Writing an Online Documentation System using Omnis Studio 10

A multidisciplinary approach to converting, displaying and searching your documentation in a web browser using Omnis Studio.

By Gary Ashford
Omnis Engineering

White Paper

July 2020

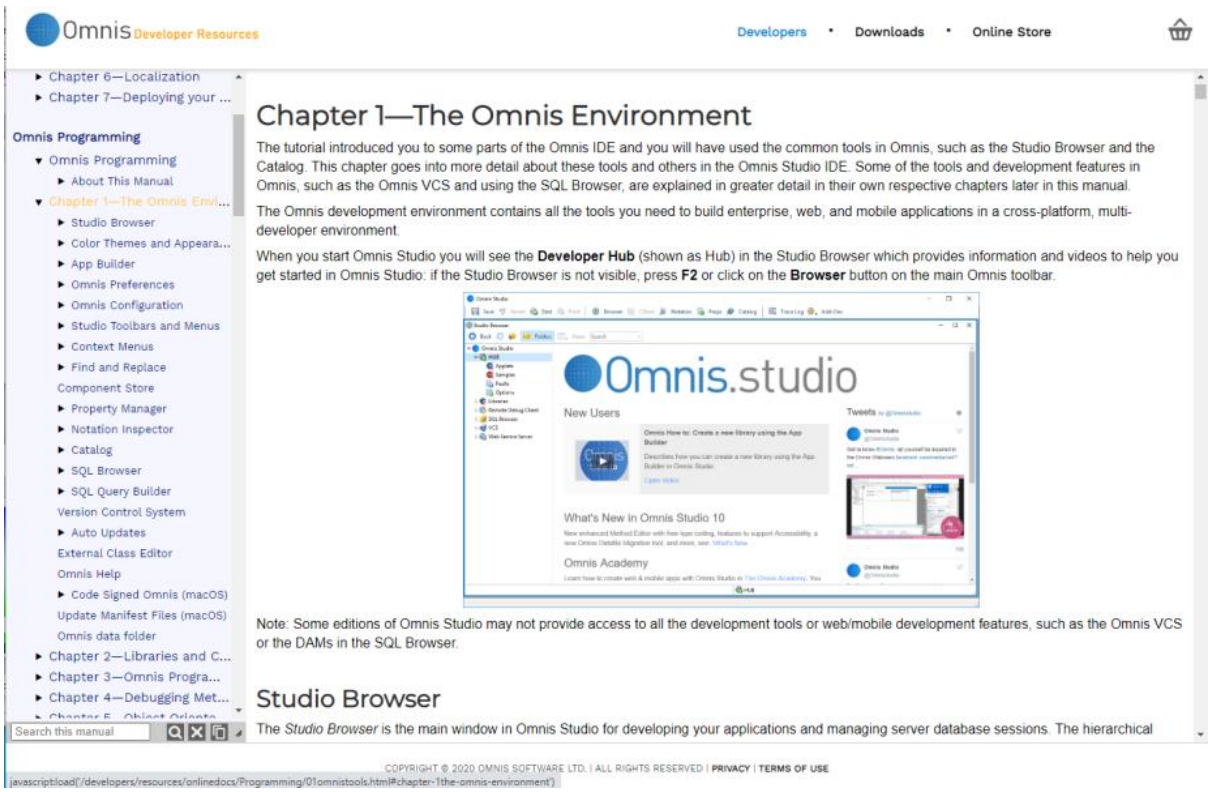
Table of Contents

Writing an Online Documentation System using Omnis Studio 10 1
Introduction 4
Converting the Source documents 5
Know your Limitations 5
A Simple Conversion 5
Using Omnis Studio 7
Extracting the Table of Contents 7
Adding Chroma-coding 7
Using CSS 8
Using JavaScript 8
Using Version Control 9
Searching the Documentation 9
Highlighting Search Terms 9
Bringing it all Together 10
Other Platforms 11
Running the Demo Project 11
Acknowledgements & Disclaimer 11
Disclaimer 11

Introduction

Some time ago we identified the need to move our main documentation online and make it part of the Omnis website. Historically, our documentation was produced as hard-copy manuals published from source documents written using Microsoft Word. This format also allowed us to produce downloadable PDF versions of the manuals from the same source documents.

