dzformatter-0.2.3 User Manual

K.Kakihara <kazuyoshikakihara@yahoo.co.jp>

2007-04-11

目 次

dzformatter の紹介

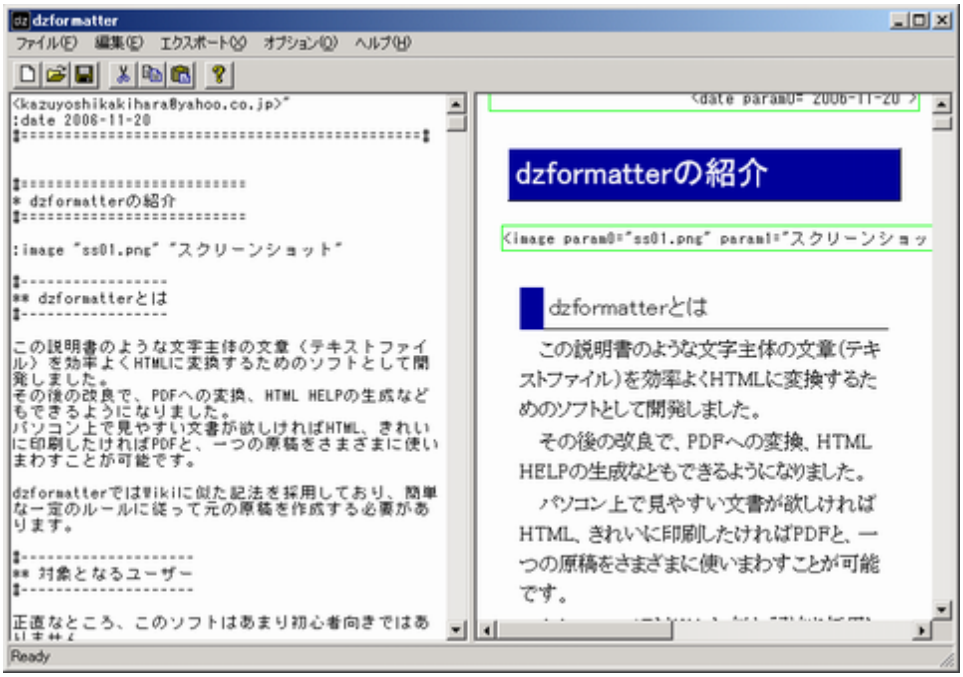

図 1: スクリーンショット

dzformatter とは

この説明書のような文字主体の文章（テキストファイル）を効率よく HTML に変換するためのソフトとして開発しました。

その後の改良で、PDF への変換、HTML HELP の生成などもできるようになりました。

パソコン上で見やすい文書が欲しければ HTML、きれいに印刷したければ PDF と、一つの原稿をさまざまに使いまわすことが可能です。

dzformatter では Wiki に似た記法を採用しており、簡単な一定のルールに従って元の原稿を作成する必要があります。

対象となるユーザー

正直なところ、このソフトはあまり初心者向きではありません。

HTML のソースって何？ というレベルではこのソフトの扱いは困難です。

手で HTML のタグは書けるけれど、それでは効率が悪すぎるとな、といったくらいのレベルの方にご利用いただければと思います。

ライセンス

本パッケージに含まれるソフトウェア、文書については、特に別途明示してあるものを除き、著作権は Kazuyoshi Kakihara に帰属します。

本パッケージは、Free Software Foundation によって発行された GNU GENERAL PUBLIC LICENSE Version 2 の定める条件の下で再頒布または改変することができます。

免責事項

本パッケージは無償で使用許諾されます。動作についての保証は一切ありません。

「あるがまま」の状態で、且つ、明示か暗黙であるかを問わず一切の保証をつけないで提供されるものとします。ここでいう保証とは、市場性や特定目的適合性についての暗黙の保証も含まれますが、それに限定されるものではありません。本パッケージの品質や性能に関する全てのリスクは使用者が負うものとします。

本パッケージに欠陥が見つかった場合も、それに伴う一切の派生費用や修理・訂正に要する費用は全て使用者の負担とします。

著作権者は、本パッケージを使用したこと、または使用できないことに起因する一切の損害について何らの責任も負いません。 そのような損害の発生する可能性について事前に知らされていた場合でも同様です。なお、ここでいう損害には通常損害、特別損害、偶発損害、間接損害が含まれます (データの消失、又はその正確さの喪失、使用者や第三者が被った損失、他のプログラムとのインタフェースの不適合化、等も含まれますが、これに 限定されるものではありません)。

