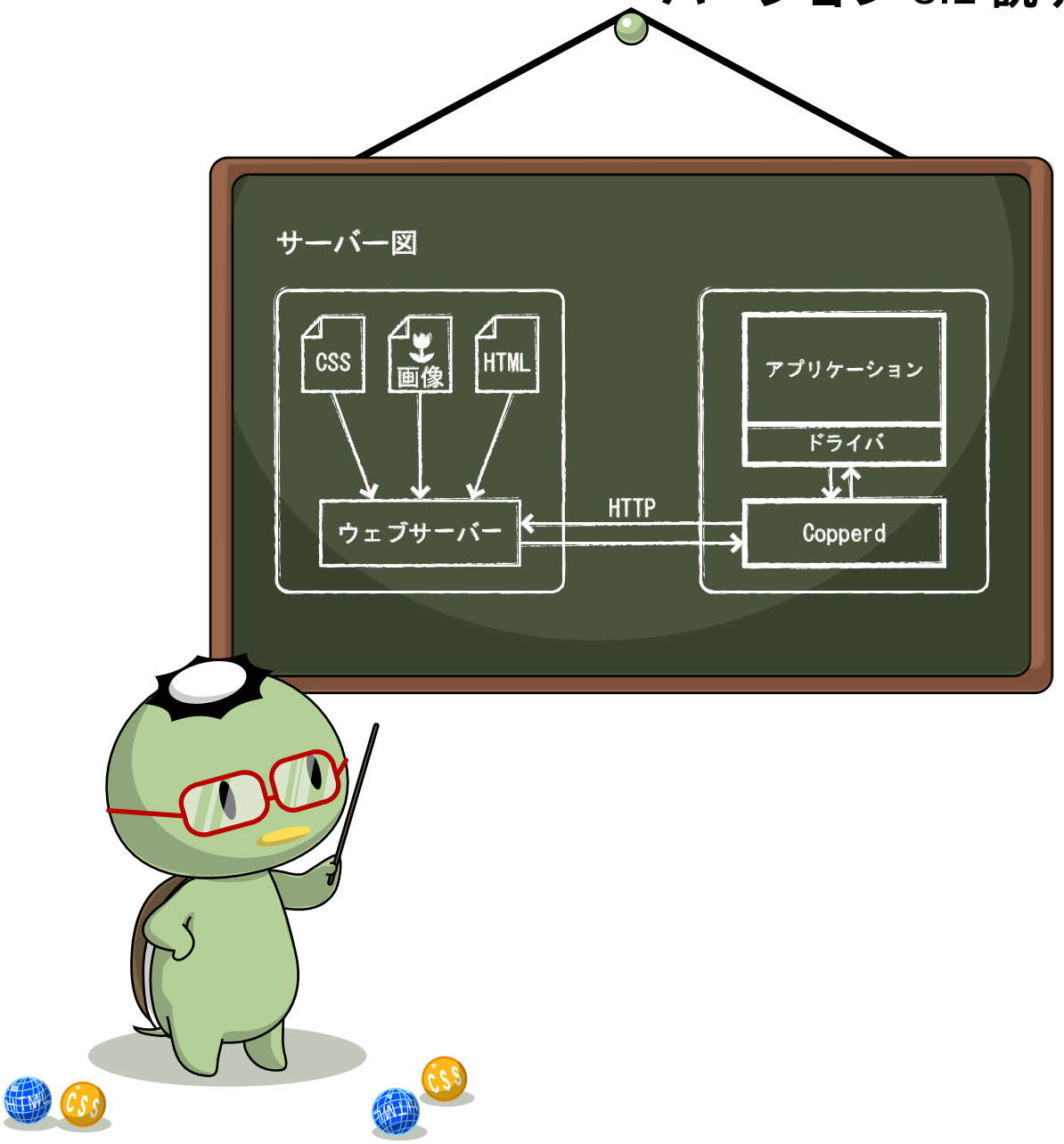
HTML→PDF変換サーバー

Copper PDF

バージョン 3.1 説明書



発売元 株式会社 GNN
2026-11-20

NOTICE

Copper PDF ©2011-2026 Zamasoft. All rights reserved.

This product includes software developed by Andy Clark.

This product includes software developed at The Apache Software Foundation (http://www.apache.org/).

This software contains code from the World Wide Web Consortium (W3C) for the Document Object Model API (DOM API), SVG Document Type Definition (DTD) and the The Simple API for CSS (SAC API).

目次

1.Copper PDF入門 1
1.1 Copper PDFの概要 1
1.1.1 Copper PDFとは 1
1.1.2 なぜCopper PDFが必要か 2
1.2 動作環境 4
1.2.1 サーバー 4
1.2.2 プログラミングインターフェース(API) 5
新インターフェース(CTIP 2.0) 5
Javaドライバ / transcode Antタスク 5
Perlドライバ 5
PHPドライバ 5
.NETドライバ 5
HTTP / RESTインターフェース 6
1.3 機能一覧 7
1.3.1 入力データ形式 7
ドキュメント 7
スタイルシート 7
ベクター画像データ 7
ラスター(ビットマップ)画像データ 8
1.3.2 PDF出力 8
1.3.3 画像出力 8
1.3.4 フォント関連機能 9
1.3.5 HTTP接続 9
1.3.6 印刷サポート 9
1.3.7 目次・ページ参照 9
1.3.8 プログラムインターフェース 10
1.3.9 その他の機能 10
1.4 とりあえず使ってみよう 11
1.4.1 Copper PDFのインストール 11
1.4.2 ウェブインターフェースでHTMLを変換する 12
1.4.3 コマンドラインアプリケーションでHTMLを変換する 13
1.4.4 プログラムからHTMLを変換する 14

2.管理者ガイド 18
2.1 セットアップ 18
2.1.1 Java実行環境 のインストール 18
2.1.2 Copper PDF の配布パッケージ 18
2.1.3 Docker（Linux） 19
2.1.4 Windows 20
付属のプログラムを使ってサービスをインストールする 20
Java Service Wrapperを使ってサービスをインストールする[2.1.4] 21
サービスの管理と動作状態の確認 21
2.1.5 Red Hat Enterprise Linux(RHEL) / CentOS 22
Copper PDFサーバーの起動・停止 23
2.1.6 Debian/Ubuntu 23
Copper PDFサーバーの起動・停止 24
2.1.7 FreeBSD 24
Copper PDFサーバーの起動・停止 25

2.1.8 その他の環境 25
2.1.9 ディレクトリ構成 26
アーカイブ内のディレクトリ構成 26
RPMまたはDebianパッケージの構成 26
2.1.10 ライセンスキー・ファイルの配置 27
2.1.11 旧バージョンからのアップデート 27
2.2 Copper PDFのツール 28
2.2.1 copper コマンドラインアプリケーション 29
形式 29
概要 29
オプション 29
説明 30
入力について 30
出力について 31
javaコマンドで実行する方法 31
2.2.2 copper-webapp ウェブアプリケーション 31
概要 31
javaコマンドで実行する方法 32
2.2.3 copperd ドキュメント変換サーバー 33
形式 33
概要 33
オプション 33
説明 35
javaコマンドで実行する方法 35
2.3 設定ファイル 37
2.3.1 Copper PDFサーバーの動作設定(copperd.properties) 38
2.3.2 SSL/TLSの設定 39
秘密鍵とCSRを用意する 40
サイト証明書を用意する 40
秘密鍵とサイト証明書をキーストアに格納する 40
2.3.3 ログの設定(logging.properties) 41
2.3.4 アクセス制御の設定(access.txt) 42
2.3.5 profilesディレクトリ 42
2.3.6 デフォルトの入出力プロパティ(default.properties) 42
2.3.7 fontsディレクトリ 43
2.4 フォントの設定 44
2.4.1 フォント設定の反映 45
2.4.2 ドキュメント中でのフォントの利用 45
2.4.3 デフォルトのフォント 46
2.4.4 フォントの種類 46
2.4.5 コアフォント 47
2.4.6 CIDフォント 51
埋め込みフォント 52
CID Identity 52
CID-Keyedフォント 52
PANOSEコード 53
参考情報 53
フォント幅情報ファイル 54
2.4.7 フォントファイルの種類 55
2.4.8 フォント設定ファイル 55
コアフォントのエンコーディング(encodings要素) 55
encodingsに含まれる要素 56

cmapファイル(cmaps要素) 56
cmapsに含まれる要素 56
コアフォント(core-fonts要素) 56
core-fontsに含まれる要素 56
letter-font 56
symbol-font 57
letter-fontおよびsymbol-fontに含まれる要素 57
CIDフォント(cid-fonts要素) 58
cid-fontsに含まれる要素 58
cid-keyed-font 58
font-file 59
font-dir 60
system-font 60
all-system-fonts 61
cid-keyed-font, font-file, system-fontに含まれる要素 62
一般フォントファミリ(generic-fonts要素) 62
generic-fontsに含まれる要素 62
2.4.9 フォント設定ファイルの設定例 63
デフォルトのフォントの変更 63

3.開発者ガイド 64
3.1 プログラムインターフェースの概要 64
3.1.1 アプリケーションからCopper PDFの機能を使うには 64
3.1.2 通信の手順 65
3.1.3 copperdへの接続・認証 65
3.1.4 メッセージハンドラの設定 66
3.1.5 プログレスリスナの設定 66
3.1.6 出力先の設定 66
3.1.7 変換結果の出力先の設定 66
3.1.8 入出力プロパティの設定 66
3.1.9 URIの解決と関連ファイル（リソース）の取得 66
リソースにサーバーからアクセスする場合 68
URIパターン 69
リソースを事前にサーバーに送る場合 70
ソースリゾルバを使う場合 70
3.1.10 設定のリセット 70
3.1.11 ドキュメント本体の送信または変換対象のドキュメントの指定 71
ドキュメント本体をサーバーに送る 71
ドキュメント本体にサーバーからアクセスする場合 71
3.1.12 複数の結果の結合[3.0.0] 71
3.1.13 変換処理の中断 71
3.2 CTIP 2.0 インターフェースの概要 73
3.2.1 接続情報 73
3.2.2 メッセージコード 73
3.2.3 サーバー情報 73
http://www.cssj.jp/ns/ctip/version 74
http://www.cssj.jp/ns/ctip/output-types[3.0.0] 74
http://www.cssj.jp/ns/ctip/fonts[3.0.4] 74
3.2.4 CTIP 2.0 プロトコルの仕様 75
3.3 Javaドライバ２ 76
3.3.1 概要 76
3.3.2 ドライバの準備 76

3.3.3 タイムアウトの設定 77
3.3.4 APIの概要 78
サーバーへの接続・認証 78
サーバー情報の取得 78
メッセージハンドラ・プログレスリスナの設定 78
出力先の設定 78
プロパティの設定 78
ソースリゾルバの設定 78
リソースの送信 78
本体の送信・変換 79
複数の結果の結合 79
処理の中断・リセット・通信の終了 79
3.3.5 サンプル 79
3.3.6 プログラミングのポイント 82
CTISessionHelperの利用 82
繰り返し処理 82
出力先(Results)の設定 83
サーバーから要求されたリソースの送信(SourceResolver) 83
MetaSource 84
複数の結果の結合 84
abortによる中断 84
3.3.7 サーブレット/JSPでの利用 85
3.3.8 ソースコード 90
3.3.9 Copper PDFのライブラリに直接アクセスする 90
3.3.10 JRubyを使う場合 91
3.3.11 Jythonを使う場合 93
3.4 Perlドライバ２ 95
3.4.1 概要 95
3.4.2 ドライバの準備 95
3.4.3 APIの概要 95
サーバーへの接続・認証 95
サーバー情報の取得 96
メッセージハンドラ・プログレスリスナの設定 96
出力先の設定 96
プロパティの設定 96
ソースリゾルバの設定 96
リソースの送信 96
本体の送信・変換 96
複数の結果の結合 96
処理の中断・リセット・通信の終了 97
3.4.4 サンプル 97
3.4.5 プログラミングのポイント 98
Content-Type，Content-Length ヘッダの出力 98
他のプログラムを呼び出して変換する 99
繰り返し処理 99
出力先の設定 99
サーバーから要求されたリソースの送信 100
複数の結果の結合 100
3.4.6 ソースコード 101
3.5 PHPドライバ２ 102
3.5.1 概要 102
3.5.2 ドライバの準備 102

3.5.3 APIの概要 102
サーバーへの接続・認証 103
サーバー情報の取得 103
メッセージハンドラ・プログレスリスナの設定 103
出力先の設定 103
プロパティの設定 103
ソースリゾルバの設定 103
リソースの送信 103
本体の送信・変換 103
複数の結果の結合 104
処理の中断・リセット・通信の終了 104
3.5.4 サンプル 104
3.5.5 プログラミングのポイント 105
Content-Type, Content-Length ヘッダの出力 105
他のプログラムを呼び出して変換する 106
繰り返し処理 106
出力先の設定 106
サーバーから要求されたリソースの送信 107
複数の結果の結合 107
3.5.6 ソースコード 108
3.6 .NETドライバ 109
3.6.1 概要 109
3.6.2 ドライバの準備 109
3.6.3 タイムアウトの設定 110
3.6.4 APIの概要 110
サーバーへの接続・認証 110
サーバー情報の取得 110
メッセージハンドラ・プログレスリスナの設定 110
出力先の設定 110
プロパティの設定 111
ソースリゾルバの設定 111
リソースの送信 111
本体の送信・変換 111
複数の結果の結合 111
処理の中断・リセット・通信の終了 111
3.6.5 サンプル 111
3.6.6 プログラミングのポイント 115
Utilsの利用 115
繰り返し処理 115
出力先(Results)の設定 116
サーバーから要求されたリソースの送信(SourceResolver) 116
SourceInfo 118
複数の結果の結合 118
Abortによる中断 118
3.6.7 ソースコード 119
3.7 Rubyドライバ 120
3.7.1 概要 120
3.7.2 ドライバの準備 120
3.7.3 APIの概要 120
サーバーへの接続・認証 120
ユーティリティ 121
サーバー情報の取得 121

メッセージハンドラ・プログレスリスナの設定 121
出力先の設定 121
プロパティの設定 121
ソースリゾルバの設定 121
リソースの送信 121
本体の送信・変換 121
複数の結果の結合 121
処理の中断・リセット・通信の終了 122
3.7.4 サンプル 122
3.7.5 プログラミングのポイント 123
Content-Type，Content-Length ヘッダの出力 123
繰り返し処理 124
出力先の設定 124
サーバーから要求されたリソースの送信 124
複数の結果の結合 125
3.7.6 ソースコード 125
3.8 Pythonドライバ 126
3.8.1 概要 126
3.8.2 ドライバの準備 126
3.8.3 API の概要 126
サーバーへの接続・認証 126
サーバー情報の取得 126
メッセージハンドラ・プログレスリスナの設定 127
出力先の設定 127
プロパティの設定 127
ソースリゾルバの設定 127
リソースの送信 127
本体の送信・変換 127
複数の結果の結合 127
処理の中断・リセット・通信の終了 127
3.8.4 サンプル 128
3.8.5 プログラミングのポイント 129
Content-Type，Content-Length ヘッダの出力 129
繰り返し処理 130
出力先の設定 130
サーバーから要求されたリソースの送信 130
複数の結果の結合 131
3.8.6 ソースコード 132
3.9 Node.jsドライバ 133
3.9.1 概要 133
3.9.2 ドライバの準備 133
3.9.3 API の概要 134
サーバーへの接続・認証 134
サーバー情報の取得 134
メッセージハンドラ・プログレスリスナの設定 134
出力先の設定 134
プロパティの設定 134
ソースリゾルバの設定 135
リソースの送信 135
本体の送信・変換 135
複数の結果の結合 135
処理の中断・リセット・通信の終了 135

3.9.4 サンプル 135
3.9.5 プログラミングのポイント 138
async/awaitによる完了待機 138
繰り返し処理 138
出力先の設定 138
サーバーから要求されたリソースの送信 138
複数の結果の結合 139
3.9.6 ソースコード 140
3.10 transcode Antタスク 141
3.10.1 Antタスクの概要 141
3.10.2 transcode タスクの使用方法 141
3.10.3 うまく動かない場合（ローカルマシンで実行する場合） 143
ライセンスが認証されない場合 143
3.11 HTTP/RESTインターフェース 144
3.11.1 概要 144
3.11.2 アクションの実行 144
3.11.3 ウェブブラウザからのアクセス 147
3.11.4 Ruby (httpclient) 147
3.11.5 Python (urllib2, urllib) 149
3.11.6 C# (.NET WebClient) 150
3.11.7 ASP.NET (C#, VisualBasic) 152
3.11.8 4th Dimension (Internet Commands) 154
3.11.9 その他のサンプルについて 157
3.12 HTTPクライアント機能 158
3.12.1 BASIC認証またはDigest認証 158
3.12.2 プロクシの設定 159
3.12.3 HTTPヘッダの送信 159
3.12.4 参照元(Referer)の送信 160
3.12.5 クッキーの送信 160
3.12.6 タイムアウトの設定 161
3.13 出力制限機能 162
3.13.1 ページ数の制限 162
3.13.2 データサイズの制限 162

4.デザイナーガイド 163
4.1 Copper PDFによる文書のレイアウト 163
4.1.1 Copper PDFで文書をレイアウトするには 163
4.2 入出力プロパティ 164
4.3 対応する入力ファイル 166
4.3.1 HTML/XMLの処理 166
ドキュメントの判別 166
キャラクタ・エンコーディング 166
文書情報 167
XSLTスタイルシートの適用 168
4.3.2 画像 168
Copper PDFがサポートする画像 168
他の画像形式の利用 168
ラスター(ビットマップ/ピクセルマップ)画像 169
GIF / PNG画像 169
JPEG / JPEG 2000画像 169
その他の画像 169
画像の解像度 169

SVG画像 169
SVG画像ファイルの参照 169
インラインSVG 170
画像を読み込めない場合 170
4.3.3 EPUB電子書籍 171
4.4 出力するファイル形式 172
4.4.1 PDFの出力 172
PDFのバージョンと機能 172
暗号化 172
ファイルの添付 174
PDFの圧縮形式 174
PDF中の画像の圧縮形式 174
PDFの表示環境のキャラクタ・エンコーディング 175
作成・更新時刻、ファイルIDの設定 175
すかし 176
印刷時だけ、または画面表示だけすかしを表示する 178
PDF/A-1bに準拠したファイルの出力 179
PDFビューワの表示設定[3.0.2/2.1.11] 179
PDFの表示の際に実行されるJavaScript[3.0.2/2.1.11] 179
4.4.2 画像の出力 180
画像出力の制約 180
画像出力の解像度 180
4.4.3 SVGの出力 180
4.5 一般的なブラウザとの互換性 181
4.5.1 互換性モードの切り替え 181
4.5.2 標準モードとmsieモードの違い 181
CSSで数字のクラス名が認識される 181
CSSで':'の代わりに'='が使用できる 182
CSSの色指定で#を省略しても認識される 183
CSSの長さ指定で単位を省略しても認識される 183
フォームの前後にマージンが設定される 183
段落と見出しのマージン上下のマージンをなくす場合がある 183
ボックスの幅または高さに100%が指定された場合、外側のボックスをはみ出さない 183
絶対位置指定ボックス、浮動ボックスの大きさが内容により拡張される 184
通常のフローのボックスが、幅が指定されたボックスにより拡張される 184
幅がautoの固定レイアウトテーブルが固定レイアウトのまま処理される 184
テーブル行に対するmax-height [css]が適用されない 184
ブロックに対するline-height [css]による高さが確保されない 184
全角スペースの間で折り返しされない 184
input要素の高さが強制される 185
マージンの設定に関わらず中央寄せされる 185
匿名のテーブルセルにより補完されない 185
インラインボックスにwidth [css]が適用される 185
text-align [css]がブロックの配置にも適用される 185
テーブルカラムのプロパティがセルに継承される 186
width [css]が設定されたテーブルセルには{white-space: nowrap;}が適用されない 186
4.5.3 既知の制限事項 186
ＭＳ明朝系フォントの文字幅について 186
サポートしていないCSSプロパティ 187
4.5.4 自動レイアウトテーブル 187

4.6 XML/HTMLの拡張機能 188
4.6.1 見出し 188
ブックマーク 188
現在ページのセクション 188
4.6.2 目次の生成 188
4.6.3 リンクとフラグメント 190
フラグメント識別子によるリンク 190
-cssj-page-ref関数 190
4.6.4 注釈 191
4.6.5 XML中でHTMLの要素と属性を使用する 191
4.6.6 ルビ[3.0.0] 192
4.7 CSSによるドキュメントのレイアウト 194
4.7.1 スタイルシートの型 194
4.7.2 メディアタイプ 194
4.7.3 ドキュメント中にスタイルシートを記述する 194
HTMLのstyle属性 194
HTMLのstyle要素 195
jp.cssj.stylesheet処理命令 196
4.7.4 外部のCSSの使用 197
HTMLのlink要素 197
CSSの@import指示子 197
xml-stylesheet処理命令 198
4.7.5 デフォルトのスタイルシート 198
4.7.6 長さの単位 199
4.8 CSSの拡張機能 200
4.8.1 名前空間 200
4.8.2 ページカウンタ 201
4.8.3 全角数字と漢数字による箇条書き番号 201
4.8.4 禁則処理 203
word-wrap[3.0.0] 204
-cssj-no-break-characters[3.0.6] 205
-cssj-break-characters[3.0.6] 206
4.8.5 圏点[3.0.4] 207
-cssj-text-emphasis-style 208
-cssj-text-emphasis-color 209
-cssj-text-emphasis 210
4.8.6 文字の影[3.0.8] 211
text-shadow 211
4.8.7 袋文字[3.0.8] 212
-cssj-text-fill-color 213
-cssj-text-stroke-width 213
-cssj-text-stroke-color 213
-cssj-text-stroke 214
4.8.8 透明化[3.0.6] 215
opacity 215
4.8.9 透明色[3.0.8] 216
4.8.10 角丸境界[3.0.6] 217
border-top-left-radius 217
border-top-right-radius 218
border-bottom-left-radius 218
border-bottom-right-radius 218
border-radius 219