For our online documentation we therefore needed a system that would readily convert our Word documents into HTML, whilst providing backward compatibility and the flexibility to make continuous updates to the source documents. It did not take long to realise that Omnis Studio was ideally suited to form at least part of this process, with help from a third-party conversion utility, plus HTML, CSS and JavaScript technologies.



Screenshot from the Omnis online documentation (https://omnis.net/developers/resources/onlinedocs/index.jsp)

Converting the Source documents

The internal format of Word documents is proprietary, so for the actual conversion of Word documents to HTML, we tried several third-party utilities, eventually deciding on a program called Pandoc; a universal document converter.

Know your Limitations

Whilst immensely powerful and providing conversion to and from many different document formats, we found that conversion from Microsoft Word documents came with caveats. The ability of Pandoc to extract embedded images as separate PNG files was immediately attractive, as was the ability to generate an embedded HTML table-of-contents based on the document headings. Simple tables, ordered and un-ordered lists, bold, italic and underlined text can also be converted. Formatting options are limited however. Fonts are basically ignored. Only Heading styles (1 to 6 in our case) are extracted and converted to HTML headings <h1> to <h6>.

Working to these limitations, we edited and produced a set of source documents that would respect this “limited palette” of formatting options. We also had to come to terms with the fact that when converting to HTML, there is a difference between carriage returns and paragraph breaks! Simply pressing ENTER in a Word document results in a paragraph break in the converted document. Simple line breaks are achieved using SHIFT+ENTER.

Similarly, HTML documents tend to collapse multiple spaces into a single space. To work around this issue, it was necessary to replace all such occurrences in the source documents with non-breaking space characters (CTL+SHIFT+SPACE).

We also needed to make sure that any existing HTML hyperlinks in the Word documents were turned into absolute/external links where necessary. Otherwise any document-relative links would present potential portability issues in converted documents.

It is also worth noting that Pandoc produces one HTML document for each Word (.docx) document. To avoid overly-long HTML documents, you may find it necessary to subdivide your source documents.

A Simple Conversion

Pandoc is a command line utility. The intricacies of the Pandoc command line syntax is left for further reading. In essence however, and using our Programming manual as an example, to convert a Word document entitled “00about.docx” to its HTML counterpart we use the command:

pandoc.exe --include-in-header=C:\Users\garya\pandoc2\htmlhead.txt --toc --extract-media="W:\onlinedocs\Programming\00about" -o "W:\onlinedocs\Programming\00about.html" "F:\VCS\Docs\Programming\00about.docx"

--include-in-header allows us to specify some HTML content to be inserted at the start of the converted document. We use this file to inject a stylesheet into the page and to insert some JavaScript that will be used later to assist with navigation and to make sure the page is displayed together with the table of contents.

--toc tells pandoc to produce an embedded table-of-contents (TOC) inside the target document. Omnis will process and extract TOCs after conversion to produce a separate HTML file containing the merged TOC entries from all converted document files.

--extract-media tells pandoc to extract images to the named folder. URLs inserted into the converted document contain links to the extracted images. Omnis post-processes these links to make sure they are correct when the target files are deployed to the web server.

-o tells pandoc where to write the converted HTML file, and its file name.

The final parameter is the path to the source document.

You will note that for consistency we are outputting the target document and the associated media images to the same location. Images are placed in a subfolder name based on the document name (00about in this example). We are also taking advantage of a mapped network drive (W:\) which allows us to output converted documents directly to our staging server. This allows us to review document changes before “pushing” them to our live website.
We also use a VCS as a source repository for our Word documents. Once updated, Pandoc is able to access these documents directly from the repository folder (F:\VCS\Docs in this case).

Obviously, to convert an entire manual like the Omnis Programming manual requires several pandoc commands; one for each document/chapter. To collate several manuals into a single documentation set requires some further post processing in order to ensure that entries in the combined table of contents all point to the correct locations.

Where required, we prepend all of our source document filenames with numbers (commencing 00, 01, 02, etc.) to ensure that the Omnis FileOps component picks up the files in the correct order.

Using Omnis Studio

To use an external utility like Pandoc, we would normally look for a convenient C/C++ API that we can access using an Omnis external component. In the absence of this (and similarly for the version control system discussed later) we simulate API calls by writing commands to a Windows batch file.

Omnis executes the batch file and waits for completion. Since the Start program command launches a separate process, Omnis monitors a "semaphore" (a simple text file) to await completion. By writing interim progress values to the semaphore file, Omnis is also able to update a progress indicator while the batch file is executing.