インストール・起動

動作環境

Windows XP で動作します。

パッケージ内容

dzformatter.zip として
以下のファイルをアーカイブしてあります。

- css（フォルダ）
  - default.css
  - その他 css ファイルいろいろ
- sample（フォルダ）
  - manual（フォルダ）
  - manual.txt（dz 文書のサンプル）
  - その他ファイルいろいろ
- script（フォルダ）
  - getimagesize.exe

- xml2xhtml.js
- スクリプトいろいろ

src（フォルダ）
- ソースファイルいろいろ

dzformatter.exe（アプリケーション本体）
index.css
index.html
LICENSE.TXT
manual.chm（ヘルプファイル）
maunal.pdf（PDF 版のマニュアル）
Readme.txt
script.cfg

インストール

インストールに際しては、適当な解凍ソフトでパッケージを展開するだけです。
レジストリ等は使用していません。

起動

アーカイブを展開して出てきた dzformatter.exe を実行してください。
特別な設定項目はありません。

アンインストール

フォルダをまるごと削除するだけです。

クイックスタート

原稿のテキストデータを dzformatter で単一の HTML に変換、表示する作業を順を追って見ていきます。

出来上がりの HTML ファイルは一つ（ページ分割しない）、スタイルシートは外部スタイルシート（ファイル名「default.css」）とします。

原稿を用意して、dzformatter を起動、その dzformatter から HTML 変換用のスクリプトを起動して HTML ファイルを生成、という手順になります。

材料の準備

- 作業用のフォルダを一つ作成してください。
- 図版を取り扱う場合は、事前に図版ファイル（png, gif, jpeg が扱えます）を用意し、作業用フォルダにコピーしておいてください。
- 何かすぐに試す材料が欲しい場合は、dzformatter の sample フォルダにある、manual というフォルダをみてください。このマニュアルの原稿と図版類があります。

原稿の作成

dzformatter を起動します。
画面の左側が原稿入力エリアで、ここに原稿テキストを入力していきます。
入力された原稿は自動的にフォーマットされて、画面右側にクイックビューが表示されます。