4.8.11 回転・拡大・変形[3.0.8] 220
-cssj-transform 220
-cssj-transform-origin 221
4.9 ページ処理機能 223
4.9.1 ページのレイアウト 223
4.9.2 ページの大きさの制約と切り落とし 225
4.9.3 2パス以上の変換処理 225
4.9.4 ページの参照 226
4.9.5 グレイスケール印刷 226
4.9.6 片面印刷と両面印刷 226
4.9.7 ページごとに生成されるコンテンツ 227
4.10 改ページ制御 232
4.10.1 用語の定義 232
4.10.2 強制改ページ 232
4.10.3 orphans [css]とwidows [css] 234
orphans [css] 234
widows [css] 235
orphans [css]とwidows [css]の競合 236
4.10.4 改ページの抑制 237
内部の改ページ抑制 237
前後の改ページ抑制 238
4.10.5 自動改ページ 238
通常のフローのブロック 239
浮動ボックス 239
4.11 テーブル内での改ページ 240
4.11.1 改ページされない場所 240
4.11.2 page-break-XXXの適用 240
4.11.3 テーブル行内部(セル内部)での改ページ 241
4.11.4 デフォルトの改ページ禁止 241
4.12 WebFont 242
4.12.1 @font-face ルール 242
font-family 243
font-style 243
font-weight 243
unicode-range 244
src 244
4.13 縦書き 246
4.13.1 -cssj-writing-mode 246
4.13.2 文書の書字方向 246
4.13.3 書字方向の混在 248
4.13.4 -cssj-direction-mode 250
4.14 多段組 256
4.14.1 -cssj-column-count 257
4.14.2 -cssj-column-width 258
4.14.3 -cssj-columns 258
4.14.4 -cssj-column-gap 258
4.14.5 -cssj-column-rule-color 259
4.14.6 -cssj-column-rule-style 259
4.14.7 -cssj-column-rule-width 259
4.14.8 -cssj-column-rule 260
4.14.9 -cssj-column-fill 260
4.14.10 -cssj-column-span 260

4.14.11 改段と改ページ 261
自動的な改段と改ページ 261
強制的な改段と改ページ 261
4.15 バーコード・QRコード 262
4.15.1 バーコードの表示 262
4.15.2 ISBNバーコード 263
4.15.3 QRコード 263
version 264
ecc 264
encmode 264
module-width 264
quiet-zone 264
4.15.4 郵便カスタマーバーコード 265
module-width 265
4.16 MathML 266

5.資料集 268
5.1 入出力プロパティー覧 268
5.1.1 文書中で設定できないプロパティ 282
5.1.2 機能限定版 282
5.2 メッセージハンドラから取得できる情報 283
5.2.1 Copper PDF 2.0以前(CTIP 1.0) 283
5.2.2 Copper PDF 2.1以降(CTIP 2.0) 283
メッセージコードのフィルタリング[3.0.0] 285
5.3 CSSプロパティのサポート状況 286
5.4 HTMLの各要素・属性のサポート状況 290
5.5 拡張機能 295
5.5.1 処理命令の拡張 295
5.5.2 CSSプロパティの拡張 295
CSSプロパティ 295
CSS関数 300
CSS識別子 301
CSSルール 302
CSS擬似クラス 302
CSS単位 302
5.5.3 XMLの拡張 302
XML要素 303
XML属性 303

1.Copper PDF入門

1.1 Copper PDFの概要

1.1.1 Copper PDFとは

Copper PDFはHTML+CSSをはじめとする文書をサーバー側でレイアウトしてPDF等に変換する、ドキュメント変換サーバーです。

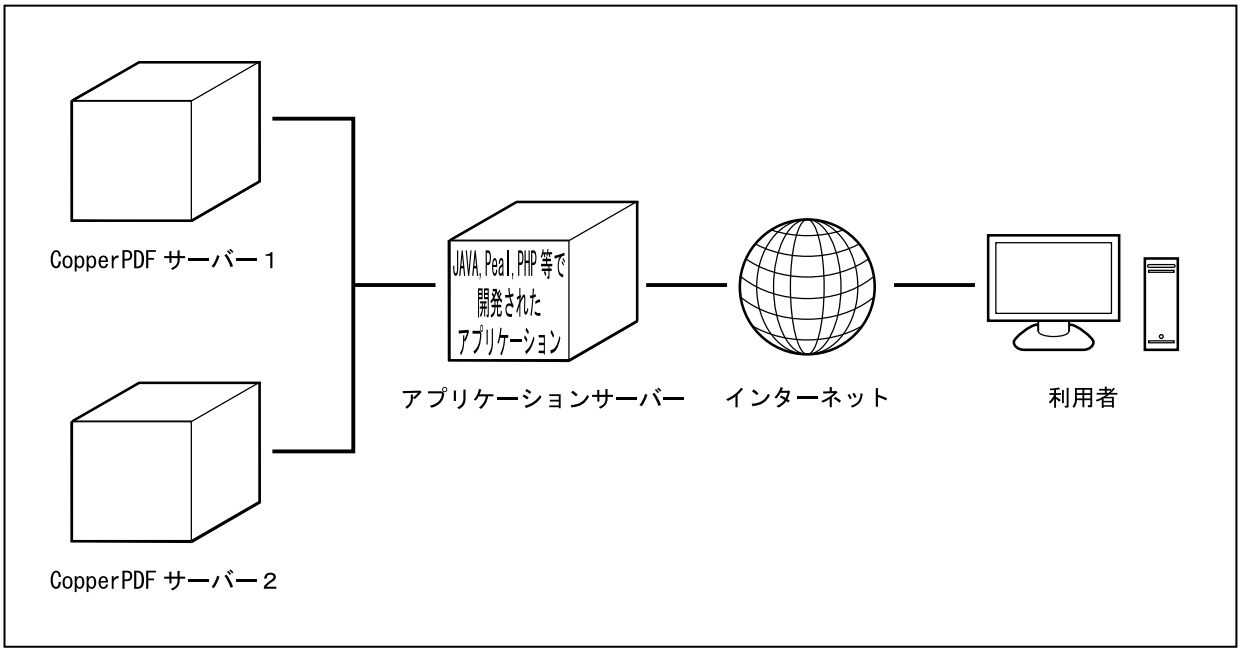
HTML, CSSをはじめ、XHTML, XML, SVG, XSLTといったウェブ向けの標準規格に対応しています。また、JPEG, PNG等の画像を読み込んでPDFに変換することができます。PDF以外の各種画像フォーマットへの変換も可能です。

Copper PDFは100% Javaアプリケーションです。Java実行環境がサポートされている、Windows, Linux, Solaris, MacOS, FreeBSD など、様々なOS上で実行可能です。

ウェブアプリケーションの開発言語として一般的なJava, Perl, PHP, C#, VB.NET, Ruby, Python向けに、使いやすく高機能なオブジェクト指向APIを用意しています。PDFのもとになるHTMLの出力は普通のウェブアプリケーションを開発する場合と同じであり、JSP, Smarty, TemplateToolkit, ASP.NETのようなテンプレートエンジンも利用することができます。各プログラミング言語からCopper PDFへ、ネットワークを介してアクセスする仕組みなので、アプリケーション側はJava実行環境を必要とせず、また負荷分散によって非常に高負荷が予想されるシステムにも対応可能です。

HTTP/HTTPSによる接続をサポートしており*[2.1.0]*、4th Dimension等、HTTPクライアントを利用できる各種開発環境からも容易に利用することができます。

図 1.1 システムの構成例



1.1.2 なぜCopper PDFが必要か

例えば、ネット上でクーポンやチケットのようなものを印刷させるために、印刷専用ページを用意し、各自のブラウザで印刷していただくとします。しかし、ほとんどのブラウザが標準に準拠するようになったと言っても、各ブラウザで完全に同じ表示を実現することは、レイアウトが複雑になればなるほど難しくなります。その点、サーバー側でPDFに変換してしまえば、完全に同じ表示を保障することができます。また、PDFでは暗号化やファイル添付等のHTMLにはない機能が利用できます。サードパーティーの製品を使うことにより、PDFに著作権管理やタイムスタンプを付与することができます。

PDFを動的に出力するためのソフトウェアとしては、各種帳票出力ソフトウェアがあります。しかし、それらはレイアウトを決めるためにプログラミングが必要であったり、独自の帳票デザインツールが必要であるため、それらの使い方を学習する必要があります。しかも、各メーカーで共通化されていないため、別の会社の製品を使おうと思ったら、また使い方を学習しなおす必要があります。また、全般的に複数ページにわたるPDFの出力は面倒です。しかし、HTML+CSSによるレイアウトであれば、既にウェブで広く使われているため、開発者が新しく学習することは最小限で済みます。HTML+CSSは従来は限られたレイアウトしか実現できませんでしたが、近年は十分に印刷にも利用できるほど高機能になりました。

HTML+CSSからPDFを出力することは、ウェブブラウザの「印刷機能」やコマンドラインを使っても可能です。しかし、普及しているウェブブラウザは画面でスクロールさせて見ることを主目的にしており、印刷への対応は十分ではありません。例えば予期しないところで途切れてしまったり、画面表示では実現できていた表示が印刷ではできなかったりといったことが起こります。また、数千ページにおよぶ文書を印刷しようとすると、一般的なブラウザは非常にメモリを消費し、動作が重くなるか、最悪の場合はクラッシュしてしまうでしょう。Copper PDFは、独自の設計により、理論的にはほぼ無制限の大きさの文書を一定のメモリ使用量でPDFに変換することができます。

冊子では目次の生成、ページ番号の付与が必要であり、特に日本語圏では日本語特有の禁則処理、両合わせ、縦組、ルビへの対応が必須と言えます。Copper PDFはこれらの機能を独自にサポートしています。

1.2 動作環境

Copper PDFは、サーバーと、プログラミングインターフェース(API)が別々に動作します。ユーザーが開発したアプリケーションからはAPIを利用してサーバーにアクセスします。アプリケーションはサーバーと同一のマシン上でも、別のマシン上でも構いません。サーバーを動作させるためのマシンにはJavaが必須ですが、アプリケーション側にJavaは必須ではありません。

1.2.1 サーバー

Copper PDFサーバーにはコマンドラインから直接PDF変換できるインターフェースと、ウェブインターフェースが付属しています。

copperd(Copper PDFサーバー)、copper(コマンドラインインターフェース)、copper-webapp(ウェブインターフェース)の実行には最低限以下の環境が必要です。copper Antタスクの実行にも同様の環境が必要です。Copper PDF 2.1.0からはtranscode Antタスクの使用を推奨します。

- Java JRE/JDK 1.8 以降
- REST(HTTP/HTTPS)の待受と copper-webapp は Java JRE/JDK 17 以降*[3.2.33]*
- 128MB以上の空きメモリ
- 50MB以上の空きディスクスペース

Copper PDF 3.2.33 から、同梱する Jetty が 12 になったため、サーバー(copperd)の REST (HTTP/HTTPS)の待受と、ウェブインターフェース(copper-webapp)の実行には Java 17 以降が必要です。Java 1.8〜16 でサーバーを起動した場合は、REST の待受を行わずに(警告をログに出して)起動し、CTIP(平文と TLS)で変換できます。copper コマンドと Java ドライバは Java 1.8 以降で動作します。

Copper PDF 2.1.x までは JDK1.4.2 以降でも動作します。

Copper PDF 3.1.x までは JDK1.6 以降でも動作します。

以下の環境を推奨します。

- Java JDK 21 以降(バンドル版に同梱しているものと同じ)
- 2GB以上の空きメモリ
- 4GB以上の空きディスクスペース

以下のOS上での動作を確認しています。

- Windows 7/8/10/11
- Red Hat Enterprise Linux(RHEL) 7/8/9/10
- Debian 13

- Ubuntu 24/25
- Solaris 10/11
- MacOS X
- FreeBSD 13/14

ウェブインターフェースには以下のブラウザとAdobe Reader等のPDFを表示できるソフトウェア必要です。

d

- Firefox/ Edge / Google Chrome / Safari など

1.2.2 プログラミングインターフェース(API)

Copper PDFを利用するアプリケーションの開発に使用する、各プログラミング言語向けのドライバはCopper PDF本体とは別に配布しています。

Copper PDF 2.1.0からは、より強力なプログラミングインターフェース(CTIP 2.0)と、HTTP/RESTインターフェースがサポートされました。HTTP/RESTインターフェースは、各開発環境から利用することができます。

新インターフェース(CTIP 2.0)

Javaドライバ / transcode Antタスク

Java 1.8 以降*[2.1.0]*

Javaドライバ 2.0.x までは JDK1.4.2 以降でも動作します。

Perlドライバ

Perl バージョン5.6.1以降*[2.1.0]*
File::Temp モジュール
IO::Socket::SSL （SSL接続をする場合）

PHPドライバ

PHP バージョン5.2.0以降*[2.1.0]*

.NETドライバ

.NET Framework 2.0以降*[2.1.0]*

HTTP / RESTインターフェース

各種開発環境（環境非依存） *[2.1.0]*

1.3 機能一覧

1.3.1 入力データ形式

ドキュメント

- HTML/XHTML (166ページ) - HTML/XHTML文書をレイアウトします。
- XML (166ページ) - XML文書をレイアウトします。
- SVG (169ページ) - SVGを直接PDF等のデータ形式に変換します*[2.1.0]*
- GIF/PNG (169ページ) JPEG (169ページ) その他Java実行環境が対応可能な画像形式 (168ページ) - 画像を直接PDF等のデータ形式に変換します*[2.1.0]*。
- EPUB (171ページ)*[3.1.0]*

スタイルシート

- CSS 2.1 (194ページ) - HTML/XHTML/XML文書をCSSでスタイル付けします。
- XSLT (168ページ) - XML文書をXSLTで変換します。
- メディアタイプの設定 (194ページ) - 印刷向け以外のスタイルシートを適用します。
- ユーザーのCSSスタイルシート (198ページ)
- ユーザーのXSLTスタイルシート (168ページ)
- WebFont(@font-face) (242ページ)*[3.0.0]* - ネットワーク上からTrueType フォント、OTF、WOFF*[3.1.0]*を読み込めます。
- 縦書き (246ページ)*[3.0.0]* - 縦書きレイアウトをサポートしています。
- 段組み (256ページ)*[3.0.0]* - 多段組をサポートしています。
- ルビ (192ページ)*[3.0.0]*
- 禁則文字の追加・除外 (203ページ)*[3.0.6]*
- 圏点 (207ページ)*[3.0.4]*
- 文字の影 (211ページ)*[3.0.8]*
- 袋文字 (212ページ)*[3.0.8]*
- 透明化 (215ページ)*[3.0.6]*
- 角丸境界 (217ページ)*[3.0.6]*
- 回転・拡大・変形 (220ページ)*[3.0.8]*

ベクター画像データ

- SVG (169ページ) - 文書中にSVG形式の絵・図を埋め込みます。

ラスター(ビットマップ)画像データ

文書中に画像を埋め込みます。

- GIF/PNG (169ページ)
- JPEG (169ページ) - JPEGをそのままPDF中に埋め込むことができます。
- その他Java実行環境が対応可能な画像形式 (168ページ) - JPEG2000等、最新の画像形式にも対応可能です。

1.3.2 PDF出力

レイアウト結果をPDFとして出力します

- PDFバージョン1.2から1.7に対応 (172ページ) - 各バージョンのPDFに対応した出力結果が得られます。
- PDF/A-1bに準拠したファイルの出力 (179ページ)*[2.1.0]*
- 画面表示/印刷時で表示制御が可能なすかし (176ページ)*[2.1.8]*
- PDFの圧縮の設定(無圧縮/Flate圧縮/ASCIIテキスト) (174ページ)
- 画像の圧縮(可逆圧縮/JPEG/JPEG2000) (174ページ)
- 添付ファイル (174ページ) - PDFに別のファイルを添付します。
- ブックマーク (188ページ) - HTMLの見出し(H1～H6)からブックマーク(しおり)を生成します。
- 暗号化 (172ページ) - 40～128ビットの暗号化に対応しています。
- パスワード・パーミッションの指定 (172ページ) - 暗号化したPDFにパスワードと利用制限をかけることができます。
- ハイパーリンク (190ページ) - HTMLのリンク(Aタグ)をPDFにも適用します。
- 文書内リンク (190ページ) - PDFにリンクを反映する他、ページ番号を出力できます。
- メタ情報(文書タイトル、Creator、CreationDate等)の設定 (167ページ)
- CreationDate, ModDate, ファイルIDの設定 (175ページ)
- ViewerPreferencesの設定 (179ページ)*[3.0.2/2.1.11]*
- オープン時に実行されるJavaScriptの設定 (179ページ)*[3.0.2/2.1.11]*

1.3.3 画像出力

各ページを画像として出力します。

- JPEG/PNG (180ページ)
- その他Java実行環境が対応可能な画像形式 (168ページ) - JPEG2000等、最新の画像形式にも対応可能です。
- 画像出力時の解像度設定 (180ページ) - 任意の解像度で画像を出力できます。
- SVG (180ページ)*[3.0.1]*

1.3.4 フォント関連機能

■ CID-Keyedフォント (52ページ) - フォントファイルを埋め込まず、大抵の環境で見ることができるPDFを出力できます。
■ 外部(CID Identity)フォント (52ページ) - 特定の環境に依存した、フォントを埋め込まないPDFを出力できます。
■ 埋め込みフォント (44ページ) - 環境に依存しないPDFを出力するために、TrueType, OpenType等のフォントを埋め込みます。
■ デフォルトのフォントの設定 (46ページ)
■ フォント幅情報・MSフォントの使用 (54ページ)
■ Unicodeサロゲートペア *[3.1.0]*

1.3.5 HTTP接続

Copper PDFが外部のサーバーからHTTP接続で文書を取得する機能です。

■ タイムアウトの設定 (161ページ)
■ ヘッダ(Referer, User-Agent等)の変更 (159ページ) - 参照元やUser-Agentを偽装することができます。
■ プロクシを介した接続 (159ページ)
■ BASIC認証/Digest認証 (158ページ)
■ クッキー (160ページ)

1.3.6 印刷サポート

■ 用紙サイズ・出力サイズの設定 (223ページ)
■ ページの拡大・縮小・変形 (223ページ)
■ トンボの出力 (223ページ) - 断裁して製本するためのトンボを表示します。
■ 欄外の描画 (225ページ)
■ ページのマージン (223ページ)
■ 片面印刷・両面印刷の切り替え (226ページ)
■ グレイスケール変換 (226ページ) - カラーの文書をモノトーンに変換します。

1.3.7 目次・ページ参照

■ 2パス以上の処理 (225ページ) - 目次やページ参照のために、文書を複数回処理できます。
■ 欄外のページ番号、見出し (227ページ) - 欄外にページ番号や、見出しを出力します。
■ 目次の生成 (188ページ) - HTMLの見出し(H1～H6)から目次を生成します。
■ ページの参照 (226ページ) - 文書中の指定した部分のページ番号を出力できます。

1.3.8 プログラムインターフェース

- Java (JRuby, Jython) (76ページ)
- PHP (102ページ)
- Perl (95ページ)
- .NET (C# / VB.NET) (109ページ)
- Ruby (120ページ)
- Python (126ページ)
- Ant (141ページ)
- 処理中のページ、内容の取得 (ページ)
- 処理状況の取得 (ページ)
- サーバーからリソースを要求 (ページ)
- HTTP/REST (144ページ) (Apache経由での接続 (ページ) / Ruby (147ページ) / Python (149ページ) / C# (150ページ) / ASP.NET (C# / VB.NET) (152ページ) / 4th Dimension (154ページ))

1.3.9 その他の機能

- MathML (266ページ) - 文書中にMathMLによる数式を埋め込みます。*[3.1.0]*
- バーコード・QRコード (262ページ) - 文書中にバーコード・QRコードを埋め込みます。*[3.1.0]*
- 改ページをせずに文書を出力する (225ページ)
- 解像度(pxの大きさ)の設定 (199ページ)
- IE互換モード (ページ) - InternetExplorerの表示に近いレイアウトモードです。
- 出力データサイズ制限 (162ページ) - DoS攻撃対策等のため出力サイズを制限できます。
- ページ数制限 (162ページ) - DoS攻撃対策等のため出力ページ数を制限できます。
- 実行経過情報取得 (283ページ) - リアルタイムで文書の変換状況をアプリケーションが取得できます。

1.4 とりあえず使ってみよう

1.4.1 Copper PDFのインストール

Copper PDFは、様々なOS(Windows, MacOS X, Linuxなど)で動かすことができます。

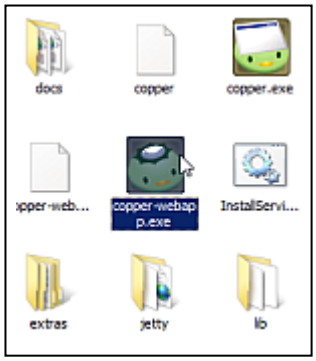
例えばWindowsで動かしてみましょう。Copper PDFはJava(JDK)を必要とします。また、Copper PDFが出力したPDFを見るためにはAdobe Readerが必要です。お手元の環境に入っていない場合は、それぞれ以下のアドレスからダウンロードしてインストールして下さい。

- http://www.oracle.com/technetwork/java/javase/downloads/index.html (Download JDK へ進んでください)
- https://get.adobe.com/jp/reader/ (Adobe Reader)

以下のアドレスから「Windows用ZIPアーカイブ」をダウンロードしてください。これがWindows版のCopper PDFの本体です。

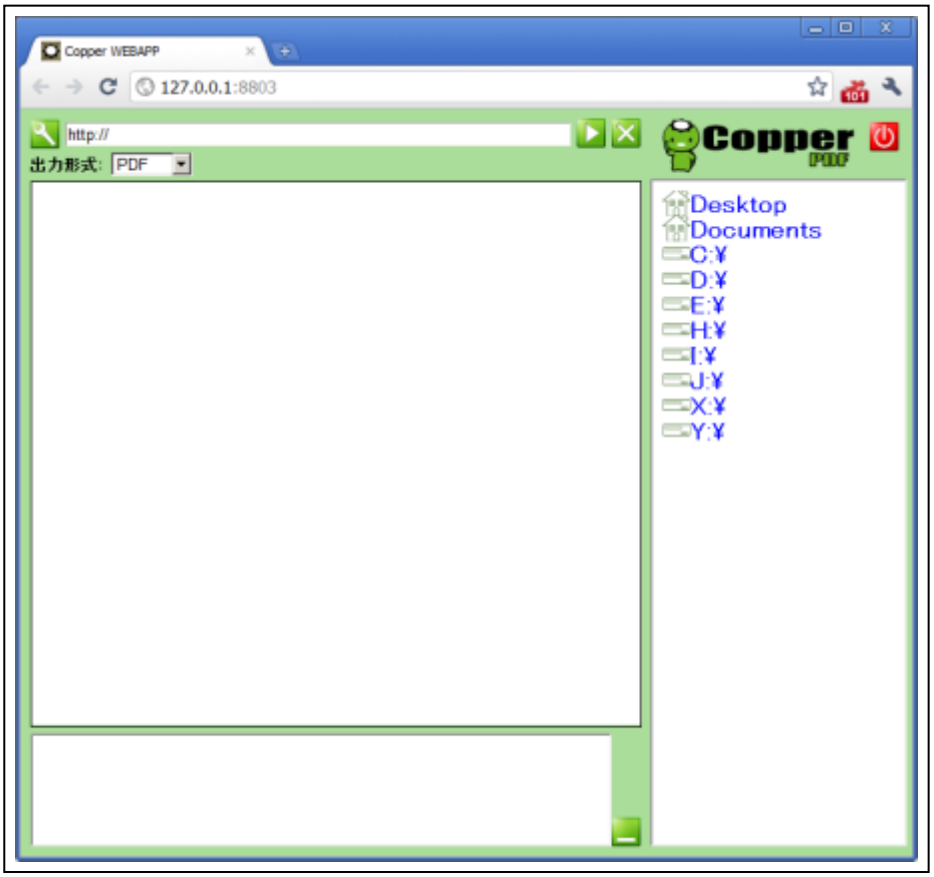
- https://copper-pdf.com/?p=90

図 1.2 copper-webappの起動



ZIPファイルを展開し、CopperPDFディレクトリ内のcopper-webapp.exeをダブルクリックして実行してください。ブラウザが起動し、Copper WEBAPP というタイトルのページが開きます。これでCopper PDFが無事に動作しました。

