autonomous software component allowing researchers to study and visualize the data set, typically provided in turn via data files that are also located in the archive. The data-set application will, in general, provide both a visual interface for raw data and code libraries for computationally manipulating this data; typically raw data files will encode serialized data structures that can be deserialized by code (e.g., C++ classes) included in the data-set application. MOSAIC includes generic implementations of a data-set explorer ("MdsX"), which can be adopted to the specific kind of information that needs to be serialized for a given publication. As a result, data-set application can then be packaged as a plugin to existing scientific software within the relevant scientific discipline.

MOSAIC Plugins

By design, MOSAIC Portals will provide a suite of plugins for existing scientific software, allowing publications and supplemental archives hosted on the portal to be read and examined within computer software associated with each publication's subject matter. For example, articles about chemistry could be read within IQMOL, a molecular visualization program; articles about cellular biology and bioimaging could be read within CAPTK (the Cancer Imaging and Phenomics Toolkit), an image-processing application; articles discussing novel computer code could be read within QT Creator, an Integrated Development Environment (IDE) for programming; etc. The advantage of accessing a publication and a supplemental archive within actual scientific software is that it allows research work to be understood, evaluated, and reused within the computing environments which scientists typically use to conduct professional research. This is different than existing science publication portals, which generally rely on web browsers to access supplemental materials like data sets and multimedia files — in this standard setup the software ecosystem wherein readers examine

published research is fundamentally separate from the software platforms where research is actually conducted. This example demonstrates how existing portals are *limited* in their ability to share research in rigorously reusable and replicable ways — and how MOSAIC offers an improvement.

Embedding presentations of their research within existing scientific software has the added benefit for authors of making their work more practically and immediately useful for the scientific community. Such presentations establish the computational framework for pragmatically deploying their techniques in real-world scientific contexts, accelerating the pace at which research work can be translated to concrete scientific (and clinical/lab/experimental) practice. As one example, research involving novel image analysis techniques could be packaged so as to target a MOSAIC plugin for bioimaging software such as CAPTK, so that readers could actually run the author's code as a CAPTK module.[2] This is important because such functional assessment and adoption of novel contributions is harder to carry out if a body of research work is described indirectly within article text, as opposed to being concretely implemented within a specific scientific application.

Another benefit of using plugins to access supplemental archives is that the host application will usually provide more sophisticated multimedia and data visualization capabilities, compared to static PDF images or even interactive web portals. Publishers have begun to develop online platforms for browsing research papers in conjunction with multimedia content such as interactive diagrams and 3D graphics — a physical model of a protein or a chemical compound, for instance, can be viewed online via WEBGL; such graphics could even be embedded as an "**iframe**" within an HTML version of the publications. Publishers consider this to be cutting-edge technology. However, the same molecular 3D model, when viewed in IQMOL, can be enhanced with many additional visual features, representing bonds, orbitals, torsion angles, etc. The multimedia experience of exploring chemistry data in custom software like IQMOL is therefore much greater than the experience of generic web multimedia, which means that the scientific software is a better forum for showcasing novel research.

MOSAIC Structured Reporting (MOSAIC-SR)

The MOSAIC structured reporting framework (MOSAIC-SR) includes tools to help authors develop interactive presentations supplementing academic documents, and specifically to use supplemental archives to document how their research has been conducted. With MOSAIC-SR, authors can implement or reuse code libraries that report on research/experiment methods, workflows, and protocols. The MOSAIC-SR information may be structured as a "minimum information checklist" conformant to standards such as those collectively gathered into the MIBBI recommendations; in this case MOSAIC-SR would be applied by implementing object models instantiating MIBBI policies. Alternatively, MOSAIC-SR reports can be derived from actual computer programs simulating research workflows, similar to BIOCODER[3] (which is included by LTS, in an updated C++ version, as one MOSAIC library). Finally, MOSAIC-SR presentations may be based on annotations applied to research/analytic code. For example, in the context of image analysis, the PANDORE project (an image-processing application) provides an "Image Processing Objectives" Ontology as well as a suite of image-processing operations that can be called from computer code. Image-analysis pipelines can therefore be explained by annotating the pertinent function-calls (which PANDORE calls "operators") with terms from the PANDORE controlled vocabulary, providing annotation targets for MOSAIC-SR presentations.

