Improving Multi-Platform Workflow at Harbour Publishing and Douglas & McIntyre

by
João Simas Ferraz
B.F.A., University of British Columbia, 2012

Research Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Publishing

in the
Master of Publishing Program
Faculty of Communication, Art and Technology

Approval

NAME:	**João Simas Ferraz**
DEGREE:	**Master of Publishing**
TITLE OF PROJECT:	**Improving Multi-Platform Workflow at Harbour Publishing and Douglas & McIntyre**

Supervisory Committee

Juan Pablo Alperin
Senior Supervisor
Assistant Professor

John Maxwell
Supervisor
Associate Professor and Director

Anna Comfort O'Keeffe
Industry Supervisor
Managing Editor, Harbour Publishing and Douglas & McIntyre
Madeira Park, British Columbia

DATE APPROVED: **August 24, 2015**

Abstract

AT HARBOUR PUBLISHING, THE PRODUCTION OF MULTI-PLATFORM MATERIALS is inefficient. Data is not effectively reused, and a lot of time is spent manually copying and reformatting data. The project described in this report aimed to develop a system that could simplify their workflow, using their promotional catalogues as a starting point. Harbour's website contains all the same data that goes into their catalogues, and since it also generates their ONIX Record, the project used it as the source. By leveraging several XML formats and some web programming, the project's prototype successfully automated most of the catalogue production process. Unfortunately, there were too many limitations to make real-world usage possible; however these were not limitations that cannot be overcome given more time and some financial investment, and the project provided an effective proof that automated workflow systems are a worthwhile investment.

Table of Contents

	Abstract	**iii**
	List of Tables, Figures & Code Snippets	**vi**
CHAPTER 1	**The Past: Introduction**	**1**
	Current Practices	4
	Current Issues	8
	Existing Market Solutions	11
CHAPTER 2	**The Present: Developing an Alternative**	**15**
	Content Streams	15
	The ONIX Structure	19
	The ICML Structure	24
	The XSL Transformation	28
	No Processing Required	30
	Simple Equivalencies	31
	Multi-Level Equivalencies	33
	Context-Dependent Processing	37
	Processing Inline Styles	40
	The PHP Implementation	46

CHAPTER 3 **The Future: Evaluation** **50**

The System Test 50

Content Problems 51

Problems with BISAC Headings 53

Prototype Shortcomings 53

Final Balance 55

Far Future: Other Applications 56

Bibliography **58**

APPENDIX **Sample ONIX Record** **59**

List of Tables, Figures & Code Snippets

TABLE 1.	Catalogue copy revision steps and staff involved.	4
FIGURE 1.	Current promotional copy workflow at Harbour Publishing.	6
FIGURE 2.	The Flying Narwhal workflow.	17
FIGURE 3.	Proposed promotional copy workflow.	18
SNIPPET 1.	ONIX representation of the tagline.	22
SNIPPET 2.	Setting up character styles in ICML.	25
SNIPPET 3.	Setting up paragraph styles in ICML.	26
SNIPPET 4.	Structure of the Story tag.	27
SNIPPET 5.	Template used to select what parts of the ONIX file to process.	29
SNIPPET 6.	The Title template.	31
SNIPPET 7.	The Product Form template.	32
SNIPPET 8.	Matching BISAC Subject Headings with the Genre template.	33
SNIPPET 9.	Matching BISAC Subject Codes with the Genre template.	34
SNIPPET 10.	The complete Genre template, including individual books matched by ISBN.	36
SNIPPET 11.	Adding "and" between second-to-last and last contributor names.	38
SNIPPET 12.	Adding role descriptors with the Contributor template.	38
SNIPPET 13.	Checking for repeated roles with the Contributor template.	39
SNIPPET 14.	The complete Contributor template.	39
SNIPPET 15.	Unescaping the Description field.	41

SNIPPET 16. Template for copying a file while making further changes. 43
SNIPPET 17. Template for applying paragraph styles. 43
SNIPPET 18. The two parts of the Character Style template. 44
SNIPPET 19. Preserving forced line breaks. 45
SNIPPET 20. Setting a dynamic parameter passed by PHP. 48

CHAPTER 1 The Past

Introduction

IN THE AUTUMN OF 2012, DOUGLAS & MCINTYRE PUBLISHERS INC. (DMPI), one of Canada's largest independent book publishers, faced bankruptcy. The company's assets were divided, and the following spring Howard and Mary White, owners of the established BC house Harbour Publishing, bought the Douglas & McIntyre imprint, which included DMPI's largest title list. While White did not merge the two companies legally nor financially, much of the existing Harbour staff was charged with carrying the Douglas & McIntyre list, supported at first by a relatively small roster of remaining DMPI employees. The workload ballooned, and there was little time to study the divergent ediatorial, production, marketing and distribution practices and identify inefficiencies on either side.

Two years later, the companies' practices have begun to align and the inefficiencies that emerge from juggling two companies with different methods are finally being addressed. Divergences in house style have been eliminated; both companies have begun to use the same print-on-demand and ebook distribution services; and accounting systems will soon use the same software.

Yet there is certainly still room for improving and streamlining workflow in several stages of production. Both Harbour Publishing and Douglas and McIntyre (henceforth referred to jointly as HPDM) produce ebook versions of nearly every title, but the conversion has been poorly integrated with the production workflow and has frequently been rushed or postponed. Certain Harbour titles are made available on KnowBC.com—a subscription-based resource maintained by Harbour primarily for local schools—but the system for posting new content is so unintuitive as to become daunting, even for simple books. The process of preparing the seasonal sales catalogues, posting new books to HPDM's respective websites and keeping both instances current and accurate is repetitive, which leaves room for error. Each of these workflows is separate, offers its own challenges and must be addressed in different ways. However, they share a common problem: inefficient reuse of information that was captured in earlier stages of the workflow.

In the Information Age, books, even printed ones, can be thought of as large sets of data, including both their narrative content and myriad metadata. In theory, this data can be retrieved and reused with minimal effort, but most publishers do not take advantage of this possibility. To present the same books in different forms, words and data need to be copied. In the early days of books, this was done by scribes with calligraphy quills, and more recently by designers with desktop publishing software. Copying data and content between formats is a long, laborious job with a significant margin for error, but a sound digital workflow can circumvent this manual process.

At the beginning of a book's in-house life cycle, when the production and marketing teams settle on its specifications and selling points and prepare early marketing materials, a vast amount of data needs to be copied—repeatedly, and from multiple sources to multiple outputs. The promotional copy used for sales and marketing, and the catalogues it is presented in, can be thought of as a microcosm of a book, as they have a similar cycle: writing, editing, deadlines, a print life and a secondary but equally—if not more—important digital life. Given the appropriate infrastructure, once the content is prepared, processing it for its multiple outputs could take as little as a few clicks—relatively effortless next to the burden of copying, pasting and manually reformatting.

As a rather self-contained process that usually takes only a few months from start to finish, they are also prime candidates for experimentation. At HPDM, the process suffers mostly from the need to produce two simultaneous catalogues within a tight schedule, compounded by the companies' individual practices and requirements, making catalogues a particularly interesting case-study. In fact, most staff members have their first exposure to upcoming titles while preparing the catalogues.

Schedules vary, but there are typically two types of catalogue: one with only new books and one that includes both the new front-list and highlights from the back-list. For the Chapters/Indigo Winter buying period, HPDM produces a short version of the catalogue that only features the front-list titles slated for release in the time frame specified by Indigo. Sometimes this is the earliest catalogue deadline, so the advance catalogue is produced first. The complete catalogue features the entire season's front-list, including those titles in the Chapters/Indigo buying period. It also highlights recent books going one or two seasons back, other popular back-list titles and an exhaustive list of books in print, each set with a different level of detail.

A front-list page typically includes a promotional description of the book, which covers content and form and touches on marketable points; short biographical notes about the primary contributors; and basic specifications: the genre, projected release month, format, price, dimensions and projected page count, a note about illustrations, sales rights and both print and ebook ISBNs. When the advanced catalogue is produced first, much of the front-list copy is simply replicated in the complete catalogue. Likewise, most of the back-list section is carried over from the previous season's catalogue.

The main book description from the catalogues becomes the basis of almost all future promotional material for the title. It is posted to the company website, and is often recycled—with appropriate revisions—for the book's jacket or back cover, press releases and other marketing materials.

THIS REPORT WILL FOLLOW THE STEPS TAKEN TOWARD IMPROVING HPDM'S catalogue workflow. We began by reviewing the established process to identify its flaws. With that information in hand, we explored existing options before developing a system finely attuned to HPDM's particular needs. The development process involved evaluating ways to organize the workflow; establishing the parameters of a system prototype; and then finding specific solutions to the important problems.

Although HPDM has some unique requirements and challenges in terms of the workflow, the types of data that it handles are unlikely to differ much from that encountered by other publishers, and our development and evaluation process offered general lessons for any workflow that reuses information. Thus, this report will cover each step and the relevant technologies in detail, in hopes that it may help other publishers who also seek to optimize digital workflows.

Current Practices

THE PROCESS OF WRITING AND REVISING PROMOTIONAL COPY IS INHERently complex. Even when there are no technological hindrances, copy follows an arduous journey from creation to final approval before it is typeset (see Table 1). HPDM's technological preferences only complicate the procedure further.

In most cases, book descriptions are born in the hands of a production assistant. The drafts are usually written in Microsoft Word, then saved to the HPDM network drive. Since the catalogue is produced early, often before the manuscript is even completed, the production assistants are not usually familiar with the titles, and must write the promotional copy based on an advanced draft or outline and what they can gather from reading the book proposal. As such, the first drafts often don't highlight all the appropriate elements, and must undergo repeated revisions. This is where the process begins to get complicated.

Table 1. Catalogue copy revision steps and staff involved.

STEP	STAFF	TASK
1	Production Assistant	Write first draft
2	In-House Senior Editor	Revise (with Track Changes)
3	Freelance Substantive Editor & Author	Revise (with Track Changes)
4	Marketing Team	Suggest revisions
5	Publisher	Approve copy
6	Designer	Typeset and layout
7	Production Assistant	Trim copy to fit
8	Designer	Finish typeset and layout

An in-house senior editor reviews the drafts and marks it up using Word's Track Changes and commenting features. This version is also saved to the network, in the same location as the previous, and with the same filename, but with the editor's initials appended at the end. The production assistant addresses the changes and comments, then adds their own initials. This initials system serves as a rudimentary form of version control: the initials indicate who has edited each file, and since every version is saved, it is possible, to a certain extent, to track who made what changes, and to recover anything that was previously deleted.

Since almost every substantive editing job is contracted out, the production assistant may have questions about the book that cannot be answered in-house. In these cases, the questions—and the latest draft of the catalogue copy—are sent to the substantive editor. These editors rarely have access to the HPDM network because they are freelancers, so files are usually sent by email. By virtue of their fresh knowledge of the book, the editors are in a good position to not only answer questions, but also make further edits to the catalogue copy for clarity, accuracy and diction. Again, changes are tracked and comments are marked. The new copy is returned via email and again saved to the network, reviewed and initialed.

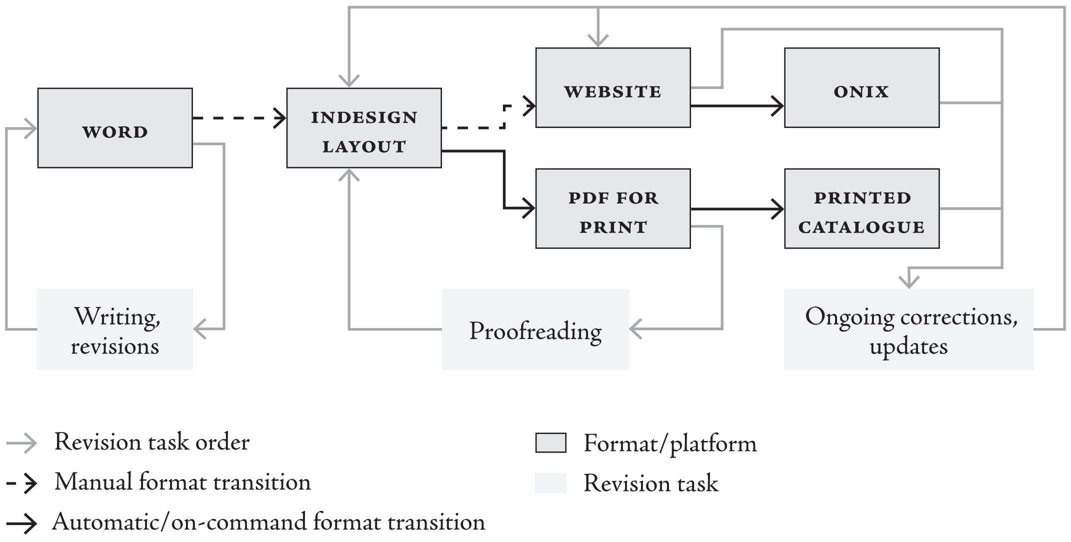
The authors must also review the copy before it is considered ready. Once again, the drafts are emailed and changes—this time usually for diction or sentence structure—are tracked, and files are saved with the author's initials appended to the filename.