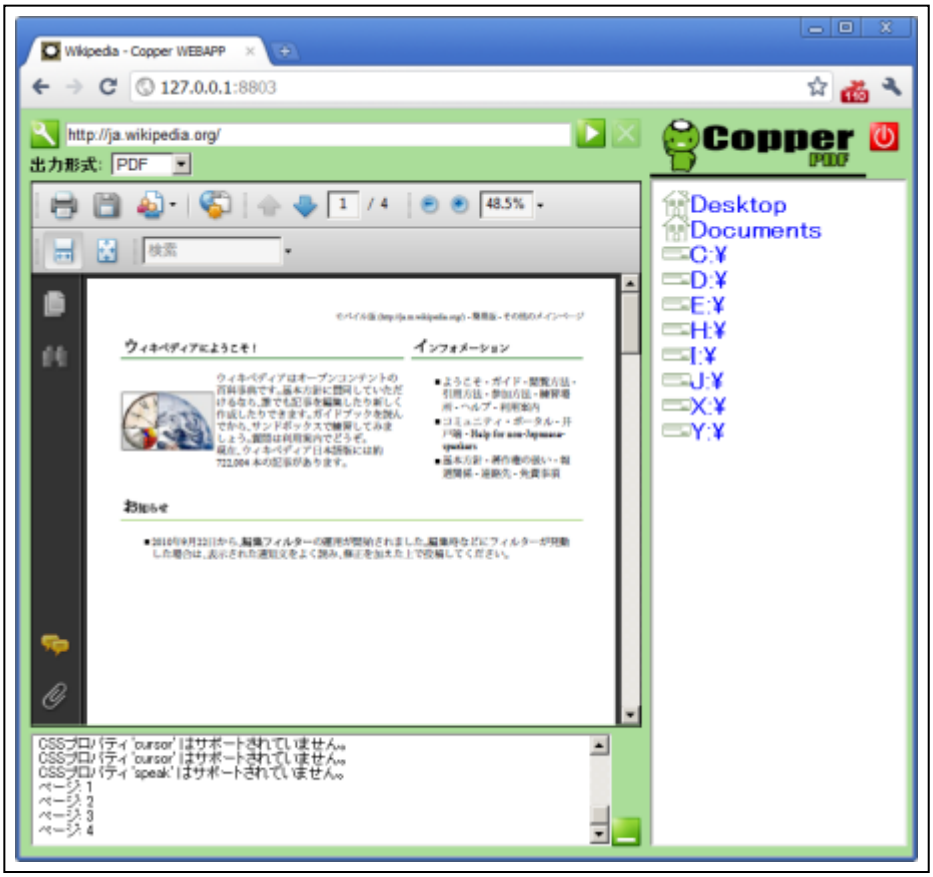
図 1.3 Copper WEBAPPの初期画面



1.4.2 ウェブインターフェースでHTMLを変換する

とりあえず、何か表示させてみましょう。例えば上部の入力欄に https://ja.wikipedia.org/ (Wikipediaのアドレス)を入力し、読込ボタンを押してください。しばらく待つと、ウィキペディアのトップページがPDF化されて、ブラウザ内に表示されます。

図 1.4 ウェブページの変換結果



同じように、https://www.w3.org/TR/xslt20/ (XSLT 2.0の仕様書)を読み込んでみてください。みるみるうちに300ページを超える仕様書のPDFが生成されます。このような大きな文書でも高速処理できることがCopper PDFの特徴です。

手元のPCの中のファイルは、右側のファイルツリーから選択して変換することができます。ドラッグ＆ドロップはできませんのでご注意ください。Copper WEBAPP を終了させるには、右上の赤いボタンを押してください。

Copper WEBAPP は、Copper PDFでPDF変換するためのHTMLデザインの補助ツールとしてご利用ください。

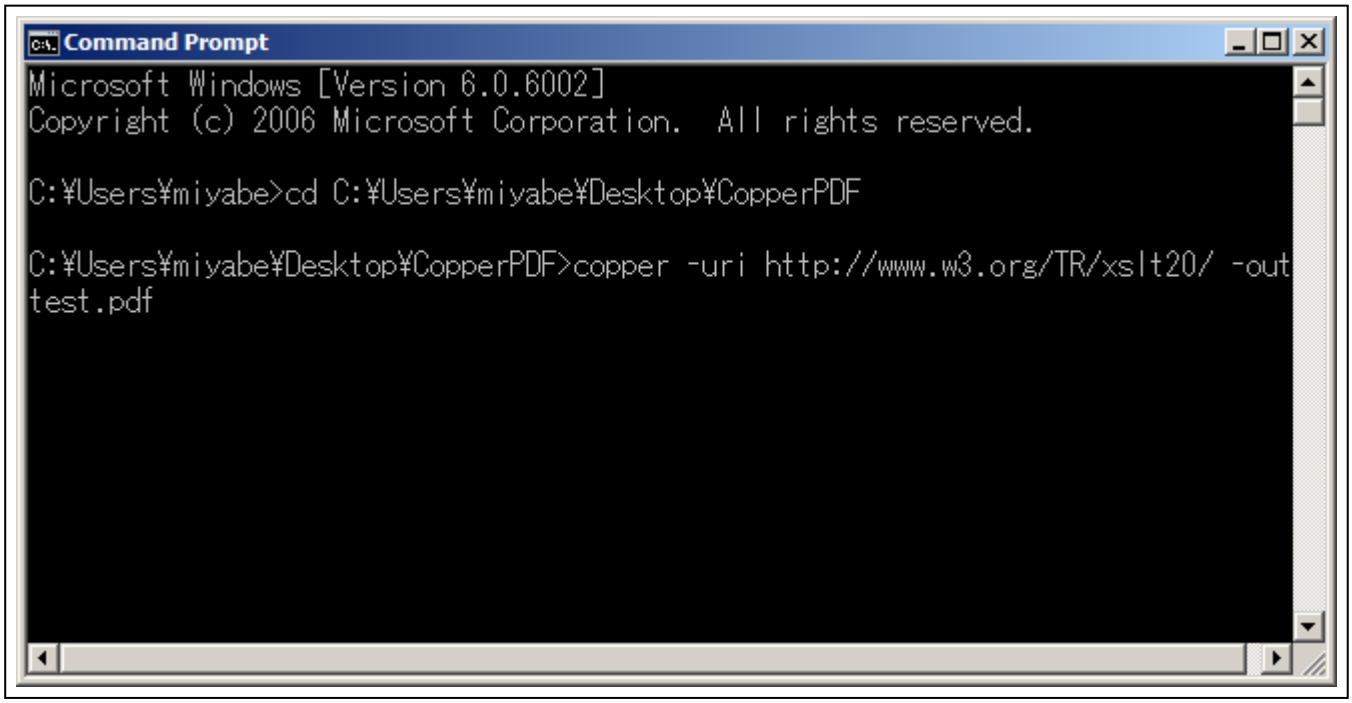
1.4.3 コマンドラインアプリケーションでHTMLを変換する

Copper PDFは、コマンドアプリケーションとしても動作します。コマンドライン上での簡単な作業や、バッチ処理には便利です。コマンド・プロンプトを起動しCopperPDFディレクトリに移動して、copper を実行してください。例えXSLT 2.0の仕様書の変換は次のとおりに行います。

例 1.1 PHPからCopper PDFを使う

```
copper -uri https://www.w3.org/TR/xslt20/ -out test.pdf
```

図 1.5 copperの実行

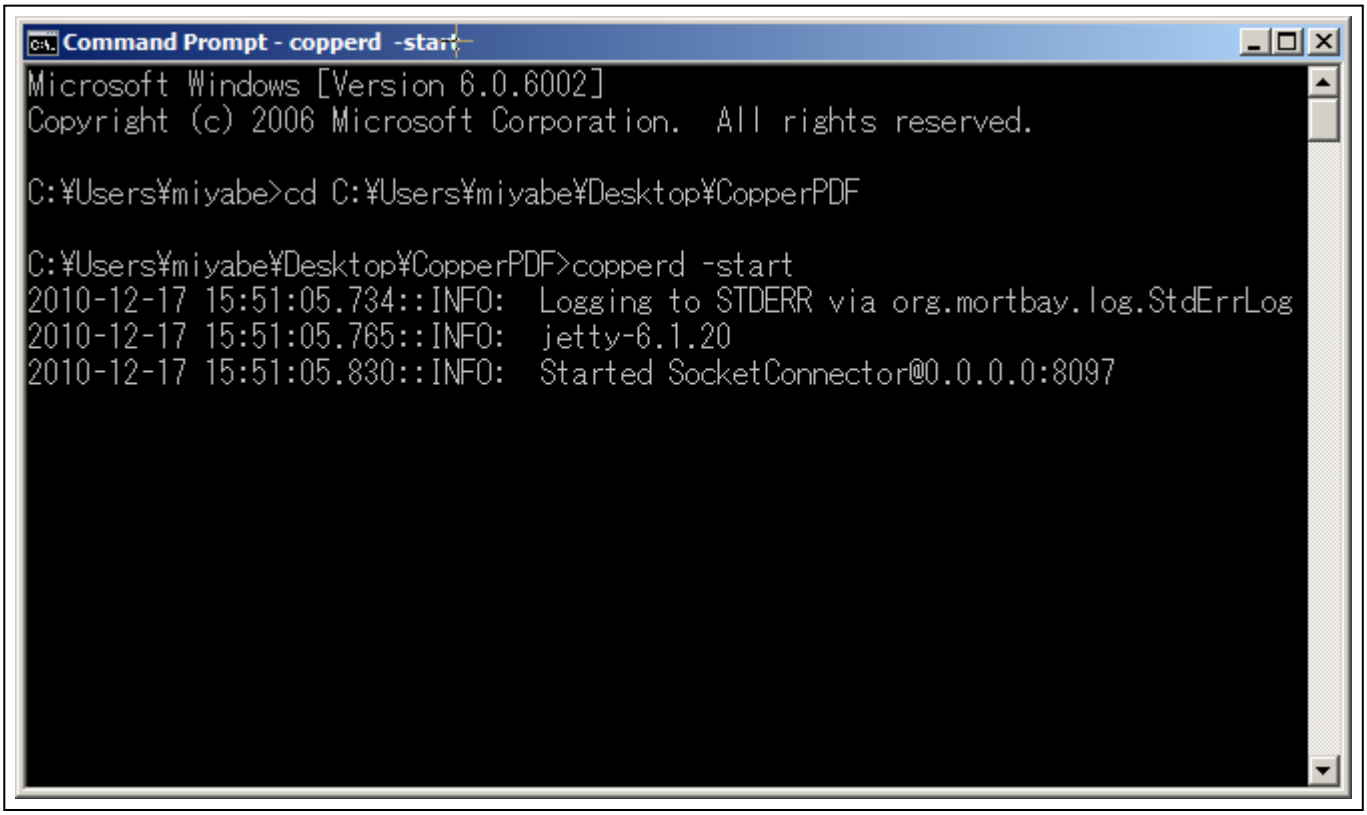


しばらくした後、変換結果としてtest.pdfというファイルが生成されます。

1.4.4 プログラムからHTMLを変換する

Copper PDFが真価を発揮するのは、ここで説明する他のプログラムとの連携です。プログラムから文書の変換を行うには、あらかじめCopper PDFサーバーを起動した状態にする必要があります。コマンドラインからサーバープログラム(copperd)を起動するには、コマンド・プロンプトを起動しCopperPDFディレクトリに移動して、copperd -start を実行します。

図 1.6 copperdの起動



つぎに、ウェブ開発で広く使われているPHPを使ってCopper PDFにアクセスしてみましょう。すでにマシンにPHPがインストールされているか、あるいはLANでつながった他のマシンにPHPがインストールされている場合は、それを使って構いません。copperdは初期設定の状態では、ローカルマシンだけからアクセスできるようになっています。confディレクトリ内のaccess.txtを編集して、

`allow` **接続元のIPアドレス**

という行を先頭に加えて、特定のマシンからのアクセスを許可するか、あるいは

```
allow *
```

という行を加えてネットワークからのアクセスを全て許可してください（セキュリティにご注意ください）。

Windowsマシンにインストールする場合は、以下のアドレスから Installer をダウンロードして、インストールしてください。

- http://windows.php.net/download/

Copper PDF のプログラムインターフェースは、本体とは別に配布されています。以下のアドレスから入手できます（Composerによるインストール推奨）。

- https://github.com/zamasoftnet/cti.php

展開してできたcti-php-2.x.xというディレクトリに、次の中身のファイルを保存してください（文字コードはシフトJISです）。

例 1.2 PHPからCopper PDFを使う

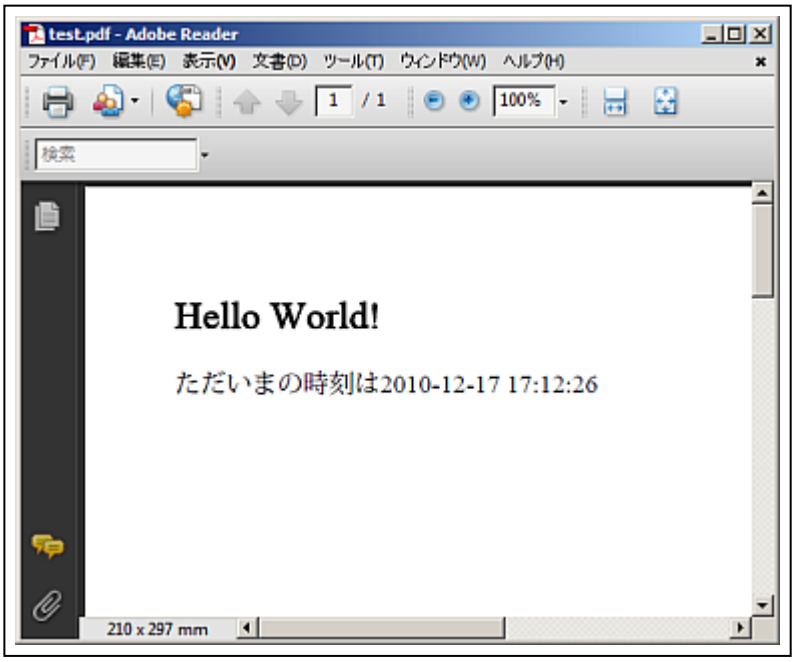
```
<?php
require_once ('code/CTI/DriverManager.php');

//セッションの開始
$session = cti_get_session('ctip://localhost:8099/',
    array('user' => 'user',
    'password' => 'kappa'));

//ファイル出力
$session->set_output_as_file('test.pdf');

//文書の送信
$session->start_main('.');
$date = date('Y-m-d H:m:s');
?>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=SJIS" />
    <title>TEST</title>
  </head>
  <body>
<h2>Hello World!</h2>
ただいまの時刻は<?php echo $date ?>
  </body>
</html>
<?php
$session->end_main();

//セッションの終了
$session->close();
?>
```

コマンド・プロンプトを起動し、cti-php-2.x.xディレクトリ内で、

```
php test.php
```

を実行してください。同じディレクトリ内にtest.pdfというファイルができれば成功です。

図 1.7 出力結果


test.pdf - Adobe Reader
ファイル(F) 編集(E) 表示(V) 文書(D) ツール(T) ウィンドウ(W) ヘルプ(H)
1 / 1
100%
検索
Hello World!
ただいまの時刻は2010-12-17 17:12:26
210 x 297 mm

2.管理者ガイド

2.1 セットアップ

この章ではCopper PDFのセットアップ(インストール)方法について説明します。

2.1.1 Java実行環境 のインストール

Copper PDFをインストールするためには、Java実行環境(JREまたはJDK)が必要です。Java実行環境は、各OSベンダが配布しているものをインストールするか、https://www.oracle.com/technetwork/java/javase/downloads/index.html (Download JDK へ進んでください) で配布されているものをインストールしてください。

必要なJava実行環境は、サーバー(copperd)とウェブインターフェース(copper-webapp)ではバージョン17以降、copperコマンドとJavaドライバではバージョン1.8.0以降の、国際化対応Java実行環境です(サーバーの17以降はCopper PDF 3.2.33から。3.2.32までは1.8.0以降)。Java RE(JRE), Java SE(JDK…開発環境およびサーバー用Java VMを含むもの)の両方で動作しますが、Java SEを推奨します。

LinuxではOpenJDKを始めとする様々なJava実行環境に対応しています。Windowsでは Copper PDF 3.2.7以降からOracle OpenJDKに対応していますがAdoptOpenJDKバンドル版を使用するか、あるいは次の説明の通り、所定の位置にJava実行環境を配置することで様々なJava実行環境に対応することができます。

AdoptOpenJDKバンドル版を使用する場合はJDKのインストールは不要です。AdoptOpenJDKバンドル版は64ビットOSのみに対応しています。

バンドルされたJava実行環境は、Windows では extras/windows/runtime 、Linux では extras/redhat/runtime に配置されています (rpm/deb パッケージ版では /var/lib/copper-pdf/extras/redhat/runtime)。非バンドル版でも、その場所に他のJava実行環境を配置するか、ディレクトリをリンクすることにより、そのJava実行環境が優先して使用されます。ただし、環境変数 JAVA_HOME が設定されている場合は、そちらが優先されます。

2.1.2 Copper PDF の配布パッケージ

Copper PDF本体は、以下のパッケージが配布されています。利用する環境に合ったパッケージをダウンロードしてください。(3.x.xの部分はCopper PDFのバージョンにより異なります)

copper-pdf-3_x_x.zip
: ZIPアーカイブ

copper-pdf-3.x.x-x.noarch.rpm
: RPMパッケージ

copper-pdf_3.x.x_all.deb
: Debian/Ubuntuパッケージ

copper-pdf-3.x.x.tar.gz
: tar.gzアーカイブ

copper-pdf-3_x_x-openjdk-bundle-x64-windows.zip
: Windows AdoptOpenJDKバンドル版

copper-pdf-3_x_x-openjdk-bundle-x64-linux.tar,gz
: Linux AdoptOpenJDKバンドル版

2.1.3 Docker（Linux）

Docker版はJava実行環境を別途インストールせずに利用できます。Dockerスターター一式の compose.release.yml、compose.license.yml、copper.ps1、copper.shを同じディレクトリへ配置してください。

Windowsでは次のコマンドで起動します。

```
.\copper.ps1 up
```

LinuxまたはmacOSでは次のコマンドで起動します。

```
chmod +x copper.sh
./copper.sh up
```

公開Dockerイメージには Noto Sans JP および Noto Serif JP があらかじめ含まれています。ライセンスキーを配置すると、和文はNotoで組まれてPDFへ埋め込まれます。ライセンスキーが無い場合は機能限定版として起動し、フォントは埋め込まれません。

商用ライセンスまたは試用ライセンスを使う場合は、ライセンスキーファイルを `user/license-key` として配置して、もう一度upを実行してください。追加フォントは `user/fonts` へ配置してrestartを実行します。起動スクリプトがキーの有無を判定し、キーと追加フォントを読み取り専用でコンテナへ渡します。ライセンスキーはDockerイメージには保存されません。

```
user/
|-- license-key        （ライセンス利用時だけ）
`-- fonts/
    `-- MyFont.otf
```

状態確認、ログ表示、更新および停止は、それぞれstatus、logs、update、downをupの代わりに指定します。

2.1.4 Windows

https://copper-pdf.com/?p=90 で配布されているZIPアーカイブ copper-pdf-3_x_x.zip または copper-pdf-3_x_x-openjdk-bundle-x64-windows.zip をダウンロードしてください。

各アーカイブを展開し、CopperPDFディレクトリを適当な場所に配置してください。アンインストールはCopperPDFディレクトリを削除するだけです



Windows上でJava6を使用する場合、Copper PDFのサービスを起動しようとすると、以下のメッセージが表示されることがあります(Windows2003/2008 Server上で現象が確認されています)。

`[174 javajni.c] [error]` 指定されたモジュールが見つかりません。
この場合jdk1.6.0_xx/binディレクトリ内のmsvcr71.dllをWindowsのsystem32ディレクトリにコピーすることで起動するようになります。

付属のプログラムを使ってサービスをインストールする



この方法では、CopperPDFディレクトリまでのファイルパスに半角英数字以外の文字(かな、漢字など) が含まれている場合はサービスの起動ができません。C:\Program Filesなど、半角英数字の文字だけで構成されるファイルパスに格納してください。ユーザーのホームディレクトリの下には置かないでください。

サービスはLocalServiceアカウントで動作します。InstallService.bat (InstallService-x64.bat)は、このアカウントがlogsディレクトリへ書き込めるように権限を設定します。

あるいは、後述するJava Service Wrapperの使用を推奨します。

アクセサリの「コマンドプロンプト」を管理者で実行し、Copper PDFのディレクトリに移動してからInstallService.bat(x64版JavaVM の場合はInstallService-x64.bat)を実行してください。サービスを削除する場合も同様にRemoveService.bat(x64 版JavaVM の場合はRemoveService-x64.bat) を実行してください。

例 2.1 Windowsでサービスをインストールする

```
C:\>cd [Copper PDFのインストールディレクトリ]
C:\[Copper PDFのインストールディレクトリ]>.\InstallService.bat
```

例 2.2 Windowsでサービスをインストールする(x64)

```
C:\>cd [Copper PDFのインストールディレクトリ]
C:\[Copper PDFのインストールディレクトリ]>.\InstallService-x64.bat
```