MOSAIC-SR can express both computational workflows that are fully encapsulated by published code and real-world protocols concerning laboratory equipment and physical materials or samples under investigation. In the latter guise, MOSAIC-SR code can employ or instantiate standardized terminologies and data structures for describing experiments — such as MIBBI policies or BIOCODER functions. In this case, the role of MOSAIC-SR code is to serve as a serialization/deserialization endpoint for sharing research metadata. Conversely, when workflows are fully implemented within software developed as part of a body of research, MOSAIC-SR can provide a functional interface allowing this code to

---

[2]This toolkit provides a good case-study for research publication because it has an innovative QT and Common Workflow Language based extension mechanism; cf. https://cbica.github.io/CaPTk/tr_integration.html.
[3]See https://jbioleng.biomedcentral.com/articles/10.1186/1754-1611-4-13.

be embedded in scientific software. For the latter, MOSAIC-SR provides a framework for modeling how a software component specific to a given research project exposes its functionality to host and/or networked peer applications. There are also instances where both scenarios are relevant — the MOSAIC-SR code would simultaneously document real-world experimental protocols and construct a digital interface as part of a workflow which is part digital ("*in silico*") and part "real-world" ("in the lab").

MOSAIC Annotations

Included in any MOSAIC plugin would be specially-designed PDF viewers for interactively reading authors' papers. In particular, these PDF applications would recognize cross-references linking between publications and their associated supplemental archives. This would allow authors to identify concepts which are discussed and/or represented both in the research paper and in the archive, annotating their papers accordingly. For example, the concept "RNA Extraction" may be discussed in a publication text, and also formally declared as one step in the lab processing as represented via BIOCODER, summarized in a BIOCODER-generated chart, and included in the supplemental archive (a similar example is used in the documentation for BIOCODER). The PDF viewer would then ensure that the phrase "RNA Extraction" in the text is interactively linked to the concordant step in the experimental process, so that readers would then be able to view the BIOCODER summary as a context-menu action associated with the phrase where it appears in the PDF file. For a different example, the phrase "Oxygenated Airflow" refers to airflow in assisted-breathing devices, such as ventilators; to ensure that the device is working properly, the equipment must be monitored to check that a steady stream of oxygen reaches the patient. Research into the design and manufacturing of ventilators and similar devices may then include "Oxygenated Airflow" both as a phrase within the article text and as a parameter in data sets evaluating the device's performance. In this situation, the publication-text location of the "Oxygenated Airflow" phrase should once again be annotated with links to the relevant part of the data set (e.g., a table column) where measurements of airflow levels are recorded.

In order to establish granular links between publication texts and data sets, MOSAIC introduces a novel "Hypergraph Text Encoding" (HTXN) system. This includes LaTeX code generators which output both LaTeX and HTXN representations of manuscripts, where the LaTeX files are provisioned with processing steps during PDF creation that generates geometric mappings between PDF viewport coordinates and HTXN annotations. Such information is then used by MOSAIC PDF viewers to cross-references publications and data sets according to a protocol which allows users to convenient switch back and forth between reading documents and visualizing data.

