Published in final edited form as:

Balisage Ser Markup Technol. 2011 ; 7: . doi:10.4242/BalisageVol7.Maloney01.

JATSPack and JATSPAN, a packaging format and infrastructure for the NLM/NISO Journal Archiving Tag Suite (JATS)

Chris Maloney[NCBI / NLM / NIH (Contractor)]

Abstract

The NISO Journal Archiving Tag Suite (JATS, formerly known as ‘the NLM DTD’) is a common format used in publication of scientific literature. JATSPack, a proposal based on the new EXPath Packaging System, provides a way to package schema customizations and software libraries associated with any of the JATS schemas, thus making it easier for developers to distribute JATS-related resources and easier for users to acquire, install, and use them. JATSPAN (the JATSPack Archive Network) is a website allowing users to share and collaborate on JATSPacks. A client-side command line tool, jatspan, interacts with the site and eases the task of maintaining local repositories of JATSPacks.

Background

“JATS”, the Journal Article Tag Suite, is a relatively new name given to the NLM Journal Archiving and Interchange Tag Suite, which is a set of XML schemas used in the publishing of scientific literature. These schemas originated initially to meet the needs of PubMed Central, for archiving and presenting journals and articles, and have grown to become widely used throughout the scientific publishing industry.

JATS was recently released as a NISO standard for trial use. The JATS schemas are downloadable from the NLM site, and come in several flavors^[1], which have different semantics and use cases. These include Archiving and Interchange (green), Journal

The Author’s contribution to this Work was done as part of his official duties as an NIH employee and is a Work of the United States Government. Therefore, copyright may not be established in the United States. 17 U.S.C. § 105. If Publisher intends to disseminate the Work outside the U.S., Publisher may secure copyright to the extent authorized under the domestic laws of the relevant country, subject to a paid-up, nonexclusive, irrevocable worldwide license to the United States in such copyrighted work to reproduce, prepare derivative works, distribute copies to the public and perform publicly and display publicly the work, and to permit others to do so.

[1]“Flavor” is my term, which I haven’t heard used anywhere else. I will use it throughout this paper to describe one of the main categories of JATS. One “flavor” roughly corresponds to one top-level DTD file, which might itself have several versions. In a detailed accounting, there are currently seven flavors:

archiving	Archiving and Interchange Tag Set - green
archive-oasis	Same as archiving but with the OASIS table model
authoring	Article Authoring Tag Set - pumpkin
books	NCBI Book Tag Set - purple
historical	A historical version of the books DTD
publishing	Journal Publishing Tag Set - blue
publishing-oasis	Same as publishing but with the OASIS table model

This is somewhat complicated by the fact that the NISO standard versions of JATS use a different version numbering scheme, and so should also be considered separate flavors, even though they are really just newer versions of the existing NLM DTDs.

Publishing (blue), and Article Authoring (pumpkin). They also come in three different formats: DTD, W3C Schema, and Relax NG. Most publishers who use the JATS choose to use the DTD format (Rosenblum 2010). Although they are still often referred to as the “NLM DTDs”, this paper will use the terms “JATS” and “JATS schemas”, to be clear that: A) the standard is now NISO, and is no longer specifically issued by the NLM; and B) it is not just DTDs, but also other schema formats. One unfortunate aspect of the JATS acronym is that the tag suite is not limited to Journal Articles, but can be used for other types of documents (books, for example).

Introduction

This paper describes JATSPack, which is a packaging format based on the new EXPath Packaging System specification. JATSPack provides a well-defined way of packaging customizations and libraries associated with any of the JATS schema. JATSPacks are bundled packages of catalog files, schema, documentation, code, and sample files, all with a coherent purpose. The JATSPack format was primarily designed to facilitate the exchange and reuse of JATS customizations, but it could also be used to package and transfer code libraries — for example, XSLT transformations to translate JATS to/from some other format. JATSPack can be thought of as a format for plugin modules, with each plugin providing a bundle of new functionality which can be easily installed into an XML toolset.

Note

JATSPack and JATSPAN are not part of the NLM/NISO JATS. JATSPack is a proposed specification that is completely independent of the tag suite. JATSPAN is a non-commercial web site with no affiliation with NLM or NISO.

JATSPack specifies a directory structure and a package descriptor file format. The package descriptor file, similar to a manifest file in other packaging formats, describes the contents, metadata about the included resources, and the dependencies of the pack.

Establishing this format will facilitate the automatic installation and maintenance of libraries and of schema extensions, which should allow easier reuse and interchange of the schema themselves. This, in turn, should allow for easier interchange of JATS document instances.

JATSPAN, the JATSPack Archive Network, is a website similar in concept to the very successful CPAN (Comprehensive Perl Archive Network). Its main purpose is to allow users to share and collaborate on JATSPacks. Authors can upload their customizations or libraries, and can search for and download JATSPacks produced by others. Every JATSPack can be downloaded in one of two formats. The contents are exactly the same, but the download files differ in their filename extension: a .xar file, which is the extension used by the EXPath Packaging System, or as a .zip file, so that the package can be used stand-alone, without any special infrastructure.

Providing a centralized repository like this will make it easier to discover packs that might be relevant to one's needs, as well as making it easier to install and manage them on the local system.

Associated with JATSPAN is a command-line client, jatspan (lowercase), which runs on a user's machine and maintains a local on-disk repository. jatspan has a simple interface and lets users look up and install packages from JATSPAN. One benefit to this architecture is that dependencies between packs can be resolved automatically. For example, if users want to install packs that depend on others, they don't have to figure out where to go to get these, and then, perhaps, be frustrated by different directory structures and installation

requirements. Instead, the dependency packs are found, downloaded, and installed automatically.

The software and the baseline set of data files for these are being developed in the “jatspan” open source project on SourceForge.

Use Cases

The following are a few examples of use cases that these specifications are designed to address.

A publisher installs support for a third-party JATS customization

Suppose a publisher wants to integrate support for document instances that conform to some customization of JATS that was not developed in-house. Very often, document instances, if they even have a document type declaration (doctype decl) at all, do not include an absolute system identifier (a URI) to reference the DTD. So, at a minimum, either an OASIS catalog entry must be added to a local catalog file to cross reference the formal public identifier (FPI) to the DTD served somewhere on the Internet, or the DTD must be downloaded and installed on the local filesystem, and the XML processors be made aware of it. This is not especially difficult, but does require a certain level of expertise and coordination of system resources.

If the customization were packaged in the JATSPack format, then the necessary setup would be more automated. It would be as easy as identifying the desired pack and using the jatspan utility to install it in the local repository. The jatspan utility automatically updates the local OASIS catalog file with a <nextCatalog> entry to point to the new JATSPack. So *any tool* that is able to use OASIS catalogs to resolve identifiers would automatically get access to the new schema files without any additional work.

The important point here is that, because of the way the JATSPack format incorporates OASIS catalog files, this system can be used with any XML processing tool, not just ones that are JATSPack-aware. This is tested and works right now for both the oXygen XML editor and the libxml2 library.

This use case is described in more detail in the TaxPub example, below.