Java Service Wrapperを使ってサービスをインストールする*[2.1.4]*

Copper PDFは、Java Service Wrapper によるサービスとしてインストールすることができます。Java Service Wrapperの全ての機能を利用し、開発元のタヌキソフトウェア有限会社によるサポートを受けるためには、別途ライセンスの購入が必要です。Java Service Wrapperを使用することにより、障害発生時の自動復旧等が可能になり、より安定して運用することができます。

Copper PDFをJava Service Wrapperで動作させるためには、Java Service Wrapperの配布物に含まれる内容を、次の通りコピーしてください。

1. CopperPDFディレクトリ直下にbinディレクトリを作り、Java Service Wrapperのbin/wrapper.exeをその中にコピーする。
2. Java Service Wrapperのsrc/bin/InstallApp-NT.bat.inをCopper PDFのbinディレクトリ内にInstallCopperPDF-NT.batという名前で配置する。
3. Java Service Wrapperのsrc/bin/UninstallApp-NT.bat.inをCopper PDFのbinディレクトリ内にUninstallCopperPDF-NT.batという名前で配置する。
4. Java Service Wrapperのlib/wrapper.dll, lib/wrapper.jarをCopper PDFのlib内にコピーする。

Java Service Wrapperの設定ファイルは、Copper PDFにconf/wrapper.confという名前で既に含まれています。Java Service Wrapperのライセンスを購入した場合は、このファイルにライセンスキーの内容を追記してください。

サービスをインストールする場合は、InstallCopperPDF-NT.batを実行してください。逆に、アンインストールする場合は、UninstallCopperPDF-NT.batを実行してください。Windows Vistaでは、必ず管理者として実行してください。サービスの起動と停止は、コントロールパネルの管理ツールから行ってください。

サービスの管理と動作状態の確認

サービスはコントロールパネルの管理ツールから起動・停止することができます。

またcopperd.exeの-statusオプションによりサービスの状態を確認することができます。

例 2.3 サービスの動作状態確認

```
C:\>cd [Copper PDFのインストールディレクトリ]
C:\[Copper PDFのインストールディレクトリ]>copperd.exe -status
* Status Report *
[Summary]
Uptime:0 days 0 h 0 min 3 s
AccessCount:0

[Threads]
Total:10
Busy:0
Free:10
Max:50

[Memory]
Total:127.06MB
Using:3.96MB
Free:123.1MB
Max:1016.13MB
```

2.1.5 Red Hat Enterprise Linux(RHEL) / CentOS

RHEL(またはCentOS）向けにはRPMパッケージを配布しています。



RHELでは、Java実行環境としてjava-(バージョン)-sun が必要です。Red Hat NetworksのSupplementaryチャンネルからyumでインストールするか、Supplement CDに収録されているものをインストールしてください。java-(バージョン)-gcj-compatでは動作しません。`alternatives --config java` コマンドでjava-(バージョン)-sunに切り替えてください。
RHELの代替としてCentOSを使用する場合は、Oracleのサイト からダウンロードしたRPMをインストールし、`rpm -i --nodeps copper-pdf-3.x.x-0.noarch.rpm` コマンドでインストールしてください。

copper-pdf-3.x.x-0.noarch.rpmをrpmコマンドでインストールしてください。アンインストールの方法は通常のRPMパッケージの場合と同じです。

例 2.4 RPMパッケージのインストール

```
# sudo rpm -ivh copper-pdf-3.x.x-0.noarch.rpm
```

例 2.5 RPMパッケージのアンインストール

```
# sudo rpm -e copper-pdf
```

Copper PDFサーバーの起動・停止

RHELではchkconfigおよびserviceコマンドでCopper PDFサーバーを管理できます。Copper PDFサーバーのサービス名はcopperdです。インストール直後は、Copper PDFサーバーの自動起動は無効化されています。

例 2.6 copper-pdfの起動

```
# sudo service copperd start
```

例 2.7 copper-pdfの動作状態確認

```
# sudo copperd -status
```

例 2.8 copper-pdfの停止

```
# sudo service copperd stop
```

例 2.9 copper-pdfの再起動

```
# sudo service copperd restart
```

例 2.10 copper-pdfの自動起動の有効化

```
# sudo chkconfig copperd on
```

例 2.11 copper-pdfの自動起動の無効化

```
# sudo chkconfig copperd off
```

2.1.6 Debian/Ubuntu

Debian/Ubuntu向けにはdebパッケージを配布しています。



Debian/Ubuntuでは、Java実行環境としてopenjdk-8-jdk以降が必要です。kaffe または java-gcj-compatでは動作しません。

aptまたはdpkg以外の方法でインストールしたJava実行環境を使う場合は、/etc/profileファイル等で、JAVA_HOME 環境変数にJavaのインストールディレクトリのパスを設定してください。
例:

```
export JAVA_HOME=/usr/local/jdk1.8.0_162
```

copper-pdf_3.x.x_all.debをdpkgコマンドでインストールしてください。アンインストールの方法は通常のdebパッケージの場合と同じです。

例 2.12 debパッケージのインストール

```
# sudo dpkg -i copper-pdf_3.x.x_all.deb
```

例 2.13 debパッケージのアンインストール

```
# sudo dpkg -r copper-pdf
```

Copper PDFサーバーの起動・停止

Debian/UbuntuではCopper PDFのサービスは/etc/init.d/copperdとして配置されます。インストール直後は、Copper PDFサーバーの自動起動は無効化された状態です。サービスの管理方法は以下の通り、通常のDebian/Ubuntuの手法に従います。

例 2.14 copper-pdfの起動

```
# sudo /etc/init.d/copperd start
```

例 2.15 copper-pdfの動作状態確認

```
# sudo copperd -status
```

例 2.16 copper-pdfの停止

```
# sudo /etc/init.d/copperd stop
```

例 2.17 copper-pdfの再起動

```
# sudo /etc/init.d/copperd restart
```

例 2.18 copper-pdfの自動起動の有効化

```
# sudo update-rc.d /etc/init.d/copperd defaults
```

例 2.19 copper-pdfの自動起動の無効化

```
# sudo update-rc.d /etc/init.d/copperd remove
```

2.1.7 FreeBSD

Java実行環境が必要です。pkg install openjdkコマンドにより、Java実行環境をインストールしてください。

copper-pdf-3.x.x.tar.gzを/optディレクトリに展開してください。次に、展開してできたcopper-pdf-3.x.xディレクトリを、copper-pdfという名前に変更してください。

copper-pdfディレクトリ内のextras/freebsd/copperdに起動スクリプトの例があります。これを/usr/local/etc/rc.d/copperdにコピーしchmod 555 /usr/local/etc/rc.d/copperdコマンドにより、実行権限を付与してください。

Java実行環境がopenjdk7の場合は、これでセットアップは完了です。他のJava実行環境を使う場合は、/usr/local/etc/rc.d/copperd.sh を編集し、export JAVA_HOME=/usr/local/openjdk8 の部分を実際のJava実行環境の場所に設定してください。

Copper PDFサーバーの起動・停止

Copper PDFサーバー起動と停止は、serviceコマンドにより行うことができます。

例 2.20 copper-pdfの起動

```
# service copperd start
```

例 2.21 copper-pdfの停止

```
# service copperd stop
```

2.1.8 その他の環境

その他の環境では、tar.gzアーカイブを使用してください。

copper-pdf-3.x.x.tar.gzを適当なディレクトリに展開してください。

Linuxあるいは UNIX系のOSの場合、展開してできたディレクトリ内のシェルスクリプト (copper, copperd, copper-webapp)を使用してください。また、デーモンをセットアップするにはextras/redhat/copperdを編集し、16行目でCOPPER_HOME環境変数にCopperPDFの配置ディレクトリを設定するようにし、適切な場所に配置してください。

例 2.22 JAVA_HOME の設定

```
COPPER_HOME=[CopperPDFの配置ディレクトリ]
```

Copper PDFに付属のシェルスクリプトは、環境変数JAVA_HOME により、Javaのインストールディレクトリを判別します。/etc/profile等で、JAVA_HOME を適切に設定してください。

例 2.23 JAVA_HOME の設定

```
export JAVA_HOME=/usr/local/jdk1.8.0_162
```

JAVA_HOME が設定されていない場合、PATH に加えられているjavaコマンドが実行されます。

各ツールの実行方法の詳細はCopper PDFのツール (28ページ)の解説を参照してください。

アンインストールは、単にCopperPDFディレクトリを削除するだけです。

2.1.9 ディレクトリ構成

アーカイブ内のディレクトリ構成

ZIPまたはtar.gzアーカイブを展開すると、copper-pdf-3_x_xというディレクトリができます。各ツールの実行ファイルは、このディレクトリの直下にあります。その他のディレクトリ構成は次の通りです。

図 2.8 ディレクトリ構成

```
copper-pdf-3_x_x
|-- conf        設定ディレクトリ
|   `-- profiles    プロファイル
|       `-- fonts   フォント設定
|-- docs        ドキュメント
|-- extras      アイコン、各プラットフォーム向けのファイル等
|-- legal       付属ライブラリのライセンス文書
|-- jetty       サーブレットコンテナ(copper-webapp用)
|-- lib         ライブラリ
|-- plugins     プラグイン[2.1.0]
|-- logs        ログディレクトリ
`-- webapp      copper-webapp
```

RPMまたはDebianパッケージの構成

RPMまたはDebianパッケージでインストールした場合は、Copper PDFの各コマンドラインツールは/usr/binと/usr/sbinに配置されます。

また、他のディレクトリの配置場所は次のとおりです。

表 2.1 RPM/Debianのディレクトリ構成

ディレクトリ	配置場所
lib	/usr/share/copper-pdf/lib
plugins*[2.1.0]*	/usr/share/copper-pdf/plugins
docs	/usr/share/doc/copper-pdf
conf	/etc/copper-pdf (/var/lib/copper-pdf/confにシンボリックリンク) ※Copper PDF 2.0系以前では実ファイルとシンボリックリンクが逆転しています。
jetty	/var/lib/copper-pdf/jetty
webapp	/var/lib/copper-pdf/webapp

ディレクトリ	配置場所
logs	/var/log/copper-pdf (/var/lib/copper-pdf/logsにシンボリックリンク)

/etc/init.d/copperdはcopperユーザーで実行されます。ファイルの読み書きが行われる /usr/share/copper-pdf/conf/profiles, /var/log/copper-pdf の各ディレクトリはcopperユーザーの所有となります。

2.1.10 ライセンスキー・ファイルの配置

Copper PDFは、そのままでは機能限定版 (282ページ)として動作します。

Copper PDFを使用するためには、ライセンスキー・ファイルをlicense-keyという名前でconfディレクトリに配置する必要があります。ライセンスキーはhttps://copper-pdf.com/?p=27で購入してください。

全ての機能を試用する場合は、https://copper-pdf.com/?p=155 で試用ライセンスキーを取得してください。

2.1.11 旧バージョンからのアップデート

confディレクトリ内の各種設定は、上位互換です。過去のバージョンの設定は、新しいバージョンで常に動作します。

RPMパッケージまたはdebパッケージは、それぞれ通常の方法でアップデートしてください。

例 2.24 RPMパッケージの更新

```
# sudo rpm -Uvh copper-pdf-3.x.x-0.noarch.rpm
```

例 2.25 debパッケージの更新

```
# sudo dpkg -i copper-pdf_3.x.x_all.deb
```

その他(Windows, .zip, .tar.gzアーカイブ)の場合は、先にconfディレクトリをバックアップし、Copper PDFを再インストールし、confディレクトリをアップデート前のものと置き換えてください。

2.2 Copper PDFのツール

Copper PDFはプログラムとして実行可能なツールで構成されています。それぞれのツールにはLinux/UNIX系OS(拡張子のない実行ファイル), Windows(拡張子が.exeのもの)向けの実行ファイルがあります。

Linux/UNIX系OSでは、次の環境変数が使用されます。

JAVA_HOME
: 使用するJava実行環境のディレクトリ("/usr/java/jdk1.8.0_162"等)です。設定されていない場合は、環境変数PATH の設定により実行できるjavaコマンドの実行環境が使用されます。

JAVA_OPTS
: javaの実行オプションです。

Linux/UNIX系OSでメモリの割り当てを設定する場合はJAVA_OPTS 環境変数を設定してください。また、Java実行環境としてJDKを使用する場合は、-serverオプションを付けることで、サーバー用のJavaVM が使われるため、若干パフォーマンスが向上します。JREでは-serverオプションは使用できないことがあるためご注意ください。

以下の例ではメモリの割り当てを最大1024MBとし、-serverオプションをつけて各ツールを実行します。

例 2.26 JAVA_OPTS の設定

```
export JAVA_OPTS="-Xmx1024m -server"
```

Javaコマンドで直接ツールを実行する方法については、各ツールの説明を参照してください。

copper
: コマンドラインアプリケーションです。コマンドラインからCopper PDFの機能を直接利用することができます。

copper-webapp
: ウェブアプリケーションです。ウェブブラウザからローカルマシン上のファイルや、ウェブ上のコンテンツを変換する、簡単なウェブアプリケーションが起動します。

copperd
: Copper PDFサーバーを起動・停止あるいは状態を確認します。プログラムインターフェースからアクセスするためのパスワードの設定もこのコマンドで行います。

2.2.1 copper コマンドラインアプリケーション



形式

copper [-h | -in <入力ファイル> | -uri <入力URI> | -v] [-ie <入力エンコーディング>] [-if <入力形式>] [-out <出力ファイル>] [-p <プロパティ名=値>] [-pf <プロパティファイル>] [-pw <パスワード>] [-s <サーバーURI>] [-sv] [-u <ユーザー>] [-t]

概要

コマンドラインでドキュメントを変換するアプリケーションです。

オプション

-h, -help
ヘルプメッセージを表示します。

-v, -version
コマンドラインアプリケーションのバージョンを表示します。

-in, --input-file ファイル
入力ファイルを指定します。

-uri, --input-uri URI
入力ファイルのURIを指定します。

-ie, --input-encoding ファイル
入力ファイルのキャラクタ・エンコーディングを指定します。

-if, --input-format ファイル
入力ファイル形式を指定します。デフォルトはtext/htmlです。

-out, --output-file ファイル
出力先ファイルを指定します。

-p 名前=値
プロパティを指定します。

-pf, --properties-file ファイル
プロパティファイルを指定します。

-s, --server URI
ドキュメント変換サーバーのURIを指定します*[2.1.0]*。省略するとローカルマシンのライブラリ "copper:direct:" を直接使用します。



Copper PDF 3.0.12以前ではCopper PDFサーバー付属のcopperコマンドでは、-sには"copper:direct:"以外使用できません。"ctip://"、"http://"等で始まるURIを指定するには、Java版ドライバ(2.1.3以降)付属のcopperコマンドを使用して下さい。

-u, --user ユーザー名

認証のためのユーザー名です*[2.1.0]*。

-pw, --password パスワード

認証のためのパスワードです*[2.1.0]*。

-t, --trust

TLS(SSL)接続で常に証明書を信頼します*[3.2.0]*。

-sv, --server-version

ドキュメント変換サーバーのバージョン情報を表示します*[2.1.0]*。

説明

コマンドラインで実行可能なドキュメント変換ツールです。

ツール実行の度にJavaVM とCopper PDFを起動するため、プログラムインターフェースを用いた処理より遅くなります。また、マニュアルの作成等、複数のドキュメントの一括変換を行う場合は transcode Antタスクを使う方が効率的です。

Linux版のcopperおよびcopper-webappは実行ユーザーのホームディレクトリに置かれた設定($HOME/.copper ディレクトリ内)を用います。詳細はprofilesディレクトリの説明を参照してください。

Copper PDF 2.1.0以降では、ネットワーク越しに起動中のサーバーに接続して、コマンドを実行することができます。-s, -u, -pwで接続情報を指定してください。ドキュメント変換サーバーのURIの記述方法は 接続情報 (73ページ) を参照してください。

入力について

- -inオプションが省略された場合は、標準入力を使用します。
- -inオプションと-uriオプションが両方指定された場合、-inで指定されたファイルのデータが使われ、ドキュメントのURIは-uriで指定されたものと見なします。
- -inオプションだけ指定された場合、-inで指定されたファイルのデータが使われ、ドキュメントのURIはそのファイルのシステムURIとします。
- -uriオプションだけが指定された場合、そのURIから取得できるデータが使われ、ドキュメントのURIも同じURIとします。**このとき-ie, -ifオプションは無視されます。**

出力について

- -outオプションが省略された場合は、標準出力を使用します。
- -outオプションが指定された場合は、指定されたファイルに出力します。

javaコマンドで実行する方法

javaコマンドで直接実行するには、libディレクトリ内のboot.jarを-jarオプションにより実行してください。また、以下のシステムプロパティを-Dオプションにより設定してください。

jp.cssj.boot.lib
: libディレクトリのパスを設定してください。

jp.cssj.plugin.lib
: pluginsディレクトリのパスを設定してください*[2.1.0]*。

jp.cssj.boot.main
: "jp.cssj.driver.cli.Main" を設定してください。

jp.cssj.driver.default
: default.propertiesファイルのパスを設定してください。

jp.cssj.copper.config
: confディレクトリのパスを設定してください。

2.2.2 copper-webapp ウェブアプリケーション

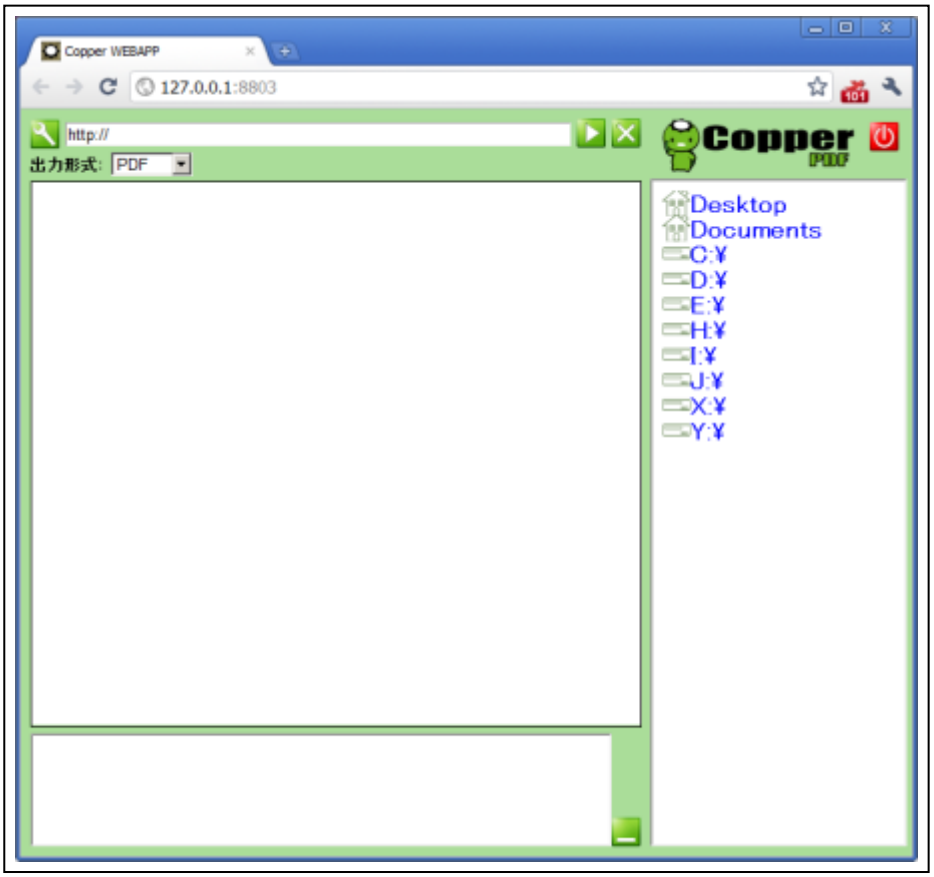


概要

ウェブアプリケーションとして動作し、ブラウザ上でドキュメントを変換するツールです。ウェブページや静的なHTMLファイルを変換してレイアウトを確認するために用いることができます。

プログラムを起動後、ブラウザで http://localhost:8803/ にアクセスすると、Copper PDFのウェブアプリケーションが利用できます。なお、ブラウザが実行可能な環境では、プログラムの起動後に自動的にブラウザが開きます。

図 2.9 Copper WEBAPPの初期画面



Windows, X Window Systemなど、GUI環境でcopper-webappを起動すると、以下のような小さなウィンドウが表示され、ブラウザの起動と、ツールの停止を行うことができます。

図 2.10 copper-webappのウィンドウ



javaコマンドで実行する方法

javaコマンドで直接実行するには、jettyディレクトリ内のstart.jarを-jarオプションにより実行してください。また、以下のシステムプロパティを-Dオプションにより設定してください。

jetty.base

jettyディレクトリのパスを設定してください。

jetty.home

jettyディレクトリのパスを設定してください。

jp.cssj.driver.default

default.propertiesファイルのパスを設定してください。

jp.cssj.copper.config
: confディレクトリのパスを設定してください。

jp.cssj.webapp.config
: copper-webappの設定ファイルのパスを設定してください。*[2.0.1]*

jp.cssj.plugin.lib
: pluginsディレクトリのパスを設定してください*[2.1.0]*。

2.2.3 copperd ドキュメント変換サーバー



形式

copperd [-cc <制御コマンド>] [-cf <ディレクトリ>] [-cp <ポート番号>] [-h | -v] [-hp <ポート番号>] [-jkp <ポート番号>] [-kill] [-p <ポート番号>] [-passwd <パスワード>] [-skp <キーのパスワード>] [-sp <ポート番号>] [-ss <キーストアのファイルパス>] [-ssp <キーのパスワード>] [-start] [-status] [-stop] [-tkp <キーのパスワード>] [-tp <ポート番号>] [-ts <キーストアのファイルパス>] [-tsp <キーのパスワード>] [-user <ユーザー名>] [-userdel <ユーザー名>]

概要

Copper PDFサーバーの起動・停止と、パスワードの設定ツールです。

サーバーの起動、停止はこのコマンドを使用するよりは、デーモンあるいはサービスを使ったほうが便利です。詳細はセットアップドキュメント (18ページ)を参照してください。