In addition to this technology for linking annotations with PDF and data-set annotations, LTS is working on frameworks for constructing annotations themselves with a rigorous semantics grounded on general-purpose models such as OpenAnnotation and the Linguistic Annotation Framework. We are, in particular, directing attention to bioimaging and other image-related use-cases via an "Annotation Exchange Format for Images" (AXFi), and to linguistic use-cases via an "Annotation Exchange Format for Text" (AXFt). The AXFt system supports graph-style annotations marking inter-textual connections on different linguistic scales. At the sentence level and smaller, these annotations are oriented to discourse-grounding models based on Cognitive Grammar; at the sentence level and larger, the primary model is Sequence Package Analysis, a discourse-representation methodology pioneered by LTS's founder Amy Neustein. [Recommed here giving links to refs on HTXN and SPA]

Part II: Interoperating with Data Sets and Scientific Repositories

Systematic use of published scientific data requires accessing information on several different levels, including outlines of research methods and protocols as well as raw research data. Several standard

models exist for how research data sets should be described, including Research Objects[4], the **FAIR** (Findable, Accessible, Interoperable, Reusable) initiative[5], **SCIXML**,[6] and **SCIDATA**[7]. Also relevant to scientific data sharing, though less general in scope, are formats for composing and/or annotating scientific publications, such as **IEXML**[8] and **JATS** (Journal Article Tag Suite). Despite (or perhaps beause of) this diversity of representational possibilities, guidelines for constructing and annotating scentific data remain open-ended: researchers who want to rigorously annotate/document their published data can do so in many different ways, which presents challenges to software that interacts with research data.

Several different kinds of applications interface with scientific data, including software where such data is *created*; web services (and potentially **API**s) where data is *hosted*; and programs which researchers use to find and download (or otherwise access) scientific data. Because data-representation paradigms vary, these applications need to be flexible enough to manipulate data and metadata packages structured in divergent ways. For example, each $\mathbb{M}$**dsX** module is designed around a particular scientific repository/portal's protocols, so the structure of distinct $\mathbb{M}$**dsX** modules may be noticeably different. Nevertheless, some general structuring principles can be developed with respect to querying and consuming repositories' resources. This motivates the idea of a general "Scientific Data Repository Model," (**SDRM**) which is concretized within code implementing clients for individual portals (e.g., $\mathbb{M}$**dsX** modules). The following section will present an overview of this **SDRM** representation, and subsequent sections will specify how **SDRM** is related to **ConceptsDB** and other technologies discussed here.

Interoperating with Science Portals via the Scientific Data Repository Model

According to the **SDRM** representation, each data set is accompanied/constituted by a collection of "assets." These include raw data files; publication texts; research method descriptions; code for accessing/examining/analyzing research data; and annotations defining interconnections between these different elements. All material within or associated with a data set is classified into one of **SDRM** "units," including the five just enumerated and several others.[9] **SDRM** units represent both serializations of data-set information, as well as **GUI** components (such as Dialog Boxes) allowing readers of publicatons (and users of corresponding data sets) to view the unit's information. Moreover, **SDRM** units, by design, encapsulate their data in type-instances (e.g., objects of **C++** classes) that can be manipulated at runtime by computer code. In short, each unit has at least three representations (serial, **GUI**, runtime), giving rise to a spectrum of **SDRM** data that can be summarized in a table (see Table 1).

In general, the data within **SDRM** units might be obtained from different sources, depending on the module — e.g., from publishers' **API**s, or from downloaded data sets, or some combination. It is better to make at least part of the **SDRM** data available without having to download data sets as a whole, because a useful overview of the data set may factor in whether users choose to download it to begin with.

When **SDRM** information is provided as part of a dataset (rather than indirectly through an **API** or some other remotely-accessed resource), the **SDRM** **GUI** components can be integrated with the data set's overall front-end. In particular, if the data set provides its own, customized **GUI** code — yielding

---

[4] See https://eprints.soton.ac.uk/271587/1/research-objects-final.pdf

[5] See https://www.researchgate.net/publication/331775411_FAIRness_in_Biomedical_Data_Discovery.

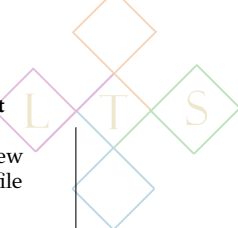[6] See http://able.myspecies.info/scixml

[7] See https://stuchalk.github.io/scidata/.