A developer writes a new translator, and shares it with others

In this use-case, suppose that a developer that has written a transformation from JATS into a new file format. It could be someone who did it for fun or, perhaps, a company has a vested interest in promoting the file format. For whatever reason, let's say that they want to share it with others and try to get others to adopt it and integrate it into their systems.

Right now, there is no standard way to present this transformation library to the world. Typically, the library would be presented for download on the developer's website, as a Zip file, with a README file that includes instructions for unpacking and installation. The actual details of packaging the library, finding a place to put it, and letting others know about it, involve a lot of detailed decisions. Each individual step is not difficult, but taken together, they present a barrier to this kind of sharing of code libraries.

All this is ripe for standardization. Indeed, this is the main *raison d'être* for the EXPath Packaging specification. The transformation languages in which these are usually written, XSLT, XQuery, and XProc, are mature enough that they can be written in a portable, system-independent way. The JATSPack format is specified and JATSPAN provides many

examples, which reduces the burden of making decisions about how exactly to package the bundle. JATSPAN provides a publically-accessible place to upload packages, so developers don't have to find places on their own websites for them. If a library were created as a JATSPack, and were put on JATSPAN, it would be easier for potential users to discover and install. If this library had dependencies on others, then those others would not have to be bundled with this package. The jatspan client program would take care of resolving these dependencies automatically.

This use case is discussed in more detail in the EPub Transforms example, below.

A publisher evaluates JATS for the first time

A publisher or a developer who is introduced to the JATS for the first time can be a bit overwhelmed. To be sure, JATS is extremely well documented, as one can readily see by browsing the NLM web site. Yet the fact that JATS have been in use for a long time, and that there are now many different branches (flavors) and versions, can make them seem, to someone who is not familiar with them, complicated.

Someone who wants to evaluate the suitability of JATS for their particular needs might want to be able to quickly set up a system that can process any JATS instance document, regardless of what flavor or version that instance document conforms to. This is difficult to do, because each flavor and each version of the JATS is distributed as a separate, flattened Zip file which includes the bundled version of all of the files needed for that particular set. For example, the books 2.3 DTD Zip file includes all of the books-specific modules, as well as all of the shared modules, and even the xhtml and mathml2 library modules.

For each flavor/version of interest, it is necessary to download and extract the Zip file, and then manually tweak the provided OASIS catalog file for that set, and then point your XML tool set to use that catalog file. Because there are so many different tag set bundles, it would be quite time-consuming to configure a system that is able to use them all. This is not seen as a problem, because this is not the primary use-case that the NLM site is designed to serve. They provide stand-alone bundles specifically so that a user who is interested in only, for example, Publishing 2.3, can download that version and not be confused by all of the others (Jeff Beck, personal communication, 3/22/2011).

But if someone is interested in configuring a system that can understand and process any JATS instance document; either for the purposes of evaluating JATS, or so that they can exchange documents with other organizations, it would be quite difficult.

To address this use case, and to provide a basis for the JATSPack architecture, I have repackaged all of the JATS DTD modules as JATSPacks, put them into a single bundle, and made the repackaged versions available on JATSPAN. This base bundle is (optionally) installed automatically when you first run the jatspan client utility, as part of its setting up the JATSPack repository. See Existing JATS DTDs are available as JATSPacks below for details.

A publisher develops a new JATS customization

Let's say that a publisher has a need to customize the JATS, in order to include domain-specific data within the source documents in their database. This is actually the primary use-case that JATSPack was designed to address, and this is the complement of the first use case described above. That use case described customizations from the user's perspective, and this use case describes them from the author's perspective.

There are a number of concerns that tend to make people reluctant to define customizations. Among the most prominent of these is that a customization to the JATS might make their documents less portable. This use case is discussed in detail in the Customizations and Compatibility section, below.

JATSPack

EXPath Packaging - the basis for the JATSPack format

JATSPack is an extension of the EXPath Packaging System (hereinafter EXPath-pkg, written by Florent Georges). EXPath-pkg is a simple, concise format for the packaging of files of various core XML technologies, including XML Schema, Relax NG, Schematron, NVDL, XSLT, XQuery, and XProc (Georges 2010). It also provides a flexible extension mechanism which is exploited by JATSPack to define additional file types, and other requirements specific to its needs.

As defined by EXPath-pkg, a package is a set of files that fulfill a common purpose. Each package has a globally unique name (which is a URI) and an abbreviated name (hereafter referred to as *abbrev*), an NCName. *Abbrev* values are case-insensitive, and, by convention, all lowercase. A component is one file within a package, and can be one of several different types.

A package includes a package descriptor (similar to a JAR manifest file) which lists the package contents and top-level metadata about the package. This metadata includes the URI name, *abbrev*, version number, and a list of dependencies. Finally, all the files of a package are arranged in a specified directory structure, and used to create a Zip file, which by convention is given a name based on the *abbrev*, the version number, and a “.xar” extension.

EXPath-pkg is an emerging standard which is already supported by several tools, including the eXist and Qizx XML databases, the Saxon XSLT and XQuery processor and the Calabash XProc processor. (Support in Saxon and Calabash is via third-party plugins.) It is also implemented as an open-source Java library, hosted as the expath-pkg project on Google Code.

The goal of JATSPack is to be a forward-compatible extension of EXPath-pkg^[2], meaning that any system that is capable of deploying EXPath packages will be able to deploy JATSPacks (although not all of the features of JATSPacks would be fully exploited). Note that, in fact, both EXPath packages and JATSPacks are forward-compatible extensions of simple Zip files, meaning that JATSPacks could be used by anyone, without any special infrastructure at all, just by unzipping them onto the local filesystem.

JATSPack extends EXPath-pkg as follows.

1. JATSPack-specific extensions to the package descriptor file. This will include the ability to record the author, release date, and other metadata. See Overall structure of JATSPacks, below, for more information about this.
2. Addition of the OASIS catalog file type. See OASIS Catalog files, below.
3. Addition of documentation file types. See Documentation, below.

^[2]At the time of this writing, there are a few areas of incompatibility between the JATSPack format and EXPath-pkg. Effort is underway to reconcile these. The specific incompatibilities are called out in footnotes which follow.

4. Sample instances documents. If the pack includes a schema customization, then it should include a set of sample documents. See Sample files and automated tests, below.
5. More specific directory structure. EXPath-pkg is very loose about the directory structure of files within the main package directory. JATSPack specifies this structure more explicitly.
6. Requirement for two-part *abbrevs*^[3]. Currently, EXPath-pkg *abbrevs* are simple NCNames. In JATSPack, they are two NCNames separated by a slash. For example, “nlmjats/archiving”.

Another source of inspiration for the JATSPack format was the One Document Does it all (ODD) format of TEI. The ODD has very clean structure, which encourages literate programming (documentation and code in the same source document), examples of usage, clear separation of concerns, and best of all, encapsulation of all the relevant data into a single document with a standardized structure (Lou Burnard et al 2005). The JATSPack format strives to be similar to ODD, and could even be thought of as encapsulating the library or customization in one document, if one expands the definition of “document” to include “Zip file” (which is quite reasonable).