Another round of revisions happens in-house: the marketing team has the opportunity to offer their input. Overall, the process can be long, with each step taking multiple rounds and filenames becoming unruly strings of initials. Finally, the production assistant sends the copy to the publisher for final approval. Changes at this stage tend to be minor.

As with most book publishers, HPDM operates on a print-first model for both its books and catalogues. Once authors and editors have approved the drafts, they need to be added to the catalogue layout in Adobe InDesign (see Figure 1). Although a catalogue template exists—a blank file with most of the page grids and paragraph styles defined—it is practical only for the advance catalogue, where all of the information is new. In the complete catalogue, so much of the back-list information is maintained from one season to the next that, in practice, it is simpler to duplicate the previous catalogue and make the necessary changes from there.

There is one final step of revision: frequently the copy provided is too long to fit in the available space on the page. Depending on how much overflow there is, the designer may trim the text themselves or request that the production assistant do so, since they are more familiar with the title.

Figure 1. Current promotional copy workflow at Harbour Publishing.



Once all of the front-list copy has been laid out, several updates need to be made to the recent and back-list sections. Some of these updates include final page counts of the current season's titles, back-list prices and changes to sales personnel. While these changes tend to be made on HPDM's websites on an ongoing basis, they are only done as a batch to the print catalogue, often using the websites for reference.

Next, the catalogue pages are sent to a proofreader, and invariably there are errors. When the advance catalogue is produced first, some front-list pages end up proofread twice—once in each version of the catalogue—and unfortunately there are occasionally errors that are not caught the first time.

Once the print catalogue is sent to the printer, it is time to enter the book information on the Harbour and D&M websites. Each website is independent and runs on different content management systems (CMS). Neither CMS is a popular, well-supported platform like Drupal or WordPress—in fact, they are both custom systems, crafted to fit the needs of each publisher. Each has its own structure, different features, different names for the same type of content and, of course, its own quirks. But there are some similarities: for example, the book description field on both websites accepts HTML, but requires it to be entered as plain text. Catalogue copy is thus usually copied, pasted into the CMS and HTML formatting is manually added (neither system offers a "what you see is what you get," or WYSIWYG, editor or even buttons for applying italics and other text formatting); other book data, such as ISBNs, must follow the same road, one item at a time.

Occasionally, some errors aren't noticed or changes become necessary after the catalogues are printed. These are immediately corrected on the website, but they must also be corrected in the latest catalogue InDesign file, to ensure they are not perpetuated in the following season's catalogue. For the remainder of the season, however, further updates will continue to be made on the website, while the catalogue is not maintained in the same way.

The Harbour CMS, although clunky, offers one special feature of note: the website data is used to generate the company's ONIX for Books Product Information Message—usually referred to simply as ONIX. This is an XML file that includes as-complete-as-possible metadata about all the books handled by a publisher or distributor. ONIX is an efficient way of disseminating title information to others down the distribution line, and keeping that information accurate. For example, a publisher can update book information on retailers' websites by modifying their ONIX feed. I will address ONIX in more detail later on (see "The ONIX Structure" on page 19).

In order to generate a complete ONIX Record for every title, the Harbour CMS offers a wide variety of fields—including many that have no front-end visibility. This is already one step ahead of D&M, whose ONIX Message is generated by the accounting software ACUMEN Book, which in turn is handled by people who do not work in the head office. Data is, again, manually copied. At this point, the Harbour CMS is the only part of HPDM workflow that reuses data in any regard. It is a model that must be followed.

FINDING A SYSTEM TO AUTOMATE HPDM'S CATALOGUE CREATION MAY BE the key to optimizing production on a greater scale. The specific systems used to streamline catalogue production and website maintenance are unlikely to apply to general book production; and it is possible that whatever solutions we find will be incomplete or too difficult to implement. However, a real-life, tangible demonstration of how content can be generated, used and reused more efficiently across platforms by using a system finely attuned to HPDM's needs may convince the staff that the transition to new systems will be worth the up-front time and financial investment, and set the companies on the right path. And the first step in this direction is to identify, specifically, the problems we face.

Current Issues

EACH STEP OF THE PROCESS DESCRIBED ABOVE OFFERS ITS OWN PROBLEMS. While it does not often snag, the initials system of version control is convoluted. The network ends up full of files that are, for the most part, copies of each other with only small changes. With copy that takes a little more time and effort to perfect, file names can end up having long strings of initials after all of the revision steps. Word's Track Changes and commenting features are both widely used and acceptably flexible, but more sophisticated version control and collaboration features are only available in recent releases, and thus not universally available. As such, Word is not the most efficient software for the type of minute, back-and-forth collaboration that this process requires.

When it comes to typesetting front-list pages, designers have two options: copying the text from Word and pasting it in InDesign, or using InDesign's

Place command. Neither option is ideal. When text is copied into InDesign from a different program, any formatting[1] that was applied to the original is lost. If the designer goes with the copy-and-paste method, they will then need to go over the text carefully to ensure all italics, small capitals, indents, and so on, are correctly applied.

The Place command, however, does import formatting. This can be a very powerful feature if styles are correctly applied to begin with, but they rarely are.[2] Word is more often than not used as a visual editor, and formatting is usually applied directly rather than through a style. For example, if the text includes a book title, which needs to be italicized, most people will use the "Italics" direct formatting button instead of Word's built-in "Book Title" style. The "Italics" button indicates that a particular set of characters need to be italic; the "Book Title" style instead indicates that those characters belong to a set that needs to be treated in the same way—which may or may not be italics, depending on the style's definition. This distinction also carries over to InDesign when text in Placed. If the designer wanted to change how book titles are presented, those italicized through a style could all be changed at once; those italicized by direct formatting would need to be individually addressed. Since the latter case is by far the most common, each small design or typesetting change could take hours to accomplish.

Bigger issues begin to arise after the print catalogue is ready and the information needs to be copied to the website. The root of the problem is that the description fields take plain text only. They offer no formatting shortcuts, which means that any HTML—such as `<i>` or `<em>` tags for italics—needs to be typed out. This is further complicated by several factors. On the one hand we have staff-related issues: historically, some production assistants have had very little familiarity with HTML; and since HPDM recruits most

1 Formatting refers to how text is presented: typeface, point size, indents, to name a few possibilities.

2 A style is a set of formatting options, which can be applied and modified in block. For example, the first paragraph of each section in this document has the same style. If I decided to add drop-caps, I would simply modify the style definition, and every first paragraph would gain a drop-cap at once. Styles are a common feature in both professional design software and word processors, but they are rarely used in the latter.

of its production assistants through co-op and internship programs, they usually only stay for four to eight months, which leaves proper training in HTML practically out of the question. On the other hand there are problems with the technology: neither website features any form of validation for the HTML in their description fields; and it is nearly impossible to access the websites' CSS files.

Over the years, so many different people have entered the book descriptions that there are many errors and inconsistencies in the HTML. For example, the prevailing instruction for formatting paragraphs in the Harbour description field was, for some time, to use an opening paragraph <p> tag, but never close it with a corresponding </p> because complete paragraphs—with both opening and closing tags—had too much space between them. This was a workaround for a simple formatting issue: the website's CSS files, which define the formatting associated with each structural tag, applied margins both above and below <p> tags, creating the extra space. However, the CSS files are not accessible to the production assistants entering the data; they are managed by the Information Technology professional who manages the website. Since he doesn't work at the office, it is thus faster to use a hack fix like unclosed tags than to find the correct, long-term solution.

The problem is more pronounced when the description field formatting is entered by hand, but persists even if pre-formatted HTML is exported from InDesign. InDesign's HTML exports are always excessively specific, always using style definitions but never deferring to defaults. If the HTML export was copied and pasted as-is, there would be a lot of unnecessary code, naming styles that do not correspond to any definitions in the website CSS. Cleaning it up could be easily automated, but very few production assistants have the technical know-how.

Further, the book descriptions aren't the only pieces of data that need to go through this cycle—the book specs do as well. For example, ISBNs are first assigned by making a note on the ISBN log—a printed list of available ISBNs. The ISBN is first typed into the same Word file as the promotional copy for the book. It is then copied into InDesign, and copied again onto the website.

ISBN mistakes come up frequently; the numbers may have been mistyped, or a title may have been assigned an ISBN belonging to the wrong company. These are usually caught and corrected before the catalogue is printed but, rarely, an error makes it to either the printed catalogue, the website or both. In this case, a correction would need to be typed at least twice—once on the website, and once in InDesign, for future catalogues. The margin for error is significant.

Genres are another example, and even more complex than ISBNs. The editors decide the appropriate genre—usually in general terms, like "poetry" or "memoir"—and it is copied into InDesign like the description. But both HPDM websites require BISAC Subject Codes, which are the industry standard and the ONIX requirement. Thus the production assistant cannot simply copy the chosen genre from InDesign into the website; they must assign BISAC codes when posting the book. Unfortunately, the available BISACs are not always a perfect match for the chosen genre. For example, Harbour frequently uses "Regional Interest" to describe all books where the primary audience is in a particular community, but there are no BISAC codes that approximate this. While this translation step cannot be completely eliminated by improving how data is reused, the error margin can be reduced by minimizing the duplication of labour.

The scale of problems faced in catalogue production is quite daunting, and their ramifications are far-reaching. With that in mind, we focused on searching for ways to improve catalogue production without considering how, or even if, those same solutions would apply in other contexts. Systems for preparing ebooks or posting new content to KnowBC.com, for example, can be pursued later.

Existing Market Solutions

Because HPDM's promotional copy process has issues at many different stages and various unique requirements, we were unable to find any workflow solutions that could cover every aspect. However, there are several existing services on the market that can help take care of individual problems and, when put together, offer a more complete solution. Services of note include Dropbox and Google Drive for the editorial stages, and BookNet Canada's CataList and Above the Treeline's Edelweiss for production.

Both Dropbox and Google Drive are cloud-based services that assist with collaboration, each with its owns strengths: Dropbox focuses on storage while Drive concentrates on creation. The key feature of Dropbox is the desktop app, which creates a folder that is automatically synchronized, whenever there are changes to it, across all computers that have access to the same files. This is very similar to the functions already in place with HPDM's office network server. The differentiating element is that Dropbox makes it easy to share subfolders and files across multiple accounts, allowing out-of-office editors and staff to access the same files as in-house personnel, with the same ease. Without the need to email files back and forth, there is also less chance of files being forgotten or lost.

With Google Drive, users can also store files on the cloud in an easy-to-share manner. Drive even offers a desktop app almost identical to Dropbox. Drive's advantage is that it also offers creation tools that are directed at collaboration. Google Docs, the web giant's free office software suite that works seamlessly with Drive, is a set of in-browser alternatives to the near-ubiquitous Microsoft Office. They offer many of the same features, including a form of Track Changes and advanced commenting, but with the bonus of live collaboration. When multiple users access the same file, they can edit and comment on the file at the same time—users can even see what others are doing as cursors move around the document in real time. While it may be uncommon for catalogue copy to be edited simultaneously by multiple people, this feature can be a wonderful resource for those times when the copy takes more work than most.

Unfortunately, Google Drive's greatest asset is also its most significant drawback: it requires people to abandon their software of preference. This can be a big issue especially since many HPDM authors have been using Word for many years, and are frequently not comfortable with online authoring platforms and even Dropbox's desktop app, which blends well with the folder systems of all major operating systems, can be hard to grasp for some.

Both Dropbox and Google Drive are well-known and established, and both are already used by HPDM for other purposes. On the production side, the existing solutions are less popular, although they are also more directed at the issue at hand: producing catalogues. BookNet Canada (BNC), a non-profit organization that champions publishing technologies and workflow, offers CataList as a solution.

BNC is in a prime position to help publishers with their workflow, particularly when it comes to catalogues. BNC's flagship program, BiblioShare, is a book metadata aggregator and sharing service. It takes the ONIX files supplied by publishers, validates them to ensure they are complete, and makes the data available to the industry as a whole, as well as the public.

CataList builds on all this accumulated data. It allows users to quickly generate digital catalogues that can be tailored to specific purposes, like the preferences of a particular store. Since these catalogues are digital and based on the supplied ONIX data, every time the book data for a particular title is modified, so are all the catalogues it appears in. This is reusable data at its finest, except that BNC does not offer any print catalogue solutions. HPDM could certainly benefit from CataList, but since the companies are not prepared to abandon paper catalogues altogether, the combined cost of both using CataList and building print catalogues indepedently became a major hindrance. A complete solution is still pending.

