Latex/XML/Concurrent/RDF Document Object Model and Plugin Framework (LXCR)

This paper will summarize the **LXCR** document object model and plugin framework and discuss its applications for testing, education, and the development of test and test-preparation materials. **LXCR** combines the requirements of a generalized, multi-format Document Object Model with specifications for embedding document-viewing as well as document-preparation tools inside software applications.

The **LXCR** Document Object Model defines a protocol for examining document structure which recognizes the dictinct structural features of different manuscript formats, particularly LaTeX, **XML**, Concurrent Markup, and graph-oriented markup formats such as **RDF**. In short, **LXCR** offers a common representation for publications structured as either LaTeX or **XML/HTML** documents (or a combination of the two), including **RDF** and Concurrent Markup which, technically, are structural extensions of **XML**. Conceptually, the **LXCR** Document Object Model effectively extends the **XML** Document Object Model (**DOM**) to incorporate unique features of LaTeX, **RDF**, and Concurrent Markup paradigms, features which are not internally part of **XML** and therefore not directly represented by **XML**'s Document Object Model.

In **XML**, the **DOM** is considered a protocol for accessing an **XML** *infoset*, which is an internal representation of **XML** documents that abstracts from surface-level syntax. Analogously, **LXCR** documents have an internal structure which can be built from different kinds of source files; as such, different markup language may be used to create the same **LXCR** document. In this manner, publishers or educational institutions can create their own **LXCR**-compatible markup language — either a specialized version of familiar languages such as **XML**, or something entirely different. By compiling the markup to **LXCR**, manuscripts in diverse surface-level formats can be used interoperably, or combined into larger documents. Because the source markup is abstracted from the internal representation, **LXCR** data structures may be more precisely described as "infosets" rather than as "documents". Nevertheless (to sustain the analogy with **XML**), the protocol for defining and accessing the structure of **LXCR** infosets is called the **LXCR** Document Object Model (**LDOM**).

Complementing this Document Object Model, the **LXCR** Plugin Framework (**LPF**) presents a standard model for embedding publication tools into software applications — in particular, scientific and/or multimedia applications which enhance the reading experience by giving readers convenient access to interactive, multimedia content.

LXCR *for multimedia and data-sets*

LXCR is designed for a contemporary publishing industry where individual books and articles are only one part of a network encompassing text, data, and multimedia resources. Traditionally, the digital version of a published document could be fully provided in a single file — most often, a **PDF** (Portable Document Format) resource. While **PDF** files remain the primary format for readers to initially find and view publications, contemporary publishers often prioritize enhancing this primary document with one or more secondary resources — ranging from multimedia files which help readers visualize or interactively experience data presented in the publication, to analytic code or data sets which help readers evaluate or even reproduce research findings. Given this data- and multimedia-oriented publishing ecosystem, it is important that publishing software (both tools for authors to create documents and tools for readers to fully experience them) properly model and implement networks which can integrate disparate reading, technical, and multimedia applications into a unified e-reading platform.

Publication Packages

Insofar as a full publication package (not just primary books/articles but also data sets, computer code, and/or multimedia files) encompasses many different kinds of digital resources, it is no longer adequate to consider one single application (e.g., a PDF viewer) as "the" e-reader used to access a publication. Instead, a network of distinct applications may be needed for readers to experience all facets of the publication package. This is especially true for publications which are linked to supplemental files that require specialized software. As a concrete example, chemistry or biomedical papers may be linked to Protein Data Bank files, which in turn are used by molecular-visualization software to create 3D graphical representations of protein structures. In order for readers to experience this multimedia content, their document viewer must be able to launch and/or send signals to the proper molecular-visualization applications, and these applications in turn must know how to respond to the data sent from the document viewer. Such inter-operability can be achieved by embedding LXCR plugins in both applications; most of the inter-application integration is then achieved by the plugins communicating with one another.