Extracting the Table of Contents

Conveniently, when Pandoc is invoked using the –toc option an unordered list containing the table of contents is embedded into each converted HTML document, placed inside <toc>…</toc> tags.

After conversion, Omnis reads each HTML file and extracts the <toc> section, writing these to interim files which are then concatenated into a single .toc file for that manual.

The table of contents is important since it is used to navigate the online documentation when displayed as two side-by-side panels on a desktop system (or as a pop-out menu on a mobile device).

Since two panels are used, Omnis processes each URL of the TOC changing them from standard links into JavaScript method calls. When the user clicks on a TOC entry the JavaScript method loads the required page into the content panel and scrolls the content into view where necessary.

Adding Chroma-coding

To display code samples in the online documentation we dedicated one of the heading styles <H6>. Initially (and by default) the CSS used by the online documentation displays code sections in a dark green, mono-space font.

```
# declare cvList1 of list type
Do cvList1.$definefromsqlclass(MySchema)
Do cvList1.$select() Returns myFlag ## sends a select
If myFlag = 0 ## checks for errors
  OK message {SQL error [sys(131)]: [sys(132)]}
End If
Do MyList.$fetch(30) Returns myFlag ## fetches 30 rows
# to fetch another 10 rows and add them to your list
Do MyList.$fetch(10,kTrue) Returns myFlag
```

We wrote a chroma-coding class designed to recognise H6 sections containing Omnis commands and notation. The chroma-coder then parses these sections and re-constructs them, placing special CSS spans around the various tokens.

While not perfect by any means, we have included the chroma-coding prototype in the conversion library accompanying this white paper. When it encounters something that looks like an Omnis variable it has no way of knowing whether it is a class, task, instance or local variable. We work around this issue by prepending variables in code samples with 'c','t','i' and 'l' respectively. There are also issues in discriminating between list and class notation but the overall effect gives a reasonable approximation to the Omnis IDE.

Using CSS

The header file imported into each HTML document during conversion contains a link to a Cascading StyleSheet (docsContent.css). This CSS file specifies the required fonts and contains entries for the normal paragraph text <p>, plus headings <h1> to <h6> where <h6> is reserved for code samples and <h5> is reserved for disclaimers and other small text. The CSS file also dictates how tables, images, ordered and unordered lists should be displayed in the content frame.

The table of contents also uses its own CSS file, and a separate CSS file contains entries for the various chroma-coding elements; commands, different variable types, functions, literals, notation, method names, and comments.

Using JavaScript

The online documentation uses JavaScript to:

- Load content when the user clicks on a TOC entry.
  A load() function loads the specified HTML file into the content iframe and scrolls the page so that the specified anchor tag is visible.
- Locate and scroll the TOC entries as the user scrolls through content.
  load() also sets-up an event listener that triggers whenever the user scrolls the content frame. The showTocHilite() function then scrolls the TOC iframe and positions a hilite around the TOC entry corresponding to the nearest visible <H1>, <H2> or <H3> heading.
- Add highlighting to the content document. An event listener on the toc.html file invokes a third-party Hiltor.js. Hilitor.apply('term') highlights occurrences of the search term in the content document.
- Expand and collapse headings in the table of contents.
  Where the TOC is collapsed, clicking on a parent node expands/collapses the node, displaying the child entries. showTocHilite() can also invoke this behaviour if the required TOC item is not currently visible.
- Copy selected links to the clipboard.
  The pageLinkToClipboard() function makes it possible to copy and paste links that will open/restore the online documentation at the currently selected position. This simplifies generation of a URL that contains parameters necessary to open and position the documentation at the given anchor location. For example:

  http://127.0.0.1:8887/Programming/06listprog.html#list-variable-values

- Invoke the search feature. The search works by issuing a RESTful request to an Omnis library. The library interrogates the search database built during the conversion process and returns information including the document URL and the hitcount that is used to add a hit counter to each applicable TOC heading.
- Control manual resizing of the TOC frame relative to the Content frame. The control works using a resize widget built in to the search toolbar.

Please inspect the accompanying docsearch.js, tocead.txt and htmlhead.txt files for further details and documentation on these functions.