On the other hand, Above the Treeline, a United States company that also aims to support the book industry, offers Edelweiss, their own alternative to CataList. It offers many of the same features, and also creates digital catalogues from ONIX metadata. As a US company, it is better attuned to—and sees more use in—the US market than the Canadian market. However, it offers an important advantage: through a partnership with Ingram Lightning Source (LSI), Edelweiss custom catalogues can be printed as easily as they are produced.[3] LSI is a short run and print-on-demand service based in the US but with the worldwide distribution power of Ingram. With Edelweiss handling digital catalogues and LSI being able to print them on demand, their partnership comes closest to a perfect solution. But with catalogue production handled largely out-of-office, the service does not grant publishers the ideal level of control.

Unfortunately, all of these platforms provide only partial solutions at best. Google Drive appears to be the best authoring option, but it offers no built-in methods to facilitate the transition to the websites and print catalogues. BNC CataList, as a natural extension of BiblioShare—which HPDM already uses—can be extremely useful, but offers no print options; meanwhile, Edelweiss and LSI, which do offer print options, don't afford as much control over the catalogue production process as we would like. Thus we are still left with a few missing pieces: integrating the HPDM websites into the workflow, bridging the useful elements of each platform and creating the print catalogue with the necessary level of control. Since the market offered no options to fulfill these roles, we decided a the ideal approach would be to create a custom solution.

3 Andrew Bromley, Ingram circular email, August 4, 2014. The partnership was announced in an email invitation to an informational webinar. We could not attend the webinar, and neither the LSI nor Above the Treeline websites mention the joint service, but we assume the service is still available.

CHAPTER 2 The Present

Developing an Alternative

IN DEVISING AN ALTERNATIVE, WE HOPED TO AUTOMATE AS MUCH OF THE workflow as possible, bring together each of the existing or available platforms, and limit the amount of human intervention required. To understand exactly how this system might look, the first thing we considered was the content stream: that is, where the content should originate, to what platforms it should flow, and in what order. Knowing where we want the stream to flow helped us understand how it could be done.

Content Streams

AS WE'VE ALREADY SEEN, THE EXISTING STREAM SEES CONTENT CREATED and edited in Microsoft Word, then copied to Adobe InDesign and finally copied again to HPDM's websites (see Figure 1). The Harbour website automatically generates the company's ONIX feed; meanwhile, the D&M ONIX is handled through an accounting system, and again much of the information needs to be copied. HPDM has no digital catalogue except for PDF versions of what is printed, exported right out of InDesign. As such, the InDesign files become, in a way, the central repository of information, but any changes or updates must be done individually in InDesign and each website. Even if the creation platform were changed to Google Drive, with its sharing and collaboration features, the content stream would be the same since Drive does not natively offer any dissemination options.

However, it is possible to use authoring software as an effective starting point for multi-platform output. A conversion utility called Pandoc can efficiently convert to and from a wide variety of formats, which in turn can be used to satisfy the requirements of several platforms: electronic and print platforms can be addressed at once, from the same source file.

This system was demonstrated by The Flying Narwhal project, a workflow experiment for magazines with which I was involved.[4] Using the The Flying Narwhal, users first transitioned content from Google Drive into HTML; then text was given tags corresponding to "title," "author" and so on to control how Pandoc processed each piece of the document. This marked-up version was then transformed into formats appropriate for print, web and iPad; for print, it was InCopy Markup Language, or ICML (see Figure 2). ICML is an XML file describing the text portion of an InDesign document, which can be Placed as a link into an InDesign file. It behaves much like an image: it must be edited externally, but any changes to the original file are updated in InDesign.[5]

The most important aspect of a content stream like The Flying Narwhal's is that output platforms—such as InDesign—were never used as intermediary platforms. Only the print version of the content ever went through InDesign; the web and iPad versions followed entirely separate routes.

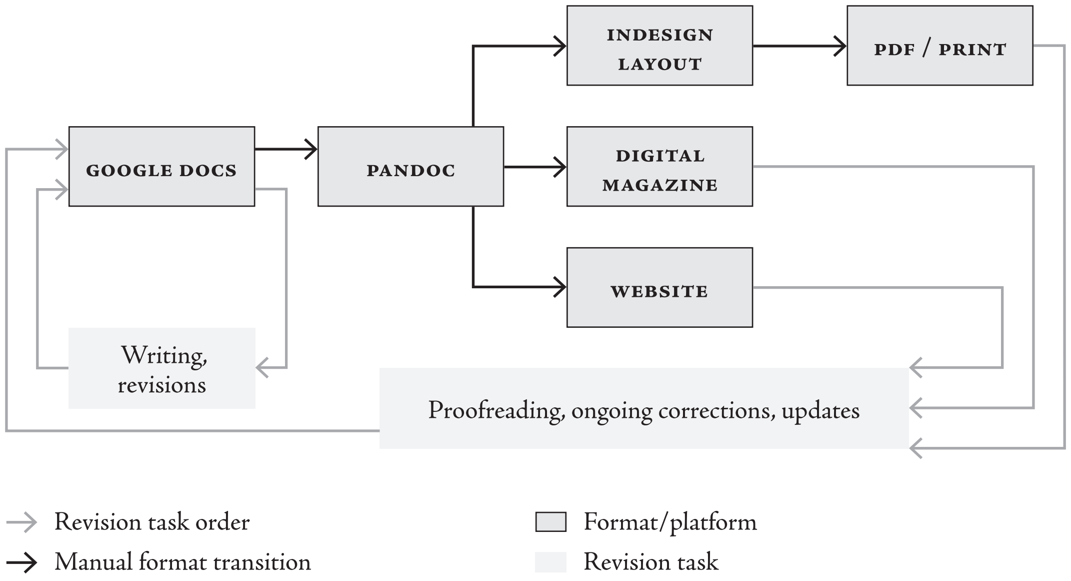
Unfortunately, most other aspects of The Flying Narwhal are incompatible with HPDM's requirements. For example, manually tagging each piece of bibliographic data in a Word document would be cumbersome, and a separate system would need to be developed to import the prepared data into the website. However, the Harbour CMS offers both creation and native dissemination to some extent. It seemed counter-intuitive, at first, to use such a complicated and limited platform as the starting point, but there were two major factors that made it an interesting candidate. The first was that, if the website is the starting point, there is little need to copy or reinterpret information. This is particularly important when it comes to basic data like publication dates, ISBN numbers and BISAC codes. The second is that the website already outputs its contents in a structured form: the ONIX feeds.

4 Till, Simas and Larkai. "The Flying Narwhal: Small mag workflow." http://tkbr.ccsp.sfu.ca/mpub/2014/04/14/the-flying-narwhal-small-mag-workflow.

5 It is in fact possible to edit content from an ICML directly in InDesign. However, it requires "checking out" the file; further external updates to the ICML will not carry through, thus compromising the system. It is usually not recommended except for proofreading corrections that affect no other platforms, performed at the very end of the process.

Structures are extremely important in allowing data to be effectively accessed, used and reused.

Figure 2. The Flying Narwhal workflow.

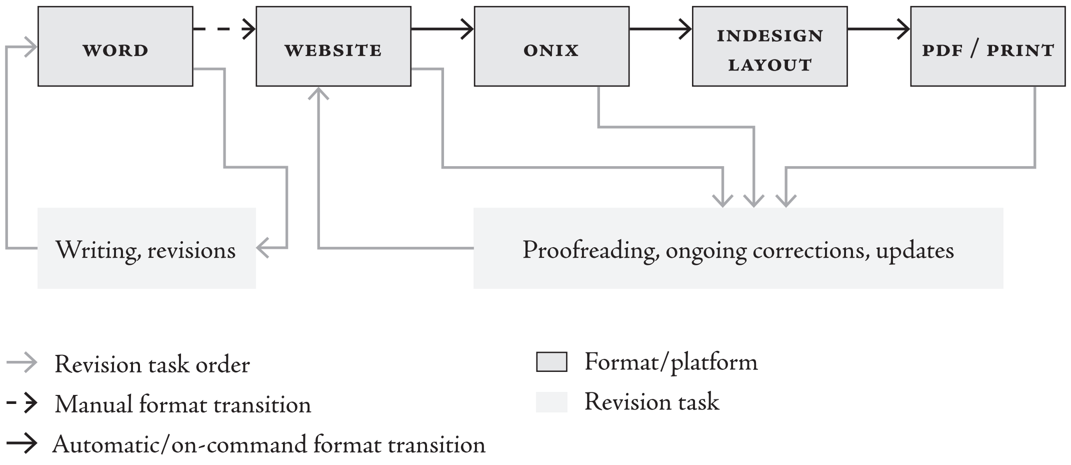


While The Flying Narwhal needed documents to be manually tagged and structured in order to be accurately transformed for every platform, the Harbour CMS already offered all the relevant fields. Data only needed to be accurately entered, not copied or labeled. And the daily ONIX message was already structured—nothing extra needed to be done to allow the data to be interpreted and reused.

Further, dissemination was built in. The visitor-facing portion of the website was updated automatically; the Harbour ONIX message was generated daily; and should HPDM adopt CataList or a similar service for its digital catalogues, the source would be the ONIX feed—and by extension, the website. One source fed one platform and most retailer websites; only print was left to be handled separately. And it would not be not difficult, at least in theory: both the ONIX message and the ICML format are based in XML. Transforming data from one XML format to another can be relatively easy.

It was clear that the website CMS should be the central repository of book data (see Figure 3). With most data entered directly into the website—leaving only the book description and similar fields to be created elsewhere first—the error margin is minimized. Then, using the website-generated ONIX to create the print catalogue ensures that all three output platforms are, at least, consistent with each other. Each of them can be proofread and, except for particular formatting issues, any correction made to the website carries through to every platform. This workflow can even survive changes to the website CMS itself, as long as the data is still presented in the ONIX feed in the same ways.

Figure 3. Proposed promotional copy workflow.



Of course, this option was not without its drawbacks, particularly when it came to author biographies and book descriptions. Those fields were still plain text boxes, with nothing to assist in styling or ensure that the HTML that is entered was valid. The problem of insufficiently trained production assistants remained. However, these were not exactly content stream issues; they could be fixed, or at least improved, by making relatively minor changes to the website CMS. For example, it would not be difficult to implement a WYSIWYG editor instead of the plain text boxes, which would make formatting much easier and leave a smaller margin for error.

Creation and collaboration, unfortunately, would not be made easier by using the website as the starting point. While it would be conceivable to have office staff write and edit content online, it was hard to imagine HPDM allowing its contracted editors, and especially authors, direct access to its website CMSs. Copy would still need to be written and edited externally, then copied onto the website. But this was the case only for the book description and author bios—nothing else needed to be sent out of the office—and good WYSIWYG editors are able to import basic formatting from Drive or Word. Thus, despite its limitations, it still appeared to be the most effective starting point for disseminating promotional copy. It left only the transition to print to be perfected.

As it was only the first experiment, we knew we could develop nothing more than a prototype. Given time constraints, we could not be able to develop every aspect of such a system at once, so we set ourselves some limitations. We would focus on the front-list titles only, since they are the most complex and most important; it should be relatively simple to later develop extra features to handle the various levels of reduced detail for the back-list. We would only ensure the user interface used to run the system was functional, rather than necessarily simple or attractive.

Unfortunately, all this only applied to the Harbour website, the only one that creates its own ONIX feed. However, as the two companies are increasingly better integrated, there are plans to eliminate divergences between the Harbour and D&M websites and have the D&M website managed by the same CMS as Harbour; eventually, D&M should match Harbour in how it produces its ONIX feed.

The ONIX Structure

A STANDARD ONIX MESSAGE INCLUDES AN EXTENSIVE SET OF METADATA about each title it describes. Harbour uses the more human-readable "Reference names" variant of the somewhat outdated ONIX 2.1 release, but even so the ONIX message still uses plenty of coded data. For example, Contributor Role A01 represents the primary writers of a work.[6] In order to extract the information we needed, it was necessary to understand how ONIX is structured and identify the relevant portions of the document. To this end, we consulted HPDM's Spring 2015 catalogues, the most recent catalogues available at time of writing, and referred to the ONIX specification and code lists provided by EDItEUR, the international group dedicated to book industry electronic infrastructure that releases and maintains the ONIX standard.