Overall structure of JATSPacks^[4]

As already mentioned, every JATSPack must have an *abbrev* and a version number. In JATSPack, in order to help ensure that the abbreviated names will be unique, they are composed of two parts, each of which is an NCName (by convention, all lowercase, and the underscore character should not be used in either part). These are separated by a forward slash in the package descriptor file, for example, “nlmjats/archiving”.

The name of the package file is specified to be *abbrev-ver.xar*. When the two part *abbrev* is used in the package filename, the forward slash is translated into an underscore. For example, “nlmjats_archiving-1.0.xar”. The actual low-level format of the file is that of a Zip file, and the directory structure within that archive is as shown here^[5].

```
[root]
  abbrev-1/
    abbrev-2/
      README.txt (optional)
      expath-pkg.xml
      catalog.xml
      dtd/
      rng/
      rnc/
      xsd/
      xslt/
      xquery/
```

^[3]JATSPack two-part abbreviations are incompatible with the existing EXPath-pkg format.

^[4]As described above, some of the requirements for the JATSPack format derive from the fact that it is an extension of EXPath-pkg, and some are JATSPack-specific.

^[5]The internal directory structure of JATSPack differs from that specified for EXPath-pkg, and is another area of incompatibility. The main difference is the location of the package descriptor file, which is in the root directory for EXPath-pkg packages. In JATSPack, it was moved to the package’s content directory. This way, it is possible to unzip the packages directly onto the filesystem, without requiring any shuffling of the file locations.

```
xproc/
doc/
samples/
resources/
test/
```

At the top two levels are directories corresponding to the two parts of the *abbrev*. Within the package contents directory are the EXPath-pkg descriptor file `expath-pkg.xml`, and optionally a `README.txt` file. Also in this directory are an OASIS catalog file and one subdirectory for each of the main file types of the package.

The `expath-pkg.xml` file format is described in Georges 2010. Extensions to the file format, as allowed by the extension mechanism, allow us to include metadata specific to JATSPacks. The 'jp' namespace prefix is used for this.

The following is an example of what the package descriptor file would look like for a hypothetical customization of the base Archiving and Interchange Tag Set, version 3.0, designed to add elements and attributes to describe filesystems^[6].

```
<package xmlns="http://expath.org/ns/pkg"
  xmlns:jp="http://jatspack.org/ns/jatspacks"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  abbrev='sys/archiving-3.0-filesystem'
  name='http://jatspan.org/jatspacks/sys/archiving-3.0-
filesystem'
  version='0.1'
  spec='1.0'
  jp:spec='0.1'>
  <title>Filesystem customization of JATS Archiving 3.0</title>
  <dependency package='http://jatspan.org/jatspacks/nlmjats/
archiving'
    semver='3.0' />
  <rdf:Description>
    <dc:creator>Chris Maloney</dc:contributor>
    <dc:date>2011-04-04</dc:date>
    <dc:description>
      This customization of the Archiving and Interchange
Tag Set, version 3.0,
      adds filesystem-related elements and attributes, for
describing computer
      filesystems within instance documents.
    </dc:description>
    <dc:language>en</dc:language>
    <dc:rights>This work is in the public domain.</dc:rights>
  </rdf:Description>
  <!-- Package contents. -->
```

[6]Note that the conventions for the abbreviated names have yet to be worked out.


```

<jp:catalog>
  <file>catalog.xml</file>
</jp:catalog>
<xslt>
  <import-uri>http://jatspan.org/ns/jatspacks/filesystem/to-
archiving.xsl</import-uri>
  <file>xslt/to-archiving.xsl</file>
</xslt>
</package>

```

The top-level `<package>` element includes attributes that give the *abbrev*, the package name (a URI), and the version of this package. The `@spec` attribute indicates the version of EXPath-pkg that this package descriptor conforms to. The `<title>` element gives a human-readable title of the package.

The `@jp:spec` attribute is a JATSPack extension to the EXPath-pkg package file format. This attribute is required for all JATSPacks, and its value gives the version of the JATSPack specification that this package conforms to.

Another JATSPack extension is the addition of the `<rdf:Description>` metadata section, which uses the XML vocabulary defined by the Dublin Core Metadata Initiative (DCMI) to allow authors to describe the package in more detail.

EXPath-pkg allows zero-to-many `<dependency>` elements, each of which indicates a dependency of this package. In this example, we see that the *filesystem* package depends on the JATS Archiving 3.0 base JATSPack.

The `<jp:catalog>` element specifies the location of the OASIS catalog file that accompanies this package. Finally, the `<xslt>` element specifies the location of an XSLT module, and the URI which is used to import it from other modules.

OASIS Catalog files

Note that there are no individual entries in the package descriptor for the top-level DTD files of this *filesystem* customization. That's because these DTD modules are specified in the OASIS catalog file, and, following the principle of DRY, are not duplicated in the descriptor. Resolution of public identifiers (FPIs) and system identifiers (URIs) to DTD modules within this package is delegated to the catalog file mechanism.

An elided view of the OASIS catalog file accompanying this package is the following.

```

<catalog xmlns="urn:oasis:names:tc:entity:xm1ns:xm1:catalog"
  prefer='public'> <
  public publicId="-//JATSPAN//DTD Filesystem Archiving 0.1//EN"
    uri="dtd/filesystem.dtd"/>
  ...
</catalog>

```

This catalog file uses the `@uri` attribute with a relative URI value to specify the location of the main DTD included with this package. An important point here is that there is nothing

special about this catalog file. It is usable by any system that can read OASIS catalog files, and does not depend on those systems' understanding the JATSPack format.

Examples

Existing JATS DTDs, and selected Relax NGs, are available as JATSPacks

The existing set of 34 complete DTDs of the JATS, as downloadable from the NLM archive_dtd and jats FTP sites, has been repackaged as JATSPacks. In addition, the latest NISO trial versions of the Relax NG schemas have been included in those JATSPacks. (The older NLM JATS versions of the JATSPacks only include the DTDs.) These can all be downloaded from JATSPAN as a single bundle. The DTDs include all seven flavors, and up to seven versions of each, going all the way back to version 1.0, and all the way up to the recently released NISO trial versions.

As described in the use case “A publisher evaluates JATS for the first time” above, each DTD version on the NLM FTP site is downloadable as a single Zip file which includes all of the files required for that version, including all the core files that the version depends on. As mentioned, this is very suitable for, for example, a publisher or author who is interested in configuring a system capable of dealing with a specific flavor and version (for example, Article Authoring 3.0) of JATS. Unfortunately, however, this method of distribution makes it difficult to implement a single system that's capable of processing instance documents that conform to *any* of the many different versions of JATS.

Because each Zip file on the FTP site is a complete set of all of the files needed by an instance of the DTD, among the complete set of all flavors and versions, there is a lot of duplication of files. This leads to some ambiguity when many of these sets coexist on the same system at the same time. If the same public identifier is used in multiple catalog files on the system, and point to document instances which are not identical, it is difficult for someone configuring the system to sort out whether the differences are substantive, and if so, which is the canonical instance.