How LPF *and* LDOM *are inter-related*

This paper will present further information about both aspects of LXCR — the Document Object Model and the Plugin Framework. To be clear, though, LDOM and LPF are closely interrelated. The data pertinent to inter-application networking, in the context of creating and reading publications, is closely linked to document structure — which in turned is expressed by a Document Object Model. For example, each supplemental multimedia resource (such as 3D graphics) is especially relevant at a specific location in the publication text. Interactive 3D views of a chemical compound (or analogously biological or physical entities such as viruses, tissue samples, vector fields, etc.) are most helpful to the reader in conjunction with locations in the text where that specific chemical, biological, or physical phenomenon is discussed. Continuing the above example, links to Protein Data Bank files should be embedded at the point in a publication where the modeled protein is mentioned or analyzed. It is not sufficient simply to note that a given multimedia file is a supplement to a given publication; instead, the link between the two resources has to be grounded at a granular location in the text — which is to say, at a specific point in the document structure. With this information, the publication viewer can add the visual cues and user actions to access the supplemental content — for example, hyprrefs, buttons, or context menu options — at the document locations which are most convenient for the reader.

This specific example — embedding visual cues and user actions within publications, so that readers can conveniently launch external software to view supplemental multimedia files — demonstrates the interrelationship between application plugins and the Document Object Model, although there are many kinds of possible inter-operation between publication viewers and scientific or multimedia applications (loading individual multimedia resources is just one use-case). The remainder of this paper will outline several different scenarios where inter-application networking is important for composing and reading manuscrips. To provide a quick summary of LXCR overall, however, note that (despite the operational relationships between DOM and plugin frameworks) currently there does not exist a standard model for combining plugins and DOMs into a unified protocol. LXCR therefore represents a novel paradigm which specifically addresses the implementational challenges of contemporary publishing technology.

The LTXH Plugin Framework (LPF)

The goal of LPF is to define a standard protocol for extending software applications so that they may be used in conjunction with software for viewing and composing publications. In an educational context, LPF plugins can augment scientific and technical applications with features that enhance their usefulness as teaching materials. LPF plugins can integrate desktop applications with course curricula, educational materials, and cloud services hosting student, instructor, and course information. In general, LPF refers not to a single plugin but rather a framework for creating plugins tailored to individual publishers, academic institutions, or educational software packages. A given scientific or technical application may host multiple LPF plugins, each tracking information structured according to the requirements of the plugin provider.

For sake of discussion, this paper will describe LPF in terms of hypothetical ETS plugins developed expressly for Educational Testing Service. To make the discussion more concrete, the paper will consider hypothetical ETS plugins for IQmol (a molecular visualization application), ParaView (data analysis and visualization software focused on statistical/quantitative data sets), MeshLab (a 3D graphics engine), Octave (an open-source Matlab emulator), QT Creator (a C++ Integrated Development Environment), and XPDF (a PDF viewer).

A suite of inter-related LPF plugins — for example, an ETS plugin suite — would contribute two kinds of functionality to their host applications: (1) tracking and presenting information specific to individual students, courses, and instructors; and (2) allowing multiple applications which each have ETS plugins to interoperate. To explain the features of the plugin suite, consider the following scenarios:

Scenario 1: A student reads textbooks, articles, or test-preparation materials which may be enhanced with multimedia content To augment the reading experience, educational texts may be supplemented with files describing visual, interactive materials which require specialized software. For example, texts about chemistry (e.g., study materials for the chemistry GRE exams) may include 3D models of chemical compouns which can be viewed with IQmol; biology texts may be illustrated with 3D tissue models which can be viewed in MeshLab; physics texts may describe equations or empirical data which may be visualized via ParaView, or Matlab simulations that could be executed through Octave. Publications could then embed these supplemental materials directly, or else include links from which the multimedia files may be downloaded. If the documents are viewed with an e-reader which itself hosts an ETS plugin — take XPDF as a case-study — then the viewer would identify the locations in the text where the multimedia files are relevant and, when the student is reading that part of the text, notify him or her of the option to automatically launch the proper application with which to access the content.