A complete ONIX Message includes a record of every book ever published or distributed by the transmitting company. It can be a very long document, and much of the information it contains is only useful to a seller. The ONIX record also includes several optional tags, and bits of data that can be expressed in multiple ways, so an in-depth knowledge of the titles it describes was useful. At the ONIX document root are the standard XML declaration, which specifies the language version and encoding, and the `<ONIXMessage>` tag, which encompasses the entire contents of the report.[7]

For each title recorded, there is a `<Product>` tag to group all of the information about that particular book. For the purposes of this analysis, we focused on *Raincoast Chronicles 23* (*RC23*), the newest edition of the serial-turned-book that Harbour was founded on, to understand the data that was relevant to catalogues. (Refer to the "Appendix: Sample ONIX" for the complete ONIX record for *RC23*.) The catalogue elements we needed to identify included:

» Title and subtitle;
» Author(s) and other contributors, and their brief biographies;
» Tagline and main book description;
» Genre;
» Price;

6 EDItEUR, "ONIX Codelists Issue 27."

7 All ONIX excerpts presented in this paper were taken from Harbour ONIX messages produced between December 2014 and January 2015.

» Publication month and year;
» Format;
» Width, height and page count;
» Sales rights;
» Print and ebook ISBNs.

The title was fairly straightforward; the `<Title>` tag lists the title and subtitle with no hierarchy or coded data.

RC23 is essentially an anthology of excerpts, so the writers were not individually listed in the ONIX. Instead, editor Peter A. Robson was the primary contributor, and Howard White was listed for his introduction. Next to the contributor names, Contributor Role is the most important piece of data: it describes the contributor's specific involvement in the title in question. For example, code B01 indicated *RC23* was "Edited by" Robson.[8]

In the ONIX, contributors' names are presented repeatedly, in multiple ways—in natural order (given names followed by surnames), inverted (surnames first) and with given and surnames in separate tags. On the front-list catalogue pages the contributor names are given in natural order, so this was the format we focused on. Others may prove useful for alphabetic organization at a later date.

The last bit of important contributor information was the Biographical Note. This was one of the tags taken from the tricky plain text boxes on the Harbour CMS (see page 9). The contents of the `<BiographicalNote>` tag were presented exactly as entered: XHTML tags like `<p>` and `<em>` may not be used in every instance.

The tagline and the book description were the other two fields that faced this same problem. These fields were also harder to identify since, despite the fact that they are comonly used in book publishing, the ONIX specification only includes them in the generic `<OtherText>` tag, using only different type codes to distinguish between them. In HPDM catalogues, the tagline is a

8 EDItEUR, "ONIX Codelists Issue 27." All ONIX codes in this paper refer to this document.

short, eye-catching line or paragraph shown near the top of the page, before the main description. This can be a brief description, a quote from the book or a review blurb, but not every book has one. *RC23* did not; but we could look at *Cam Tait: Disabled? Hell No! I'm a Sit-Down Comic!* for its tagline (see Snippet 1). The ONIX code list does not identify any type codes for "tagline," so the Harbour CMS uses type code 02 "Short description/annotation."

Snippet 1. ONIX representation of the tagline.

```
<OtherText>
  <TextTypeCode>02</TextTypeCode>
  <Text textformat="02">
    <p>"I have cerebral palsy much like I have
    blue eyes…"</p>
  </Text>
</OtherText>
```

The book's genres were probably the most difficult to understand because of how they are coded in ONIX 2.1. There are two relevant subject tags: `<BASICMainSubject>` and `<Subject>`. Of the two, `<Subject>` had the more complex structure, listing both the code scheme—HPDM always uses BISAC—and the code itself; but `<BASICMainSubject>` was but more important. The odd name for the tag is an artifact of when BISAC subject headings were called BASIC, and the name cannot be changed.[9] Since the tag name identifies the scheme, all it contained was the BISAC code.

Since different distributors can offer the same book for different prices, the `<Price>` tag was not by itself but nested inside the `<SupplyDetail>` composite, which included the supplier, the price and other distribution data. Only prices listed with Harbour Publishing as the supplier matter to Harbour catalogues.

9 EDItEUR, "ONIX Product Record Format," 77.

The remaining information was much easier to identify, although we still relied on the code lists for many of them. In HPDM catalogues, only the release month is listed for most books. ONIX carried the complete Publication Date, from which we could extrapolate the month. The book format—paperback, cloth or accordion-folded pamphlet—was identified by the `<ProductForm>` tag. Width and height were both given in `<Measure>` tags using different Measure Type Codes, while page count was given simply in the `<NumberOfPages>` tag. Equally straightforward was the Illustrations Note, which indicates whether a book includes any illustrations or photographs, and occasionally how many. The `<SalesRights>` tag usually included `<RightsTerritory>`, when the territory was "World," but `<RightsCountry>` replaced it if the company had only Canadian rights.

Finally, the HPDM catalogues list the print and ebook ISBNs for each title. ISBNs were found in the `<ProductIdetifier>` tag. Unfortunately, the Harbour ebook program is fairly new, and the ONIX does not include any ebook records, or mentions ebooks at all.

One part of the ONIX record was relevant to the catalogue but could not be used: the cover image. Most front-list books do not have covers ready at the time the catalogue is produced, so placeholders are selected instead. These images never go on the website or ONIX. Further, the cover files that are eventually posted online are a much lower resolution than ideal for print, at least at the size required for front-list titles. It was best to simply add images directly in InDesign.

Finally, the ONIX record did include bits of information that, although they are not shown in the catalogue, could be useful as selection criteria for the system, like the publisher and imprint. D&M has no imprints, while Harbour only has one (Lost Moose); however, Harbour distributes Nightwood Editions, which includes their blewointmentpress imprint. Other useful data included the copyright year and product availability, both found in tags with straightforward names.

The ICML Structure

In Adobe's software suite, ICML is produced by InDesign's sister program, InCopy. InCopy files are used to control the text of a project independently from the design. In our system, the ICML can be Placed in InDesign, but it cannot edited directly; rather, any change to the website can be quickly carried through to InDesign by overwriting the ICML.

An ICML file only concerns itself with what InDesign calls a Story—any piece of continuous text. When text from one page or text box overflows into another, it is a continuation of the same Story. For our purposes, it would have made sense for our catalogue to have each as a separate Story since the information about each front-list title is independent from the others. However, an ICML document can only contain one Story. We decided that, to avoid having to create a multitude of ICML files, which would have needed to be individually placed, we would run all the titles together in a single story, and thus a single ICML file.

To ensure our output was valid and harnessed as much flexibility as ICML allows, we had to understand how the ICML format is structured. Adobe offers extensive and detailed documentation of its XML formats,[10] but the document is very dense. Thus most of our understanding was gleaned from examples—valid ICML files generated through The Flying Narwhall and Ickmull.[11]

In an ICML file, after tags specifying the document format and its the role of the file within the InDesign environment, the `<Document>` element encompasses the body of the ICML file. This includes two main sections: style definitions and the Story. Only the Story is required, but any paragraph or character styles, colours, fonts and so on must be defined before they can be used in the Story.

10 Adobe, "IDML File Format Specification."

11 Ickmull converts HTML from WordPress posts into ICML by performing an XML-based transformation. Their method as similar to the one we chose, so Ickmull served as a reference both for the ICML output and for the transformation. Maxwell, "Ickmull Project Summary."

A complete and functional InDesign template was also required for this prototype. An ICML file can carry specific formatting instructions like font and type size, but this is not ideal. Not only would it have significantly increased the length of the document, it would have made it difficult to change the design of future catalogues. It was best to keep structure and design separate; however, our ICML did have to ensure the visual styling options were correctly applied. Titles needed to be styled differently from subtitles, for example; and the first paragraph of each book description or author biography needed to be given a specific, unindented style. To this end, the ICML could simply name the styles that should be applied to the text, and the definitions would be kept in the InDesign template. If the ICML named a style that the template did not contain, it would be added to the document when the file was Placed, allowing it to be formatted after the fact.

In the ICML file, we first had to name the character styles. These affect only specific characters, and override the paragraph styles; they can also be automatically applied by paragraph styles and, in some cases, combined. For example, small caps and italics were character styles. As Snippet 2 shows, the `<RootCharacterStyleGroup>` element collects all `<CharacterStyle>` tags, which in turn define the character styles. Since we were not assigning any formatting to the styles, all we needed were the Self and Name attributes. Self is a unique identifier, which we used later to apply the style; Name is what it should show up as on the InDesign user interface. Name was important because it needed to match the template definitions—any discrepancies, including capitalization, prevented the style from being matched to a definition in the template.

Snippet 2. Setting up character styles in ICML.

```
<RootCharacterStyleGroup
Self="onix2icml_character_styles">
  <CharacterStyle Self="CharacterStyle/Italic"
  Name="Italic"/>
  <CharacterStyle Self="CharacterStyle/SmallCaps"
  Name="Small Caps"/>
  …
</RootCharacterStyleGroup>
```

Paragraph styles work in nearly the same way, except that they are applied to the paragraph as a whole, and cannot overlap. We wanted to have one style for each bit of information that would be styled differently, so that they could be formatted independently. Again we had `<RootParagraphStyleGroup>` holding everything else; but since we had so many paragraph styles, we divided them into groups using `<ParagraphStyleGroup>` tags to keep them organized (see Snippet 3). As with the character styles, Self and Name were the only required properties.

Snippet 3. Setting up paragraph styles in ICML.

```
<RootParagraphStyleGroup
Self="onix2icml_paragraph_styles">
  <ParagraphStyleGroup Self="ParagraphStyleGroup/
  Frontlist" Name="Frontlist">
    <ParagraphStyle Self="ParagraphStyle/
    Frontlist/Title" Name="Title"/>
    …
  </ParagraphStyleGroup>
  <ParagraphStyleGroup Self="ParagraphStyleGroup/
  Specs" Name="Specs">
    <ParagraphStyle Self="ParagraphStyle/
    Frontlist/Specs/Genres" Name="Genres"/>
    …
  </ParagraphStyleGroup>
</RootParagraphStyleGroup>
```

The `<Story>` element makes up the bulk of the ICML document, so it was the elements we had to pay the most attention to in making sure all of our data showed up correctly in InDesign. The `<Story>` includes all of the visible text, plus tags that assign the wanted styles (see Snippet 4).Within it goes the tags that apply the paragraph styles, `<ParagraphStyleRange>`; and within that we applied the character styles with `<CharacterStyleRange>`. When there were no character styles to apply, we used the InDesign default "[No character style]," which did not need to be defined. At the centre was the `<Content>` element, wrapping the text itself. All of these elements were required; if any were missing, the text would not show up in InDesign.

`<ParagraphStyleRange>` elements are repeatable within the Story, while `<CharacterStyleRange>` is repeatable within the paragraphs. A new `<CharacterStyleRange>` tag needed to be used every time the style changes, whether it is to one of the defined styles or back to the default [No character style]. At the end of each `<ParagraphStyleRange>` element, the break tag `<Br/>` told InDesign to insert a standard carriage return and create a new paragraph.

Snippet 4. Structure of the Story tag.

```
<Story>
  <ParagraphStyleRange AppliedParagraphStyle="Para
  graphStyle/Frontlist/ParaNoIndent">
    <CharacterStyleRange AppliedCharacterStyle="Ch
    aracterStyle/[No character style]">
      <Content>When the first edition of </Content>
    </CharacterStyleRange>
    <CharacterStyleRange AppliedCharacterStyle="Ch
    aracterStyle/Italic">
      <Content>Raincoast Chronicles</Content>
    </CharacterStyleRange>
    …
    <Br/>
  </ParagraphStyleRange>
  …
</Story>
```

The XSL Transformation

WITH A PROPER UNDERSTANDING OF THE DATA WE NEEDED FROM ONIX and how it should be represented in the ICML, with proper separation of content and design, we could finally approach the conversion itself. The best option was to use an Extensible Stylesheet Language (XSL) file to automate the transformation (know as XSLT). An XSL document describes how data from the original XML file, in this case the ONIX feed, should be reformatted and used in the output file, the ICML.[12] This is done by identifying data patterns and describing associated behaviours. For example, the XSLT can define that, for each `<ContributorRole>` tag with a value of B01, the output file should read "Edited by." But the possible transformations go far beyond that, and can be much more complex. It was important to understand not only the technical way in which XSLTs function—including XPath, a language XSL relies on to identify the locations of elements within the source document[13]—but the various ways the source data can be transformed and reinterpreted.

The accuracy of our transformation hinged on two main factors: that the system could pick up all of the relevant information, and that it could present this information with all the necessary formatting and accompanying labels. For example, the width and height needed to be presented accompanied by an inch sign (double prime) and separated by a dimension sign. However, these symbols were nowhere in the data itself. Thus, the system had to copy the width and height data, add the appropriate symbols then apply the ne-

12 W3Schools, "XSLT Introduction."

13 W3Schools, "XSLT Introduction."

cessary styles. Or, in the case of the contributors, the system needed to be able to identify every contributor to a book and list their names with their appropriate roles identified. If a book had an author and a foreword writer, for example, they needed to be differentiated: "Author Name, with a foreword by Foreword Writer."