The JATSPack base bundle eliminates this redundancy, factors shared modules out into a **core** package, and cleans up a few inconsistencies. The result is a well-defined directory structure, with well documented dependencies. Each individual package comes with an OASIS catalog file that only has entries for the file in that particular module. A top-level catalog file is provided that uses the <nextCatalog> element to include all of the others.

These JATSPack versions of the schema are 100% compatible with those on the NLM site, and because of this, the JATSPack versions of the NISO DTDs are fully compliant implementations of the NISO standard^[7].

The Relax NG Zip files downloadable from the NLM site do not include OASIS catalog files, and so to include these files in the JATSPacks, it was necessary to create entries for them. This is discussed in more detail in Namespaces and Relax NG, below.

[7]When doing this work, I faced a question, and reached a conclusion, which might be controversial. Because I moved files into a different directory structure, but I wanted them to remain usable by systems without OASIS catalog files, using the relative system identifiers, I had to change these identifiers in the modules that reference the moved files. I also made other minor edits to reconcile cases where, for example, two different instances of a document were referred to by the same public identifier (FPI). I made these changes, without changing the FPIs, for those files that were changed. I think this is the right decision, for the following reason. The significance of an FPI is that any tool that resolves it correctly will get identical results. The FPI specifies a *logical resource* — a DTD or an external parsed entity — not a *specific byte sequence*. With this change, the resolution of any given FPI will still result in the exact same entity replacement text, when all of the sub-entities are correctly resolved.

Journal Publishing 3.0 Preview Stylesheets as a JATSPack

These are a set of preview stylesheets for the Publishing tag set, version 3.0, that are available for download from the NLM website. They are authored by Wendell Piez, and are described very nicely in Piez 2010, from the Proceedings of JATS-Con, 2010.

These were repackaged into the JATSPack format. This was very straightforward, and involved the following steps:

1. Assigning an *abbrev*: “html/publishing-3.0-preview”; a name, which is a URI: “http://jatspan.org/jatspanks/html/publishing-3.0-preview”, and a version: “1.0”.
2. Creating the requisite directory structure.
3. Moving files around. The files in this library comprise 14 XSLT files, nine files of documentation, a CSS resource, and a readme.txt file.
4. Reimplementation of all of the shell stylesheets as individual steps defined within a single XProc pipeline. At the top level of the preview stylesheets are seven “shell” XSLT files, which are basically pipelines. One of the seven should be used depending on choices of citation style, desired output format, and content filtering (print only). Unfortunately, these shell stylesheets use Saxon-specific extensions and depend on one of the non-free versions of Saxon. It makes sense to implement these as an XProc pipeline, with three parameters for the choices described above.
5. Writing the package descriptor file. This involved assigning absolute import URIs to the outward-facing XProc and XSLT modules.
6. Zipping it into the file html_publishing-3.0-preview-1.0.xar, and uploading it to JATSPAN.

By itself, repackaging this library in this way could not be considered an improvement over the current deployment of this tool, as a Zip file. However, there are a few advantages. Since this package is on JATSPAN, it can be searched for and discovered on JATSPAN. The documentation can be browsed by anyone from the JATSPAN site, without downloading and extracting the package. Anyone using the jatspan client utility could install it automatically with the following simple command.

```
jatspan install html/publishing-3.0-preview
```

Anyone with an EXPath-pkg enabled system would have instant programmatic access to the XProc stylesheets and XSLT modules, through their import URIs. Normally this would require some integration effort, but with JATSPack, it is automatic.

Furthermore, there are no disadvantages to packaging this way. The JATSPAN site allows anyone to download this package as a .zip file (exactly the same file as the .xar, but with a different extension), which can then be unzipped on the filesystem and used just as easily as before.

Another advantage is that other JATSPacks, written by other authors, could declare a dependency on this package. This would simplify the installation and integration of those later packages. And this point makes a good segue to the next example.

EPub Transformation

At the JATS-Con last year, Laura Kelly gave an excellent presentation on the EPUB format and a set of transformations for converting JATS documents into EPUB (Kelly 2010).

She wrote the transformations in standards-conforming, system independent, XSLT 2.0 and XProc. The effort to repackage this as a JATSPack was quite minimal, and involved basically the same steps as above. The *abbrev* assigned to this JATSPack is “epub/jats-to-epub”.

This set of transformations uses the preview stylesheets described above. The original Zip file distribution of this library includes a copy of the XSLT files from the preview stylesheets that are used. In the JATSPack, those have been removed, and replaced with a dependency declaration in the package descriptor file. The advantage of this is that users can avoid having two copies of the same library (the preview stylesheets) on their system at the same time. In this simple example, it would not be a big problem. But the problem of multiple copies of the same resources can get severe as the number of packages and libraries increases.

Accessible Tables Stylesheet

The next example is not a JATSPack at all; it is an EXPath-pkg. My colleague Martin Latterner wrote a very nice XSLT stylesheet which takes as input an XHTML table. The stylesheet computes, for each table cell, which horizontal and vertical headers correspond to that cell. The stylesheet then adds classes to the cells and headers to explicitly encode the associations.

This is necessary so as to properly code tables for compliance with Section 508 of the U.S. Rehabilitation Act of 1973, providing for accessibility to the disabled, in particular the visually impaired. With tables marked up like this, screen readers can assist visually impaired users by reading, for any given table cell, the headers associated with that cell. Figure 1 illustrates the result of this transformation.

Given that tables can have very complicated structures, with multiple @colspan and @rowspan cells in different places, this is not a trivial algorithm.

In addition to the XSLT, Martin also wrote a JavaScript module for testing. The JavaScript reacts to mouseover events, and causes the current cell and its associated header cells to be highlighted with a different background color, as shown in Figure 1.

This stylesheet provides a very specific bit of functionality, and is something that other users could benefit from, so it is suitable for packaging and putting on a public repository. However, since it operates on XHTML tables, and is not JATS specific, it is more appropriate to package it as an EXPath-pkg and to put it on CXAN (the Comprehensive XML Archive Network), so that is what we did.

The reason this is included as an example in this paper is to illustrate the important fact that JATSPack systems are backwards-compatible with EXPath-pkg. Any system set up to use JATSPacks will also be able to install and use any of the EXPath-pkgs on CXAN.

To use this particular accessible tables stylesheet, one would install the EXPath-pkg, and then simply import the stylesheet using the import URI defined in the package descriptor. It could be wrapped in an XProc step, or the template could be invoked directly from another stylesheet.

TaxPub

TaxPub is a customization of JATS in DTD form, which was described in a paper presented at last year's JATS-Con (Catapano 2010). The extension allows for the encoding of literature of biological taxonomy, and in particular, taxonomy treatments, which are blocks

of well-structured markup with very specific semantics. It is maintained as an open-source project on SourceForge, with documentation on the project home page.

The TaxPub customization comprises a set of DTD files, which defines a set of new element names with a “tp:” pseudo-namespace-prefix (e.g. “tp:taxon-treatment”).