Using Version Control

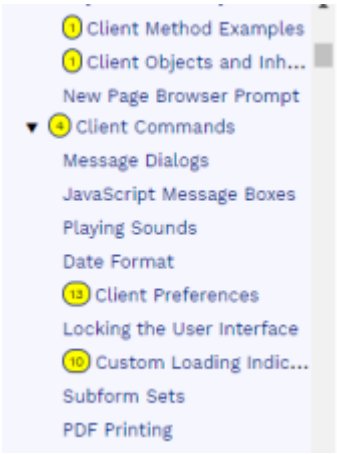
If your VCS or SVN application supports a command line interface, you can incorporate an "SVN update" or checkout phase into the Omnis library. For simplicity we have omitted this from the sample library however.

Using a semaphore system like that employed by the converter, it would be possible to pause execution pending completion of the VCS update.

Searching the Documentation

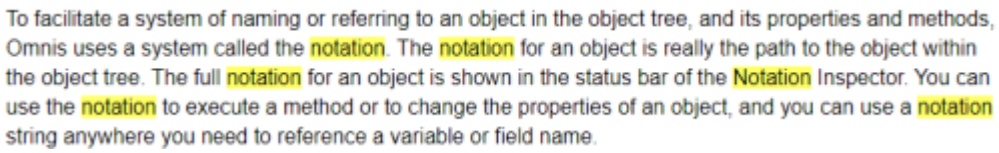
We devised a simple searching mechanism that basically extracts the text content (excluding formatting tags) from each HTML document and stores it in an external relational database.

A second Omnis library then listens for RESTful requests issued via the Search feature on the web page. The app searches the database for the supplied search term and orders the results in terms of relevance. The response contains a JSON array that the JavaScript can use to add "hit counters" to items on the TOC frame.



Highlighting Search Terms

Once the Table of Contents has been decorated in this way, clicking on one of the decorated links invokes some additional JavaScript that highlights occurrences of the search term in the content document. We found a convenient third-party JavaScript file for this purpose available from http://www.the-art-of-web.com/javascript/search-highlight/ (hilitor.js).



A cancel button on the search bar allows the TOC decoration to be removed; highlight tags are stripped from the TOC document and the table of contents is collapsed, ready for the next search.

Bringing it all Together

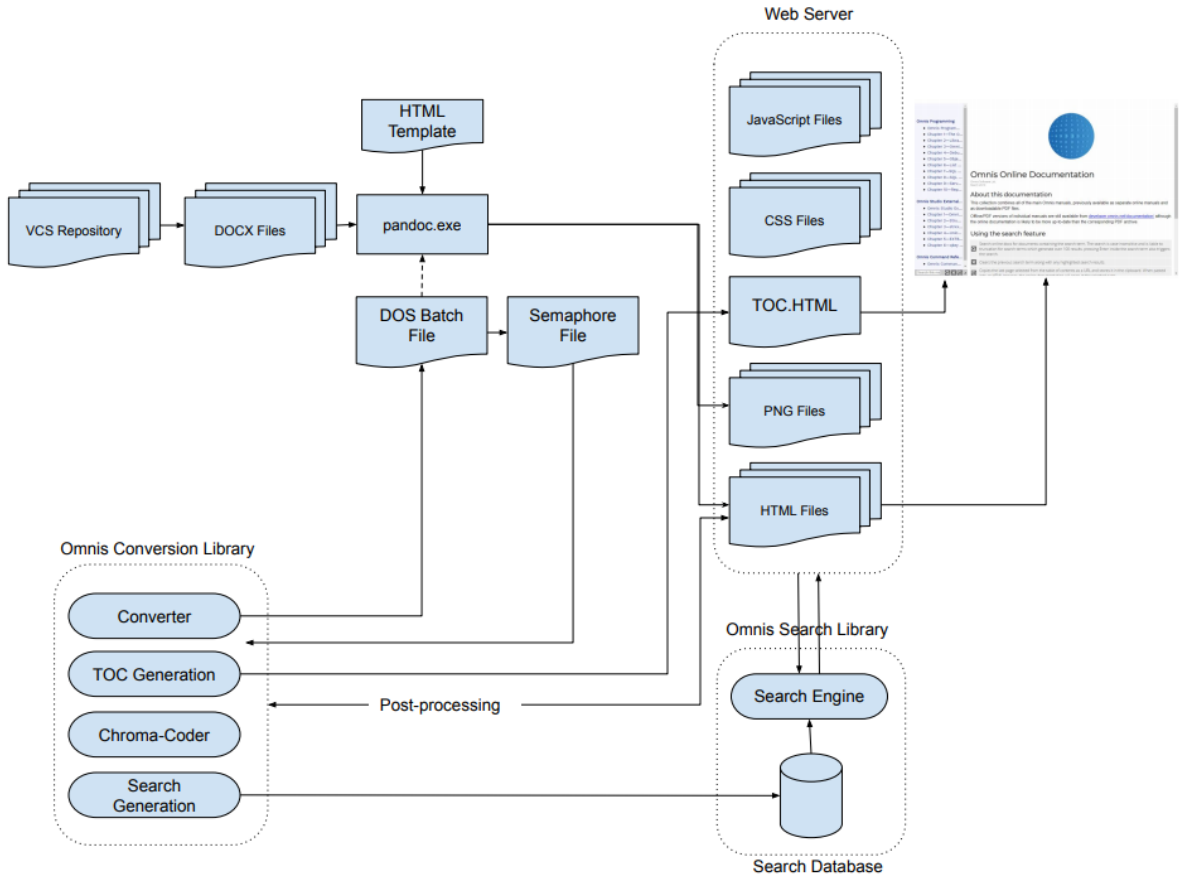
As you can see from the diagram below, the online documentation system is much more than an Omnis library. In fact there are two libraries; one which is used to re-generate and post-process the online documentation from the source documents, and can be shutdown afterwards.