A hypothetical XPDF plugin, for example, would identify the correct application to use to view multimedia content based on the file type. This feature could also be refined via a cloud service — information provided by instructors could include notation of the kinds of software used for any particular course. This cloud-hosted data might, for instance, indicate that a specific course is using

IQmol as a pedagogical tool, and indicate that cheminformatic files linked to teaching materials for the course should always be viewed with IQmol. Once the XPDF plugin identifies the proper multimedia software to use, it can lanch the application on the student's computer and — assuming the target application also has an ETS plugin — send that application signals identifying which files to load into the application session. The XPDF plugin may also send information about the current student and class/curriculum, which the target application may use to personalize the User Interface according to students' or instructors' preferances (see the next scenario for more about personalization).

For multimedia content which is not specific to specialized technical softwar — that is, generic multimedia formats such as audio, video, or panoramic-photography — LPF plugins for PDF or ePub viewers can present this content directly, in separate windows detached from the principle document viewer, rather than routing the files to external software.

Scenario 2: Authors or instructors analyze publications' content to develop teaching materials When documents are read in an educational context, they become part of a larger ecosystem that extends beyond publication itself — an ecosystem of assessment, test preparation, curricular design, and overall teaching environments (labs, classrooms, and so forth). This superimposes additional criteria on publications: why is a given reading assignment included in a course curriculum? How can students' understanding of the reading be tested? How can the publication be used in the context of the actual pedagogical environment where students are active (in labs with specific equipment and software, for example)? How can instructors use or compose supplemental material to help students master the primary texts — explanatory reviews, for instance, but also visualization tools, glossaries, additional readings, and other resources which promote students' intuitive understanding of the subject material?

These are questions which become relevant for any document which is used in an instructional context — including but not limited to publications originally created for educational use (such as textbooks or test-preparation materials). Such use-cases illustrate the value of machine-readable Document Object Models which provide structured access to publication content, as opposed to the merely visual access offered by PDF files. Pedagogically using publications can involve extracting information from texts, or laying on additional content: inserting review questions at strategic points in the text; compiling glossaries indicating when technical terms are first used; foregrounding graphics and figure illustrations and encouraging students to explore them in depth (for instance, performing for themselves the calculations which are plotted in a statistical diagram; or identifying the axes, scales of measurement, dimensions, free and dependent parameters, and other mathematical details of a quantitative illustration). Both of these kinds of operations — extracting or adding content — require computational access to a Document Object Model.

These use-cases reveal the limitations of presenting digital manuscripts solely in PDF (or e-Pub or HTML) formats — in general it is difficult or impossible to pull information from these formats analogous to how algorithms may traverse an XML DOM. For this reason, certain publishers and electronic journals offer machine-readable views on their publications as supplements to the human-readable PDF files; however, this practice is not widely adopted in educational settings. Also, many publishers use XML as an internal representation for the publishing workflow (editing, typesetting, compositing, etc., often through legacy software originally implemented for SGML); however, those in-house files are not usually made available to the public (even purchasers or subscribers who have digital rights to the final publication). This situation explains the importance of a multi-format Document Object Model: by publishing machine-readable infosets capturing the content of human-readable manuscripts, as supplemental materials to manuscripts themselves, it is easier for educational technology to integrate publications into a digital learning platform.

Computer code which operates on LXCR Document Object Models can therefore act in consort with LPF plugins. For example, an LDOM component could define review questions or glossary definitions linked to specific points in the text; this supplemental content may then be packaged along with the original publication. With suitable LPF plugins, the document viewer can then process the review material and superimpose pedagogical questions, comments, or instructions on the underlying document text.

Scenario 3: A student launches a scientific application which is used as a pedagogical tool Teachers often instruct students to download and install software relevant to course curriculum, and this software can potentially be an essential part of the course content. Instructors may (1) use the visualization capabilities of these domain-specific applications to help students understand the concepts covered in class; (2) provide instruction in how to use the software as part of the curriculum; (3) evaluate students' understanding of the software as part of their assessment of students' mastery of the curriculum; or (4) use applications' analytic features as an overview of analytic or quantitative methodologies relevant to the course's subject matter. In the case of IQmol, features such as energy minimization, plotting orbitals, calculating vibrational frequencies, and many other chemphysical computations provide an overview of scientific concepts which might be covered in a Chemistry class.