オプション

-h, -help
: ヘルプメッセージを表示します。

-v, -version
: Copper PDFサーバーのバージョンを表示します。

-start
: サーバーを起動します。

-stop
: サーバーを停止します。

-kill
: サーバーを強制停止します。

-status
動作中のサーバーの状態を表示します。

-cc, --control-command 制御コマンド文字列
制御コマンドを指定します。

-cf, --config ディレクトリ
設定ディレクトリを指定します。

-cp, --control-port ポート番号
制御ポートを指定します。

-p, --port ポート番号
サービスポートを指定します。

-passwd パスワード
接続パスワードを変更します。

-user ユーザー名
追加またはパスワードを変更するユーザーです*[2.1.0]*。

-userdel ユーザー名
削除するユーザーです*[2.1.0]*。

-hp, --http-port ポート番号
HTTP/RESTインターフェースを起動するポート番号です*[2.1.0]*。

-sp, --https-port ポート番号
HTTP/RESTインターフェースをSSLで起動するポート番号です*[2.1.0]*。

-ss, --https-keystore
SSLに使用するキーストアのファイルパスです*[2.1.0]*。

-ssp, --https-keystorepassword
SSLに使用するキーストアのパスワードです*[2.1.0]*。

-skp, --https-keypassword
SSLに使用するキーのPKCS12パスワードです*[2.1.0]*。

-tp, --tls-port ポート番号
TLSインターフェースを起動するポート番号です*[3.0.0]*。

-ts, --tls-keystore
TLSに使用するキーストアのファイルパスです*[3.0.0]*。

-tsp, --tls-keystorepassword
TLSに使用するキーストアのパスワードです*[3.0.0]*。

-tkp, --tls-keypassword
TLSに使用するキーのPKCS12パスワードです*[3.0.0]*。

説明

各プログラミング言語向けのインターフェースを利用するには、ドキュメント変換サーバーを常駐させる必要があります。また、ドキュメント変換サーバーへはネットワークを介してアクセスできるため、ドキュメント変換サーバーと、それを利用するアプリケーションを別々のマシン上で動かすことができます。

copperdの設定ファイルはcopperd.propertiesです。サービスポート(-p)、制御ポート(-cp)、制御コマンド(-cc)、はデフォルトでは設定ファイルのものが使われ、

Copper PDF 2.1.0以降ではHTTPポート(-hp)、SSLポート(-hs)とSSLに使用するサーバーキーの情報(-ss, -ssp, skp)を設定可能です。SSLに使用するサーバーキーの設定方法はSSLの設定 (39ページ)を参照してください。

上記の設定は、デフォルトでは設定ファイルのものが使われ、オプションは設定ファイルによる設定を上書きします。

サーバーの停止(-stop)および状態の取得(-status)には制御用ポートと、制御用コマンドを用います。起動時(-start)の設定と、実行中のサーバーを制御する際の設定は同じである必要があります。

-stopコマンドは、現在接続中のプログラミングインターフェースの処理が全て完了してからサーバーを停止します。-killコマンドは全ての処理を強制的に中断してサーバーを停止します。

copperdにはパスワード設定機能(-passwd)があります。パスワードファイルは暗号化されており、直接編集できないため、このツールを利用して編集してください。Copper PDF 2.1.0以降では-user, -userdelオプションによりユーザーの追加と削除ができます。

javaコマンドで実行する方法

javaコマンドで直接実行するには、libディレクトリ内のboot.jarを-jarオプションにより実行してください。また、以下のシステムプロパティを-Dオプションにより設定してください。

user.home
logsディレクトリが置かれているディレクトリのパスを設定してください。

java.util.logging.config.file
logging.propertiesファイルのパスを設定してください。

jp.cssj.boot.lib
libディレクトリのパスを設定してください。

jp.cssj.plugin.lib

pluginsディレクトリのパスを設定してください*[2.1.0]*。

jp.cssj.boot.main

"jp.cssj.copper.Main"を設定してください。

jp.cssj.driver.default

default.propertiesファイルのパスを設定してください。

jp.cssj.copper.config

confディレクトリのパスを設定してください。

2.3 設定ファイル

confディレクトリの構成は次の通りです。

図 2.11 設定ディレクトリの構成

```
conf
|-- license-key
|-- copperd.properties
|-- logging.properties
|-- password.txt
|-- access.txt
`-- profiles
    |-- default.properties
    `-- fonts
        |-- fonts.xml
        |-- truetype
        |-- warrays
        |-- afms
        |-- cmaps
        `-- encodings
```

confディレクトリの直下には次の設定ファイルが収められています。

license-key
ライセンスキーファイルです。

copperd.properties
Copper PDFサーバーの動作設定です。

logging.properties
ログの設定です。

password.txt
パスワードの設定です。編集にはcopperdコマンドを使ってください。

access.txt
アクセス制御設定です。

また、profilesディレクトリ内には次のファイルがあります。

default.properties
デフォルトの入出力プロパティです。

fonts/fonts.xml
使用するフォントの設定です。

2.3.1 Copper PDFサーバーの動作設定(copperd.properties)

copperd.propertiesはCopper PDFサーバー(copperd)の動作に関する設定です。この設定の変更を反映するにはCopper PDFサーバーの再起動が必要です。

ファイルの形式は Javaのプロパティファイル です。各行に 名前=値 の形式で記述してください。= 記号、改行はそれぞれ\= \nで記述します。\の次の改行と空白は無視されます。

例えば以下の場合、key1 の値は a=b であり、key2 の値は a, b, c, d, e, f です。

例 2.27 propertiesファイルの記述例

```
key1 =  a\=b
key2 =  a, b, c,\
        d, e, f
```

propertiesファイルには、日本語のカナ、漢字等をそのまま記述できません。日本語を含むpropertiesファイルは、Javaに付属の native2ascii ツールにより、マルチバイト文字をエスケープした形式に変換してください。

各プロパティの説明は以下の通りです。

表 2.2 copperd.propertiesのプロパティー覧

プロパティ	説明
jp.cssj.cssjd.port	サービスポートです。 このポート番号がプログラムインターフェースの接続先のポート番号となります。
jp.cssj.cssjd.control-port	制御用ポートです。サーバーの状態の取得や停止のためにこのポートを使います。このポートへのlocalhost以外からのアクセスはブロックされます。
jp.cssj.cssjd.control-command	制御用コマンドです。制御用ポートからアクセスする場合の「合言葉」のようなものです。通常は変更する必要はありません。
jp.cssj.cssjd.timeout	接続タイムアウト(秒数)です。 プログラムインターフェースとの間で指定された時間データがやりとりされなかった場合、接続を切断します。
jp.cssj.cssjd.minThreads	最小スレッド数です。 プログラムインターフェースの接続を待ち受けるために準備しておく最小限のスレッド数です。高負荷が予想される状況では大きめの値を設定しておくと処理が効率化しますが、使用するメモリは増大します。

プロパティ	説明
jp.cssj.cssjd.maxThreads	最大スレッド数です。 copperdが同時に処理できるプログラムインターフェースからの接続の最大数です。高負荷が予想され、なおかつシステムのリソースに余裕がある場合は大きな値を設定するのが有効ですが、リソースに余裕がない環境では、大きな値を設定することで逆に非効率になることがあります。
jp.cssj.cssjd.backlog	接続のバックログ数です。 未処理の接続がこの数を超えた場合、処理に空きができるまで以降の接続は拒否します。
jp.cssj.cssjd.http.port	HTTPポート番号です。 指定したポートで、HTTP/RESTインターフェースを起動します*[2.1.0]*。
jp.cssj.cssjd.https.port	SSL(HTTPS)ポート番号です。 指定したポートで、SSLでHTTP/RESTインターフェースを起動します*[2.1.0]*。
jp.cssj.cssjd.https.keyStore	SSLに使用するキーストアのファイルパスです*[2.1.0]*。
jp.cssj.cssjd.https.keyPassword	SSLに使用するキーのPKCS12パスワードです*[2.1.0]*。
jp.cssj.cssjd.https.keyStorePassword	SSLに使用するキーストアのパスワードです*[2.1.0]*。
jp.cssj.cssjd.tls.port	TLSポート番号です。 指定したポートで、TLSによるプログラムインターフェースを起動します*[3.0.0]*。
jp.cssj.cssjd.tls.keyStore	TLSに使用するキーストアのファイルパスです*[3.0.0]*。
jp.cssj.cssjd.tls.keyPassword	TLSに使用するキーのPKCS12パスワードです*[3.0.0]*。
jp.cssj.cssjd.tls.keyStorePassword	TLSに使用するキーストアのパスワードです*[3.0.0]*。

2.3.2 SSL/TLSの設定

Copper PDFは高速な独自の通信プロトコル(CTIP)の他、Copper PDF 2.1.0からはHTTPによる接続をサポートしました。さらに、SSL(HTTPS)による暗号化された接続も可能です。

Copper PDF 3.0.0からはTLSによる、暗号化された高速な接続をサポートしました。

SSLを有効にするためには、キーペアを用意してください。以下はOpenSSLとJDK付属のkeytoolを使用する方法です。

秘密鍵とCSRを用意する

以下のコマンドで秘密鍵とCSR(証明書署名要求)を生成してください。パスフレーズは任意に設定して構いません。

例 2.28 秘密鍵の生成

```
openssl genrsa -out ssl.key 2048
```

例 2.29 CSRの生成

```
openssl req -new -key ssl.key -out ssl.csr
```

サイト証明書を用意する

サイト証明書は、前に生成したCSRを認証局に送付することで入手することができますが、テストや単に通信の暗号化の目的であれば自己署名によるサイト証明書を使用することができます。以下のコマンドでサイト証明書を生成してください。

例 2.30 自己署名によるサイト証明書の生成

```
openssl x509 -in ssl.csr -out ssl.crt -req -signkey ssl.key -days
3650
```

秘密鍵とサイト証明書をキーストアに格納する

秘密鍵とサイト証明書を利用するには、JDK付属のkeytoolによって管理されるキーストアファイルに格納する必要があります。最初に、以下のコマンドで秘密鍵とサイト証明書のペアをPKCS12形式のファイルにまとめてください。このとき、設定するパスワードがPKCS12パスワードです。

例 2.31 PKCS12キーペアを生成

```
openssl pkcs12 -inkey ssl.key -in ssl.crt -export -out ssl.pkcs12
```

次に、PKCS12ファイルをキーストアに格納してください、このとき、キーストアのパスワードを設定します。

例 2.32 PKCS12キーペアをキーストアに格納

```
keytool -importkeystore -srckeystore ssl.pkcs12 -srcstoretype
PKCS12 -destkeystore keystore
```

以上の手順で生成したキーをCopper PDFサーバーのSSL/TLS通信に使用する場合は、設定ファイルまたはコマンドラインオプションでキーストアファイルのファイルパスと、PKCS12パスワード、キーストアのパスワードを設定してください。

2.3.3 ログの設定(logging.properties)

logging.propertiesはCopper PDFサーバーのログ設定ファイルです。この設定の変更を反映するにはCopper PDFサーバーの再起動が必要です。ログ設定ファイルはjava.util.loggingのログプロパティファイルの形式で記述してください。

例 2.33 ログの設定例

```
# Logging levels by categories.
jp.cssj.handlers = jp.cssj.logging.FileHandler, \
    java.util.logging.ConsoleHandler
jp.cssj.level = INFO

# File
jp.cssj.logging.FileHandler.level = INFO
jp.cssj.logging.FileHandler.formatter =
java.util.logging.SimpleFormatter
jp.cssj.logging.FileHandler.encoding = UTF-8
# ログを保持する日数を設定する（Copper PDF 3.1.2 以降）
jp.cssj.logging.FileHandler.backlogs = 365

# Console
java.util.logging.ConsoleHandler.level = OFF
java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter
```

Copper PDFには日付でログをローテーションするログハンドラ(jp.cssj.logging.FileHandler)が用意されています。ハンドラのプロパティは次のとおりです。

表 2.3 jp.cssj.logging.FileHandlerのプロパティ

名前	デフォルト	説明
jp.cssj.logging.FileHandler.directory	copperdの標準のlogsディレクトリ	ログの出力先ディレクトリ
jp.cssj.logging.FileHandler.prefix	copperd.	ログファイル名の先頭の文字列
jp.cssj.logging.FileHandler.suffix	.log	ログファイル名の末尾の文字列
jp.cssj.logging.FileHandler.level	ALL	ログレベル
jp.cssj.logging.FileHandler.filter	N/A	ログのフィルタ
jp.cssj.logging.FileHandler.formatter	java.util.logging.SimpleFormatter	ログのフォーマッタ
jp.cssj.logging.FileHandler.encoding	UTF-8	ログファイルのキャラクタエンコーディング

名前	デフォルト	説明
jp.cssj.logging.FileHandler.backlogs	ログファイルを永久に保存する	過去のログファイルを保存する日数

2.3.4 アクセス制御の設定(access.txt)

access.txtはIPアドレスでアクセスできるクライアントを制限する設定です。このファイルの変更は自動的に検出されるため、変更後にCopper PDFサーバーを再起動する必要はありません。各行のルールを

`[allow|deny] IPアドレス`

という形式で記述します。IPアドレスはIPv4, IPv6のどちらの形式も使用可能です。

allowは指定されたアドレスのアクセスを許可し、denyはアクセスを拒否するルールです。クライアントからのアクセスがあった場合は、先頭から順に検索し、最初に一致したルールが適用されます。

IPアドレスを * にすると、全てのクライアントにルールを適用します。

例 2.34 アクセス制御の設定例

```
allow   127.0.0.1
deny    *
```

2.3.5 profilesディレクトリ

profilesディレクトリ内の設定は、Copper PDFの出力結果に関係するものです。デフォルトの入出力プロパティやフォントの情報が含まれます。

Linux版のcopperdおよび、Windows版の各種ツールはconfディレクトリ内のprofilesディレクトリの設定を使用しますが、Linux版の copper copper-webapp は初回実行時に実行ユーザーのホームディレクトリに.copperという名前のディレクトリをつくり、そこにprofilesディレクトリを作成します。copper, copper-webappを使用する各ユーザーはホームディレクトリの設定を編集する必要があります。

copper-webappの設定ファイルは、初回実行時に /WEB-INF/client.propertiesから各ユーザーの$HOME/.copper/webapp.proprtiesにコピーされます。*[2.0.1]*

2.3.6 デフォルトの入出力プロパティ(default.properties)

default.propertiesはデフォルトの入出力プロパティを設定するものです。入出力プロパティはcopperコマンドの引数やcopper-webappの設定ウィンドウ、プログラムインターフェースによって上書きされます。default.propertiesは、その初期値をあらかじめ設定するものです。

default.propertiesの変更は実行中のCopper PDFサーバーにより自動的に検知されますので、変更の度にCopper PDFサーバーを再起動する必要はありません。

ファイルの形式は Javaのプロパティファイル です。各プロパティは入出力プロパティの節で解説されていますが、１つだけsystem.fontsという特別なプロパティがあります。system.fontsは後で次に説明するフォント設定ファイルをdefault.propertiesファイルからの相対パスで指定したものです。

2.3.7 fontsディレクトリ

フォント関連の設定です。詳細はフォントの設定の章で解説します。

2.4 フォントの設定

Copper PDFはフォントを埋め込まないPDFと、フォントを埋め込んだPDFの両方を出力することができます。デフォルトではフォントを埋め込みません。また、機能限定版ライセンスではフォントの埋め込みはできません。フォントを埋め込まないPDFはサイズが小さくなりますが、表示する環境によっては全く意図しないフォントが使われてしまったり、文字化けが生じてしまったり、まったく表示できないことがあります。重要な文書、特に同じ表示が保障される必要がある場合は、フォントを埋め込んでください。



公開Dockerイメージには Noto Sans JP および Noto Serif JP が含まれています。公開Dockerイメージでは既定で埋め込みフォントを使う設定になっており、ライセンスキーがある場合は、serif・sans-serifの和文がNotoで組まれてPDFへ埋め込まれます(欧文は従来どおりTimes・Helvetica)。機能限定版ではフォントを埋め込めないため、Notoは使われず、従来どおり埋め込まない明朝・ゴシックになります。追加フォントを使う場合は、利用条件を確認したフォントファイルをDockerスターターの `user/fonts` へ配置し、`copper.ps1 restart` または `./copper.sh restart` を実行してください。

`user/fonts` だけがコンテナ内の専用ディレクトリへ読み取り専用で配置されるため、組み込みのNotoフォントは隠れません。独自のfonts.xml、CMapまたは幅情報が必要な場合は、フォントプロファイル一式を別途設定してください。

埋め込むフォントの指定は、Copper PDF 3.0.0 からはCSSの@font-faceルール(WebFont)を使うこともできますが、高速に処理させるためには、あらかじめ設定しておくことを推奨します。フォント関連の設定はprofilesディレクトリ 内のfontsディレクトリに含まれています。まだ、デフォルトの設定では、ここにフォントファイルも置くことになります。fontsディレクトリ内の構成は次のとおりです。

図 2.12 fontsディレクトリの構成

```
fonts
|-- fonts.xml
|-- truetype
|-- warrays
|-- afms
|-- cmaps
`-- encodings
```

fonts.xml

フォント設定ファイルです。Copper PDFが使用するフォントの設定が記述されています。

truetype
: 出荷時のfonts.xmlの設定では、このディレクトリ内のフォントファイルを自動的に読み込むようになっています。使用したいフォントをここにコピー(あるいはシンボリックリンク)して、設定を反映してください。truetypeというディレクトリ名はCopper PDF 2.0.2までTrueTypeフォントしか使用できなかったためで、実際はOpenType CFF/Type2(.otf)フォント、WOFF*[3.1.0]*にも対応しています。

warrays
: CID-Keydフォントを使用するために、既定のフォントの幅情報が配置されています。

cmaps
: CID-KeydフォントのためのCMapファイルが配置されています。

afms
: コアフォントのフォントメトリックス情報がAFM形式で配置されています。

encodings
: コアフォントのための文字名と各エンコーディングとの対応表が配置されています。

2.4.1 フォント設定の反映

Copper PDFはfonts.xmlを解析し、フォントへアクセスするための情報をfonts.xmlが配置されているのと同じディレクトリに、fonts.xml.dbというファイル名で保存します。Copper PDFはfonts.xmlの変更を自動検出するためfonts.xmlへの変更はすぐに反映されます。

ただし、OSへのフォントのインストールや、truetypeディレクトリ内におけるフォントファイルの追加・削除・置き換え等を自動検出しません。変更を反映させるためには、touchコマンド等でfonts.xmlファイルのタイムスタンプを変更してください。

2.4.2 ドキュメント中でのフォントの利用

ドキュメント中で使用されるフォントを特定するためのCSSプロパティは`font-family`[css](ファミリ名), `font-weight`[css](太さ), `font-size`[css](大きさ), `font-style`[css](スタイル)の４つです(`font`[css]プロパティでまとめて指定することもできます)。指定されたファミリ名、太さ、スタイルから、記述された文字を表記できる最も近いフォントが選択されます。

該当する太さのフォントが見つからない場合は、機械的に文字の輪郭を拡張して太いフォントがつくられます(逆に指定した太さより細いフォントがない場合、機械的に細いフォントをつくることは**ありません**。最も細いフォントが使われるだけです)。

また、font-style [css]にitalicまたはobliqueが指定されたとき、斜体スタイルのフォントが見つからない場合は、フォントを機械的に傾けます。同時にfont-weight [css]が指定されている場合は、以下の表のとおり、font-style [css]の指定がitalicかobliqueかで選択されるフォントが異なることがあります。

表 2.4 フォントスタイルの選択

条件	italic指定の場合	oblique指定の場合
太さとスタイルが一致するフォントがある	太さとスタイルが一致するフォントを使用する	太さとスタイルが一致するフォントを使用する
太さだけ一致するフォントがある	太さが一致するフォントを傾けて使用する	太さが一致するフォントを傾けて使用する
スタイルだけ一致するフォントがある	スタイルが一致するフォントを太くして使用する	スタイルが一致するフォントを太くして使用する
太さだけ一致するフォントとスタイルだけ一致するフォントがある	スタイルが一致するフォントを太くして使用する	太さが一致するフォントを傾けて使用する
太さもスタイルも一致するフォントがない	他の条件が一致するフォントを太くして傾けて使用する	他の条件が一致するフォントを太くして傾けて使用する

2.4.3 デフォルトのフォント

ドキュメント中でフォントが指定されていない場合、あるいは指定されたフォントが見つからない場合は、output.default-font-family [io] により設定されたフォントが使われます。これはデフォルトではserif(ローマンあるいは明朝体)です。

CMapに該当するコードがない、コアフォントにも該当する文字がない、かつインストール済みのフォントにも該当する文字がないといった理由で、どうしても表示できない文字がドキュメント中に記述された場合は、代わりに16進数でユニコードを表す組文字(のような文字)が表示されます。

2.4.4 フォントの種類

Copper PDFがサポートするフォントには コアフォント、CID-Keyed フォント CID Identityフォント、埋め込みフォント、の４種類があります。コアフォントは常に利用可能ですが、埋め込みフォント、CID Identityフォント、CID-Keyedフォントのどれを利用するかは output.pdf.fonts.policy [io] によって選択されます。また、Copper PDFの拡張CSSプロパティ -cssj-font-policy [css] によってドキュメント中で指定することもできます。デフォルトではコアフォントとCID-Keyedフォントだけが使われます。

output.pdf.fonts.policy[io] または -cssj-font-policy[css] に cid-keyed, cid-identity, embeddedを指定することで、それぞれCID-Keyedフォント、CID Identityフォント、埋め込みフォントを切り替えることができます。コアフォントは常に使用されますが、"-core"を指定することで除外することができます*[3.0.0]*。またスペース区切で複数指定することも可能です*[2.0.1]*。"embedded cid-keyed"のようにスペース区切りで複数指定された場合は、最初に指定されたものから優先的に使用されます*[2.0.9]*。ただし、コアフォントの優先度は常に最低となります。また、outlinesを指定すると、PDFでembeddedフォントをテキストではなくアウトライン化した状態で表示することができます*[3.1.1]*。

PDF/A-1を出力する場合は、上記の設定に関わらず、埋め込みフォントだけが使われます*[2.1.0]*。

outlinesを指定すると、埋め込みフォントが使われ、なおかつ全てのフォントがアウトライン化されます。*[3.1.1]*。outlines指定はPDF/A-1でも有効です。

2.4.5 コアフォント

コアフォントは、ほとんどのPDF表示環境が標準的にサポートしているフォントで、フォントの埋め込みをせずに表示することができます。14種類あることから、コア14フォントとも呼ばれます。コアフォントはさらにletter(欧文文字・記号だけで構成されるもの)、symbol(欧文文字・記号以外の文字を含むもの)の2種類に分けられます。

次の表はコアフォントの一覧です。

表 2.5 コアフォント

正式名称	略称	ファミリ名	太字	斜体	種類
Times Roman	Times-Roman	Times			letter
Times Bold	Times-Bold	Times	✔		letter
Times Italic	Times-Italic	Times		✔	letter
Times Bold Itatdc	Times-BoldItalic	Times	✔	✔	letter
Helvetica	Helvetica	Helvetica			letter
Helvetica Bold	Helvetica-Bold	Helvetica	✔		letter
Helvetica Oblique	Helvetica-Oblique	Helvetica		✔	letter
Helvetica Bold Oblique	Helvetica-BoldOblique	Helvetica	✔	✔	letter
Courier	Courier	Courier			letter
Courier Bold	Courier-Bold	Courier	✔		letter
Courier Oblique	Courier-Oblique	Courier		✔	letter
Courier Bold Oblique	Courier-BoldOblique	Courier	✔	✔	letter