Besides the opening tags that define the XSL file's properties, an XSLT is composed of templates, which dictate how the contents of an XML file should be modified. Templates can be simplified into two elements: the match criteria and the behaviour. Whenever the XSLT engine finds something that meets the match criteria, it executes the behaviour. The main template for our purposes matched the element `<ONIXMessage>`—that is, the entire document. This template set up the ICML basics—including the paragraph and character styles and the beginning of the Story—then called a sequence of other templates to parse the ONIX file and populate the Story with the appropriate information (see Snippet 5). The Select attribute of the `<xsl:for-each>` tag specified that the following templates would be applied to each Product element in the ONIX file where the Publication Date is in 2015. Since this was hard-coded into the template at first, it could not be changed by the user—we will discuss selection criteria later on. Then, each `<xsl:apply-templates>` element included a Select attribute that specified what tags it would be applied to. For example, every time it found a Title tag, it executed all the templates that matched the `<Title>` tag or any elements within it. Applying templates in this way gave us control over the order in which the data was output, since it did not show up in the ONIX in the same way we needed it for the catalogue.

Snippet 5. Template used to select what parts of the ONIX file to process.

```
<xsl:template match="ONIXMEssage">
  …
  <xsl:for-each select="Product[PublicationDate >
  20150000]">
```

```
        <xsl:apply-templates select="Title"/>
        <xsl:apply-templates select="Contributor[1]"/>
        <xsl:apply-templates
        select="OtherText[TextTypeCode = '02']"/>
        …
      </xsl:for-each>
    </xsl:template>
```

Some of those Select attributes contained more than just the name of the relevant tag. For example, "`select="Contributor[1]"`" meant it would only apply the template to the first contributor element. Since there were multiple contributors to some books, but we wanted them all to be processed together, matching only the first one prevented the template from being applied repeatedly. Similarly, "`select="OtherText[TextTypeCode = '02']"`" only matched `<OtherText>` tags where its TextTypeCode element was equal to 02—that is, the tagline. If no further specification was given, it was because there was no chance of error in the ONIX file at hand.

We identified five different types of templates that we needed to get accurate results. The simplest were those where little or no processing was required—the data only needed to be copied. Then there were sets of data that needed to be translated, from their coded representation in ONIX to a more human-readable form. Some of these were simple one-to-one relationships, while others, more complex, required multiple steps and conditional translations. Some data needed to be treated differently depending on its relation to other data points. Finally, to preserve formatting like italics within the description fields, we had to create an entirely separate file so that those could be processed independently. We will discuss each template in turn.

No Processing Required

The templates for Title, Subtitle and IllustrationsNote were the simplest, because the system only needed to repeat the information it found. In Snippet 6 (see page 31), every time the engine found a `<TitleText>` element in the ONIX message, it output the appropriate `<ParagraphStyleRange>` and `<CharacterStyleRange>` tags. `<xsl:value-of select=".">` instructed the engine to copy the contents of the TitleText it matched, placing the book's title within the `<Content>` tag—the period refers to the current element. The template for Price was almost the same, except that it added the currency code "CAD" and a dollar sign before the tag's value.

Snippet 6. The Title template.

```
<xsl:template match="TitleText" >
  <ParagraphStyleRange AppliedParagraphStyle="Para
  graphStyle/Frontlist/Title">
    <CharacterStyleRange AppliedCharacterStyle="Ch
    aracterStyle/[No character style]">
      <Content><xsl:value-of select="." /></
      Content>
    </CharacterStyleRange><Br />
  </ParagraphStyleRange>
</xsl:template>
```

Simple Equivalencies

Slightly more complex were those templates that needed to define equivalencies. In some cases this was a simple matter of matching the code to the related term, such as the one for ProductForm (see Snippet 7). The basics were the same as the no-processing templates, but the `<xsl:choose>` element allowed for different behaviours. For example, when the value of the Product Form in question was "BB," the template filled the `<Content>` tag with "Cloth." The templates for Sales Rights and Publication Date were very similar, except that the latter did not check the entire contents of the matched element, but only the digits that identified the month.

Snippet 7. The Product Form template.

```
<xsl:template match="ProductForm">
  <ParagraphStyleRange AppliedParagraphStyle="Para
  graphStyle/Frontlist/Specs/OtherSpecs">
    <CharacterStyleRange AppliedCharacterStyle="Ch
    aracterStyle/[No character style]">
      <Content>
        <xsl:choose>
          <xsl:when test=". = 'BB'">Cloth</
          xsl:when>
          <xsl:when test=". = 'BC'">Paperback</
          xsl:when>
          <xsl:when test=". = 'BF'">Pamphlet</
          xsl:when>
        </xsl:choose>
      </Content>
    </CharacterStyleRange><Br />
  </ParagraphStyleRange>
</xsl:template>
```

The next step up was the Measure template. In the HPDM catalogues, the height, width and page count are given in a single line. However, if the title in question is a pamphlet, the page count is replaced with "8-fold," since the pamphlets are accordion-folded and don't follow a traditional book pagination. Because of this, the Measure template had to check the value of `<ProductForm>` to determine how to present the data.

ISBNs posed a different problem entirely, which we addressed with a two-step template. HPDM catalogues differentiate between print and ebook ISBNs by attaching the format at the end, in parentheses, so the ISBN template applied nearly the same transformation as the Product Form template above. However, the ISBN is shown hyphenated in HPDM catalogues, while in the ONIX it is not, so we had to insert the hyphens. Fortunately, all recent

HPDM books follow a standard hyphenation pattern. Thus, it was only a matter of parsing the ISBN to insert hyphens at the correct intervals. We called a separate template for this, so that this hyphenation system could be used by both print and ebook ISBN templates without having to be coded twice. The ebook ISBN may not be included in the ONIX yet, but will be in the future.

Multi-Level Equivalencies

The biggest transformation challenges came from the contributor names and the genres. These were still essentially equivalency templates, but they were very complex and required multiple steps. In the case of genres, the problem was variety. To begin with, there are fifty top-level BISAC subject headings, such as Fiction, Art and Science. Thankfully, not all fifty are likely to be used by HPDM, and in most cases this top-level heading may provide enough detail. Most of the time, the genre listed in the catalogue is a subject heading—or is closely related to one, such as the genre "Children's" and the BISAC headings Juvenile Fiction and Juvenile Nonfiction.

Since there are too many genre possibilities to predict them all, a direct one-to-one translation, as was done for Product Form, for example, was impossible. But we could deal with many of them by breaking the process up into parts. We began by listing those genres that matched exactly to general BISAC headings. The subject codes follow the format AAA000000, where the first three letters are the heading code (see Snippet 8). The Substring function above captured the first three characters of the code; if those characters were "ART," it wrote "Art," and so on. We only needed to list the subject headings that were likely to be used by HPDM.

Snippet 8. Matching BISAC Subject Headings with the Genre template.

```
<xsl:choose>
  <xsl:when test="substring(.,1,3) = 'ART'">Art</
  xsl:when>
```

```
 …
</xsl:choose>
```

But there are some cases where the genre is more specific. The most common ones were Memoir, Short Fiction and First Nations. "Personal Memoir" is found in the Biography & Autobiography category; and both "Short Stories (single author)" and "Anthologies (multiple authors)" are found under Fiction.

The First Nations subject is more complicated, as it variously falls under multiple categories. The "Native American" subject—BISAC's closest equivalent to First Nations—is found under Art, Fiction and History, among others. All of these possible BISAC codes had to converge under the First Nations genre in the catalogue.

Another complicating factor was that occasionally two genres are listed in the catalogue, usually First Nations or Regional Interest being one of them. It was important that the system be able to tell when to print only one subject and when to print two.

We took care of all these issues by adding to our template a list of specific subject codes and their corresponding genres, and giving them precedence over the generic headings (see Snippet 9). Here we checked for matches to the entire code. Note that we placed the `<xsl:choose>` for the generic headings within an `<xsl:otherwise>` tag, meaning that the system would only use those headings if it did not match one of the specific codes above.

Snippet 9. Matching BISAC Subject Codes with the Genre template.

```
<xsl:choose>
  <xsl:when test=". = 'ART041000'">Art / First
  Nations</xsl:when>
  …
  <xsl:otherwise>
    <xsl:choose>
```

```
          <xsl:when test="substring(.,1,3) =
          'ART'">Art</xsl:when>
          …
        </xsl:choose>
      </xsl:otherwise>
    </xsl:choose>
```

However, we were still left with some problematic cases where the BISAC codes assigned to the book did not accurately reflect the desired genre. This problem could have arisen if the production assistant creating the book record on the CMS made poor BISAC choices; but more likely is that there were simply no BISAC codes that corresponded to the genres we wanted to show in the catalogue. Regional Interest stands out: it is a very common genre at Harbour, but there is no BISAC equivalent. Part of the problem is that Regional Interest is not region-specific; it simply refers to any title that is of particular interest to one community.

RC23, for example, is largely targeted at the west coast of British Columbia, where the Raincoast Chronicles serial began; but the BISAC Regional Themes code list does not include the West Coast as a whole. Thus, production assistants were forced to assign BISACs that did not reflect the "Regional Interest" genre. *RC23* has Canadian History as its main BISAC and Canadian Fiction as a secondary subject. These are both accurate, to a certain extent, but *RC23* is neither entirely a history book nor a fiction book—but it is entirely a regional, West Coast book.

It was impossible for the system to distinguish Regional Interest titles based simply on their BISAC codes; there were no indications in the data for when a book with a History subject code should be labelled "Regional Interest" instead of "History," for example. With this in mind, we hard-coded the genre based on the ISBN (see Snippet 10). This was not ideal since it meant that changing the BISACs would not update the genre, but it allowed Regional Interest to be applied accurately. Again we had the problematic cases, defined by ISBN, taking precedence over the specific subjects, which were now nested in their own `<xsl:otherwise>` element. Of course, the list of problematic cases will grow significantly with each season, and will likely need to me amended frequently.

Snippet 10. The complete Genre template, including individual books matched by ISBN.

```
<xsl:template match="BASICMainSubject">
  <ParagraphStyleRange AppliedParagraphStyle="Para
  graphStyle/Frontlist/Specs/Genres">
    <CharacterStyleRange AppliedCharacterStyle="Ch
    aracterStyle/[No character style]">
      <Content>
        <xsl:choose>
          <xsl:when test="../
          ProductIdentifier[ProductIDType = '15']/
          IDValue = '9781550177107'">Regional
          Interest</xsl:when>
          …
          <xsl:otherwise>
            <xsl:choose>
              <xsl:when test=". =
              'ART041000'">Art / First Nations</
              xsl:when>
              …
              <xsl:otherwise>
                <xsl:choose>
                  <xsl:when
                  test="substring(.,1,3) =
                  'ART'">Art</xsl:when>
                  …
                </xsl:choose>
              </xsl:otherwise>
              </xsl:choose>
            </xsl:otherwise>
          </xsl:choose>
        </Content>
      </CharacterStyleRange><Br />
    </ParagraphStyleRange>
  </xsl:template>
```

Context-Dependent Processing

Although the contributor line had less diversity of input options, the range of output formats was extensive. In this case, the way the data should be presented depended on more than just the data itself. When the only contributor was the writer, the name was listed by itself. But if the single contributor was a photographer, as in the case of *Vancouver Light*, the contributor line had to specify "Photographs by David Nunuk." When there were multiple contributors, all their names were given on the same line, but how they should be written out depended on their roles. For example, *RC23* should read "Edited by Peter A. Robson, with an Introduction by Howard White." The picture book *Orca Chief* was written jointly by two people, and so the contributor line should read "Roy Henry Vickers and Robert Budd." The two authors of *Cam Tait*, though, needed to be given different treatment—"Cam Tait with Jim Taylor"—since Taylor's role was more in helping Tait tell his story. Being able to handle all the varieties of contributor lines was crucial for the system's accurate translation of the data.

As a result, the contributor line was handled by one of the most complex templates in our XSLT. It would have been easy to simply repeat the name of every contributor with a comma between them, for example, but we needed much more than that. Switching the last comma for the word "and" solved part of the problem (see Snippet 11): it inserted the contributor's name, then checked what position the current `<Contributor>` tag was in relative to others in the same `<Product>`. When it was the last or only contributor, it did nothing; when it was second-to-last, it added "and" after the name; for any other position it added a comma.