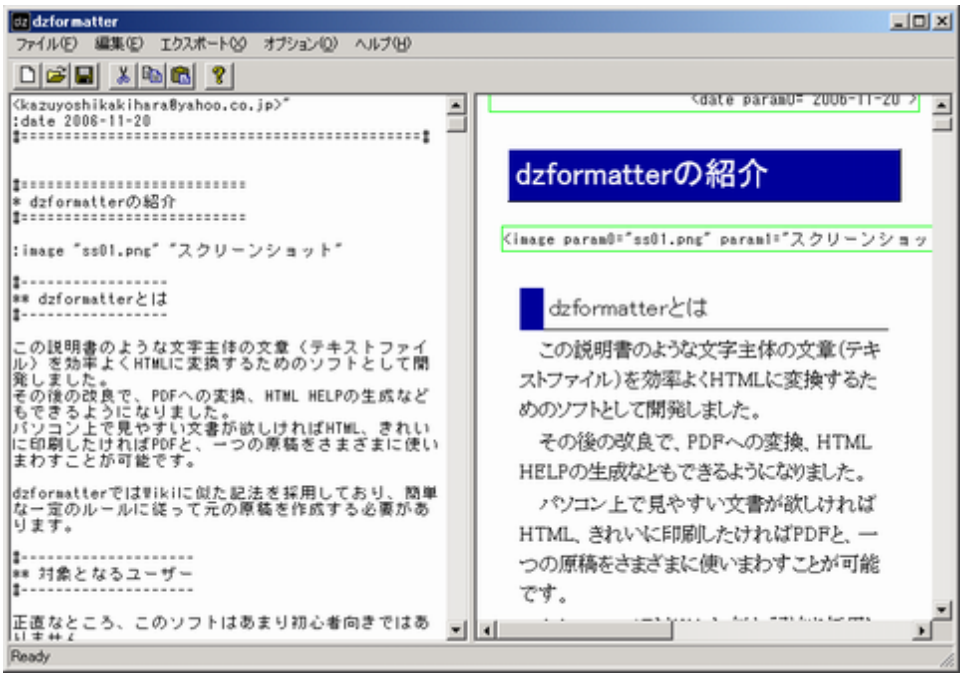

図 2: 画面イメージ

元になる原稿は、以下の簡単なルールに従って作成します。

- 文章は普通にテキストで記述します。
- 行頭に「#」か「;」がつけば、その行はコメント扱いとなり、HTML に変換される際に消えます。

- 行頭に「*」が一つつけば大見出し、「**」と二つつけば中見出し、「***」で小見出しとなります。「*」と見出しの間には半角の空白を一つ入れます。
- 行頭に「-」がつけば箇条書きとなります。「-」と箇条書きの文章の間には半角の空白を一つ入れます。
- 将来に備えて、細かいルールもいくつかありますが、それらについては後述します。とりあえず、文頭では「;#:*->!|」の文字が使えず、文中では円記号 (¥) とアンダースコア (_) が使えないと思っていてください。これら特殊文字の使い方については後述します。

このようにして作成した文書を、以後、dz 文書と呼びます。

dz の「d」は「dictation（口述筆記)」の「d」、「z」は「easy（ = 読みが ez)」の「z」です。

HTML への変換

- dzformatter のメニューバーの「エクスポート」から「単一 HTML(TOC なし)」を選択してください。
- 出力先のファイル名を指定するダイアログが現れますので、先ほど用意した作業フォルダ内に適当な名前を付けて保存してください。
- HTML への変換でエラーが発生しなければ、HTML に変換されたファイルのプレビューが表示されます。

できあがったファイルの取り出し

作業フォルダ内に、成果物の HTML ファイル（拡張子 html）が生成されているはずです。

できあがりの HTML ファイルを見れば大体わかると思いますが（わかってほしいですが)、HTML として再配布するには、この HTML だけでなく、CSS ファイル、画像ファイル類もあわせて取り出す必要があります。

dzformatter の操作

[ファイル]

新規作成

原稿ファイルを新たに書き始めます。

開く

処理の対象となる原稿ファイルを開きます。

上書き保存

現在入力中の原稿ファイルを保存します。

名前をつけて保存

現在入力中の原稿ファイルに名前をつけて保存します。

終了

dzformatter を終了します。

[**編集**]

元に戻す

一つ前の入力を取り消します。

やり直し

「元に戻す」で取り消した作業をやり直します。

切り取り

選択範囲を切り取り、クリップボードにコピーします。

コピー

選択範囲をクリップボードにコピーします。

貼り付け

クリップボードの内容をカーソル位置に貼り付けます。

すべて選択

編集画面内のすべての文字を選択します。

外部エディタ起動

外部テキストエディタを起動して原稿を編集します。
外部テキストエディタで編集が終わったらファイルを保存して終了してください。
原稿ファイルを XML 形式の中間ファイルに変換し、表示します。

[**エクスポート**]

プレビュー

自動プレビューが OFF の場合、プレビューを更新します。

単一 HTML（TOC なし）

原稿ファイルを 1 ページの HTML ファイルに変換します。

単一 HTML（TOC あり）

原稿ファイルを 1 ページの HTML ファイルに変換します。
ページの上部に Table Of Contents を作成します。

マルチ HTML

原稿ファイルを大見出し（原稿ファイルで、*一つで始まる行）ごとに分割して、複数の HTML ファイルに変換します。
見出しだけを集めた目次ファイルもあわせて作成します。

HtmlHelp

HTML Help Workshop **がインストールされている必要があります。**
Microsoft Windows 用の HtmlHelp を作成します。

テキストファイル

原稿ファイルから dz 文書固有の記号類を取り除いたテキストファイルに変換します。

PDF

LaTeX **がインストールされている必要があります。**
また、LaTeX のインストール状態に合わせて、xml2pdf.js を編集し、変数を調整する必要があります。
デフォルトでは、原稿ファイルを A4 横 1 段組の PDF ファイルに変換します。

XML（中間ファイル）

dzformatter から、各種フィルタに出力する際に使用される XML ファイルをそのまま出力します。

[**オプション**]

外部エディタ設定

編集メニューの「外部エディタ起動」で起動するエディタを設定します。
設定を取り消したい場合は、dzformatter.exe と同じフォルダにある「EDITOR」というファイルを削除します。

自動プレビュー

ここにチェックが入っていると、エディタ画面で文字が入力されるたびに自動的に右のブラウザ画面が更新されます。

開発コンセプト

似たようなコンセプトのソフトはいくらでもあるのに、なぜ改めてこのようなソフトを開発したのか、他の類似のソフトとどこが違うのかを説明します。

文書中心であること

既存の商用 HTML 作成ソフトは概してレイアウトに主眼がおかれたものでした。
それはそれでよいのですが、たとえば Windows アプリケーション用に HTML ヘルプファイルを作るような場合には、このレイアウト機能がかえってわずらわしかったりするものです。
インターネットに接続してグラフィカルなブラウザで見るだけが HTML ではない、文書に主眼を置いた HTML を作らなければならない場合もある、ということで、文書主体の HTML を作成する際に力を発揮するソフトが必要だと考えました。