To facilitate the use of scientific applications as teaching tools, LPF plugins help instructors personalize the applications which their students use in conjunction with course curricula. This personalization can have several dimenions, including: (1) manipulating the User Interface to prioritize concepts pertinent to each course; (2) enabling the application to present course-specific instructions to the student, such as instructions and assignments; (3) tracking a suite of resources curated for the specific class; and potentially (4) allowing students to send questions to the instructor with screenshots and application-state information. Again using IQmol as a case-study, an ETS plugin could show students a list of molecular examples discussed in class (based on data provided by the instructor) and allow students to view the corresponding 3D molecular graphics accordingly; instructors could also rearrange the IQmol menus and toolbars, to foreground those analytic tasks which are relevant to the course curriculum.

Scenario 4: A student launches an Integrated Development Environment (IDE) which is used as a teaching tool Using IDEs is a prerequite for most courses in computer science and computer programming, and in this context LPF plugins may be used as with other specialized software. An ETS plugin for QT Creator, for example, could load source and project files curated for individual classes, and display instructions or assignments for students based on instructor input. The use-cases for IDE plugins, however, extend beyond computer science proper, and include any scenario where students would write computer code as a learning aid or part of an assignment. Physics students might write algorithms to approximate answers to equations which lack closed-form solutions; biology students might write code to examine genetic patterns; chemistry students might develop simulations of materials' behavior in different force fields. In these situations LPF plugins would allow students to get information from instructors, within the IDE itself, about the goals and requirements for a code-writing exercise.

A further use-case for IDE plugins is for building other LPF-enabled applications. For example, many scientific applications can be built from source on students' computers, with the aid of IDEs such as QT Creator. In these cases instructors can provide students with application code, perhaps modified according to the course curriculum (including with their own LPF plugins). For example, QT-based applications such as IQmol, MeshLab, and XPDF can be built directly from QT Creator. An ETS plugin for QT Creator could then be the first tool which students use at the start of a course, with that plugin obtaining information from a Cloud service about which applications are needed for the course. Behind-the-scenes tasks such as defining project files and setting up build environments can then be performed automatically via the plugin, helping ensure that the student's system has the necessary prerequisites to run all the course-related software.

Scenario 5: Authors Develop Educational Materials Targeting Specific Applications In this scenario, authors compose books, articles, or test-preparation materials with the anticipation that readers will use specific software applications to enhance their reading experience. This sort of interrelationship between publications and external software is presupposed at a rudimentary level as soon as authors link documents to specialized multimedia files. For example, files in the ParaView Data (PVD) format are intended to be used by the ParaView software; as a result, documents which reference such files presuppose that readers will have ParaView installed on their computer, for the full reading experience. Similarly, files in cheminformatic formats such as Protein data bank (.pdb) or Chemical markup language (.cml) need to be opened with chemistry-related software such as IQmol.

In some cases, however, authors may desire a more rigorous degree of interop between document viewers and scientific/multimedia software. This situation applies when it is useful for readers not view particular multimedia files in domain-specific applications, but for the applications to be reconfigured so as to emphasize features or capabilities relevant to the publication which readers are studying. Applications may temporarily modify their layout or appearance for reasons similar to those identified for Scenario 2: instructors may wish to foreground certain kinds of analyses which are pertinent to course curricula, or to group together files which are interrelated in the context of the course (e.g., based on study requirements for an upcoming class session). Similar use-cases apply to publications which interoperate with domain-specific software. The difference between the current scenario and the earlier use-cases is that Scenario 2 assumed that domain-specific applications will receive data from a cloud service, curated by instructors, to personalize their appearance and features for each course. By contrast, the current scenario assumes that similar customization data is provided by publications themselves. Therefore, authors can personalize the software which enhances the reading experience, similar to how instructors may customize domain-specific software for each course.