正式名称	略称	ファミリ名	太字	斜体	種類
Symbol	Symbol	Symbol			symbol
ITC Zapf Dingbats	ZapfDingbats	ZapfDingbats			symbol

CSSの`font-family`[css]プロパティによるフォントの指定は、正式名称、略称、ファミリ名のいずれでも可能です。ファミリ名を使用した場合は、`font-style`[css], `font-weight`[css] プロパティの指定により、同じファミリ名を持つフォントのうち、スタイルと太さが最も一致するフォントが自動的に選択されます。

コアフォントのフォントメトリックス情報(文字の幅などの情報)がAFM(Adobe Font Metrics)ファイルとして、fonts/afmsディレクトリに収められています。これらのファイルをユーザーが変更する必要はありません。

各文字とPDF内で使用される文字コードとの対応表が fonts/encodingsディレクトリに収められています。UNICODE.txtはletterフォントの文字名とユニコードとの対応表です。symbol.txtとzdingbat.txtはそれぞれSymbol, ITC Zapf Dingbatsのためのユニコード対応表です。これらのファイルをユーザーが変更する必要はありません。

以下はletterフォントで使用できる文字セット(WinAnsiEncodingエンコーディング)の文字一覧表です。それぞれの文字は8ビット(16進数で2桁)のユニコードに対応しており、縦が上位桁、横が下位桁です。ただし、80から9Fまでのコードで空いている部分は対応

する文字がないことを表し、下に4桁の16進数字がある文字には、同じ文字に対して2つのコードがあります。例えばダブルダガー(‡)を表示する場合はドキュメント中で‡または&#x2021と表記してください。

表 2.6 WinAnsiEncodingの文字一覧

-	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8	€ 20AC		‚ 201A	*f* 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	‰ 2030	Š 0160	‹ 2039	Œ 0152		Ž 017D	
9		‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	š 0161	› 203A	œ 0153		ž 017E	Ÿ 0178
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

以下はSymbolで使用できる文字の一覧です。文字の下の数字は16進ユニコードです。

表 2.7 Symbolの文字一覧

0020 00A0	! 0021	∀ 2200	# 0023	∃ 2203	% 0025	& 0026	∍ 220B	(0028	) 0029	* 2217	+ 002B	, 002C	− 2212	. 002E	/ 002F
0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
≅ 2245	Α 0391	Β 0392	Χ 03A7	Δ 0394 2206	Ε 0395	Φ 03A6	Γ 0393	Η 0397	Ι 0399	ϑ 03D1	Κ 039A	Λ 039B	Μ 039C	Ν 039D	Ο 039F
Π 03A0	Θ 0398	Ρ 03A1	Σ 03A3	Τ 03A4	Υ 03A5	ς 03C2	Ω 03A9 2126	Ξ 039E	Ψ 03A8	Ζ 0396	[005B	∴ 2234	] 005D	⊥ 22A5	_ 005F
¯ F8E5	α 03B1	β 03B2	χ 03C7	δ 03B4	ε 03B5	φ 03C6	γ 03B3	η 03B7	ι 03B9	φ 03D5	κ 03BA	λ 03BB	μ 00B5 03BC	ν 03BD	ο 03BF
π 03C0	θ 03B8	ρ 03C1	σ 03C3	τ 03C4	υ 03C5	ϖ 03D6	ω 03C9	ξ 03BE	ψ 03C8	ζ 03B6	{ 007B	| 007C	} 007D	~ 223C	
€ 20AC	ϒ 03D2	′ 2032	≤ 2264	⁄ 2044 2215	∞ 221E	ƒ 0192	♣ 2663	♦ 2666	♥ 2665	♠ 2660	↔ 2194	← 2190	↑ 2191	→ 2192	↓ 2193
° 00B0	± 00B1	″ 2033	≥ 2265	× 00D7	∝ 221D	∂ 2202	• 2022	÷ 00F7	≠ 2260	≡ 2261	≈ 2248	… 2026	| F8E6	— F8E7	↵ 21B5
ℵ 2135	ℑ 2111	ℜ 211C	℘ 2118	⊗ 2297	⊕ 2295	∅ 2205	∩ 2229	∪ 222A	⊃ 2283	⊇ 2287	⊄ 2284	⊂ 2282	⊆ 2286	∈ 2208	∉ 2209
∠ 2220	∇ 2207	® F6DA	© F6D9	™ F6DB	∏ 220F	√ 221A	· 22C5	¬ 00AC	∧ 2227	∨ 2228	⇔ 21D4	⇐ 21D0	⇑ 21D1	⇒ 21D2	⇓ 21D3
◊ 25CA	〈 2329	® F8E8	© F8E9	™ F8EA	∑ 2211	⎛ F8EB	⎜ F8EC	⎝ F8ED	⎡ F8EE	⎢ F8EF	⎣ F8F0	⎡ F8F1	⎧ F8F2	⎣ F8F3	⎢ F8F4
	〉 232A	∫ 222B	⌠ 2320	⎮ F8F5	⌡ 2321	⎞ F8F6	⎟ F8F7	⎠ F8F8	⎤ F8F9	⎥ F8FA	⎦ F8FB	⎤ F8FC	⎫ F8FD	⎦ F8FE	

以下はZapfDingbatsで使用できる文字の一覧です。文字の下の数字は16進ユニコードです。

表 2.8 ZapfDingbatsの文字一覧

0020 00A0	2701	2702	2703	2704	260E	2706	2707	2708	2709	261B	261E	270C	270D	270E	270F
2710	2711	2712	2713	2714	2715	2716	2717	2718	2719	271A	271B	271C	271D	271E	271F
2720	2721	2722	2723	2724	2725	2726	2727	2605	2729	272A	272B	272C	272D	272E	272F
2730	2731	2732	2733	2734	2735	2736	2737	2738	2739	273A	273B	273C	273D	273E	273F
2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	274A	274B	25CF	274D	25A0	274F
2750	2751	2752	25B2	25BC	25C6	2756	25D7	2758	2759	275A	275B	275C	275D	275E	
F8D7	F8D8	F8D9	F8DA	F8DB	F8DC	F8DD	F8DE	F8DF	F8E0	F8E1	F8E2	F8E3	F8E4		
	2761	2762	2763	2764	2765	2766	2767	2663	2666	2665	2660	2460	2461	2462	2463
2464	2465	2466	2467	2468	2469	2776	2777	2778	2779	277A	277B	277C	277D	277E	277F
2780	2781	2782	2783	2784	2785	2786	2787	2788	2789	278A	278B	278C	278D	278E	278F
2790	2791	2792	2793	2794	2192	2194	2195	2798	2799	279A	279B	279C	279D	279E	279F
27A0	27A1	27A2	27A3	27A4	27A5	27A6	27A7	27A8	27A9	27AA	27AB	27AC	27AD	27AE	27AF
	27B1	27B2	27B3	27B4	27B5	27B6	27B7	27B8	27B9	27BA	27BB	27BC	27BD	27BE	

2.4.6 CIDフォント

欧文以外の文字をPDFに含める場合は、CIDフォントを用います。フォントを埋め込む方法と、フォントを埋め込まない方法があり、フォントを埋め込まない方法には、さらにCID IdentityとCID-Keyedフォントの２種類の方法があります。どの種類のフォントを利用するかは、入出力プロパティ`output.pdf.fonts.policy`[io]の設定によります。

埋め込みフォントまたはCID Identityを使用する場合、Copper PDFにフォントをインストールする必要があります。Copper PDFの出荷時の状態では、truetypeディレクトリ配置したフォントが自動的に読み込まれるようにfonts.xmlが設定されています。

埋め込みフォント

PDFにフォントの字体データそのものを埋め込む方式で、環境に関係なく、確実に同じ字体で文字が表示されることが保証されます。

フォントを埋め込む場合、文書中で使用されている文字のフォントだけをPDFに含めるため、ファイルサイズは最小限に抑えられますが、出力されたPDFは編集や加工には適しません。

表示や印刷の見栄えに厳密さが求められる場合や、広く配布する文書、あるいは長期保存する文書に適しています。

CID Identity

フォントの埋め込みをせず、グリフID(フォントファイルに含まれる文字の番号)をそのままPDF内に記述する方式です。Copper PDFの動作環境と、PDFを表示する環境に同一のフォントがインストールされている必要があります。グリフIDは特定のフォントファイルに依存するため、PDFを表示する環境にインストールされたフォントファイルが異なれば、似たような書体のフォントであっても文字化けが発生するか、全く表示できなくなります。

同一のマシン上や、同じ組織内での編集や加工、印刷を目的とする文書に適しています。フォントの埋め込みをする場合にくらべて、出力されるPDFのサイズは小さくなります。

CID-Keyedフォント

CID Identityフォント同様にフォントの埋め込みをしませんが、Adobe社により公開されているコード体系(CMap)を使用します。

PDFを表示する環境で利用可能な、書体の近いフォントが自動的に選ばれるので、特定のフォントファイルに依存することはありません。ただし、使用できる文字はAdobe社が提供するCMapファイルで定義された文字に限られます。また、全ての環境でPDFを表示できることが保証されるものではありません。

特定の言語環境のWindowsやMacOS、特定のバージョン以降のAdobe Readerなど、表示・編集・加工する環境が比較的限られ、表示や印刷の見栄えに厳密さが求められない文書には有効です。PDFのファイルサイズは最も小さくなります。

PANOSEコード

CID-Keyedフォントの場合、表示環境にインストールされた、書体の近いフォントを選ぶために、PANOSE-1(パノーズ)という10桁のコードを使います。PDFでは、さらにクラスID・サブクラスIDというコードが先頭に付けられるため、12桁となります。

クラスID・サブクラスIDと、PANOSE-1コードはTrueTypeまたはOpenTypeフォントから、ツールを使って取得することができます。Microsoft社が配布しているFontTools.exeに含まれるttfdump.exeというツールを利用すると便利です（配布終了）。

以下はmsgothic.ttcからフォントの情報を、out.txtというテキストファイルに書き出すコマンドです。-c1は.ttcファイル(複数のTrueTypeフォントを含むファイル)の1番目のフォント情報を取得するためのオプションで、2番目のフォント情報を取得する場合は-c2のように指定してください。.ttfまたは.otfファイルではこのオプションは不要です。

例 2.35 ttfdump.exeによるフォント情報の取得

```
C:\TTFDump>ttfdump.exe "C:\Windows\Fonts\msgothic.ttc" -c1 >
out.txt
```

以下の部分(OS/2テーブル)のsFamilyClassがクラスID・サブクラスIDで、PANOSEが10桁のPANOSE-1コードです。この場合、フォント設定ファイル中ではPANOSEコードを 8 1 2 11 6 9 7 2 5 8 2 4 と指定してください。

例 2.36 抽出されたフォント情報の例

```
... 略
'OS/2' Table - OS/2 and Windows Metrics
---------------------------------------
Size = 96 bytes (expecting 96 bytes)
  'OS/2' version:           3
... 略 ...
  yStrikeoutSize:           13
  yStrikeoutPosition:       66
  sFamilyClass:             8    subclass = 1
  PANOSE:                   2 11  6  9  7  2  5  8  2  4
  Unicode Range 1( Bits 0 - 31 ): E00002FF
  Unicode Range 2( Bits 32- 63 ): 6AC7FDFB
略 ...
```

参考情報

https://monotype.github.io/panose/pan1.htm
PANOSE-1コードの仕様です。

https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6OS2.html
TrueType フォントのOS/2テーブルの仕様です。

http://download.microsoft.com/download/f/f/a/ffae9ec6-3bf6-488a-843d-b96d552fd815/FontTools.exe
Microsoft社のFontToolsのダウンロードURLです（配布終了）。ここでダウンロードしたFontTools.exe を実行すると、展開されたファイルの中にttfdump.exeがあります。

フォント幅情報ファイル

文字列をレイアウトするとき、各文字の幅や基底線の情報が必要です。CID-Keyedフォントは、フォントそのものは含まないフォント幅情報ファイルをもとにレイアウトします。フォント幅情報ファイルはfonts/warraysディレクトリに既定のものが用意されています。

出荷時のfonts.xmlの設定では、文書中がMincho, Gothicというフォント名で、明朝体とゴシック体のフォントが使われるようになっています。それぞれserif, sans-serifというCSS総称フォント名にも結び付けられており、またMinchoはデフォルトのフォントです。つまり、CID-Keyedフォントが利用する設定では、対応するフォントがない場合は全てMinchoとなります。これらのフォントは、実際には多くの日本語表示環境に入っていると考えられる、等幅の明朝体、ゴシック体フォントが使用されるようにPANOSEが設定されています。

Copper PDF 2.1.10 以降では、MS明朝、MSゴシック、MS P明朝、MS Pゴシック、MS UI Gothicの幅情報が用意されています。出荷時のfonts.xmlの設定では、それぞれの名前で文書中から使用できるようになっています。これらのフォントを使うと、日本語Windows環境を対象とした軽量なPDFを生成することができます。ただし、MS系フォントが入っていない環境では、隣り合う文字が重なったり、離れすぎてしまったりという現象が発生します。

手持ちのフォントから幅情報を抽出し、新たにフォント幅情報ファイルを作成する場合は、付属のツールを使ってください。以下のように、java コマンドで直接 jp.cssj.sakae.pdf.tools.WArrayTool を実行してください。クラスパスとクラス名に続く引数はそれぞれ、cmapファイル、Javaエンコーディング名、フォントファイルです。複数のフォントを含むファイル(.ttc)の場合、さらにフォントの番号を続けることができます。

例 2.37 幅情報の抽出

```
java -cp lib/*.jar \
    jp.cssj.sakae.pdf.tools.WArrayTool \
    conf/profiles/fonts/cmaps/UniJIS-UTF16-H \
    UTF-16BE \
    /usr/share/fonts/truetype/kochi/kochi-mincho.ttf \
    > conf/profiles/fonts/warrays/kochi-mincho.txt
```

2.4.7 フォントファイルの種類

埋め込みフォント、CID Identityフォントを使用する場合、あるいは後述するようにCID-Keyedフォントのためにフォントの幅情報を抽出するためには、Copper PDFが動作する環境にフォントファイルがインストールされている必要があります。

Copper PDFがサポートするフォントファイルの種類は以下の通りです。

TrueType "OpenType(TrueType flavor)"
: 一般的にTrueTypeと呼ばれるフォントファイルです。.ttfまたは.ttc(複数のフォントを含むもの)という拡張子のファイルとして配布されています。

OpenType CFF/Type2
: 一般的に単にOpenTypeと呼ばれる、CFF/Type2形式のOpenTypeフォントファイルです。.otfという拡張子のファイルとして配布されています。Copper PDF 2.0.3からサポートされました。

また、Copper PDFは上記のフォントファイル以外に、Java実行環境によりサポートされるフォントファイルを使用することができます。SunのJava実行環境(1.5.0以降)はTrueTypeに加えてType1フォント(.pfa, .pfb)、F3フォント(.f3b)をサポートしています。

2.4.8 フォント設定ファイル

PDF出力に使用するフォントの情報はフォント設定ファイル(fonts.xml) に記述してください。フォント設定ファイルはXML形式で、ルート要素はfontsです。フォント設定ファイルには各種ファイルのファイルパスの情報も含まれており、ファイルパスはフォント設定ファイルからの相対パスとなります。

fonts要素には次の要素が含まれています。

コアフォントのエンコーディング(encodings要素)

コアフォントのエンコーディング情報のファイルを設定します。通常は編集する必要はありません。

letterフォント(SynbolとZapfDingbats以外のフォント)を使用する場合、使用できる文字セットにはStandardEncoding、MacRomanEncoding、WinAnsiEncodingの3種類があります。Copper PDFの出荷時にはWinAnsiEncodingが設定されています。core-fonts要素のencoding属性 の指定を変更することで切り替えることができます。

encodingsに含まれる要素

表 2.9 encoding要素

属性	必須	説明
src	✔	グリフコードとグリフ名の対応を記述したファイルです。

cmapファイル(cmaps要素)

CID-Keyedフォントのエンコーディング情報のファイルを設定します。通常は編集する必要はありません。

cmapsに含まれる要素

表 2.10 cmap要素

属性	必須	説明
src	✔	Adobe Japan 1-4コードと文字コードの対応を記述したファイルです。
java-encoding	✔	文字コードのJavaエンコーディング名です。

コアフォント(core-fonts要素)

コアフォントの設定です。通常は編集する必要はありませんが、使用するエンコーディングを変更したり、ドキュメントから参照する際の別名を追加することができます。

表 2.11 core-fonts要素

属性	必須	説明
unicode-src	✔	ユニコードと文字名の対応を記述したファイルです。
encoding	✔	エンコーディング名です。encodingsで設定されたものの中から選択できます。

core-fontsに含まれる要素

core-fonts要素にはletter-font, symbol-fontのいずれかを含むことができます。

letter-font

letter-font要素は通常の欧文フォントの設定です。

表 2.12 letter-font要素

属性	必須	説明
src	✔	AFMファイルです。
encoding		core-fontsのencoding属性を上書きします。

symbol-font

symbol-font要素は記号フォント(SymbolまたはZapfDingbats)の設定です。

表 2.13 symbol-font要素

属性	必須	説明
src	✔	AFMファイルです。
encoding-src	✔	ユニコードとグリフコードの対応を記述したファイルです。

letter-fontおよびsymbol-fontに含まれる要素

letter-fontおよびsymbol-font内にalias要素を追加することで、ドキュメント中の`font-family`[css]で参照することができる別名が追加されます。

表 2.14 alias要素

属性	必須	説明
name	✔	フォントの別名です。

letter-fontおよびsymbol-font内のinclude, exclude要素により、フォントが使用される文字範囲を指定できます*[2.0.3]*。includeにより、フォントが使用される文字範囲を明示することができ、excludeにより除外する文字範囲を明示することができます。include, excludeの記述がない場合は、利用可能な全ての文字でフォントが利用されます。

表 2.15 include要素

属性	必須	説明
unicode-range	✔	カンマで区切ったユニコード範囲(詳細は後述)。

表 2.16 exclude要素

属性	必須	説明
unicode-range	✔	カンマで区切ったユニコード範囲(詳細は後述)。

unicode-rangeはU+に続く16進数によるユニコードとハイフンを用います。例えば、日本語のかなを含める文字範囲の指定は次の通りです。

例 2.38 ハイフンによる文字範囲

```
<include unicode-range="U+3030-30FF" />
```

また下位の桁をワイルドカードに置き換えることもできます。以下の記述は U+1F00-1FFFと書くのと等価です。

例 2.39 ワイルドカードによる文字範囲

```
<include unicode-range="U+1F??" />
```

複数の文字範囲はカンマで区切ってください。

例 2.40 複数の文字範囲

```
<include unicode-range="U+3030-30FF,U+1F??" />
```

CIDフォント(cid-fonts要素)

cid-fontsに含まれる要素

cid-fonts要素にはcid-keyed-font, font-file, font-dir, system-font, all-system-fontsのいずれかを含むことができます。

cid-keyed-font

cid-keyed-font要素はフォントファイルを使用する代わりに、フォントの幅情報を記述したファイルを利用してCID-Keyedフォントを定義するものです。

表 2.17 cid-keyed-font要素

属性	必須	説明
name	✔	フォント名です。
italic		フォントを斜体にする場合はtrue、そうでない場合はfalseを設定してください。
weight		フォントの太さです。100から900まで100刻みの値で設定してください。400が普通の太さです。
panose		PDFのFontDescripterのPanoseフィールドに対応する値です。クラスID、サブクラスID、10桁のPANOSE-1コードの順でスペース区切りで記述した12の数字から構成されます。
cmap	✔	横書きのCMap名です。

属性	必須	説明
vcmap		縦書きのCMap名です。
warray	✔	フォント幅情報ファイルです。

font-file

font-file要素はフォントファイルを直接指定します。

表 2.18 font-file要素

属性	必須	説明
name		フォント名フォントデータから取得されますが、ここで上書きすることもできます。
src	✔	フォントファイルです。
index		複数のフォントを含むTTCファイルの中で、使用するフォントの番号です。省略した場合は0です。TTCファイルでない場合は無視されます。
types	✔	フォントのタイプをスペース区切りで記述します。値は次のいずれかです。 **embedded** 埋め込みフォント **cid-identity** CID Identityフォント **cid-keyed** CID-Keyedフォント
italic		フォントが斜体かどうかはフォントデータから取得されますが、ここで上書きすることもできます。斜体にする場合はtrue、そうでない場合はfalseを設定してください。
weight		フォントの太さはフォントデータから取得されますが、ここで上書きすることもできます。100から900まで100刻みの値で設定してください。
cmap	✔ (type="cid-keyed"の場合)	横書きのCMap名です。
vcmap		縦書きのCMap名です。

font-dir

font-dir要素は指定したディレクトリに存在するフォントファイルを直接まとめて読み込みます。

表 2.19 font-dir要素

属性	必須	説明
dir	✔	フォントファイルが格納されるディレクトリです。
types	✔	フォントのタイプをスペース区切りで記述します。値は次のいずれかです。 **embedded** 埋め込みフォント **cid-identity** CID Identityフォント

system-font

system-fontはJava実行環境を利用してフォントを読み込みます。OSやウィンドウシステムにインストールされたフォントを名前で指定できますが、縦書きなどフォントの一部の機能に制約があります。

表 2.20 system-font要素

属性	必須	説明
name		フォント名です。フォントデータから取得されますが、ここで上書きすることもできます。
src	✔ srcまたはfileが必要	システムにインストールされたフォント名です。
file *[3.0.0]*	✔ srcまたはfileが必要	フォントファイルです。font-file要素による読み込みでは、Copper PDF独自のプログラムで読み込みますが、こちらではjava.awt.Fontを使用します。通常はfont-file要素を使ってください。