効率が最優先であること

文字主体であるという前提で、GUI で対話的に HTML を作成していくやり方よりも、はるかに効率よく大量の HTML を作成できるような策として dzformatter を考案・開発しました。
dz 文書の d は dictation（口述筆記）の「d」です。口述筆記をするのと同じくらいの効率で文書を作成できればと願いました。

正確な HTML を作成できること

なぜワープロソフトの HTML 出力機能では満足できなかったのか、の答えがこれになります。

正確な HTML であればあるほど、その後の再利用が容易になりますので、単にブラウザで表示できればよしとするのではなく、そこそこ正確な（strict に近い）HTML 文書を作成できるように努力しました。

実際に、後ほどこれが幸いして、PDF への変換等が可能になりました。

商業印刷の原稿として耐えうること

最近は商業印刷において、XML 組版が採用されることがあります。

dz 文書の仕様策定において、XML 変換した状態で、商業印刷でも使える程度の機能は確保したいと考えました。

この都合上、他の wiki 書法にありがちな<pre>タグ相当の機能やテーブル作成機能を割愛することにしました。

<pre>タグやテーブルに関してはパソコンの環境の影響を受けやすいために、別途図版として組むようにしなければ、最終出力でそれなりの見栄えになるという確証が得られなかったためです。

改造が容易であること

当初はすべて C++で開発していたのですが、こういうものはユーザーがいろいろ自由に手を加えられたほうが便利だろうと考えなおし、dzformatter を初めて世間一般に公開した際（Version 0.1.0）に、基本部分は改めて JScript で書き直しました。

その後、ユーザビリティを考慮した結果、中間 XML ファイルまでは C++で生成し、それを JScript で好きなフォーマットに変換するという構成にしました。

最後まで C++にせず、中間ファイルをはさむようにしたのは、そうしたほうが、後々の改造が容易になるからです。

こういった趣旨ですから、同梱のスクリプト類は自由に改造・再配布していただいて結構です。

インストール・アンインストールが容易であること

気軽に使えていらなくなったらすぐ捨てられるようなソフトを目指しました。

そのためには、.Net フレームワーク必須などと言うのは、論外でした。

Perl だの Python だの Ruby だのといったインタープリタのインストールが前提になるのも、なるべく回避したいと思いました。

そこで、最初から Windows XP で利用できる JScript 主体でソフトを開発し、ユーザーインターフェース部分のみを C++で開発することにしました。

なお、Perl.exe をパッケージに同梱してしまって、Perl で開発するというアイデアはありだと思っていますので、将来的には、Perl でクロスプラットフォームということを考えるかも知れません。

元原稿がテキストのままでも読みやすいこと

いちいち dzformatter にかけなくても、人の目で元の文書が読めるならば、PDA などで原稿を作成することが可能ですから。

dz 文書の仕様

基本ルール

以下に述べる特殊記号を除き、文章はそのまま変換されます。
文字コードは SHIFT-JIS、改行は CR+LF です。

文中で使う特殊記号

文中で特殊な意味を持つ記号は「¥（円記号）」および「_（アンダースコア）」の 2 つです。この説明書を原稿のままのテキストファイルで見ると、これらの記号の前に一つ余計な円記号がついているのが分かると思います。HTML や PDF に変換して見ている人には、記号前の一つ余計な円記号は見えません。

円記号（エスケープ文字）

円記号はエスケープ文字です。
文中で円記号を使うと、その円記号が消え、次に現れた特殊記号が、特殊記号としての意味を無くし、普通の文字として変換されます。文中で円記号を文字として使いたい場合は円記号を二つ並べればいいわけです。
文頭で「*」を使いたい場合には、円記号を前に置き、「¥*」とすればいいわけです。