Although it is not a prerequisite for LPF plugins, one tool which can help authors customize their audience's reading experience is the Hypergraph Text Encoding Protocol (HTXN), formulated in conjunction with LPF. HTXN explicitly supports information sharing between document viewers and external software, such as scientific/multimedia applications. While the "signals" routed via HTXN between applications may simply be instructions to load and display a single file, they may also be more complex data structures which describe how the target application should adopt to the pedagogical role of presenting multimedia or technical content specifically linked to a publication which the current user is reading. HTXN is discussed in greater detail below.

Note also that "external software" referenced by a publication may actually be a unique application developed specifically for that document. Examples of this scenario include publications which are paired with open-access data sets, when the data sets themselves include code for accessing and analyzing the published data. Contemporary standards, such as the "Research Object" format, encourage authors not only to share raw data but to develop code libraries which help readers verify research findings. These project-specific code libraries, particularly if they include GUI components, can therefore play the role of multimedia applications which augment the reading experience, enhancing the publications which present research findings in a more conventional, less interactive fashion.

The HTXN Protocol

HTXN defines a protocol for character-encoding manuscripts and defining document structures via graphs, whose nodes correspond to ranges in a character stream. HTXN therefore uses "standoff annotation" (i.e., character encoding and document structure are defined in isolation from one another), and can be used to encode manuscripts in many different markup formats. LXCR and HTXN are autonomous technologies (each may be used apart from the other), but they form a natural pairing to encode both the character/orthographic data and the document structure of manuscripts for publication.

The central goal of HTXN is to support the new generation of publishing technologies, where conventional document formats are increasingly being supplanted by digital, multimedia reader experiences. The conventional manuscript (the "primary" resource which is cited and downloaded) is then networked with a package of supplemental (or "secondary") resources. The HTXN protocol is designed to rigorously document these multimedia networks, enabling e-readers and domain-specific applications to be integrated so that readers may easily access and experience multimedia content.

The generic term "multimedia content" actually encompasses multiple phenomena:

Multimedia Files Individual files representing audio, video, or 3D graphics content. These files may be linked from specific locations in the primary manuscript, or even embedded within manuscripts when they are published in PDF format.

Data Sets and Data Visualization Publishers increasingly emphasize sharing research data alongside texts, so readers can verify or even attempt to replicate claimed results. Data sets are also a form of multimedia content because, apart from being aggregates of raw data, data sets are almost always accompanied by interactive, visual content: charts, diagrams, or plots to visualize the information holistically, or interactive tools to examine or navigate through the data set at finer scales.

Application Networks Another genre of multimedia content involves resources which may only be experienced through specialized software. This classification encompasses content from particular scientific or technical domains, which is encoded in domain-specific formats: representations of molecular structures, archaeological sites, image-processing data, wave-forms for signal processing, sentence-parses for linguistic analysis, and so forth. To conveniently access this kind of multimedia, readers need to use software which can send signals to the specialized applications having the capability to recognize the domain-specific formats and translate them to interactive, visual presentations. In short, publication viewers (e.g., e-readers) need to participate in multi-application networks, where data can be sent and received between each component. Publishers can provide this functionality to readers by implementing special e-readers and, in addition, writing plugins (or collaborating with external application developers) to ensure that applications networked with e-readers are properly aligned with the e-readers themselves.

Publications-as-Applications In some cases, publications themselves are a form of multi-layered multimedia content. This applies to publications which are not simply read from start to end, but instead naturally lend themselves to a reading process which navigates back and forth between different sections of the text, or juxtaposes different sections to be visible at the same time. A canonical example of such layered reading is testing materials and test preparation, where exam questions, instructions, supplemental materials (such as passages for reading-comprehension assessment), and comments or analyses about answers (in the case of prep materials), each form different layers which students may wish to view side-by-side. In these cases, e-readers cannot simply treat the publication as one single ePub or PDF file. Instead, the manuscript needs to identify text segments which can be factored into different layers, and the e-reader needs to implement text-viewers which allow each layer to be viewed in separate windows, with readers able to juxtapose and position the windows as desired.