Snippet 11. Adding "and" between second-to-last and last contributor names.

```
<xsl:for-each select="../Contributor">
  <xsl:value-of select="PersonName" />
  <xsl:choose>
    <xsl:when test="position() = last()" />
    <xsl:when test="position() = last()-1"> and </
    xsl:when>
    <xsl:otherwise>, </xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
```

Unfortunately, this still did not differentiate between contributor roles, so we expanded on it. We added an `<xsl:choose>` selector before the contributor's name attached a brief descriptor depending on the contributor's role (see Snippet 12).

Snippet 12. Adding role descriptors with the Contributor template.

```
<xsl:choose>
  <xsl:when test="ContributorRole[1] =
  'B01'">edited by </xsl:when>
  <xsl:when test="ContributorRole[1] =
  'A12'">illustrated by </xsl:when>
  …
</xsl:choose>
```

By itself, though, this added the descriptor to each contributor, which could get messy when there were multiple contributors with alternate roles, such as co-editors. To correct this, we wrapped it in an `<xsl:if>` element (see Snippet 13). The `<xsl:if>` element defined that the role descriptor would only be attached when it was the first contributor, and when the role was different from that of the preceding contributor.

Snippet 13. Checking for repeated roles with the Contributor template.

```
<xsl:if test=”position() = 1 or ContributorRole[1]
!= preceding-sibling::Contributor/
ContributorRole[1]”>
  <xsl:choose>
    <xsl:when test=”ContributorRole[1] =
    ‘B01’”>Edited by </xsl:when>
    …
  </xsl:choose>
</xsl:if>
```

Putting the pieces together, we created the following template shown in Snippet 14. Note that we were also using the word “and” after the contributor name when the next contributor had the same role as the current one. The only remaining issue was that we could not predict every possible role that may become relevant, so we may need to add more role translations as the system is used.

Snippet 14. The complete Contributor template.

```
<xsl:template match="Contributor[1]">
  <ParagraphStyleRange AppliedParagraphStyle="Para
  graphStyle/Frontlist/AuthorName">
    <CharacterStyleRange AppliedCharacterStyle="Ch
    aracterStyle/[No character style]">
      <Content>
        <xsl:for-each select="../Contributor">
```

```
            <xsl:if test="position() = 1
            or ContributorRole[1] != pre-
            ceding-sibling::Contributor/
            ContributorRole[1]">
              <xsl:choose>
                <xsl:when test="ContributorRole[1] =
                'B01'">Edited by </xsl:when>
                …
              </xsl:choose>
            </xsl:if>
            <xsl:value-of select="PersonName" />
            <xsl:choose>
              <xsl:when test="position() = last()"
              />
              <xsl:when test="position() = last()-
              1 and ContributorRole[1] = fol-
              lowing-sibling::Contributor/
              ContributorRole[1]"> and </xsl:when>
              <xsl:otherwise>, </xsl:otherwise>
            </xsl:choose>
          </xsl:for-each>
        </Content>
      </CharacterStyleRange><Br />
    </ParagraphStyleRange>
  </xsl:template>
```

Processing Inline Styles

Attempting to process the three fields that are written in HTML—the tagline, the main description and the contributor biographical notes—brought the issue of poor HTML standards on the website back to the forefront. Each of these was likely to have multiple paragraphs as well as italics, bold and other formatting. Although any HTML in these fields is rendered on the front-end of the website, the program that generates the ONIX file escapes the HTML, rendering it inactive, and processes it all as plain text. For example, the essential angle brackets, < and >, are converted to "`&lt;`" and "`&gt;`" respectively. Escaping prevents the HTML from interfering with the ONIX structure itself, while retailer websites are able to unescape the HTML and render it correctly. If included in the ONIX as XHTML, the bad tags would make the document essentially unintelligible to any system trying to read it since, in the past, so many tags have been left unclosed. However, including them as plain text also prevented us from being able to use the XSLT to apply the correct paragraph breaks and character styles.

To be able to do that, we needed to unescape the HTML and make it valid as XHTML again. However, simply unescaping the HTML did not make it processable as XML, so we saved the unescaped result into a separate file. When the new file was loaded, it read as valid XML. In our XSLT, then, we added templates like the one in Snippet 15. The `<Description>` tag acts as a simple identifier to make the data easy to process later. The Normalize Space and Disable Output Escaping functions ensured the output was properly unescaped. After we ran the first transformation, we needed to run another on the result. For this, we needed a second XSL file.

Snippet 15. Unescaping the Description field.

```
<xsl:template match="OtherText[TextTypeCode =
'01']">
  <Description>
    <xsl:value-of select="normalize-space(./Text)"
    disable-output-escaping="yes" />
  </Description>
</xsl:template>
```

Unfortunately, here we hit another obstacle. If the original HTML presented errors, our interim file would, too, now that the HTML was unescaped.

This caused an error and prevented the second XSLT from being processed. It would be impractical to individually fix every HTML problem, so we searched for an automated solution. We decided it was best to modify the Perl program that creates the ONIX. It already included some text processing, and this way we would prevent the website from being affected.

Some of the most common HTML problems included unclosed tags; inconsistent use of capital letters in tags; using the HTML flavour of a tag rather than the XHTML variant;[14] and a variety of ways to create paragraphs. Some paragraphs were defined properly, using `<p>` tags, but others were separated simply by typing empty lines, or using two break tags, and occasionally a combination. This inconsistency made it difficult to for our transformation to accurately identify where one paragraph ended and another began.

Of all these issues, however, only the first could not be fixed in the ONIX Perl program. It was impossible to automatically know where a tag like `<i>`, which indicates italics, was supposed to end. But it was possible to change all XHTML tags to lowercase, swap tags to their XHTML version, and trade all the likely ways of marking paragraphs for complete `<p>` tags. This still left some room for error, but those problems could be fixed manually on the website when they were identified.

With the revised ONIX file, out first transformation yielded—usually—valid results, so we proceeded with our second XSLT. We first needed to copy the entire contents of the original file, then apply any appropriate templates (see Snippet 16). The "`match="@*|node()"`" portion ensured this template acted on every element of the interim file. Together, `<xsl:copy>` and `<xsl:apply-templates>` duplicated the previous iteration while checking if any elements required further processing. Since this XSL file only included templates for processing the Tagline, Description and Author Bio, everything else was left unchanged.

14 In HTML, some tags can be left unclosed without causing trouble, most notably the line break tag `<br>`. In XML and XHTML, every tag needs to be closed; in case case of line breaks, the correct tag is `<br/>`, which is self-closing (opening and closing tags are one).

Snippet 16. Template for copying a file while making further changes.

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

Within those three tags, however, the next step was to identify the paragraph breaks, and apply the appropriate paragraph styles (see Snippet 17). In HPDM catalogues, the first paragraph of the main description and of each author bio is given a special, unindented style, while all paragraphs thereafter are indented. The template above applied a different Paragraph Style to the first paragraph. Note also that this template did not add the `<CharacterStyeRange>` and `<Content>` tags: since there may be multiple character styles in each paragraph, we could not have the entire paragraph wrapped in a single `<CharacterStyleRange>`.

Snippet 17. Template for applying paragraph styles.

```
<xsl:template match="Description">
  <xsl:for-each select="p">
    <xsl:choose>
      <xsl:when test="position() = 1">
        <ParagraphStyleRange AppliedParagraphStyl
        e="ParagraphStyle/Frontlist/ParaNoIndent">
          <xsl:apply-templates select="./node()"/>
          <Br/>
        </ParagraphStyleRange>
      </xsl:when>
      <xsl:otherwise>
        <ParagraphStyleRange AppliedParagraphStyl
        e="ParagraphStyle/Frontlist/ParaIndent">
```

```
          <xsl:apply-templates select="./node()"/>
          <Br/>
        </ParagraphStyleRange>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:template>
```

The template in Snippet 17 also called for further templates to be applied within each `<p>` tag. Those were the templates that applied the inline styles. The template in Snippet 18 applied the [No character style] when the text was not wrapped in any other tags. We had a similar template that applied the "Italics" style whenever the text was wrapped in `<i>` or `<em>` tags, and another that applied "AuthorName" when it was wrapped in `<b>` or `<strong>`.[15] The second part of the above template constructed `<CharacterStyleRange>` with the attributes requested in the first part, and added the `<Content>` tag; this was actually used by every other character style template. As with the ISBN hyphenation template, this reduced repetition.

Snippet 18. The two parts of the Character Style template.

```
<xsl:template match="Tagline/p/
text()|Description/p/text()|AuthorBio/p/text()">
  <xsl:call-template name="CharacterStyle">
    <xsl:with-param name="StyleName">[No character
    style]</xsl:with-param>
  </xsl:call-template>
```

15 In the catalogue, bold text is only used as a design choice, including as a way to set the author's name apart in their biographical note. This bolded style is applied automatically in InDesign, but since not all author names follow a predictable pattern, it may be necessary to specify what part of the note is the author's name. For this, we can repurpose the `<b>` and `<strong>` tags.

```
</xsl:template>
…
<xsl:template name="CharacterStyle">
  <xsl:param name="StyleName"/>
  <CharacterStyleRange>
    <xsl:attribute name="AppliedCharacterStyle">
      <xsl:value-of select="concat('CharacterSty
      le/', $StyleName)"/>
    </xsl:attribute>
    <Content><xsl:value-of select="."/></Content>
  </CharacterStyleRange>
</xsl:template>
```

Most other formatting, such as web links, are unnecessary in the print catalogue, so we used a template exactly like the one above to add [No character style]. We kept them separate since the purpose of the template is different—to simplify the code, rather than to reinterpret it—and it is possible that, in the future, we may want to apply special formatting to these tags after all.

Finally, since most paragraph breaks were converted to the standard `<p>` tag, we could preserve intentional breaks. We simply replaced each `<br/>` tag with the code that InDesign understands as a forced line break character (see Snippet 19).

Snippet 19. Preserving forced line breaks.

```
<xsl:template match="br">
  <CharacterStyleRange>
    <Content>
      <xsl:text>&#x2028;</xsl:text>
    </Content>
  </CharacterStyleRange>
</xsl:template>
```

The PHP Implementation

WITH THE SOURCE, THE PROCESS AND THE DESIRED OUTPUT IN PLACE, WE turned to the engine that would make it all happen. There are multiple ways to perform an XSL transformation, but PHP stood out as the most suitable in this case. PHP is a web programming language that acts on the server, and presents the processed file to the user. This meant that the specifics of the transformation could be kept away from the user, so people with varying levels of familiarity with programming would be able to use the system. Our program would be hosted on the Harbour server, so it would be available to everyone who can access the Harbour CMS; and using a server-side program also minimizes issues that could arise from using different operating systems.[16]

Implementation can be one of the biggest challenges for a system like this, since there are always several requirements to satisfy. We had to accommodate various levels of expertise and different approaches to what a catalogue can be while affording as much flexibility as possible. Thus, first and foremost, we had to provide some level of instruction on how the system worked.

Second, users needed to be given some control over the output. When generating a catalogue, we didn't want the system to process every book HPDM has ever published. Usually we wanted books to be selected based on how recent they were, but it would be extremely interesting for marketers and sales personnel to be able to create catalogues following custom criteria, similar to how CataList and Edelweiss operate.

Third, the system should be able to be improved as easily as possible. As the system was used more and more, it was likely that unexpected cases would arise, and our XSLTs would not be prepared to handle them. So we had to be able to update the XSL files to accommodate these cases, and be sure that they would not be a problem again in the future. That way, the more the system was used, the more accurate it would become.

16 W3Schools, "PHP 5 Introduction."

Fourth, we needed some way to identify and correct errors. Errors of various kinds were expected to arise as the system was used, so users needed to know how to correct them or who to contact for assistance. A simple HTML error that could be corrected by editing the book data on the website should be differentiated from a system error that must be corrected in the XSL or PHP files.

Finally, the system should, of course, be relatively easy to use. The main issue here was that it should be as easy as possible to carry out corrections and updates: after a change was made to the website data, the less effort it took to update the same data in InDesign, the better.

As far as instructions went, we only included basic steps and links to the ONIX code lists and tutorials on XSLT and XPath syntax, which would be used in writing the selection criteria. We also included a link to the most recent ONIX file, which could serve as a reference for identifying the required paths. More extensive instructions could be written later, when the system was ready for general use.

The next step was to set up a basic processor. Thankfully, PHP provides several built-in functions for processing XSLTs. All we needed to do was write in some error messages, so that if the processing failed the user was notified of where the problem happened; then ensure both steps of the transformation were run, one after the other. Again, we kept our error messages simple: if the error occurred in processing the second step, it was likely an HTML issue that can be fixed on the website. The interim XML could then be opened and checked for specific errors. Otherwise, the problem was in the PHP or XSL files, so the user is instructed to ask us for help.