[8] See https://www.semanticscholar.org/paper/IeXML%3A-towards-an-annotation-framework-for-semantic-Rebholz-Schuhmann-Kirsch/1d72a56b6576117c62f388a5f2193965e4c7e293.

[9] For instance, data in some fields — such as genomics — sometimes becomes too large to be conveniently downloaded to an ordinary computer; in this case, portals such as GenBank offer query services to restrict vary large data sets into small result-sets that researchers can examine directly. In these cases, the steps required to obtain information via large, remotely-hosted data steps would potentially be encapsulated into a distinct **SDRM** unit classified outside the main **SDRM** categories. Another potential example of **SDRM** would be statements related to funding, potential conflicts of interest, and so forth.

SDRM Unit	Runtime Object	Serialization	**GUI** Component
Raw Data Files	File and archive objects identifying file types, preferred applications, software prerequisites, etc.	Structured review of archive layout (e.g., the Research Object Bundle **manifest** file)	Filesystem archive view plus information on file types and sizes
Publication	Machine-Readable Text (e.g., **HTXN**)	Machine-Readable Serializaton (e.g., **HTXN**-compatible LaTeX)	**PDF** Viewer
Research Method	Research Protocol Encoding (e.g., **BioCoder**)	Research Protocol Serialization (e.g., an **MIBBI**-based markup language)	Dialog Box to view "Minimum Information" checklist
Dataset-Specific Code Library	Descriptions of library dependencies, compile steps, inter-type relations (e.g. corresponences between data object and **GUI** types), etc.	Serialized build information	Dialog to set compile/user options; **IDE** integration
Annotations on Data, Code, and Text	Integrated Annotation Object Model	Annotation Encoding (e.g. LTS's "Annotation Exchange Format")	Annotation-related context menus within **PDF** viewers and dataset **GUI** elements

Table 1: Representations for **SDRM** Units

what LTS calls a "Data-Set Application" — then such **GUI**s may include dialog boxes and similar components representing **SDRM** data. Data-Set Applications (which may be implemented with the aid of $\mathbb{M}$**dsX** libraries) are discussed in the next section.

Data-Set Applications

Dataset Creator ($_{ds}$C) is a framework for constructing data sets which include computer code based on the **QT** application-development platform. Dataset Creator takes advantage of the **QT** platform to construct Research Objects with exceptional **GUI** and data-mining capabilities. **QT**, the leading native cross-platform development toolkit, is a comprehensive framework encompassing a thorough inventory of programming features — networking, **GUI** implementation, file management, data visualization, **3D** graphics, and so forth. Data sets based on **QT** require users to obtain a copy of the **QT** platform, but **QT** is free for non-commercial use and easy to install — importantly, **QT** is wholly contained in its own folder and does not affect any other files on the user's computers (in this manner **QT** is different than most software packages, which usually demand a "system install").

By leveraging the **QT** platform, $_{ds}$C enables standalone, self-contained, and full-featured native/desktop applications to be uniquely implemented for each data set, distributed in source-code fashion along with raw research data. Adopting such a data-curation method makes data sets easier to use across a wide range of scientific disciplines, because the data sets are freed from having to rely on domain-specific software (software which may be commonly used in one scientific field but is unfamiliar outside that field). In addition, Research Objects composed with $_{ds}$C can be integrated into Multi-Application Networks (which are described above) because the dataset applications are autonomous native **GUI** applications that can easily interoperate via **QT** messaging protocols.