This has been repackaged as a JATSPack and put on the JATSPAN website. The following is a summary of the steps involved:

1. Assigned a unique **abbrev**: “taxpub/schema”, and version: “0.1”.
2. Created the specified directory structure:

```
taxpub/
  schema/
    0.1/
      dtd/
      doc/
      samples/
```

3. Moved the TaxPub DTD-specific .dtd and .ent files into the dtd directory. There are five of these.
4. Removed the no-namespace-prefix versions of the DTD. These have been deprecated.
5. Fixed the relative system identifiers in each of the TaxPub modules so that they reflect the fixed directory structure of a JATSPack installation. This is similar to what was done for the base JATS modules. It is not strictly necessary, but has the benefit that the same modules can be used on systems that don't support OASIS catalog files.
6. Fixed the DOCTYPE declarations of the sample files. As downloaded, the included sample files' DOCTYPE declarations did not use formal public identifiers (FPIs), and used only relative system identifiers. For example:

```
<!DOCTYPE article SYSTEM “../tax-treatment-NS0.dtd”>
```

These were changed to use FPIs, and the relative system identifier was changed to reflect the new directory structure. For example:

```
<!DOCTYPE book PUBLIC
  “-//TaxonX//DTD Taxonomic Treatment Publishing DTD v0
  20100105//EN”
  “../dtd/tax-treatment-NS0.dtd”>
```

This allows these sample files to be used in the automated tests. They also now serve as examples of the proper DOCTYPE declaration to use for instance

documents of these types. If it is desired that documents of this type be easily exchangeable between systems, then it is crucial that document instances be properly self-identifying.

7. Created a catalog.xml file in the base “taxpub” directory. This cross references each of the defined FPIs in this customization to it’s correct module in the package. For example:

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
  prefer='public'>
  <public publicId="-//TaxonX//DTD Taxonomic Treatment Publishing
DTD v0 20100105//EN"
    uri="dtd/tax-treatment-NS0.dtd"/>
  ...
</catalog>
```

8. Moved the documentation into the docs directory.
9. Created a new README.txt file, adapted from the existing one, and adding a bit of information about the new structure.
10. Zipped the result, gave it the name **taxpub_schema-0.1.xar**, and then uploaded it to the JATSPAN website.

To see the advantages of this packaging format, let’s consider a very specific use-case. Suppose that I am a user who works with the oXygen XML editor, and I find that I have a need to read, edit and validate TaxPub files.

If TaxPub were not packaged in a JATSPack, the process of adding support to oXygen would not be difficult, but would require a little bit of expertise and some time. I would have to download the Zip file, unpack it, and read the README file to see if there are any specific instructions I needed to worry about. Then I would find a place for it to reside on my filesystem and move the files there. The tricky bit is then to set up oXygen to understand the public identifier, and have it retrieve the correct DTD. I could either add an entry to my own personal catalog file (if I have one), add it to oXygen’s master catalog file, or set up a “document type association” inside oXygen.

Now let’s suppose instead that I will use the JATSPack version of TaxPub. Of course, there is a bit of up-front configuration required, to set up a JATSPAN repository and point oXygen to it. But that is a one-time operation, which then allows me instant access to any number of JATSPacks. Also, this one-time setup is actually quite trivial, and easier than installing the support for the single TaxPub document type as described above. The setup involves two steps:

1. Installing the JATSPAN software, and running it for the first time. This establishes a JATSPAN repository on the local filesystem.
2. Telling oXygen where to find the master catalog file of the JATSPAN repository, by adding an entry in the “XML Catalog” preferences box. Note that *any* tool that is able to use OASIS catalog files could be pointed to this JATSPAN master catalog file, and would have the automatic ability to reference all of the schema files of all of the installed JATSPacks.

After the initial setup is done, to install support for TaxPub as a JATSPack, I need to enter one command:

```
jatspan install taxpub/schema
```

That is it — everything else is done automatically. When the TaxPub JATSPack is downloaded, a <nextCatalog> entry is added to the JATSPAN repository master catalog file, so oXygen instantly resolves any public identifiers defined by TaxPub. After entering the above command, I can validate and process TaxPub documents in oXygen.

Customizations and Compatibility

The original motivation for developing the JATSPack specification was to devise a means to package schema customizations together with software libraries that allow users to transform documents to and from the new schema. This is still, perhaps, the primary use-case.

There is an ongoing sea change in the way that scientific research is published and presented. The lines between traditional definitions of media types, such as journal articles, books, wikis, blog posts, presentations, etc., are continually getting blurred (Owens 2010). This implies that the number of different types of content included with scientific publications is growing rapidly. Often it is most appropriate to include disparate types of data with the original source documents, and that means either customizing the JATS schema, or providing some other means of including it, such as (for example) using the flexible <named-content> elements, and adding appropriate layer validation.

The TaxPub example illustrates that the JATSPack/JATSPAN infrastructure can facilitate interchange of documents and data between organizations. A common theme overheard at last year's JATS-Con was that often, individual publishers' versions of their JATS DTDs are not interchangeable. In other words, PMC's version of JATS is not the same as Highwire's, even if they nominally use the same flavor and version of the tag suite. With a small amount of up-front work to resolve discrepancies and repackage customizations, this interchange problem can be helped, if not solved.

The TaxPub example also serves to illustrate that the cost of doing compatible customizations need not be high. This can lead to easier reuse of existing XML vocabularies. The XML landscape is vast, and vocabularies have been defined for wide array of specialized topic domains — biological taxonomy is just one example. As mentioned previously, and as TaxPub illustrates, often it is very appropriate and beneficial to mix markup from other vocabularies directly into the instance documents. If the customization is deployed as a JATSPack, then one could make the customization without having to worry quite so much about breaking compatibility with existing systems, and the ability to exchange documents.

But often one wants to do more with a document than just validate it, and that is where the true power of EXPath-pkg, and by extension JATSPack, lies. Customizations can also be packaged with software libraries in XProc, XQuery, and XSLT, that allow users to transform documents to and from the new schema. This would facilitate making forwards-compatible schema customizations.

First, let's review what is meant by backwards and forwards compatibility, in the context of document schemas. For this, I highly recommend Orchard 2004. In a nutshell, backwards compatibility means that existing (version A) documents can be used by new XML

processing systems, and forwards compatibility means that newer documents (version B) can be used by existing processing systems.

A very successful model of forwards compatibility, that we are all familiar with, is the “must ignore” pattern of extensibility of HTML. This stipulates that HTML renderers must ignore any tags that they don’t understand. In effect, this is a forwards-compatibility extension substitution rule. It allows future designers to customize the HTML schema, adding elements and attributes, while being able to predict how document instances in the new schema will be processed by old systems. The “must ignore” rule is one substitution rule, but it is very limited. If designers could include a set of transformations with the new schema, then those designers could define their own substitution rules.

TaxPub as an example of customization

TaxPub serves as a good example of this type of customization. It includes many new elements that deal specifically with taxonomy treatments, and thus introduces the problem of forwards compatibility:

TaxPub, as an extension, provides semantics beyond what is available in the base DTD through creating newly named elements - thus lending itself to domain-specific application. However, TaxPub instances may not be easily processed by applications already familiar with the Publishing DTD.

— Catapano 2010

This problem could be addressed by writing a set of XSLT stylesheets and including them with the TaxPub JATSPack. These stylesheets would transform the new elements into plain-old-JATS. This would make it easier for third-party systems to install meaningful and useful support for TaxPub documents. Unlike the “must ignore” rule, the substitution rules written in XSLT could provide for richer markup in the result document, displaying the new content in visually distinctive ways, and perhaps even preserving the semantic content by means of special attribute keywords (for example).

Designing a customization that is both forwards and backwards compatible, and compatible with other already-existing schema, is not trivial. With the JATSPack format, and a set of examples and detailed how-to instructions, some of the work required can be made easier.

Style checkers

The definition of “customization” can be broad. A customization does not necessarily include a new DTD (or Relax NG or XSD schema). Customizations are often implemented as the addition of layered validation implemented in, for example, Schematron, that imposes a set of style rules that are more specific to a particular organization’s use of the JATS. Sometimes these validation rules are coded simply as XSLT stylesheets that are applied to instance documents, and return a predefined report format.

According to Beck 2010, the results of a survey of publishers indicated that, “Half of the respondents said that they impose rules other than schema validation on their content.”

Since JATSPack supports Schematron, XSLT, and other formats, there is no reason that these sorts of style checkers couldn’t also be packaged as JATSPacks and made available to third parties. Since JATSPack/JATSPAN imposes rules on versioning and the uniqueness of identifiers, it can help to safeguard against incompatibilities that can result from ad-hoc sharing of these sorts of libraries.

Namespaces and Relax NG

JATSPack can also ease the migration away from DTDs and towards Relax NG for JATS schema. There are several reasons why this is desirable. DTDs do not properly support XML namespaces, and so hamper interoperability. Using namespaces will allow third parties to create customizations without having to worry so much about name clashes that can result from mixing vocabularies.

“The major design point of XML namespaces is to allow decentralized extensions.”

— Orchard 2004

That decentralized customizations and extensions occur is a fact. I don’t think anyone would argue that decentralized customizations is a bad thing, or should be prohibited (even if that were possible). Yet, an inherent problem is that they can be difficult to manage and control. The JATSPAN infrastructure is designed to adapt to this decentralization, while at the same time providing specifications and imposing requirements, such that the proliferation of new schema doesn’t become a problem. It does this by requiring that the formal public identifiers (FPIs) and URIs used to identify resources are well defined and unique, and also by virtue of its fixed directory structure, ensuring that separate individual JATSPacks fit together on the same system without clashing.

Invoking the TaxPub customization as an example again, in the paper given at JATS-Con last year, the author describes some of the inherent difficulty of working with DTDs:

The lack of robust namespace support in DTD removed the option of importing external schemas into TaxPub. This would make synchronization less onerous, for example, were it decided to include Darwin Core elements in TaxPub. It also would enable the inclusion of XML data in TaxPub instances themselves rather than on linking to them as external documents.

— Catapano 2010

There are other reasons for recommending the use of Relax NG over DTDs, and an entire paper could be written on just that topic.

However, there are two problems that must be overcome before Relax NG schemas can be used easily for JATS documents. The first is the lack of URI names for these Relax NG resources, and the second is that there is no standard way for documents to identify the Relax NG schema to which they conform.

The first problem, the lack of URI names, is inherent in the current JATS infrastructure. Relax NG schema are identified by URIs, and not FPIs. In other words, URIs are used as the names of these resources, and there is no other name. That this is a problem was eloquently described by Norman Walsh in his blog post from 2004 titled, “On the Web, My Name is 266 North Pleasant Street” (Walsh 2004). The current recommendations of the W3C lead inexorably to the choice that new resources, such as Relax NG files, should be identified by absolute, canonical, persistent, and stable HTTP URIs.

NLM has declined to issue such URIs for these resources. They are worried about the potential for excessive server traffic driven by automated tools that process these URIs, similar to the experience of the W3C.

Without these URI identifiers, the Relax NG schema cannot be referenced in the instance documents in a clear, unambiguous way. And as mentioned before, crucial to interchange of document instances is that they identify themselves.

With the JATSPack/JATSPAN architecture, I have attempted to address this problem in the following ways.

- First of all, JATSPack, since it is based on EXPath-pkg, supports the inclusion of Relax NG XML and compact-notation files. These should reside in the “rng” and “rnc” subdirectories of the package root, respectively.
- Every JATSPack is assigned a name, which is an absolute URI. This includes the base JATS packages. For these, I assigned HTTP URI names that use the “jatspan.org” domain. For example, the JATSPack implementation of the NISO trial version of the Article Authoring schema, version 0.4, has the name “http://jatspan.org/jatspacks/nisojats/articleauthoring/0.4”.
- The OASIS catalog files included with each JATSPack includes a <rewriteURI> entry which allows the identification of any resource within the JATSPack by using an absolute URI starting with the URI of the JATSPack.
- This provides a *de facto* canonical absolute URI name for each of these resources.
- JATSPAN may serve these resources from these absolute URIs, but may not. That decision is TBD.
- Regardless, that URI can be safely used to identify these resources within instance documents.

An example should make this clear. The NISO-JATS 0.4 version of the Article Authoring Relax NG schema is downloadable from the NLM FTP site. The Zip file does not include an OASIS catalog file. The schema resource itself does not have a canonical URI name.

While repackaging this as a JATSPack, I assigned a URI name to this package of “http://jatspan.org/jatspacks/nisojats/articleauthoring/0.4”. I added the following <rewriteURI> entry to the OASIS catalog file for this package.

```
<rewriteURI uriStartString="http://jatspan.org/jatspacks/nisojats/
articleauthoring/0.4/"
    rewritePrefix="/" />
```

In effect, this assigns a URI name to the master Relax NG schema file of this package, and that name is “http://jatspan.org/jatspacks/nisojats/articleauthoring/0.4/rng/JATS-articleauthoring0.rng”. Please feel free to use this name in any context, to refer to this resource. As mentioned above, JATSPAN might serve this resource at that URL, but might not; it is not guaranteed.

The second problem is that there is no standard, agreed upon way for instance documents to identify themselves as conforming to a particular Relax NG schema. The philosophy behind this fact is that schema validation should be a separate process that is not specified by the instance documents. In other words, the idea is that the system, and not the document, should decide what schema to apply, and that for a given document, any of a number of schema might be applied, depending on the context.

This is a nice theory, but in practice it has impeded the adoption of Relax NG in this particular domain.

The oXygen XML editor defines a processing instruction (PI) that can be used within an instance document. For example, given the URI described above, an instance documents could now use this PI to identify itself as a document conforming to this Relax NG schema:

```
<?oxygen RNGSchema="http://jatspan.org/jatspacks/niso-authoring/0.4/rng/
JATS-articleauthoring0.rng"
    type="xml"?>
```

This is very processor-specific, and therefore is not a good general solution.

Perhaps the most straightforward way of making the association is by using a namespace on the root node of the document. But the immediate problem with this is that the NISO-JATS Relax NG schemas will fail if any non-null namespace is given on the root node of any document.

Solving this particular problem is out of the scope of this paper, but I would like to suggest that the Relax NG schema files delivered with JATS be modified to specify a canonical namespace for each of the document types. Failing that, this could always be done as an independent customization, in a separate JATSPack.

More JATSPack features and recommendations

Documentation

It is recommended that each JATSPack includes documentation, including structured documentation of any schema extensions. The exact format of this documentation is TBD, but the goal is that it should seamlessly integrate with existing JATS documentation. That is, the documentation should simply “plug in” to the documentation provided by the JATSPacks that come before it in the dependency tree. For example, hyperlinks to element or attribute descriptions, between the documentation for the various packs and between packs and the base JATS documentation, should resolve correctly. This aspect of the format is still under exploration.

Code libraries in XSLT, XQuery, and XProc

JATSPacks can include library functions that pull out specific data from instance documents, so that the data is easily accessible from any JATSPack-enabled system. This is a way to provide different “views” on instance documents that could be appropriate for different purposes. JATSPacks that import and customize others could also extend the library functions defined in the imported packages. This is analogous to the object-oriented programming paradigm of creating a derived class from a parent class, and overriding or extending certain methods.

As described above under “Customizations and Compatibility”, in order to facilitate interchange of document instances, authors of JATSPacks that customize the schema are encouraged to provide a stylesheet for conversion to and from the “standard JATS”.

Also, for these customization JATSPacks, authors are encouraged to supply XSLT stylesheets that import and extend the Journal Publishing Preview Stylesheets. In this way, a complete preview stylesheet for the new JATS customization would be available for use, instantly upon installation of the new JATSPack. If the stylesheet which converts to “standard JATS” is provided, then the implementation of this preview stylesheet is trivial —

it is just the baseline preview stylesheet applied to the output of the “standard JATS” stylesheet.

Sample files and automated tests

In order to enable interchange of documents and of the software that processes them, instance documents must be self-identifying. Among the recommendations for proper JATSPacks which supply DTD customizations, sample files should be included which have the appropriate DOCTYPE declaration for that customization, which use the correct formal public identifier (FPI). The jatspan client utility, when installing a new JATSPack, performs automatic tests, which include validation of each of the sample files. In addition to validating them in place, this step includes a test in which each sample file is copied to a separate temporary directory and then validated. This ensures that the FPI is used to resolve the DTD, not the system identifier, and guarantees that instance documents modeled after the JATSPack sample files can be exchanged between different systems.

Additionally, any type of JATSPack can include its own predefined test in the form of an XQuery function which returns a boolean true (pass) or false (fail). Since all of the resources inside a JATSPack are addressable by using URIs relative to the “name” URI of the JATSPack itself (given in the package descriptor), these test could operate on the included sample files, to verify the integrity of the package as a whole, and that the software library operates correctly on the target system.

JATSPAN

As mentioned in the introduction, JATSPAN is a website at jatspan.org, and is based on the concept of the CPAN website, providing a place where users can upload and share JATSPacks.

There are three complementary, interlocking faces of the JATSPAN site.

1. A repository of JATSPacks, and a website allowing users to upload and download from that repository.
2. A client application jatspan, which enables users to maintain a local installation of JATSPacks.
3. A set of RESTful web services (these are envisioned for the future).

The initial implementation of this site is in Perl, and is being developed on the jatspan open-source project on SourceForge.

The “phase 2” implementation of JATSPAN will copy the Servlex/eXist architecture of the CXAN site as described in Georges 2011.

jatspan.org web site

Here is a list of some of the features of the JATSPAN website:

- Anyone can browse the list of JATSPacks, see their descriptions (which are extracted automatically from the package descriptors) and download those of interest. JATSPacks can be downloaded in one of two forms (that differ only in the filename extension): XAR files (which have the .xar extension) or Zip files (which have the .zip extension).
- Users can also browse the JATSPack documentation on the JATSPAN site, without having to download the package.

- Authors of JATSPacks can use the JATSPAN site to check that their preferred *abbrev* is not already in use.
- Registered users can upload JATSPacks. These are unpacked on the server, automatically checked for consistency (for example, that the package descriptor file is valid) and instantly made available to other users for browsing and download.
- The site automatically runs the automated document-generation tools XSLStyle and xqDoc over the XSLT and XQuery JATSPack components, respectively, and makes the generated documentation instantly available on the site.

The site, in its alpha incarnation, is implemented as a set of Perl CGI scripts based on the Catalyst web framework.

jatspan client program

I would like emphasize again that JATSPacks are usable on any system without any special infrastructure. Downloading the package as a Zip file and extracting it to the local filesystem could be done exactly the way it is done now, and the included schema, documentation, and library files are just as usable.

However, there are additional advantages to setting up a local repository and using the jatspan client program to manage it. The jatspan client is a simple program that users can download and install. The interface is through the command line, and it is implemented as a Perl script.

When it is run for the first time, it will ask for a location to create the local jatspan repository. This is a directory on the local filesystem, which is the root directory to which downloaded JATSPacks are extracted. The client program then creates this directory and writes a jatspan master OASIS catalog file to it, as well as performing a few other setup tasks.

The master OASIS catalog file can be used by tools, such as oXygen, to resolve identifiers defined within JATSPacks. These XML tools should be set up to point to this catalog file. As described above, this only needs to be done once, and from that point on, those tools will be able to resolve any identifiers defined by any JATSPack that is installed on the local system.

Also as part of setup, the user can choose whether or not to download and install the repackaged base JATS bundle, described in Existing JATS DTDs are available as JATSPacks above. The advantage to choosing to install it is that all of the flavors and versions of JATS will then be available immediately. But it is also possible to use the system without installing the base JATSPacks. Because of the system of resolving dependencies automatically, the required base JATS DTD files will be available as soon as the first JATSPack is installed. In other words, they could also be installed one-by-one, as the need arises.

After setup, the client can install any desired JATSPack by entering the command