The other library runs in combination with the web server and is responsible for processing search requests and returning the results via RESTful requests.

For demonstration purposes, we have designed the search library and the conversion library to run using the same instance of Omnis Studio 10.1. The search library uses the same “DOCGEN” session opened by the converter. For simplicity, the demo libraries also use a SQLite database to store the search information. Our live system uses PostgreSQL which is a lot more scalable.

To make the demo work, we recommend installing the Windows IIS component and following Technote TNJS0003 which describes how to configure IIS for use with Omnis web apps.
(The omnisrestisapi.dll is required in order to support RESTful request to the search library).
The “webdocs” output produced by the converter can then be copied into the \inetpub\wwwroot folder for local testing.

Please note that the demo libraries and source code have been simplified as much as possible but are presented “warts and all”. You are welcome to use any or all of these files as the basis for your own online documentation system, but please note we will not actively support this code and wish to draw your attention to the disclaimer at the end of this document.



Simplified diagram showing key components of the online documentation system.

Other Platforms

The demonstration libraries and converter presented at the end of this whitepaper were developed for use with a Windows operating system. Many of the cross-platform aspects of the project will also run on macOS and Linux subject to the availability/suitability of utilities such as pandoc for those platforms. In practice, the conversion library, web server and search engine can potentially run on separate machines with different operating systems. Our live system uses a Linux webserver. The search engine also runs "headless" on the Linux server. The converter runs on Windows and uses a mapped network drive to output its files directly to our web (staging) server.

Running the Demo Project

To accompany this whitepaper, we have included libraries and utilities that will work for example when the containing folder is placed on your desktop. We have also included a set of deprecated Word source documents for demonstration purposes only. A set of pre-converted HTML files is also included.

The accompanying files can be obtained from
https://omnis.net/developers/resources/documentation/whitepapers.jsp

Open the docgenDemo library first. This should create and open a sample SQLite database.
You can preview the web documents directly from the webDocs folder, but we recommend copying the folder contents to your web server folder. The search feature in particular will not work unless the web server is configured correctly for use with the omnisresisapi.dll and the Omnis Studio $serverport also needs to be set! There may also be permissions issues using the table of contents to display content unless the JavaScript executes in the context of a web server, e.g. Microsoft IIS.

Acknowledgements & Disclaimer

Pandoc

Pandoc – a free to use universal document converter: https://pandoc.org

CollapsibleLists.js

An object allowing lists to dynamically expand and collapse.
Created by Stephen Morley - http://code.stephenmorley.org/ - and released under the terms of the CC0 1.0 Universal legal code: http://creativecommons.org/publicdomain/zero/1.0/legalcode

Hilitor.js

Original JavaScript code by Chirp Internet: www.chirp.com.au
Please acknowledge use of this code by including this header.
http://www.the-art-of-web.com/javascript/search-highlight/

Disclaimer