アンダースコア（ルビ、強調、その他）

特定の文字列にルビをふったり、文字列を強調したりしたい場合にアンダースコアを使用します。
ルビをふるには、アンダースコアと [ ]（ブラケット）を使用します。
ルビをふりたい文字列のところで_文字列 [もじれつ] とすると、「もじれつ」の部分が「文字列」にかかるルビに変換されます。
文字列を強調するには、アンダースコアと*（アスタリスク）を使用します。
強調したい文字列のところで_文字列*とすると、「文字列」の部分が強調されます。色がつくのか、太字になるのかといった、どのような強調がされるかは、ブラウザその他の設定によります。
また、dzformat の拡張にもアンダースコアを使用することがあります。
例えば_文字列 {link ”http://www.example.com”} とすると、「文字列」の部分に http://www.example.com へのハイパーリンクが張られたりします（現時点ではまだ隠し機能に近い扱いです）。

行頭で使う特殊記号

「;(セミコロン)」「#(シャープ)」　コメント

; あるいは#で始まる行はコメント行です。変換されません（変換されずに消えます)。

「:(コロン)」　1 行コマンド

:で始まる行は、1 行で完結するコマンドです。
:とコマンド名の間にスペースは空けません。

:image "ファイル名" "キャプション"　その行に画像を貼りこむことができます。

:title "タイトル"　文書のタイトルを指定することができます。

:author "著者名"　文書の著者名を指定することができます。

:date "日付"　文書の作成日付を指定することができます。

「*(アスタリスク)」　見出し

*で始まる行はセクション区切りになり、*に続く文字がセクションのタイトルになります。
「*」で大見出し、「**」で中見出し、「***」で小見出しとなります。
*と見出し文字の間には半角スペースを一つ挟みます。

「-(ハイフン)」　箇条書き

-で始まる行は箇条書きになります。
-と箇条書き文字の間には半角スペースを一つ挟みます。
他の行頭記号が現れるか、空行が現れるまでが、一まとまりの箇条書きブロックになります。
以下のように「-」を複数並べることで入れ子の箇条書きを作ることができます。
- 箇条書き
- 箇条書き
– 入れ子の箇条書き
– 入れ子の箇条書き
- 箇条書き

「>(グレーターザン)」　引用

>で始まる行は引用文です。
>と引用文の間には半角スペースを一つ挟みます。
他の行頭記号が現れるか、空行が現れるまでが、一まとまりの引用ブロックになります。
以下のように「>」を複数並べることで入れ子の箇条書きを作ることができます。
> 引用文
> 引用文
>> 入れ子の引用文
>> 入れ子の引用文
> 引用文

「!」（エクスクラメーションマーク）　見出し付箇条書き

!で始まる行は見出し付きの箇条書きになります。
!と見出し文字の間には半角スペースを一つ挟みます。
見出しと箇条書き文書は「:」（コロン）で区切ります。
他の行頭記号が現れるか、空行が現れるまでが、一まとまりの箇条書きブロックになります。
! 見出し:箇条書き
! 見出し:箇条書き
! 見出し:箇条書き

「|」（パイプ）

| で始まる行は汎用のブロックになります。なぜこのようなものを用意したかは「応用」の項目で別途説明します。

応用

デザインの変更

css フォルダ内にいくつかのスタイルシートを用意してあります。

default.css gray.css と同じ内容です。

gray.css モノクロ基調のデザインです。

blue.css 青基調のデザインです。

green.css 緑基調のデザインです。

red.css 赤基調のデザインです。

fsf_gray.css gray.css と同じ配色ですが、フォントサイズが固定になっています。

fsf_blue.css blue.css と同じ配色ですが、フォントサイズが固定になっています。

fsf_green.css green.css と同じ配色ですが、フォントサイズが固定になっています。

fsf_red.css red.css と同じ配色ですが、フォントサイズが固定になっています。

htmlhelp.css HTML HELP 用に特化した CSS です。

出来上がりの文書の論理構造まで変更したい場合は、変換スクリプトそのものを書き換えてください。

中間ファイルについて

dzformatter では、dz 文書を一旦、整形式の XML ファイルに変換し、それから HTML 形式に変換するという手法をとっています。

後々の再利用を考えて、このような仕様にしました。

中間ファイルを取り扱うスクリプトを作成することで、dzformatter の機能を大きく変えることが可能です。

また、中間 XML ファイルに対して直接 XSLT や css を適用するというのもありです。

なお、現在のところ、XML スキーマは用意できておりません。

タグにクラス名、ID を付加

行頭に「*」や「-」などの特殊記号を使っている場合、たとえば「**(class1#id1) ○○○」というように、記号に続いて丸カッコ（パーレン）、クラス名、#（シャープ）記号、ID を補記することにより、HTML に変換後、そのタグに class=”class1” id=”id1”といったアトリビュートを付加することができます。