```
jatspan install abbrev
```

This will install the latest version of the JATSPack indicated, along with all of its dependencies. It will also update the master catalog file with <nextCatalog> entries, pointing to the catalog file provided with each new installed JATSPack.

As an implementation detail, note that the JATSPAN server uses many of the same Perl functions as the jatspan client, to perform many of the same sorts of tasks. For example, when a user uploads a JATSPack to the server, the server “installs” this JATSPack in its repository, in a manner quite complementary to the operation performed when users download JATSPacks to their client machine.

Future possibilities

The next-phase implementation of JATSPAN will copy the Servlex/eXist architecture of the CXAN site as described in Georges 2011. Besides the obvious advantage that this is a more flexible architecture, it can take advantage of the XML resources that the site is designed to serve, it can also provide a model for how to set up a flexible, extensible XML processing toolchain for organizations that need to develop one. This architecture is dubbed the JATSPack Application Framework, or JATSPAF. Figure 2 shows the relationships among all of these entities.

The following is a list of features that could be implemented in a future release of the JATSPAN website, that would enhance its usefulness. These are listed primarily to illustrate the benefits that this architecture and this infrastructure could provide.

- Faceted search on uploaded JATSPacks, with facets defined for element and attribute names, authors, dependencies, documentation, etc. When JATSPAN is implemented in eXist, this is a logical next step.
- JATSPacks could be usable directly off of JATSPAN, without installing them to a local machine. This would involve pointing an XML tool at the master OASIS catalog file on JATSPAN, which includes <nextCatalog> entries for all of the JATSPacks uploaded to date. This would allow users to run tests and use a JATSPack without installing it locally, for evaluation purposes, in the same way that they now can browse the JATSPack documentation before downloading the package.
- JATSPAN could provide instance document tools, such as a validator and document previewer. This would let anybody upload a document that purports to conform to any of the JATS schema (including any customized version implemented as a JATSPack), and validate it and/or preview it. These are similar to the services provided by the PMC style checker and article previewer.
- JATSPAN can be configured to do more thorough checking of uploaded JATSPacks. For example, when a new pack is uploaded, checking that system ids and public ids are unique, and that resources identified by system ids and/or public ids are not changed (it could issue warnings if they are). Also verifying that there are no circular dependencies.
- JATSPAN could implement RSS feeds, allowing users to subscribe to see when new JATSPacks, or new versions of JATSPacks of interest, are uploaded, for example.
- JATSPAN could be configured with a discussion forum, or an issue tracker, such that users could comment on, rate (“like”), or issue trouble tickets about uploaded JATSPacks.
- A “Roma for JATS” could be developed and served from JATSPAN. Roma is a web-based GUI tool for building customizations to the TEI schema. The output of the “Roma for JATS” would be a complete JATSPack.

Conclusions

Throughout this paper, I have touted the potential benefits of the JATSPack format and the JATSPAN site. These benefits include:

- making it easier to deploy systems that support a wide variety of XML formats;
- facilitating the creation, distribution, and maintenance of schema customizations and libraries of code that would be associated with them;
- allowing developers to easily exchange these customizations and libraries; and thereby
- facilitating the exchange of scientific documents and data, while preserving their rich semantic content.

To fully realize the possible benefits, much more work would be required. However, I believe that establishing this specification and web site is a good step in the right direction.

One of the goals of this proposal is to help publishers and digital repositories to produce XML systems that are more open. Sharing of schema customizations and libraries would reduce the amount of duplicated effort that is expended when adapting systems to different XML document types. In order to achieve this goal, this format will have to obtain some measure of support within the publishing community. The challenge lies in persuading systems architects that this format would benefit them, so that they will expend the effort to adapt their systems to support it. By providing a “starter set” of JATSPacks, as described in the examples section, I hope that I’ve shown that: A) creating JATSPacks is easy, and B) sharing JATSPacks will benefit everybody.

Of course, there is nothing inherently JATS-specific in the JATSPack format. The ways in which JATSPack extends EXPath-pkg, as described above, are primarily to add schema of various formats, sample documents, documentation, and OASIS catalog files. It would be nice if these extensions could be rolled into the EXPath-pkg specification itself. That might eventually obviate the need for the JATSPAN website; it could be merged with CXAN. Whether that would help or hurt the cause of getting the format adopted is anybody’s guess.

Comments, suggestions, and help are all very welcome. Please visit the project page on Sourceforge.

Acknowledgments

I would like to thank Kim Tryka, Florent Georges, Abe Becker, and Rebecca Orris for valuable feedback on this paper. Also, thanks to the anonymous reviewers who provided useful suggestions.

This research was supported in part by the Intramural Research Program of the NIH, National Library of Medicine.

I’d also like to thank you, the reader. If you’ve gotten this far, and are even reading the acknowledgements, that is quite impressive! As a small lagniappe, here is a nice Unicode snowman for you: ❄!

Bibliography

- [Beck 2010]. Beck, Jeff. Are We There Yet? An introduction to the first Journal Article Tag Suite Conference [Presentation slides]. 2010.
<http://jats.nlm.nih.gov/jats-con/program/2010/presentations/beck.pptx>
- [Lou Burnard et al 2005]. Burnard, Lou; Rahtz, Sebastian. One Document Does It All [Presentation slides]. 2005. <http://www.tei-c.org/Talks/2005/Sofia/odds.pdf>

- [Catapano 2010]. Catapano, Terry. TaxPub: An Extension of the NLM/NCBI Journal Publishing DTD for Taxonomic Descriptions. Proceedings of the Journal Article Tag Suite Conference. 2010. <http://www.ncbi.nlm.nih.gov/books/NBK47081/>
- [Georges 2010]. Georges, Florent. Packaging System, EXPath Candidate Module 11 November 2010 [Specification]. 2010. <http://expath.org/spec/pkg/20101111>
- [Georges 2011]. Georges, Florent. CXAN: a case-study for Servlex, an XML web framework. XML Prague 2011 Conference Proceedings. 2011. <http://www.xmlprague.cz/2011/files/xmlprague-2011-proceedings.pdf>
- [Kelly 2010]. Kelly, Laura. JATS to EPUB: Unraveling the Mystery. Proceedings of the Journal Article Tag Suite Conference. 2010. <http://www.ncbi.nlm.nih.gov/books/NBK47314/>
- [Orchard 2004]. Orchard, David. Extensibility, XML Vocabularies, and XML Schema. XML.com. October. 2004 <http://www.xml.com/pub/a/2004/10/27/extend.html>
- [Owens 2010]. Owens, Evan. The Evolving Information Ecostructure of Publishing. Journal Article Tag Suite Conference. [Presentation]. 2010. <http://videocast.nih.gov/summary.asp?Live=9729&start=18474> (video); <http://jats.nlm.nih.gov/jats-con/program/2010/presentations/owens.pptx> (slides)
- [Piez 2010]. Piez, Wendell. Fitting the Journal Publishing 3.0 Preview Stylesheets to Your Needs: Capabilities and Customizations. Proceedings of the Journal Article Tag Suite Conference. 2010. <http://www.ncbi.nlm.nih.gov/books/NBK47104/>
- [Rosenblum 2010]. Rosenblum, Bruce. NLM Journal Publishing DTD Flexibility: How and Why Applications of the NLM DTD Vary Based on Publisher-Specific Requirements. Proceedings of the Journal Article Tag Suite Conference. 2010. <http://www.ncbi.nlm.nih.gov/books/NBK47101/>
- [Walsh 2004]. Walsh, Norman. On the Web, My Name is 266 North Pleasant Street [Blog post]. March. 2004 <http://norman.walsh.name/2004/03/03/266NorthPleasant>

Head	Head	Head	
Head	Data	Data	
Head	Data	Head	Head
Head	Data	Data	Data

Figure 1. Accessible Table Illustration

Example of a table marked up with accessibility classes. The data cell pointed to by the mouse pointer (the blue cell) has several header cells associated with it. Those header cells are highlighted in red.

Figure 2. Relationships among software and systems

The architecture for the JATSPack / JATSPAN systems borrows heavily from the work already done by Florent Georges.