属性	必須	説明
types	✔	フォントのタイプをスペース区切りで記述します。値は次のいずれかです。 **embedded** 埋め込みフォント **cid-identity** CID Identityフォント **cid-keyed** CID-Keyedフォント
italic		フォントが斜体かどうかはフォントデータから取得されますが、ここで上書きすることもできます。斜体にする場合はtrue、そうでない場合はfalseを設定してください。
weight		フォントの太さはフォントデータから取得されますが、ここで上書きすることもできます。100から900まで100刻みの値で設定してください。400が普通の太さです。
cmap	✔(type="cid-keyed"の場合)	横書きのCMap名です。
vcmap		縦書きのCMap名です。

all-system-fonts

all-system-fontsはJava実行環境が利用可能なフォントを全て読み込みます。

表 2.21 all-system-fonts要素

属性	必須	説明
dir *[3.0.0]*		フォントファイルが格納されるディレクトリです。font-dir要素による読み込みでは、Copper PDF独自のプログラムで読み込みますが、こちらではjava.awt.Font を使用します。通常はfont-dir要素を使ってください。
types	✔	フォントのタイプをスペース区切りで記述します。値は次のいずれかです。 **embedded** 埋め込みフォント **cid-identity** CID Identityフォント

cid-keyed-font, font-file, system-fontに含まれる要素

フォントの名前はフォントデータから取得されますが、cid-keyed-font, font-file, system-fontにalias要素を追加することにより、さらにフォントの別名を追加できます。記述方法はcore-font, symbol-fontのalias と同じです。

cid-keyed-font, font-file, system-fontにinclude, exclude要素を追加することにより、有効な文字範囲を指定することができます。記述方法はcore-font, symbol-fontのinclude, exclude と同じです。

一般フォントファミリ(generic-fonts要素)

`font-family` [css] で指定できるserif, sans-serif, monospace, fantasy, cursiveという 5種類の一般フォント・ファミリに対応するフォントを指定するものです。

generic-fontsに含まれる要素

generic-fontsにはCSSの一般フォントファミリ名に対応する5つの名前の要素、serif, sans-serif, monospace, fantasy, cursiveが含まれます。

`font-family` [css]等で指定するフォント名には、実際のフォント名以外に5種類の一般フォントファミリ名を指定することができます。generic-fontsは、この一般フォントファミリ名と実際のフォントとの対応付けをするものです。

表 2.22 serif要素

属性	必須	説明
font-family	✔	カンマで区切ったフォント名を優先順位の高いものから順に記述します。

表 2.23 sans-serif要素

属性	必須	説明
font-family	✔	カンマで区切ったフォント名を優先順位の高いものから順に記述します。

表 2.24 monospace要素

属性	必須	説明
font-family	✔	カンマで区切ったフォント名を優先順位の高いものから順に記述します。

表 2.25 fantasy要素

属性	必須	説明
font-family	✔	カンマで区切ったフォント名を優先順位の高いものから順に記述します。

表 2.26 cursive要素

属性	必須	説明
font-family	✔	カンマで区切ったフォント名を優先順位の高いものから順に記述します。

2.4.9 フォント設定ファイルの設定例

デフォルトのフォントの変更

output.pdf.fonts.policy[io]にembeddedを指定するか、CSSで{-cssj-font-policy: embedded;}をしていると 埋め込みフォントが使用されますが、出荷時のフォント設定ファイルでは、大抵の場合は全ての文字が 🈸 のような組み文字になってしまいます。これは、デフォルトのフォントが一般フォントファミリのserifになっており、serifはCID-Keyedフォントにした対応付けてないためです。

output.default-font-family[io]でデフォルトのフォントを変更するか、あるいはフォント設定ファイルを修正して、一般フォントファミリを埋め込み可能なフォントに対応させます。

例えばtruetypeディレクトリにIPAexフォント (https://ipafont.ipa.go.jp/) を配置した場合は、generic-fontsの部分を以下のように設定することで、フォントの埋め込みに、デフォルトでIPAフォントが使われるようになります。

例 2.41 例のタイトル

```
<generic-fonts>
    <serif font-family="Mincho,IPAex明朝,UniKS-Myungjo,UniCNS-
Ming,UniGB-Song" />
    <sans-serif font-family="Gothic,IPAexゴシック,UniKS-
Gothic,UniCNS-Ming,UniGB-Heiti" />
    <monospace font-family="Gothic,IPAexゴシック,UniKS-
Gothic,UniCNS-Ming,UniGB-Heiti" />
    <fantasy font-family="Comic-Sans-MS,Gothic,IPAexゴシック,UniKS-
Gothic,UniCNS-Ming,UniGB-Heiti" />
    <cursive font-family="Comic-Sans-MS,Mincho,IPAex明朝,UniKS-
Myungjo,UniCNS-Ming,UniGB-Song" />
</generic-fonts>
```

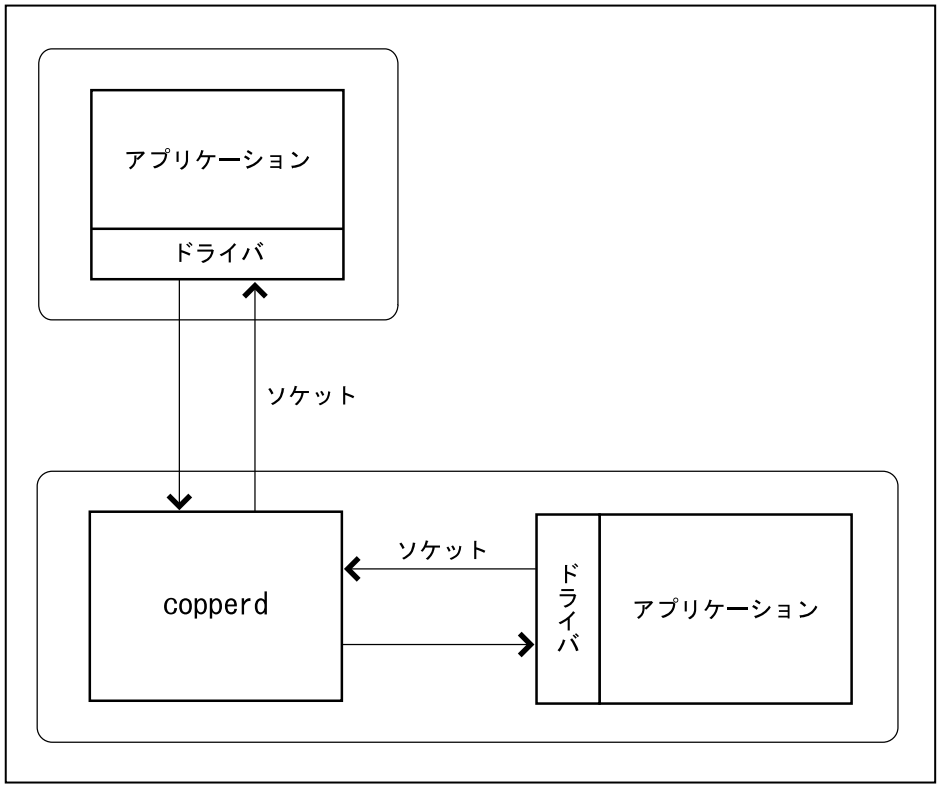
3.開発者ガイド

3.1 プログラムインターフェースの概要

3.1.1 アプリケーションからCopper PDFの機能を使うには

OSのバックグラウンドで動作しているCopper PDFサーバー(copperd)を、アプリケーションから呼び出すためには、ソケット通信を使います。通信用のドライバは、各言語ごとに配布しています。これは、一般的なデータベースサーバーと同様のしくみです。また、Copper PDF 2.1以降ではHTTPによる接続をサポートしており、HTTPクライアントプログラムやライブラリを使用して接続することができます。いずれにしても、ネットワーク越しの接続が可能です。

図 3.13 アプリケーションからcopperdにアクセスする



通信方式には次の２種類があります。

CTIP 2.0 (73ページ)*[2.1.0]*
: 高速で信頼性の高い通信方式です。

HTTP/REST (144ページ)*[2.1.0]*
: HTTPベースの通信方式です。普通のHTTPクライアントを使うことができます。CTIP 2.0と同等の機能を持ちます。

CTIP 2.0 によるアクセスのために、現在のところJava, Perl, PHP, Python, .NET, Node.jsのドライバと、Antタスクが用意されています。これらのプログラミング言語では、プログラミングインターフェースから高速にアクセスできます。Java版のドライバとAntタスクはHTTP/RESTに対応しており、全く同じプログラムで両方のプロトコルを切り替えることができます。

HTTP/RESTでは、ドライバは必要とせず、各プログラミング言語からHTTPクライアントを利用してアクセスすることができます。CTIP 2.0 と比較して、おおむね70%程度にパフォーマンスが低下しますが、ほとんどの場合は問題となることはありません。

3.1.2 通信の手順

ドライバとcopperdとの通信は、以下の手順が基本となります。

1. copperdへの接続・認証
2. メッセージハンドラの設定
3. プログレスリスナの設定
4. 変換結果の出力先の設定
5. 入出力プロパティの設定
6. リソースの送信・アクセス許可
7. ソースリゾルバの設定*[2.1.0]*
8. 設定のリセット*[2.1.0]*
9. ドキュメント本体の送信または変換対象のドキュメントの指定
10. 変換処理の中断*[2.1.0]*
11. 通信の終了

上記のうち、2〜8の手順は省略されるか、順序が入れ替わっても構いません。10は、ドキュメントの変換中に、処理を中止したいときに実行します。CTIP 2.0では、通信の終了をせずに、2に戻って手順を再実行することができます。以下、各手順について順を追って説明します。

3.1.3 copperdへの接続・認証

copperdにアクセスするためには、copperdのホスト名、ポート番号を知る必要があります。これらの設定はcopperdの設定(copperd.properties)によります。ローカルマシンで初期設定のままCopper PDFを動かしている場合、ホスト名はlocalhost(あるいは127.0.0.1(IPv4)または::1(IPv6))、CTIPのポート番号は8099(CTIP 2.0)、HTTP/RESTのポート番号は8097です。

簡単なセキュリティ機能として、ユーザーIDとパスワードにより認証があります。サーバーの初期設定の状態ではユーザーIDは"user"、パスワードは"kappa"ですが、サーバー側でcopperdコマンドにより変更することができます (33ページ)。

copprdはクライアントのIPアドレスによりアクセスを制限します。**初期設定の状態ではローカルマシン(127.0.0.1(IPv4)または::1(IPv6))からのアクセスだけが許可されています。**これはサーバー側のアクセス制御の設定(access.txt)を編集することで (42ページ)起動中に変更することができます。

3.1.4 メッセージハンドラの設定

メッセージハンドラは、変換処理の過程で出力された警告やエラー、処理情報を受け取るためのインターフェースです。

3.1.5 プログレスリスナの設定

プログレスリスナはサーバー側でのデータの処理状況をプログラムが知るためのインターフェースです。クライアント側で処理状況をユーザーに通知させ、待ち時間の目安にするためのものであるため、データの処理状況は必ずしも正確なものではありません。あるいは、一切処理状況が通知されないこともあります。

3.1.6 出力先の設定

変換結果はストリーム、ファイル等に出力することができます。出力先の指定方法は、プログラミング言語によります。Copper PDFはウェブ上での利用を重視しているため、クライアントのブラウザに送る方法は必ず用意されています。

3.1.7 変換結果の出力先の設定

ドキュメントを変換した結果得られるPDF、画像等のデータはクライアント側で受信してドライバにより構築されます。出力結果はファイルに保存するか、あるいはウェブアプリケーションであればそのままユーザーに送り出すことができます。

CTIP 2.0では、複数の結果を受信することができます。例えば、複数ページの文書を複数の画像ファイルとしてクライアント側で保存することができます。

3.1.8 入出力プロパティの設定

ドキュメントの変換方法の詳細は入出力プロパティによって細かく指定することができます。利用可能な入出力プロパティのリストは入出力プロパティ一覧 (268ページ)を参照してください。

3.1.9 URIの解決と関連ファイル（リソース）の取得

ドキュメントからは、様々な形で関連するCSSや画像ファイル等のリソースが参照されます。リソースはURIによって参照されます。URIには絶対URIと相対URIがありますが、Copper PDFは必ずドキュメントを実際の、あるいは仮想的なURIに結びつけ、相対URIはドキュメントのURIを基準に解決されます。例えば、ドキュメントのURIが

*http://copper-pdf.com/docs/document.html*であった場合、ドキュメント中に`<img src="../images/photo.jpeg">`という記述があれば、*http://copper-pdf.com/images/photo.jpeg*に存在する画像が使われます。

ドキュメントのURIを相対URIとすることができます。もっとも簡単なURIは "."（カレントディレクトリ）で、このとき相対URIで参照されたリソースの解決後のURIは相対URIのままとなります。

Copper PDFが、あるURIで参照されるリソースにアクセスする場合、3通りの方法があります。

1つは、サーバーから直接アクセスする方法です。file: で始まるURIであれば、サーバーマシン上のファイルを取得しようとします。http: で始まるURIであれば、HTTPでネットワークから取得しようとします。Copper PDFは、最初はこれらのアクセスを禁止するように設定されているため、プログラムでアクセス許可する必要があります。

図 3.14 copperdがリソースにアクセスする場合(サーバーマシン上のファイル)

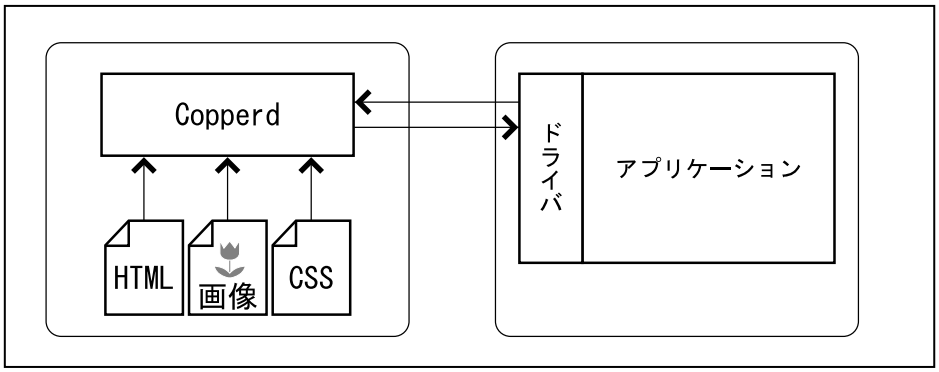
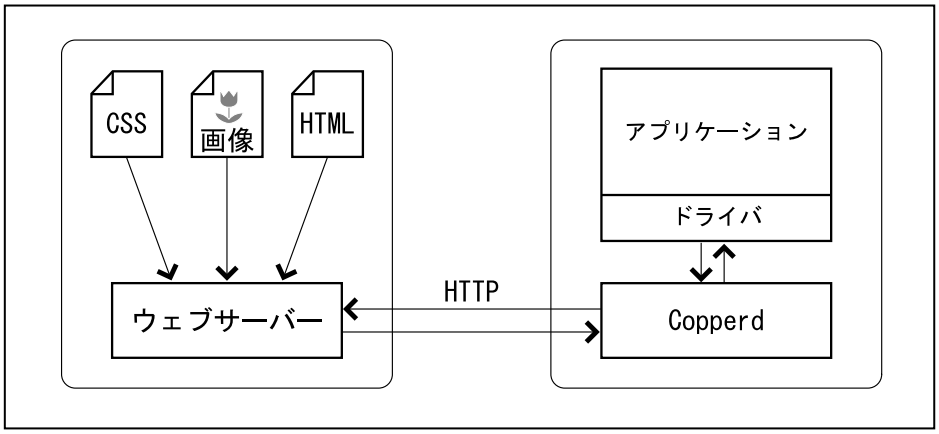
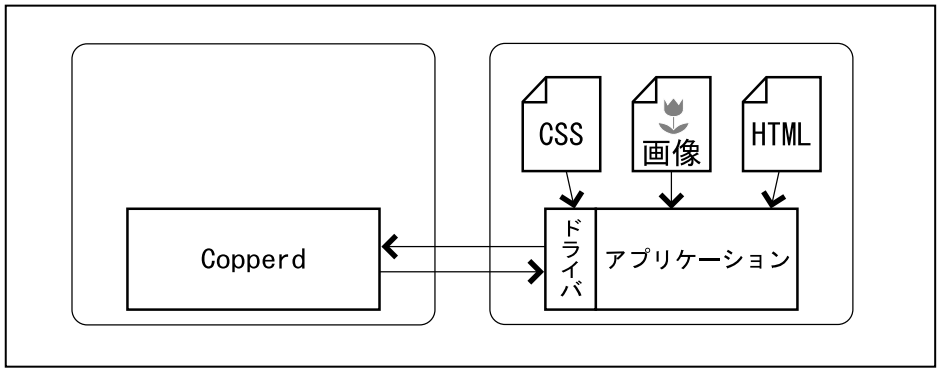


図 3.15 copperdがリソースにアクセスする場合(ネットワークからの取得)



2つめは、ドライバによって、事前にクライアントからサーバーにリソースを送る方法です。リソースを仮想的なURIに結びつけて事前に送っておくと、サーバーはそのURIのリソースを取得する際に、実際のURIにアクセスするのではなく、事前に送ったデータを使用します。

図 3.16 クライアント側からcopperdにリソースを送る場合



3つめはソースリゾルバ*[2.1.0]*によって、サーバーが必要としたリソースをクライアントが送信可能であるかを問い合わせ、送信可能なリソースを都度サーバーに送る方法です。

サーバーは最初にクライアントから送られたリソースがあるかどうかをチェックし、なければ実際にURIアクセスすることを試みます。また、既にクライアントから送られたリソースがある場合は、クライアントにリソースの送信を要求しません。そのため優先順位は2番目の方法、3番目の方法、1番目の方法の順ということになります。

リソースにサーバーからアクセスする場合

必要とするリソースがドライバから送られていなかった場合、copperdは実際にそのURIで表される場所にアクセスしてデータを取得しようと試みます。

Copper PDFはHTTPクライアントを持っており、HTTP(https://で始まるURI)、SSL(https://で始まるURI)によりアクセス可能なデータを取得することができます。またCopper PDFが動作しているローカルマシン上のファイル(file://で始まる、ブラウザでローカルマシン上のファイルを開く場合のURI)へアクセスすることができます。データスキームURI(data:で始まる、URI中にデータを含むURI)もサポートしています。その他のデータへはjava.net.URL を利用してアクセスするため、Java実行環境がサポートするコンテンツハンドラに依存します。

この方法の利点は、変換対象の文書から参照されているリソースを、copperdその都度自動的に集めてくることです。また、リソースがサーバーのディスク上にある場合は、事前にドライバがリソースを送る(この作業は実質的には、copperdがアクセスできる場所にファイルをコピーする作業です)手間が省けるため、パフォーマンス上有利です。

欠点として、セキュリティの問題が挙げられます。変換対象となる文書を誰でも編集できる場合、そこにサーバー上のファイル名を指定することで、サーバー上のファイルが盗まれてしまう可能性があります。また、http://で始まるURIを使うことで、他のサーバーへの不正なアクセスのための踏み台にされる恐れがあります。そのため、ネットワークの構成を含めて十分に注意して運用することが必要となります。

無制限にサーバー上のリソースにアクセスされるのを防止するため、copperdがアクセスするリソースを制限する簡単なセキュリティ機能が用意されています。サーバー上のリソースを利用するには、適宜アクセス許可を行う必要があります。



デフォルトの状態では、サーバーから全てのリソースへのアクセスが禁止されています copperdが文書に関連するCSSや画像等にアクセスするためにはアプリケーションで明示的に許可する必要があります。

URIパターン

リソースへのアクセス許可・制限は、URIパターンによって指定します。本文の変換を始める前に、クライアントは、利用できるURIと除外するURIのパターンを指定します。

URIパターンにはワイルドカードを使うことができます。"*"というワイルドカードは、'/'(スラッシュ)以外の任意の文字列を表します。"**"というワイルドカードは、それに加えて'/'も含めることを表します。ワイルドカードの例は以下の通りです。

- *http://www.company.com/** というパターンは、*http://www.company.com/* 直下の全てのリソースを現します。
- *http://www.company.com/image/*** は、*http://www.company.com/image/* 以下の全てのリソースを表します。
- *http://www.company.com/**.css* は、*http://www.company.com/* 以下の全てのCSSファイルを表します。

アクセスの制御は指定された順に行われます。

例えば、最初に次のパターンへのアクセスを禁止したとします。

- *http://www.company.com/secret*

次に以下のパターンへのアクセスを許可したとします。

- *http://www.company.com/***

このとき、*http://www.company.com/style.css* へのアクセスは許可されますが、*http://www.company.com/secret/image.jpeg* へのアクセスは禁止されます。

逆に、最初に以下のパターンへのアクセスを許可したとします。

- *http://www.company.com/***

この場合は、後の指定に関係なく *http://www.company.com/* 以下へのアクセスが全て許可されてしまいます。

http: または https: で始まるURIでは、%エスケープした文字は、デコードされたものとして比較されます。また、パターンで * にマッチするようにするには、代わりに %2A を記述してください。例えば *http://www.company.com/%61bc %2A/** というパターンは *http://www.company.com/a%62c*/def* というURIにも *http://www.company.com/ab%63%2A/def* というURIにもマッチします。*[2.1.7]*

ローカルファイルのパターンは*file:///var/data/***のような形になります。絶対ファイルパスは*file:///*（'/'は３つ）で始めてください。Windowsの場合は*file:///C:/User/John/***のようにします。Windowsでも大文字・小文字は区別されます。ドライブレター（'C:'など）は必ず大文字にしてください。

リソースを事前にサーバーに送る場合

ドライバがリソースをcopperdに送る際には、リソース本体のデータとリソースの仮想的なURIに加え、リソースのMIME型およびキャラクタ・エンコーディングを送ることができます。MIME型とキャラクタ・エンコーディングは必須ではなく、省略された場合は拡張子やファイルの内容をもとにサーバー側で自動的に判断されます。また、画像などのバイナリデータではキャラクタ・エンコーディングは無意味です。

仮想的なURIは、*file:///var/data/image.gif*のような絶対URIや、*data/style.css*のような相対URIです。copperdはこれらのURIで表されるリソースが必要になった場合、そのURIで表される実際の場所にアクセスすることはせず。クライアント側から送られたデータを使います。おなじURIで2度リソースを送った場合は、前のリソースは後に送られたリソースで上書きされます

この方法は、事前に画像などのデータをサーバーに送り出す手間がかかる分、パフォーマンス上不利になります。また、アプリケーションは、本文から参照されているリソースを事前に把握している必要があります。

一方で、変換対象やリソースを1つ1つアプリケーションで指定するため、予期しなかったファイルにアクセスされてしまうといった、セキュリティ上の危険は少なくなります。

ソースリゾルバを使う場合

ソースリゾルバは、サーバー側で必要とされたリソースのURIをクライアントに通知し、可能であればクライアントからデータを送信するためのインターフェースです。ドキュメントから参照されるリソースは、事前にクライアントから送ることができますのが、そのためには、ドキュメントからどのリソースが参照されているのかを、クライアントが知っていることが前提となります。しかし、HTMLからCSSが読み込まれ、さらにCSSから別のCSSや画像が参照されるといった複雑な状況では、アプリケーションで事前に必要なリソースを解析することは困難です。CTIP 2.0では、サーバー側でまず処理を開始し、必要になったリソースをその都度クライアントに要求し、ドライバは要求されたリソースをサーバーに送るか、あるいはリソースの不存在を通知することができます。