これにより、スタイルシートで部分的にデザインを変更したり、出来上がりの HTML を再加工する際の目印にしたりといったことが可能になります。

PDF に変換

配付状態の dzformatter では A4 横組み 1 段の機能しかありませんが、縦組み横組み 2 段組、A4・A5 サイズ出力など、元の原稿に触れることなく、変換スクリプトの起動オプションの調整だけで思うがままです。

事前の準備

LaTeX のセットアップが完了していることが前提になります。

LaTeX のインストールについては、三重大学の奥村先生のホームページが参考になります。また、厳密に調査したわけではありませんが、LaTeX の場合、ファイル名に空白文字が含まれているとうまく動作しないかも知れません。

ファイル名はなるべく半角アルファベットと数字だけにしておくことをお勧めします。

変換スクリプトの設定

script フォルダ内の xml2pdf.js を使用します。

xml2pdf.js を適当なテキストエディタで開き、30 行目前後にある texbasepath の値を LaTeX のインストール先に合わせて正確に設定してください。

この変数の値が LaTeX の環境変数の基準となります。

上記の奥村先生のホームページのとおりに Windows 用の LaTeX をインストールしたのであれば、

var texbasepath=”C:/usr/bin”

となります。

変換スクリプトの実行

ここまでくれば、後は HTML への変換と手順は同じです。

作業用フォルダを作り、必要な画像データ等をとりまとめたうえで、dzformatter で変換処理してください。

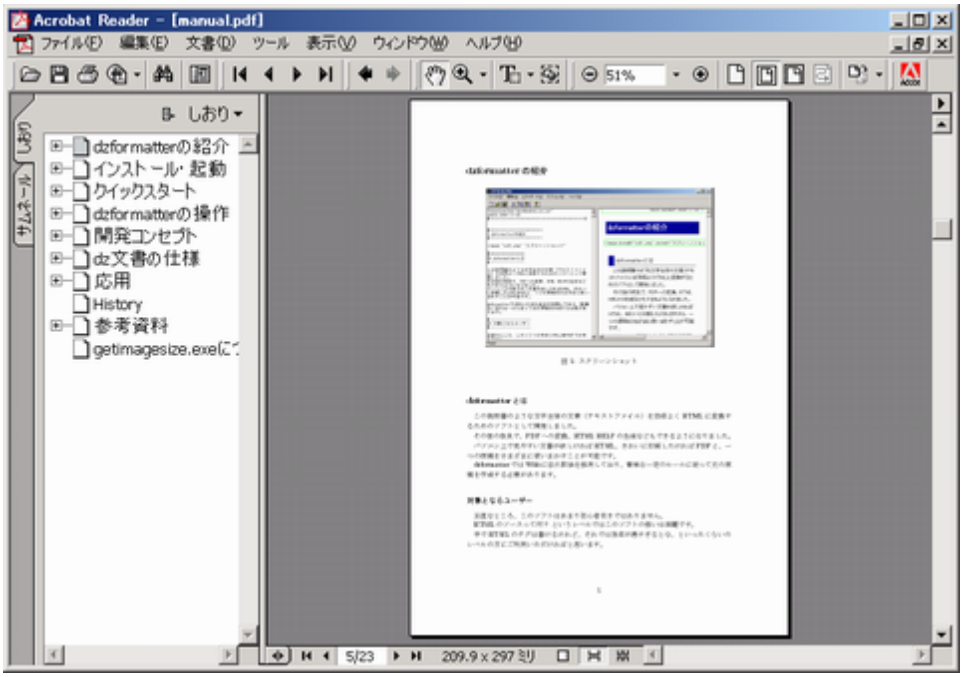

図 3: PDF 変換の例

Html Help を作成

dz 文書から Html Help を作成することができます。

HTML Help Workshop の準備

Microsoft 社のホームページから HTML Help Workshop を入手し、インストールしておいてください。

変換スクリプトの設定

script フォルダ内の xml2xhtml_multi.js を使用します。

xml2html_multi.js を適当なテキストエディタで開き、25 行目前後にある htmlhelp_hhc の値を、HTML Help Workshop のインストール先に合わせて正確に設定してください。

デフォルトのインストールであれば、

var htmlhelp_hhc = ”C:¥¥Program Files¥¥HTML Help Workshop¥¥hhc.exe”;

となります。

変換スクリプトの実行

dzformatter で整形すれば、元のファイル名の拡張子を hhp に変えたファイルが出来、さらにそれが hhc.exe でコンパイルされ、chm ファイルが出来上がります。