The key to the system was being able to set custom criteria. Allowing the user to decide which books were selected for the ICML meant that any number of catalogues could be generated, for any reason. To do so, we first had to prepare our original XSLT to accept external definitions by setting a parameter—essentially a variable—at the beginning of our document, after the stylesheet and output properties but before the first template (see Snippet

20). Only the parameter name was necessary, but we also provided a default value in case none is set.

Because of the nature of the two languages, variables passed from PHP to XSL are interpreted as a string—that is, a simple series of characters—and cannot not be understood as a valid XPath expression. We had to implement an XSLT add-on function called dyn:evaluate, which converts a string into an XPath expression.[17] After adding a short line of code to make sure the XSLT engine understood the add-on, we substituted the Select attribute in our master template with the parameter and wrapped it with dyn:evaluate (see Snippet 20). Now that we had the parameter defined and placed in the template, we could use PHP to set its value.

Snippet 20. Setting a dynamic parameter passed by PHP.

```
<xsl:param name="criteria" select="//
Product[PublicationDate > 20150000]" />
…
<xsl:for-each select="dyn:evaluate($criteria)">
  <xsl:apply-templates select="Title"/>
  …
</xsl:for-each>
```

With the parameter correctly sent by PHP and understood by the XSLT, we could move forward and add a simple form to allow the user to define the parameter. Manually setting the parameter as a variable in the way required by the XSLT is too complex for most users, who will not have the time to review the XPath tutorial and function references every time a catalogue is required. However, this would allow the system to be in use earlier, while a more user-friendly interface could be developed later.

The form also provided a set of other helpful functions. First, there was a "Generate" button that updated the working ONIX, a copy of the ONIX

17 "How do I pass and XPath expression."

message that would be regularly sent to vendors. While the regular ONIX was generated daily, however, this copy could be generated on command to immediately include any corrections. Second, we offered a download link for the ONIX file, as well as for each of the XSLTs. The XLSTs could also be updated by uploading a new file. Clicking the "Submit" button activated the process: it uploaded any replacement XSLTs, then applied the first one to the to the working ONIX. If this step was successful, it would apply the second XSLT to the interim file; if there were no HTML errors, this would produce the ICML file, which it automatically downloaded to the user's computer. This meant that, whenever corrections were entered in the CMS, it only took a click to update the working ONIX, another to generate a new ICML and—in InDesign—one final click to update the file.

HAVING SORTED THROUGH THE ONIX FILES FOR THE RELEVANT DATA, SET up the necessary XSL transformations to output valid ICML, and implemented effective ways to select which books to process and to carry out revisions, we had an ONIX-to-ICML prototype ready for its final test.

CHAPTER 3 The Future

Evaluation

IN DEVELOPING ANY TYPE OF PRODUCT, IT IS IMPOSSIBLE FOR THE CREators to foresee everything that could possibly come up in using it. It is almost guaranteed that some feature judged important will go unused, while others will be found to be missing or incomplete. This is the purpose of prototypes and alpha and beta tests: to judge the effectiveness of the product in achieving its design goals and identifying areas for improvement.

The System Test

OUR ALPHA TEST WAS COMPRISED OF TWO COMPLETE RUN-THROUGHS OF the system: execute a first processing attempt; identify and correct HTML errors; update the ONIX and process it again; and place the ICML file into the InDesign template. If the information came through as expected, the system could be considered successful. For the first run-through we created a catalogue of the two latest seasons of Nightwood Editions titles, to be used in grant applications. We targeted books that both were published since the beginning of July 2014 and had Nightwood Editions set as the publisher's name.

Since the Nightwood list is smaller and less varied than the Harbour list, we performed the same run-through with the most recent Harbour titles. This was especially useful in evaluating the accuracy of translated data since there was an existing print catalogue—produced in the traditional manner, before this system was developed—that we could compare the output to. While the transformation was once again successful—after fixing the same kinds of HTML errors as the Nightwood run-though—this attempt shed some light on other discrepancies. Further, we had HPDM's managing editor as audience in this step to provide other perspectives and further input.

In both cases, the first step of the transformation appeared to be successful, but—as expected—the second step was interrupted by HTML errors such as those described in the previous chapter. There were several unclosed paragraph tags, a handful of poorly written URL links and a few miscoded special characters. After these problems were manually fixed in the CMS and a new copy of the ONIX was generated, the transformation ran smoothly for both steps, and yielded an ICML file. The conversion system, at least, seemed to be working correctly.

Placing this file into the prepared InDesign template, though, revealed further problems. Many were simple, fixable problems with the content itself. One was rather an issue of compatibility between the way the data is stored and how it needs to be used and presented. Others were shortcomings of the system itself; some easily addressed and some major barriers to the system's real-world application.

Content Problems

The first content problem is rather an extension of something we thought we had overcome: paragraph breaks. Many standard paragraph breaks were indicated by a single `<br/>` tag. Since the ONIX generator was preserving single `<br/>`s, they were then converted by the XSLT to the forced line break character, which does not separate paragraphs. Unfortunately, since we do want to preserve forced breaks in some cases, we could not automate a solution in Perl as we had done before.

The second content problem is also an extension of something previously discussed: improper workarounds. In a few books where the tagline was a review quote or promotional blurb, the quote author's name was indented using spaces, which made them inconsistent with other paragraphs. One tagline also had italic tags around it, which were preserved in the ICML. Both of these were probably done to imitate the style of the catalogue, but infringe on the principle of separating design and content. In this case, both italics and indents were a stylistic choice, and should have been applied through CSS instead.

One particular tagline had three review quotes in it, which unsurprisingly took up more space than was allotted and caused the book's main description to overflow into other pages. That particular book was not making use of the existing website fields for review quotes, and instead relying on the tagline for that. Like with the italics in the tagline, this is simply another case of a user finding the fastest or easiest option that fits their needs, rather than the correct one. To rectify the issue, we decided to keep the most prominent review quote as the tagline, and moved the other two into the appropriate fields.

As discussed in Chapter 2, the First Nations and Regional Interest genres were problematic, but some unexpected genre issues arose when the primary BISAC code assigned to a title did not match its designated genre listing as shown in the catalogue. This could be a relatively common problem since the staff members making genre decisions for the catalogue are rarely the same as those making BISAC selections for the website; and while both choices may be equally accurate, they are frequently different.

For example, the book *Cape Scott and the North Coast Trail*, about a recently completed trail on northern Vancouver Island, was expected to be listed as "Travel," and the "TRV" subject heading was coded in the system. But the primary BISAC for *Cape Scott* was in fact SPO018000, the code for "Hiking." The Sports heading had been left out of the XSL, which resulted in the *Cape Scott* genre line being left empty. This raised the question: should we change *Cape Scott*'s BISAC to a TRV code, or accommodate the SPO heading? The decision was to correct the BISAC.

A similar type of problem involved contributors, when their roles were not accurately entered in the system. The authors of *Cam Tait* were differentiated in the catalogue by the way their names were presented: "Cam Tait with Jim Taylor." But on the CMS, both were given the same contributor role A01, suggesting both were primary authors. Thus the resulting author line was inaccurate: "Cam Tait and Jim Taylor." The solution was to change the contributor role for Jim Taylor to A02 "With," and ensure it was accurately accounted for in the XSL file.

All of these content issues could be easily fixed in the CMS. However, it is important to note that the genre and contributor role problems were not simply HTML errors that would cause text to display with the wrong formatting. They were factual inaccuracies in the ONIX data about these books, which meant booksellers and libraries could end up grouping them with the wrong set, for example. At the moment, there are no systems in place to verify this type of information, so much of it goes unnoticed and unchanged. Thus, our catalogue system also presented an unexpected but very valuable feature in helping HPDM identify where our ONIX may be wrong, and prevent similar mistakes from being committed in the future.

Problems with BISAC Headings

From the point of view of a marketer looking for a catalogues of specific genres, the system faces one major problem. It would be easy to create a catalogue of all books on art, for example, by matching the first three characters in BASIC Main Subject, as we did in the genre processing template in the XSL. But since the First Nations genre is fragmented across several BISAC headings and Regional Interest has no BISAC equivalent whatsoever, it would be nearly impossible to select every book in those genres. In the case of First Nations, the criteria would have to include the BISAC codes of every subheading about First Nations; for Regional Interest, it would have to include the ISBNs or titles of every book in that category.

To make these catalogues possible, the system needs some sort of shorthand catch-all code that can simplify these selectors. Unfortunately, we have yet to identify the correct approach for this issue.

Prototype Shortcomings

As might be expected, the prototype suffered a number of shortcomings, some of which we identified from the start or encountered during the development process. For example, the prototype did not incluence the process of writing and editing promotional copy, which remains a complicated process. The website CMS also presented a number of obstacles: Since there is no ebook

ISBN field in the CMS, it needs to be manually typed into the InDesign file, leaving the margin for error unchanged in that case. Manual entry errors also remain a problem with the tagline, description and author bio fields, where the CMS needs to adopt WYSIWYG editors to minimize formatting and coding mistakes. And until the D&M ONIX can match Harbour's in style and ease of access, the system will at best assist in one half of the production process.

Another set of shortcomings was only identified at the end, during the complete system run-though, with HPDM's managing editor was present. For example, editors might agree to omit certain text from the tagline, description or bios in the printed catalogue, where there is limited space, but wish to have it included on the website and ONIX, where space is open. The system currently offers no way to hide text from certain platforms.

Further, when text is flowed into InDesign, it can result in bad breaks—when a word breaks from one line to the next in an undesirable location. This could be a proper name that should not break at all, or a hyphenated word that should be kept from breaking anywhere but the hyphen. Again, there is currently no way for the system to handle these typographical requirements; tweaking the hyphenation settings in the InDesign template can help minimize the problems, but there will always be exceptions.

A close look at the InDesign document revealed another typographical problem, and a rather important one: several special characters were not being preserved. En-dashes were coming out as hyphens, em-dashes as double hyphens, and all quotation marks were coming out straight instead of curly. The catch was that these characters were, in almost every case, correctly entered in the CMS and rendered on the website accordingly. The issue was arising in the generation of the ONIX. Most likely there was an encoding inconsistency between how those characters were saved and how they were processed. After tinkering with the Perl program[18] that creates the ONIX, we

18 The Perl program that generates the ONIX was not an expected component to this prototype. With very limited understanding of Perl, we could only modify the program based on what we gathered from a cursory study of the code.

managed to rescue the en- and em-dashes, but the quotes remained straight. This is a major issue that remains unsolved, and is likely the biggest barrier to real-world use that the system faces.

Another drawback is that we rely on typed-out criteria. In fact, although we initially placed a good-looking user interface outside the prototype parameters, a UI overhaul is badly needed. The current interface is too complex for an unaccustomed user. It requires users to have at least some level of familiarity with XSLT, XPath and the ONIX structure to be able to use the system, which few have. Further, the web form currently does not remember the last criteria used, so if the user navigates away from the page, they will need to reenter the criteria when they visit it again. This can be incredibly onerous if the criteria are particularly long or if it takes several rounds to clear all the errors. A more advanced and intuitive user interface is necessary. If the criteria could be set by using drop-down menus or checkboxes, for example, the system could be usable without any knowledge of the XSLT and XPath languages.

Finally, it is imperative that the system be able to handle both front-list and back-list titles. This was outside the prototype parameters, but most catalogues need to feature books with varying levels of prominence. The main obstacle here is that recent back-list titles, which are listed with a medium level of detail, feature only a shortened version of the description, usually one paragraph long. The Harbour CMS currently does not feature any fields to this effect, although the ONIX code lists offer some possibilities with the `<OtherText>` element. This field needs to be added to the CMS and ONIX before this aspect of the system can be developed.

Final Balance

GIVEN SUCH A LARGE NUMBER OF PROBLEMS WITH THE SYSTEM, CAN WE say it was successful? The fact is that, as it stands, the system remains unusable in a real-world environment. However, it was only a prototype: the fact that it doesn't work now doesn't cloud the possibilities for its future. It did facilitate the dissemination of information, as it set out to do; it sped up the process of preparing the catalogue front-list for printing, missing only a way to handle the reduced detail of the back-list; and the groundwork for creating custom catalogues has been laid, pending only an improvement to the user interface. The margin for error was smaller, inconsistencies were easier to spot and it could even help identify problems with the ONIX data itself. Despite its shortcomings, the prototype showed that it can become a significant aid in preparing catalogues.

In fact, in a recent meeting, the production team expressed their wish that the system could be ready in time to prepare the Fall 2015 catalogue. In particular, the automation would significantly speed up the process of updating discounted prices, which in some cases appear in multiple sections of the catalogue. Unfortunately, this was not possible because of the short timeline.