Because every data set is unique, each Dataset Application will necessarily include some code specific to that one Research Object. However, $_{ds}$C will provide a core code base and file layout which is shared by all $_{ds}$C data sets by default. This common core is structured in part by the goal of developing Dataset Applications in a **QT** context; for instance, $_{ds}$C projects are structured to use **QT**'s "qmake" build system as the primary tool for compiling data-set code. The common $_{ds}$C code therefore includes qmake project files which support compiling application with several build configurations. In this framework, data-set users are classified into several different roles — in addition to ordinary users (specifically, researchers who want to work with and draw information from data sets but have no development connection to these data sets themselves), $_{ds}$C recognizes roles for authors, editors, testers, and other users who are responsible for bringing data sets into publication-ready form to begin with. Depending on the administrative role, data set code can be compiled with additional features (e.g., unit testing features).

Another core component of dsC is LᴬTᴇX code that authors may use when preparing documents accompanying their data set. These LᴬTᴇX files encompass special functionality for defining code annotations and semantically significant points in article text, such as sentence and paragraph boundaries. This LᴬTᴇX code can be used in conjunction with a pre-processor that generates LᴬTᴇX files from a special input language. The goal of these text-processing technologies is to improve the interoperability between research papers and data sets. In particular, the LᴬTᴇX pre-processors (and subsequent LᴬTᴇX-to-**PDF** converters) generate **HGXF** files which store information about textual and **PDF** viewport coordinates specifying the location of semantically meaningful elements such as sentences and annotations. These files are then zipped and embedded in **PDF** files. With dsC, the resulting **PDF** files can then be loaded into a customized **PDF** viewer capable of reading the embedded **HGXF** data. This allows the **PDF** application to utilize the embedded information so as to provide a more interactive reading experience — for instance, viewing annotations or copying sentences via context menus, where viewport data maps cursor position to textual elements visible on the **PDF** page. These features provide an application-level interface between the **PDF** viewers (considered as **GUI** components) and the corresponding **GUI** components in Dataset Applications.

With proper customization, both the **PDF** viewers and the dsC Dataset Applications can interoperate, with **PDF** context menus calling up windows in the Dataset Application's **GUI** implementation, and vice-versa. For example, researchers reading the **PDF** version of a scientific article can launch the Dataset Application to explore some detail mentioned in the text. This is an example of where microcitations are practically useful: any microcitable element in a document (such as a table, column, row/record, or analytic procedure formalized as a procedural asset associated with a data set) can be linked to a corresponding **GUI** element in the Dataset Application. For example, a statistical parameter — mentioned by name in the text, and perhaps represented in serialization within raw data — can be mapped to a **GUI** table column, and specifically the column header; this is then an annotation target, in the sense that for readers to gain more information it is proper to link mentions of the relevant scientific concept in article text to the column header as a graphical element that can be made visible. The link is operationalized by implementing a procedure to show the **GUI** window where the table is located, and ensure that the column header lies in the visible portion of the screen, as a response to readers on the **PDF** side signaling a desire for information on the annotated text element. In the opposite direction, database elements can be annotated with links that can used by the Dataset Application to launch a **PDF** window opened to the page and location where the corresponding concept is discussed in the article.

In order to properly model this semantic, viewport, and data set data integration, dsC uses **HTXN**, described earlier, for document representation. With dsC, **HTXN** files are not only associated with data set assets; they are also machine-readable document encodings that can be introduced into publication repositories and other corpora oriented toward machine-readability. Authors can host **HTXN** files within data sets and link to them via services such as CrossRef, thereby ensuring that a highly structured, machine-readable version of their papers is available for text and data mining. The **HTXN** protocol is also useful for encoding natural-language content which becomes part of a data set as data assets in themselves; for example, linguistic case-studies, or (in the biomedical context) patient narratives.

In order to establish annotation cross-references between publications and data-set code, dsC provides a code-annotation system based on hypergraphs. Further documentation of hypergraph structuring principles and how they apply to representing computer code can be found in the LTS paper reviewing **ConceptsDB**.