対象となる Html Hel ファイルが開いているとコンパイルに失敗しますので、注意してください。

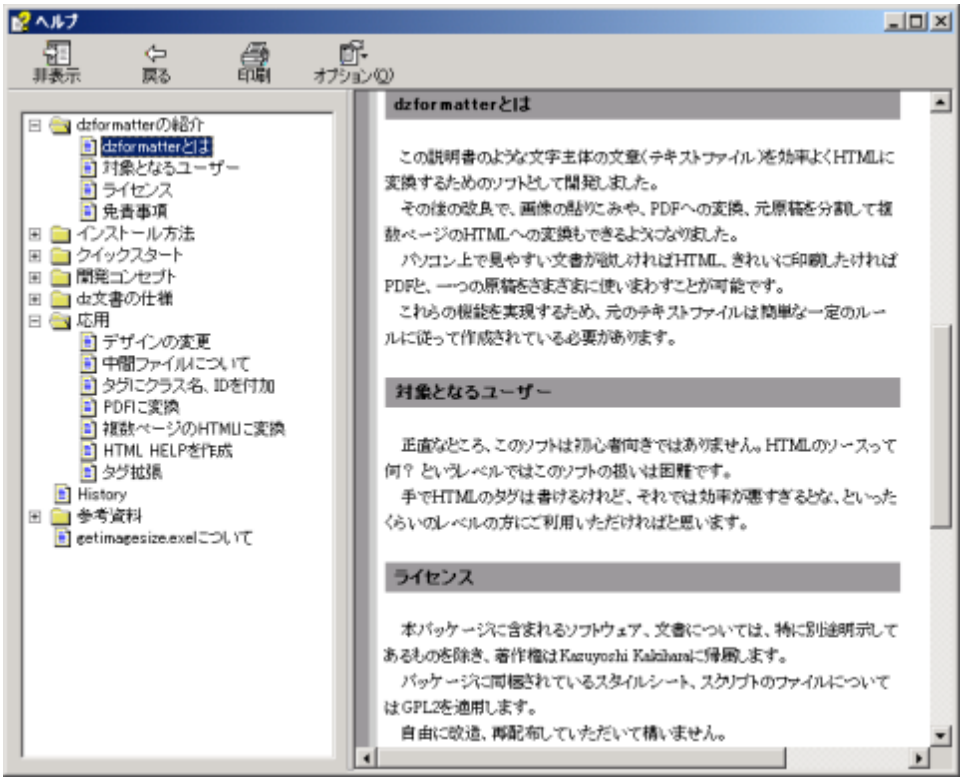

図 4: Html Help ファイルの例

タグ拡張

行頭に「|」（パイプ記号）を置くと、ユーザー定義のタグが作成されます。
具体的には
|{ タグ名 (クラス名#ID) パラメータ 0 パラメータ 1}
| タグ内の要素 1
| タグ内の要素 2
とすると、中間ファイルにおいて
<タグ名 class=”クラス名” id=”ID” param0=”パラメータ 0” param1=”パラメータ 1”>
<div_inner>タグ内の要素 1</div_inner>
<div_inner>タグ内の要素 2</div_inner>
</タグ名>
という XML ノードが生成されます。
これを受ける形で xml2html.js を改造すれば、自分の好きなブロック型のタグを dzformatter に追加することができます。
また文中で_タグ内の要素 { タグ名 (クラス名#id) パラメータ 0 パラメータ 1} と書くと、中間ファイルにおいて
<タグ名 class=”クラス名” id=”ID” param0=”パラメータ 0” param1=”パラメータ 1”>タグ内の要素</タグ名>
という XML ノードが生成されます。
やはりこれを受ける形で xml2html.js を改造することにより、自分の好きなインライン型のタグを dzformatter に追加することができます。

スクリプトの起動オプション変更

dzformatter.exe と同じフォルダに script.cfg というファイルがあります。
1 行ごとに変換用スクリプトをメニューに表示する際の名前、コマンド名、起動パラメータなどが記述されており、dzformatter の起動時に読み込まれるようになっています。
書式は 1 行ごとに
タイトル<tab>スクリプトホスト名<tab>スクリプト名<tab>コマンドラインオプション<tab>拡張子
となっており、各項目はタブで区切られています。

タイトル dzformatter のメニューに表示される名称です。

スクリプトホスト名 JScript や VBScript なら cscript.exe を設定します。Perl なら perl.exe を絶対パスで指定することになります。