HTXN represents publication manuscripts using structures which rigorously document publications' multimedia content and multi-application networking requirements. This detailed multi-media support has several dimensions:

1. Defining points in the manuscript where multimedia files are linked or embedded: this involves annotating locations in the manuscript with hyper-references to multimedia files (audio, video, etc.) which readers should be able to access when they reach the corresponding point in the text.
2. Establishing granular cross-references between publications and multimedia content: this is a more complex case where manuscript locations have to link *to* or *from* limited *portions* of the corresponding multimedia resources. For example, a passage in the manuscript may discuss a single sample within a data set; or may explicate a particular facet of the data set, such as an individual column in a tabular information space, or a specific set of statistical parameters against which quantitative operations are

performed. These scenarios call for bi-directional cross-references between the data set and the publication, wherein the granular data-set facet topically relevant to the corresponding manuscript location (the sample, table-column, parameters, etc.) is formally isolated and declared as a reference-target.

3. Cross-referances may also be defined between publications' non-textual or non-paragraph content and corresponding multimedia resources. For example, tables or diagrams visually presented in a manuscript may be liked to statistical data from which the figures are derived. A similar situation applies when visuals inluded in a publication are linked to multimedia resources which represent the same information in a different experiential register: a PDF document may include a two-dimensional graphic which is created by taking a camera shot of a **3D** model, which readers may also experience with a **3D** graphics engine; or a publication may reproduce a graph or scatter-plot derived from a data set, where data visualization software can represent the same information in a more interactive medium, with parameters plotted as curves or surfaces in a **3D** ambient space, or where systems are visualized as systems evolving over time.

The Proposed ETS Plugin Framework

As proposed, the ETS Plugin Framework would create a common code base that can be used to implement ETS-specific plugins to applications spanning a range of academic disciplines and subject areas. The primary goal of these plugins would be to connect e-reader software — applications for viewing test-preparation and course materials — with domain-specific software which instructors may use as teaching aids. With respect to GRE exams in fields such as biology, chemistry, or physics, domain-specific applications might include bioinformatics, molecular visualization, or physical simulation tools, respectively. ETS plugins would help scientific applications become more valuable as classroom tools. In particular, with ETS plugins these applications can (1) receive signals from document viewers (such as PDF viewers) to automatically display multimedia content and (2) display course and instructional materials. As a concrete example, consider molecularal visualization software such as IQmol. Via an ETS plugin, IQmol could (1) render a **3D** image given data, in a chemical file format, received from an e-reader with a similar plugin; and (2) display instructions, questions, assignments, or course-related content (such as graphics for chemical compounds discussed in class) provided by instructors.

ETS Plugin Cloud Services

Supplementing the functionality described in the preceding paragraph, Cloud Services can be used to connect individual applications to content and curricula specific to individual courses. Because most scientific applications are implemented as desktop software, the hosting of the cloud back-ends to support these features would be a natural fit for LTS's "Native Cloud/Native" (NCN) protocol, which is specifically designed to integrate desktop front-ends with Cloud/Native services. According to this architecture, NCN services would host information specific to individual courses, students, and instructors. When a student launches an application with an ETS plugin, the plugin would retrieve data pertaining to that student and to his or her classes. As appropriate, the plugin could then instruct the host application to load specific content, and/or present questions or instructions supplied by the instructor. For example, IQmol could load a list of molecular files corresponding to chemicals studied in the students' class. The ETS Plugin Cloud back-end would also be used by document (e.g., PDF) viewers to obtain information needed to properly route signals to other applications using ETS Plugins.

Student Dashboard

In addition, one feature which should be common to all ETS Plugins is a "dashboard", or a separate window aggregating the student's information. The dashboard's design would be roughly as follows: student information would be divided into four or perhaps five tabs, with labels such as "My Courses", "My Library", perhaps "My Tests", "My Applications", and "My Account". Under the Courses tab, students select one course to focus on, which would cause the Library and Tests tabs to prioritize tests, test preparation, and readings assigned for that course. Under the Applications tab, students could see a list of all applications on their computer which have the ETS Plugin installed, or which are used as pedagogical tools for their courses, or which they are instructed to install (the plugin could identify required software which students have not yet installed based on information provided by the ETS Cloud Service).