Hypergraph Data Serialization and "Grounding"/"Meta-Serialization"

As a database engine, **ConceptsDB** is focused on data persistence — that is, storing information so that it may be shared among different instances of one or several applications; and preserving information between application-sessions, where each "session" refers to the time during which one user launches, uses, and ultimately exits an application. In addition to data persistence, however,

ConceptsDB also encapsulates theoretical research into optimal strategies for data *modeling*, which includes representations during application runtimes and serializations for sharing data between heterogeneous end-points. **ConceptsDB** introduces the idea of *infoset classes*, which are intermediate structurations of data type-instances, organized so that interconnected clusters of infoset class objects can be modeled as hypergraphs. Infoset classes are guidelines for marshaling runtime values into structures that can be persisted in **ConceptsDB**. At the same time, infoset classes can also be used as preliminary encodings for data sharing and serialization.

CsAF (the **ConceptsDB** Application Framework) is designed around the premise that the primary goal of any data modeling system (and, as a surface-level encoding of data to be shared, any markup language) is to encapsulate data which is used by and present in one application, to be subsequently re-used by a different application — or the same application at a future time. There is an application which *creates* a data structure, and a (not necessarily different) application which *consumes* that structure. "Consuming" in this context means reading the data so that it may be incorporated into some computational process and/or rendering the data so that it may be viewed by a human user. Of course, the consumer-application may also be a creator, adding new data or revising that which it has consumed.

Data sharing is "correct" insofar as the consumer-application reads, understands, displays, and allows human users (as well as, potentially, computer algorithms) to manipulate the data in ways which are functionally equivalent to the creator-application. When this functional equivalence is achieved, the two applications can be said to be *aligned*. Creator-to-consumer alignment can happen in one of two ways. One option is that the data model shared between the two applications is formalized so that different applications are naturally interoperable because they are implemented according to similar guidelines. This may be called "coincidental alignment." There are, in turn, two pathways toward coincidental alignment. On the one hand, data-model formalization may stipulate criteria on creator and consumer applications to ensure that any creator is aligned with any consumer. Such a data model would essentially provide a "checklist" of requirements to be satisfied, procedures to be implemented, and so forth, so that applications are only considered to be creators and/or consumers by fulfilling the checklist contract.

On the other hand, the formalization may focus on the medium through which data shared between creator and consumer is structured and encoded. This second mode of standardization — coincidental alignment based on data-serialization formats — is arguably the most common strategy for enforcing alignment (even if it is not identified in these terms). For example, **XML** schemata represent constraints on any creator of information encoded by conformant **XML** documents; given these constraints, the **XML** *consumers* can guarantee alignment with creators by incorporating the schema as a contract the creator obeys. The consumers need not have any knowledge of the creators "apart from" the fact the **XML** they create conforms to the schema.

Whether by explicit application-level requirements or via contracts on the medium by which shared data is encoded (e.g., **XML** schemata), coincidental alignment can occur without any explicit coordination between the code used by creator and consumer applications, respectively. As an alternative model, however, we can identify *deliberate* alignment, wherein the consumer aligns with a creator by explicitly referring to and mimicking the creator's data models and procedures. To facilitate deliberate alignment in this sense, creator applications can architect the relevant code to prioritize transparency and reuse — e.g., factoring this code into a semi-autonomous library which the consumer application can use directly, or use as the basis for its own library.

In contrast to many data models, **CsAF** openly embraces deliberate rather than coincidental alignment in this sense. Data-sharing protocols are not only necessary application features; they are also proxies for a scientific/technical formalization of the principles underlying a given data profile. Standardizing only a surface-level realization (e.g., **XML** schemata) fails to rigorously capture this scientific dimensions. One way to document the shape of any relevant data is simply to explain how an **XML** document will be structured insofar as it encodes data accordingly[10]; however, the structure of an **XML** document does not, in and of itself, present a clear picture of how the information which