スクリプト名 xml2xhtml.js などのスクリプト名です。

起動オプション スクリプトを起動する際のスイッチ項目です。

拡張子 変換後のファイルの拡張子です。

History

- 2006.10.01 Version 0.1.0 公開
- 2006.10.28 Version 0.1.0a 公開（テキストファイル生成用スクリプトを追加）
- 2006.11.25 Version 0.2.0 公開（ユーザーインターフェース一新）
- 2006.11.27 Version 0.2.1 公開（ユーザーインターフェース改良）
- 2007.03.18 Version 0.2.2 公開（メモリ確保に関する不具合および UI 関連の不具合修正）
- 2007.04.11 Version 0.2.3 公開（getimagesize.exe のバグ修正）

参考資料

wiki 各種

プレーンテキストを HTML に変換するというアイデアそのものは wiki から借用しました。

単に自分一人で使う分には、wiki で十分でしたし、wiki をベースに自分専用の CMS を作ってみたりもしました。

ですが、グループワークなどで、複数の人間で HTML を作ろうとした場合、wiki ではセットアップがあまりに大げさ過ぎると思いました。

WWW サーバーがいらないように改造したとしても、Perl なり PHP なりをセットアップしないことには動作しないのですから。

参考:

YukiWiki ホームページ（Perl による wiki）

PukiWiki ホームページ（PHP による wiki）

LaTeX および ASCII EWB

普通のテキストファイルにコマンドを書き込んでソースファイルを作成し、それを TeX でコンパイルすることにより美しい印刷物を作成します。

LaTeX は TeX を使いやすくするためのプリプロセッサ、ASCII の EWB（Editor's Work-Bench）は、その LaTeX をさらに使いやすくするためのフロントエンドといったところでしょうか。

印刷物の生成を主体に考えた場合これはなかなかのソリューションではないかと自分では思ったのですが、ワープロしか知らない人にとってはかなりとっつきにくい代物だったようです。

結局、dzformatter に PDF 生成機能を付加する際、dz 文書から LaTeX を経由することで期待通りの成果を得られるようになりましたので、LaTeX 関連の調査は無駄にはなりませんでした。

参考:
三重大学奥村先生の TeX Wiki
ASCII Editor's WorkBench ホームページ

はてな記法

wiki の一種とも言えなくはないですが、これを元に印刷物を作るというサービスもあるようなので、他の wiki とは別扱いで研究しました。

HTML にも変換できて、PDF にも変換できて、という意味では、私の好みに最も近いものでした。

ですが、これはそもそもの企画の趣旨が違っているものなので、参考にとどめるしかありませんでした。

参考:
はてな ホームページ

reStructuredText

基本的なコンセプトは Wiki や LaTeX と同じ、簡易なマークアップ言語です。

ただし、パーサーを通して変換しなくても、もとのソースのままでも人間の目で余裕で読めるのが特長です。

わたしの要望をほとんど満たす規格だったのですが、残念なことにパーサーが Python で書かれているのです。

これをグループで使うには、皆のパソコンに Python をインストールして回らないといけなくなるので、利用を断念しました。

参考:
reStructuredText ホームページ

DocBook JepaX

DocBook は構造化文書の作成を主目的にした SGML/XML です。

JepaX は「JEPA 出版データフォーマット標準化研究委員会」が主体となって策定した、印刷物の制作を主目的にした XML です。

このどちらかの形式に変換できれば、後々の応用が楽だろうと思いました。

dz 文書は、タグの名称こそ違え、かなり JepaX を意識した構造になっています。

参考:
DocBook ホームページ
JepaX ホームページ

XSL Formatter

XML を PDF 化するための商用ソフトウェアです。アンテナハウス株式会社が制作しています。

dz 文書をうまく変換してこのソフトに流し込むことができれば、かなり可能性が広がるのではないかと思いつつも、個人が趣味で買うレベルの商品ではありませんので、今のところ、アンテナハウスさんのホームページを眺めているばかりです。

アンテナハウス ホームページ

getimagesize.exe について

dzformatter には getimagesize.exe というユーティリティが同梱されています。

これは、jpeg, png, gif のファイルについて、画像の大きさをピクセル単位で取得するプログラムです。

getimagesize.exe <filename>

とすれば標準出力の 1 行目に画像の幅が、2 行目に高さが出力されます。

何らかのエラーが生じた場合は、それぞれ、負の数字が出力されます。

適当につくったものですが、ライセンスとしてとりあえず GPL2 を適用することとし、ソースを同梱しておきます。