While some of the missing features require immediate attention and others can be handled in the middle or long term, all are in the plans for the ONIX-to-ICML system, and some are already being addressed. The background work required for merging D&M and Harbour CMSs is currently under way, for example, and several improvements to the system are scheduled to be implemented shortly thereafter, including ebook ISBNs and a WYSIWYG editor for the descriptive fields. With these changes and only a few further enhancements to the XSLTs and their PHP handling, including the necessary UI improvements, the ONIX-to-ICML system could be usable in time for the Spring 2016 catalogue, to be produced around September of 2015, and fully functional the following season.

Far Future: Other Applications

THE PROTOTYPE PROVED THAT WORKFLOW SYSTEMS ARE A USEFUL INVESTMENT. Any system that makes it easy and straightforward to reuse information, or disseminate it to multiple platforms, saves time, effort and consequently money. Although it will take an extra dose of each of those elements to get any system into action, the long-term benefits of a well-designed and easily updated system will ultimately outweigh the costs.

In fact, other book publishers have already accepted the benefits of reusable data and adopted digital-first workflows. At F+W, a company that describes itself as "A Content + eCommerce Company," their book publishing workflow is managed by a custom CMS derived from the Librios workflow software.[19] By using semantic tags to describe the contents of a book, the F+W production team can easily export the same title to an ebook, where the tags match CSS definitions, and to InDesign, where they match the styles in a template, at the same time.

While the ONIX-to-ICML prototype was at most partially successful, several HPDM staff members are excited about the prospect of a fully functional system. Although the prototype needs to be improved and expanded before any further steps are taken, the possibility that HPDM will eventually adopt a digital-first workflow is distinct. It is my hope that HPDM will eventually follow the example of F+W and embrace reusable data for all production streams.

19 Cunningham, "How tagging can streamline your workflow."

Bibliography

Adobe Systems Inc. "IDML File Format Specification, version 7.0." 2010. https://www.adobe.com/content/dam/Adobe/en/devnet/indesign/cs55-docs/IDML/idml-specification.pdf.

Cunningham, Colleen. "How tagging can streamline your print + digital workflow." BookNet Canada blog, January 22, 2015. http://www.booknetcanada.ca/blog/2015/1/22/how-tagging-can-streamline-your-print-digital-workflow.html.

EDItEUR. "ONIX for Books Codelists Issue 27." October 17, 2014. http://www.editeur.org/files/ONIX%20for%20books%20-%20code%20lists/ONIX_BookProduct_Codelists_Issue_27.html.

———. "ONIX for Books Product Information Message Product Record Format." January 2006.

———. "What are 'Reference names' and 'Short tags'?" http://www.editeur.org/74/faqs#q10.

Harbour Publishing. ONIX Messages (various). http://harbourpublishing.com/ONIX.

"How do I pass an XPath expression as an XSL param using PHP?" StackOverFlow, January 5, 2015. http://stackoverflow.com/questions/27774013/how-do-i-pass-an-xpath-expression-as-an-xsl-param-using-php.

Maxwell, John. "Ickmull Project Summary." November 2011. https://code.google.com/p/ickmull/

Till, Kaitlyn, Shed Simas and Velma Larkai. "The Flying Narwhal: Small mag workflow." April 14, 2014. http://tkbr.ccsp.sfu.ca/mpub/2014/04/14/the-flying-narwhal-small-mag-workflow.

W3Schools. "PHP 5 Introduction." http://www.w3schools.com/php/php_intro.asp.

———. "XSLT Introduction." http://www.w3schools.com/xsl/xsl_intro.asp.

APPENDIX Sample ONIX Record

THE FOLLOWING IS A COPY OF THE COMPLETE ONIX RECORD FOR *RAINCOAST Chronicles 23* retrieved on December 17, 2014.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ONIXMessage SYSTEM "http://www.editeur.
org/onix/2.1/reference/onix-international.dtd">
<ONIXMessage>
  …
  <Product>
    <RecordReference>1550177109</RecordReference>
    <NotificationType>03</NotificationType>
    <RecordSourceType>01</RecordSourceType>
    <RecordSourceName>Harbour Publishing</
    RecordSourceName>
    <ProductIdentifier>
      <ProductIDType>02</ProductIDType>
      <IDValue>1550177109</IDValue>
    </ProductIdentifier>
    <ProductIdentifier>
      <ProductIDType>03</ProductIDType>
      <IDValue>9781550177107</IDValue>
    </ProductIdentifier>
    <ProductIdentifier>
      <ProductIDType>15</ProductIDType>
      <IDValue>9781550177107</IDValue>
    </ProductIdentifier>
    <Barcode>10</Barcode>
    <ProductForm>BC</ProductForm>
    <ProductFormDetail>B102</ProductFormDetail>
    <NoSeries/>
    <Title>
```

```
      <TitleType>01</TitleType>



      <TitleText>Raincoast Chronicles 23</
      TitleText>

  <Subtitle>Harbour Publishing 40th Anniversary
  Edition</Subtitle>
</Title>
<Contributor>
  <SequenceNumber>1</SequenceNumber>
  <ContributorRole>B01</ContributorRole>
  <PersonName>Peter A. Robson</PersonName>
  <PersonNameInverted>Robson, Peter A.</
  PersonNameInverted>
  <NamesBeforeKey>Peter A.</NamesBeforeKey>
  <KeyNames>Robson</KeyNames>
  <BiographicalNote>&lt;p&gt;Peter A. Robson
  has authored hundreds of articles and has
  served as editor for several magazines, in-
  cluding &lt;em&gt;Pacific Yachting&lt;/
  em&gt;, &lt;em&gt;Cottage&lt;/em&gt; and
  &lt;em&gt;Cottage Life West&lt;/em&gt;. He has
  also authored or co-authored books about com-
  mercial fishing, forestry, towboating and salmon
  farming. He is a regular at Harbour Publishing's
  Friday martini hour and lives in Garden Bay,
  BC.&lt;/p&gt;</BiographicalNote>
  <CountryCode>CA</CountryCode>
</Contributor>
<Contributor>
```

```
<SequenceNumber>2</SequenceNumber>
<ContributorRole>A24</ContributorRole>
<PersonName>Howard White</PersonName>
<PersonNameInverted>White, Howard</
PersonNameInverted>
<NamesBeforeKey>Howard</NamesBeforeKey>
<KeyNames>White</KeyNames>
<BiographicalNote>&lt;b&gt;Howard White&lt;/
b&gt; was born in 1945 in Abbotsford, British
Columbia. He was raised in a series of
camps and settlements on the BC coast and
never got over it. He is still to be found
stuck barnacle-like to the shore at Pender
Harbour, BC. He started &lt;i&gt;Raincoast
Chronicles&lt;/i&gt; and Harbour Publishing
in the early 1970s and his own books include
&lt;i&gt;A Hard Man to Beat&lt;/i&gt; (bio),
&lt;i&gt;The Men There Were Then&lt;/i&gt;
(poems), &lt;i&gt;Spilsbury's Coast&lt;/i&gt;
(bio), &lt;i&gt;The Accidental Airline&lt;/i&gt;
(bio), &lt;i&gt;Patrick and the Backhoe&lt;/
i&gt; (childrens'), &lt;i&gt;Writing in the
Rain&lt;/i&gt; (anthology) and &lt;i&gt;The
Sunshine Coast&lt;/i&gt; (travel). He was award-
ed the Canadian Historical Association's Career
Award for Regional History in 1989. In 2000,
he completed a ten-year project, &lt;i&gt;The
Encyclopedia of British Columbia&lt;/i&gt;. He
has been awarded the Order of BC, the Canadian
Historical Association's Career Award for
Regional History, the Stephen Leacock Medal for
Humour, the Jim Douglas Publisher of the Year
```

```
  Award and a Honorary Doctorate of Laws Degree
  from the University of Victoria. In 2007, White
  was made an Officer of the Order of Canada. He
  has twice been runner-up in the Whisky Slough
  Putty Man Triathlon.</BiographicalNote>
  <CountryCode>CA</CountryCode>
</Contributor>
<NoEdition/>
<Language>
  <LanguageRole>01</LanguageRole>
  <LanguageCode>eng</LanguageCode>
</Language>
<NumberOfPages>160</NumberOfPages>
<IllustrationsNote>80 B&amp;W photos and
illustrations</IllustrationsNote>
<BASICMainSubject>HIS006000</BASICMainSubject>
<Subject>
  <SubjectSchemeIdentifier>10</
  SubjectSchemeIdentifier>
  <SubjectCode>FIC051000</SubjectCode>
</Subject>
<AudienceCode>01</AudienceCode>
<OtherText>
  <TextTypeCode>01</TextTypeCode>
  <Text textformat="02">&lt;p&gt;When the first
  edition of &lt;em&gt;Raincoast Chronicles&lt;/
  em&gt; was produced by a couple of novice pub-
  lishers in the unlikely location of Pender
  Harbour in 1972, it boldly announced that it
  was going "to put BC character on the rec-
  ord." Printed in sepia ink and decorated with
  the rococo flourishes characteristic of that ex-
```

travagant era, the unclassifiable journal-cum-serial-book about life on the BC coast struck a nerve and in time became something very close to what it set out to be--a touchstone of British Columbia identity. Soon the term "Raincoast," which had been coined by the editors, was appearing on boats, puppet theatres, interior decorating firms and at least one other publishing enterprise. &lt;/p&gt;&lt;br /&gt;

&lt;p&gt;&lt;em&gt;Raincoast Chronicles&lt;/em&gt; also created another publishing enterprise--Harbour Publishing. Many of the stories that started out as articles in the Chronicles grew into books and so the White family was more or less forced to get into book publishing to deal with them. That undertaking went on to publish some six hundred books (and counting!) about every possible aspect of BC and, in 2014, celebrated its fortieth anniversary in the biz. To honour that occasion this special double issue of &lt;em&gt;Raincoast Chronicles&lt;/em&gt; takes a tour down memory lane, selecting a trove of the most outstanding stories in all those Harbour books and republishing them in one volume. &lt;/p&gt;&lt;br /&gt;

&lt;p&gt;Here are some of Canada's most exciting and iconic writers--Al Purdy, Anne Cameron, Edith Iglauer, Patrick Lane and Grant Lawrence, to start a long list. Here also are stories of disasters at sea, scarcely believable bush plane feats, eerie events at coastal ghost towns and a First Nations elder who has

```
  seen so many sasquatches he finds them sort of
  boring. Full of great drawings and photos, this
  jumbo anniversary edition of &lt;em&gt;Raincoast
  Chronicles&lt;/em&gt; is a feast of great Pacific
  Northwest storytelling.&lt;/p&gt;</Text>
</OtherText>
<Imprint>
  <NameCodeType>01</NameCodeType>
  <NameCodeValue>HAROP</NameCodeValue>
  <ImprintName>Harbour</ImprintName>
</Imprint>
<Publisher>
  <PublishingRole>01</PublishingRole>
  <PublisherName>Harbour Publishing</
  PublisherName>
</Publisher>
<CountryOfPublication>CA</CountryOfPublication>
<PublishingStatus>02</PublishingStatus>
<PublicationDate>20150328</PublicationDate>
<CopyrightYear>2015</CopyrightYear>
<SalesRights>
  <SalesRightsType>01</SalesRightsType>
  <RightsTerritory>WORLD</RightsTerritory>
</SalesRights>
<Measure>
  <MeasureTypeCode>01</MeasureTypeCode>
  <Measurement>11</Measurement>
  <MeasureUnitCode>in</MeasureUnitCode>
</Measure>
<Measure>
  <MeasureTypeCode>02</MeasureTypeCode>
  <Measurement>8.5</Measurement>
```

```
      <MeasureUnitCode>in</MeasureUnitCode>
    </Measure>
    <SupplyDetail>
      <SupplierName>Harbour Publishing</SupplierName>
      <ReturnsCodeType>02</ReturnsCodeType>
      <ReturnsCode>Y</ReturnsCode>
      <AvailabilityCode>NP</AvailabilityCode>
      <ProductAvailability>10</ProductAvailability>
      <ExpectedShipDate>20150227</ExpectedShipDate>
      <OrderTime>1</OrderTime>
      <PackQuantity>1</PackQuantity>
      <Price>
        <PriceTypeCode>01</PriceTypeCode>
        <ClassOfTrade>gen</ClassOfTrade>
        <PriceAmount>24.95</PriceAmount>
        <CurrencyCode>CAD</CurrencyCode>
      </Price>
      <Price>
        <PriceTypeCode>01</PriceTypeCode>
        <ClassOfTrade>gen</ClassOfTrade>
        <PriceAmount>24.95</PriceAmount>
        <CurrencyCode>USD</CurrencyCode>
      </Price>
    </SupplyDetail>
  </Product>
</ONIXMessage>
```