the document represents is semantically organized. Even though **XML** is processed by computer programs, one cannot even determine from an **XML** document or schema which **XML** elements (if any) correspond to data types recognized by applications which read and/or write the corresponding **XML** code.[11] In short, because applications can read or use data from an **XML** document in different ways, the document's structure does not itself provide a clear picture of how the specific code that reads the **XML** is organized. Such uncertainty becomes significant when one wishes to use markup specifications as an indirect strategy for documenting parameters and features of the data structures which are serialized via the relevant markup language. In effect, markup/text-based serialization operates on two levels: on the one hand, the specific document provides an encoding of data conforming to a given structure; but, at a more abstract level, one can model the relationship between the surface-level document structure and the application-level data structures thereby serialized. This second level of detail is usually implicit and unstated, but in **CsAF** technology, such "meta-serialization" — a term we are using to suggest the idea of providing meta-data *about* a serialization — is directly formulated through a notion of "grounding," which involves adding supplemental markup clarifying how documents instantiating a serialization format relate to the coding protocols and data types of software that reads or creates these documents.

For maximum expressivity, **CsAF** introduces a meta-serialization system that can be applied to languages more flexible than **XML**, notably **TAGML**.[12] In effect, **CsAF** provides parsers for an extended version of **TAGML** which includes an additional "grounding" layer. Grounding, in this context, means describing how elements in the markup — nodes, attributes, and character sequences — are "grounded" in application-level types, data fields, and other programming constructs. **CsAF** provides parsers for this "Grounded **TAGML**" (**GTAGML**) language as well as converters to output serialized data in more conventional formats (e.g., **XML**). **CsAF** also provides code libraries for extracting data from **GTAGML** documents.

By design, **GTAGML** schemata operate on two levels: first, they constrain the organization of **GTAGML** files themselves; and second, they stipulate how **GTAGML** document structure relates to the type systems of code libraries serializing and deserializing **GTAGML** files. To model the second level of metadata, **GTAGML** introduces a hypergraph-based type model organized around "infosets," which are structured overviews of application-level data types. To fully utilize **GTAGML** features, programmers may consequently compose infoset classes (described earlier), as wrappers around ordinary data types (e.g., **C++** classes), whose instances are serialized via nodes or character strings within **GTAGML** documents. Infoset classes provide hypergraph "views" onto type-instances, and act as a bridge between data types and their associated **GTAGML** schema.

Because **GTAGML** is not only a hypergraph-serialization format, but also a representational paradigm for "meta-serializaton" data, **GTAGML** is closely interconnected with ς**DB** data models. ς**DB** recognizes numerous forms of hypergraph structuring elements and levels of organization; each of these can be annotated via **GTAGML** to document the relation between **GTAGML** node patterns and **ConceptsDB** hypergraph formations (or any construction which can be mapped to ς**DB**). More information on the structural principles underlying ς**DB** hypergraph categories can be found in the ς**DB** documentation.

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

---

[10] Potentially, such specification can be a single **XML sDTD** file, or an **XML** sample, providing a convenient reference point for developers to grasp the underlying data model.

[11] For example, the **XML** portion of **OME-TIFF** (the principal Open Microscopy Environment imaging format) includes an explicit **Image** element (which gathers up all significant image metadata); an application reading **OME-TIFF** files might, therefore, introduce a single datatype — analogous to (and maybe wrapping an) **itk::Image** from the Insight Toolkit imaging libraries — bundling the data in that part of the **OME XML**. In this case, there will be a one-to-one correspondence between **XML** structure and application-level data types, at least for that one **Image** node. On the other hand, software reading **OME-TIFF** information might not manipulate images directly, but rather pull out other kinds of metadata, such as an experimenter's name or description of the microscopic setup. In this case, the application might not have an explicit "object" representing the image itself, but it may still read information about the image from child nodes of the **XML Image** element.

[12] See http://www.balisage.net/Proceedings/vol21/print/HaentjensDekker01/BalisageVol21-HaentjensDekker01.html.