3.1.10 設定のリセット

CTIP 2.0では、メッセージハンドラ、プログレスリスナ、入出力プロパティの設定と、サーバー側で受信済みのリソースを全てリセットすることができます。リセットにより、全ての状態は接続直後に戻ります。

3.1.11 ドキュメント本体の送信または変換対象のドキュメントの指定

最後にドキュメント本体をCopper PDFが変換するために必要な情報を送ります。リソースの場合と同様、データをサーバーに送る方法と、サーバー側からデータを取得する方法があります。

ドキュメント本体をサーバーに送る

変換対象のドキュメント本体をクライアントからサーバー側に送る場合は、必ずドキュメントを仮想的なURIと結びつける必要があります。また、必須ではありませんが、ドキュメントのMIME型とキャラクタ・エンコーディング、予測されるドキュメントサイズを明示することができます。

サーバー側でのドキュメントの変換処理は、本体の送信と並行して行われます。

ドキュメント本体にサーバーからアクセスする場合

サーバーに変換対処のドキュメントのURIを送ることにより、サーバー側で変換対象の文書を取得することができます。ドキュメント中の相対パスは、ドキュメントのURIを基点に解決します。ドキュメントのURIに対しては、URIパターンによるアクセス許可とは無関係にアクセス可能です。

ドキュメントのURIは、事前にサーバーに送ったリソースでも構いません。ことのき、事前に送ったリソースがドキュメント本体となります。

リソースに対するアクセス禁止設定は、ドキュメント本体のURIには適用されません。

3.1.12 複数の結果の結合 *[3.0.0]*

通常は、ドキュメント1つの変換処理に対して結果が出力されますが、複数の結果を結合するモードに移行することにより、複数のドキュメントの変換結果を結合して1つの処理結果とすることができます。

このモードでは、各々の変換処理で結果を出力することはせず、一連の変換処理の終了をクライアントが明示した時点で、結合された結果が出力されます。

`processing.pass-count`[io]により 2パス以上の変換処理 (225ページ)を行う場合、それぞれのドキュメントが複数のパスで変換され、その結果が結合されます。複数のドキュメントを通して2パス以上の変換処理を行う場合は `processing.middle-pass`(ページ) を使ってください。

3.1.13 変換処理の中断

CTIP 2.0では、変換処理の最中に処理を中断することができます。クライアントから処理の中断を要求されてから、実際に処理が中断されるまでには、若干の遅れが生じます。

処理を中断する際には、なるべくきりのよいところまで処理を継続して、途中のページまでが含まれたPDFのようなファイルが生成されるようにするか、ファイルが破壊されても強制的に停止する２つのモードを選ぶことができますが、必ず要求どおりに処理が終わる保障はありません。

本文を送っている途中で中断した場合も、本文のストリームは閉じてください(残りを書く必要はありません)。サーバーは中断のあとに届いた本文を読み捨て、クライアントがストリームを閉じたところで中断を知らせます。中断のあとも、同じセッションはリセットして続けて使えます。

3.2 CTIP 2.0 インターフェースの概要

3.2.1 接続情報

まず、サーバー側が各接続方式に対応するように設定されていることを確認してください。ドライバ側では次の形式のURIで接続情報を設定します。

`ctip://ホスト名:ポート/`

例えば、ローカルマシンの8099版ポートでCopper PDFサーバーが動作している場合は、`ctip://localhost:8099/` というアドレスをドライバに渡します。

TLSによる暗号化通信*[3.0.0]*を用いる場合は、`ctips://localhost:8094/` のようにURIを設定してください。

Java用ドライバはHTTP/RESTによる接続にも対応しています。HTTPを使う場合は `"http://127.0.0.1:8097/"`、SSL(HTTPS)を使う場合は `"https://127.0.0.1:8096/"`、のようにURIを設定してください。

3.2.2 メッセージコード

処理状況や警告・エラーとして、2バイトのメッセージコード、メッセージに付随する値、人間が読める形式の文字列の3つの値がドライバに渡されます。メッセージコードは16進数で表記し、以下の通りクラス分けされます。

1XXX
: 処理情報。文書のタイトル、出力したページの番号等です。

2XXX
: 警告メッセージ。通常の運用でも発生する可能性のある軽微なエラーを示すもので、処理が続行されます。

3XXX
: エラーメッセージ。アプリケーション等の問題によるエラーで、処理が中断されます。

4XXX
: 深刻なエラー。サーバープログラムのバグやシステムの障害によるもので、処理が中断されます。

全てのメッセージコードの一覧は資料集 (283ページ)を参照してください。

3.2.3 サーバー情報

ドライバによりCopper PDFサーバーの環境に関する情報を得ることができます。情報の種類はURIで指定します。

http://www.cssj.jp/ns/ctip/version

Copper PDFのバージョン情報です。これは以下の形式のXMLです。

例 3.1 バージョン情報

```
<?xml version="1.0" encoding="UTF-8"?>
<version>
<long-version>Copper PDF バージョン番号/ビルド番号</long-version>
<name>Copper PDF</name>
<number>バージョン番号</number>
<build>ビルド番号</build>
<copyrights>著作権表示</copyrights>
<credits>
付属ライブラリの著作権教示
</credits>
</version>
```

http://www.cssj.jp/ns/ctip/output-types *[3.0.0]*

Copper PDFがサポートする出力形式です。これは以下の形式のXMLです。

例 3.2 出力形式情報

```
<?xml version="1.0" encoding="UTF-8"?>
<output-types>
<type name="PDF" mimeType="application/pdf" suffix="pdf"/>
<type name="PNG" mimeType="image/png" suffix="png"/>
<type name="JPEG" mimeType="image/jpeg" suffix="jpg"/>
<type name="WBMP" mimeType="image/vnd.wap.wbmp" suffix="wbmp"/>
<type name="BMP" mimeType="image/bmp" suffix="bmp"/>
<type name="SVG" mimeType="image/svg+xml" suffix="pdf"/>
...
以降
<type name="フォーマット名" mimeType="MIME型" suffix="拡張子"/>
の繰り返し
...
</output-types>
```

http://www.cssj.jp/ns/ctip/fonts *[3.0.4]*

利用可能なフォントの一覧です。これは以下の形式のXMLです。

例 3.3 フォント情報

```
<fonts>
<font name="Courier-Bold" weight="700" type="core" direction="ltr">
<alias name="Courier"/>
<alias name="Courier-New"/>
</font>
<font name="Courier-BoldOblique" italic="true" weight="700"
type="core" direction="ltr">
<alias name="Courier"/>
<alias name="Courier-New"/>
</font>
<font name="Courier-Oblique" italic="true" weight="400" type="core"
 direction="ltr">
<alias name="Courier"/>
<alias name="Courier-New"/>
</font>
...
以降
<font name="フォント名" (italic="true") weight="太さ" type="タイプ"
direction="方向">
<alias name="別名"/>
...
</font>
の繰り返し
...
</fonts>
```

italic="true" 属性は斜体のフォントだけに付きます。weightは100～900までの数値です。typeはcore, cid-keyed, cid-identity, embeddedのいずれかで、それぞれコア14フォント、CID-Keyedフォント、外部フォント、埋め込みフォントに対応します。directionはltr, tbのいずれかで、それぞれ横書き用フォント、縦書き用フォントに対応します。

3.2.4 CTIP 2.0 プロトコルの仕様

プロトコルの仕様書は、以下のアドレスで公開しています。
https://htmlpreview.github.io/?https://github.com/zamasoftnet/cti.java/blob/main/docs/ctip-v2.xhtml

3.3 Javaドライバ2

3.3.1 概要

Java用ドライバは、ストリーム（java.io.InputStream/java.io.OutputStream）からストリームへの変換ができることが特徴です。ユーティリティークラスを利用して、ファイルからストリーム、ストリームからファイル、ファイルからファイルなど、あらゆる入出力に対応できます。

また、サーブレット/JSPの出力をキャプチャして変換するためのクラスが用意されています。PDFのもととなるテンプレートをJSP, JSF, Velocity, Tapestryなど、ウェブ開発で広く使われているJavaベースのテンプレート言語によりデザインできます。

3.3.2 ドライバの準備

Java用ドライバはGitHubの https://github.com/zamasoftnet/cti.java で公開されています。JitPackを利用してMavenまたはGradleで取得することができます。最新バージョンはリリースページで確認してください。

Mavenの場合は `pom.xml` に以下を追加してください。

例 3.4 Mavenでのインストール

```
<repositories>
  <repository>
    <id>jitpack.io</id>
    <url>https://jitpack.io</url>
  </repository>
</repositories>
<dependencies>
  <dependency>
    <groupId>com.github.zamasoftnet</groupId>
    <artifactId>cti.java</artifactId>
    <version>vX.X.X</version>
  </dependency>
</dependencies>
```

Gradleの場合は `build.gradle` に以下を追加してください。

例 3.5 Gradleでのインストール

```
repositories {
    maven { url 'https://jitpack.io' }
}
dependencies {
    implementation 'com.github.zamasoftnet:cti.java:vX.X.X'
}
```

Mavenなどのビルドツールを使わない場合は、GitHubのリリースページからアーカイブをダウンロードすることもできます。

https://github.com/zamasoftnet/cti.java/releases

アーカイブには以下が含まれています。

cti-driver-x.x.x.jar
: ドライバ本体（CTIP、REST、CLIが利用可能）

cti-driver-min-x.x.x.jar
: 最小構成のドライバ（REST、CLIは利用不可）

apidoc
: API ドキュメント（Javadoc）

lib
: サンプルのコンパイルに必要なライブラリ

examples
: サンプルプログラム

ドライバの窓口となるクラスはjp.cssj.cti2.CTIDriverManagerです。例えばlocalhostの8099番ポートで起動しているcopperdに、ユーザーID"user"、パスワード"kappa"で接続するには、以下のようにします。

例 3.6 copperdへの接続

```
//ドライバクラスのインポート
import jp.cssj.cti2.CTIDriverManager;
import jp.cssj.cti2.CTISession;
...

CTISession session = CTIDriverManager.getSession
("ctip://127.0.0.1:8099/", "user",
        "kappa");

//各種操作
...
```

Java用ドライバはHTTP/RESTによる接続にも対応しています (73ページ)。

3.3.3 タイムアウトの設定

Java版ドライバ2.1.4以降から、一定時間通信がない状態で自動的に通信を切断するタイムアウトに対応しています。

以下のようにURLパラメータで、timeoutをミリ秒単位で指定できます。

```
ctip://127.0.0.1:8099/?timeout=10000
```

ctips, http等他のプロトコルではタイムアウトは無効です。

3.3.4 APIの概要

ここではAPIによるアクセスの概要で説明した各手順に対応する関数を列挙します。各関数の詳細は オンラインのAPIドキュメント（Javadoc）を参照してください。

サーバーへの接続・認証

- public static CTIDriver getDriver(URI uri)
- public static CTISession getSession(URI uri) throws IOException
- public static CTISession getSession(URI uri, Map props) throws IOException
- public static CTISession getSession(URI uri, String user, String password) throws IOException
- public CTISession getSession(URI uri, Map props) throws IOException, SecurityException

サーバー情報の取得

- public InputStream getServerInfo(java.net.URI uri) throws IOException

メッセージハンドラ・プログレスリスナの設定

- public void setMessageHandler(MessageHandler messageHandler) throws IOException
- public void setProgressListener(ProgressListener progressListener) throws IOException

出力先の設定

- public void setResults(Results results) throws IOException

プロパティの設定

- public void property(String name, String value) throws IOException

ソースリゾルバの設定

- public void setSourceResolver(SourceResolver resolver) throws IOException

リソースの送信

- public void resource(Source source) throws IOException
- public OutputStream resource(MetaSource metaSource) throws IOException

本体の送信・変換

- public void transcode(Source source) throws IOException, TranscoderException
- public OutputStream transcode(MetaSource metaSource) throws IOException
- public void transcode(URI uri) throws IOException, TranscoderException

複数の結果の結合

- public void setContinuous(boolean continuous) throws IOException
- public void join() throws IOException

処理の中断・リセット・通信の終了

- public void abort(byte mode) throws IOException
- public void reset() throws IOException
- public void close() throws IOException

3.3.5 サンプル

次の例は、ストリームに送ったHTMLをPDFに変換してファイルとして保存します。

例 3.7 ストリームに送ったHTMLをPDFに変換

```
package jp.cssj.cti2.examples;

import java.io.File;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.URI;

import jp.cssj.cti2.CTIDriverManager;
import jp.cssj.cti2.CTISession;
import jp.cssj.cti2.helpers.CTIMessageHelper;
import jp.cssj.cti2.helpers.CTISessionHelper;
import jp.cssj.resolver.helpers.MetaSourceImpl;

public class Example1 {
    /** 接続先。 */
    private static final URI SERVER_URI = URI.create
("ctip://127.0.0.1:8099/");

    /** ユーザー。 */
    private static final String USER = "user";

    /** パスワード。 */
    private static final String PASSWORD = "kappa";
```

```
    public static void main(String[] args) throws Exception {
        // 接続する
        CTISession session = CTIDriverManager.getSession
(SERVER_URI, USER,
                PASSWORD);
        try {
            // test.pdfに結果を出力する
            File file = new File("test.pdf");
            CTISessionHelper.setResultFile(session, file);

            // リソースの送信
            {
                PrintWriter out = new PrintWriter(new
OutputStreamWriter(
                        session.resource(new MetaSourceImpl
(URI.create("style.css"),
                                "text/html")), "UTF-8"));
                try {
                    // CSSを出力
                    out.println("p {color: Red;}");
                } finally {
                    out.close();
                }
            }

            // 出力先ストリームを取得
            {
                PrintWriter out = new PrintWriter(new
OutputStreamWriter(
                        session.transcode(new MetaSourceImpl
(URI.create("."),
                                "text/html")), "UTF-8"));
                try {
                    // 文書を出力
                    out.println("<html>");
                    out.println("<head>");
                    out.println("<meta http-equiv='Content-Type'
content='text/html; charset=UTF-8'>");
                    out.println("<link rel='StyleSheet'
type='text/css' href='style.css'>");
                    out.println("<title>サンプル</title>");
                    out.println("</head>");
                    out.println("<body>");
                    out.println("<p>Hello World!</p>");
                    out.println("</body>");
                    out.println("</html>");
```

```
                } finally {
                    out.close();
                }
            }
        } finally {
            // セッションを閉じる(忘れやすいので注意!)
            session.close();
        }
    }
}
```

次の例は、サーバー側からネットワーク上のウェブページアクセスしてPDFに変換します。

例 3.8 サーバー側からウェブページにアクセスしてPDFに変換

```
package jp.cssj.cti2.examples;

import java.io.File;
import java.net.URI;

import jp.cssj.cti2.CTIDriverManager;
import jp.cssj.cti2.CTISession;
import jp.cssj.cti2.helpers.CTIMessageHelper;
import jp.cssj.cti2.helpers.CTISessionHelper;

public class Example2 {
    /** 接続先。 */
    private static final URI SERVER_URI = URI.create
("ctip://127.0.0.1:8099/");

    /** ユーザー。 */
    private static final String USER = "user";

    /** パスワード。 */
    private static final String PASSWORD = "kappa";

    public static void main(String[] args) throws Exception {
        // 接続する
        try (CTISession session = CTIDriverManager.getSession
(SERVER_URI, USER,
                PASSWORD)) {
            // test.pdfに結果を出力する
            File file = new File("test.pdf");
            CTISessionHelper.setResultFile(session, file);

            // エラーメッセージを標準出力に表示する
            session.setMessageHandler(CTIMessageHelper
                    .createStreamMessageHandler(System.err));
```

```
            // ハイパーリンクとブックマークを作成する
            session.property("output.pdf.hyperlinks", "true");
            session.property("output.pdf.bookmarks", "true");

            // https://copper-pdf.com/以下にあるリソースへのアクセスを許
可する
            session.property("input.include", "https://copper-
pdf.com/**");

            // ウェブページを変換
            session.transcode(URI.create("https://copper-
pdf.com/"));
        }
    }
}
```

他のサンプルはドライバのexamples/srcに収められています。

3.3.6 プログラミングのポイント

CTISessionHelperの利用

結果の出力先、リソースの送信、ファイルの変換等のよく使われる操作が、jp.cssj.cti2.helpers.CTISessionHelper のstaticメソッドにまとめられています。

例えば、事前に関連するCSSを送信してHTMLファイルを変換する処理は、次のように簡単に書けます。

例 3.9 ファイルを変換する

```
...
CTIDriverManager.sendResourceFile(session, new File("test.css"),
"text/css", "UTF-8");
CTIDriverManager.transcodeFile(session, new File("test.html"),
"text/html", "UTF-8");
...
```

繰り返し処理

出力先を変え、transcodeメソッドを繰り返し呼び出すことで、同じセッションで何度もドキュメントを変換することができます。送信済みのリソース、設定済みのプロパティは同じセッションで維持されます。同じセッションのまま初期状態に戻すには reset を呼び出してください。

出力先(Results)の設定

出力先が単一のファイルやストリームの場合は、CTISessionHelperの setResultFile か、setResultStream を使ってください。これらのメソッドは内部的に jp.cssj.cti2.results.SingleResultクラスを使用しています。

jp.cssj.cti2.results.Resultsインターフェースは、複数の出力結果を受け取るためのインターフェースです。CTISessionのsetResultsメソッドに渡します。

複数の結果をファイルとして出力する場合は、jp.cssj.cti2.results.DirectoryResults を使用してください。このクラスは、指定したディレクトリに、1から開始する連番の前後に指定した文字列をくっつけたファイル名で結果を出力します。次の例では変換結果の各ページを、resultsディレクトリ内に"image[通し番号].jpeg"という名前で別々のJPEG画像として出力します。

例 3.10 ディレクトリに結果を出力する

```
...
session.property("output.type", "image/jpeg");
session.setResults(new DirectoryResults(new File("results"),
"image", ".jpeg"));
...
```

さらに複雑な処理が必要な場合は、Resultsインターフェースを実装するクラスを用意する必要があります。Resultsインターフェースは jp.cssj.rsr.RandomBuilder に依存しますが、RandomBuilder にはファイルとストリームにデータを出力する実装(jp.cssj.rsr.impl.FileRandomBuilder, jp.cssj.rsr.impl.StreamRandomBuilder) が用意されています。

サーバーから要求されたリソースの送信(SourceResolver)

CTISessionのsetSourceResolverで、ソースリゾルバ(jp.cssj.resolver.SourceResolver) を設定すると、サーバーから要求されたリソースを都度送信できるようになります。

jp.cssj.resolver.composite.CompositeSourceResolver の static メソッド、createGenericCompositeSourceResolver を呼び出すと、file:, http:, data:で始まるURIによのリソースを取得できるSourceResolverの実装が返されます。

CompositeSourceResolverをそのまま使用すると、クライアントのファイルシステム上のファイルをドキュメントから参照可能になってしまうため、注意してください。次の例のように jp.cssj.resolver.restricted.RestrictedSourceResolver を使用すると、アクセス制限をかけることができます。

例 3.11 アクセス制限をしてSourceResolverを使う

```
...
RestrictedSourceResolver resolver = new RestrictedSourceResolver(

CompositeSourceResolver.createGenericCompositeSourceResolver());
resolver.include("file:/home/miyabe/data/**");
session.setSourceResolver(resolver);
...
```

MetaSource

CTISessionの resource, transcode メソッド等では、データの仮想URI、MIME型、キャラクタ・エンコーディング、予測されるデータサイズを MetaSource インターフェースにより渡します。

通常は用意されている jp.cssj.resolver.helpers.MetaSourceImpl という実装を利用してください。

複数の結果の結合

複数の結果を結合したものを得るためには、setContinuous(true) を呼び出した後、transcodeを複数回呼び出し、最後に join を呼び出してください。

例 3.12 2つの結果の結合

```
...
session.setContinuous(true);
CTIDriverManager.sendResourceFile(session, new File("test.css"),
"text/css", "UTF-8");
CTIDriverManager.transcodeFile(session, new File("test.html"),
"text/html", "UTF-8");
session.transcode(URI.create("http://print.cssj.jp/"));
session.join();
...
```

abortによる中断

CTISessionのabort メソッドは文書の変換処理を中断しますが、transcodeメソッドは処理の間ブロックするため、別スレッドからabortを呼び出す必要があります。abortは引数によって、強制的に中断するモードと、きりのよいところまで処理して、一応利用可能な結果を出力するモードの２つがあります。後者のモードは、例えば大きなPDFファイルを出力中に処理を中断して、途中までの出力結果を見たい場合に有用です。ただし、1ページ目を出力される前に中断してしまった場合など、読み込み可能なデータが出力できないことも起こり得ます。

3.3.7 サーブレット/JSPでの利用

以下はjavax.sevlet を使用する場合の例です。jakarta.servlet を使用する場合は、jp.cssj.cti2.helpers.jakartaパッケージ内のクラスを使ってください。

サーブレットで、クライアントに対してドキュメントの変換結果を送る場合は jp.cssj.cti2.helpers.ServletHelper クラスの setServletResponse メソッドを使ってください。このメソッドは内部で jp.cssj.cti2.helpers.ServletResponseResults クラスを使用しており、出力結果に合わせてContent-Type, Content-Lengthヘッダを適切に設定します。

サーブレットやJSPが出力するデータをキャプチャしてCTISessionに渡す場合は、jp.cssj.cti2.helpers.CTIHttpServletResponseWrapper クラスを使用してください。このクラスは、キャプチャしたデータをリソースか、メインドキュメントとしてCTISessionに渡します。ServletResponse を CTIHttpServletResponseWrapper によりラップして、RequestDispatcherにより、他のサーブレット/JSPに転送すると、転送先のサーブレット/JSPによる出力をキャプチャします。あるいは、フィルタを使う方法があります。

次に紹介するサンプルプログラムは、ドライバのexamples/webappディレクトリにあります。このサンプルは、source.jspの出力を2通りの方法で変換します。１つめは、/pdf/ で始まるパスへのアクセスを、サーブレットで転送する方法です。/pdf/source.jspに対するアクセスを、RequestDispatcherにより/source.jspに転送し、転送先の出力を変換します。２つめは、/source.jspに対するアクセスを文字通りフィルタリングして変換する方法です。

それぞれjp.cssj.cti2.examples.SampleHttpServlet という名前のサーブレットと、jp.cssj.cti2.examples.SampleFilterという名前のフィルタを使う場合のweb.xmlの記述は次のとおりです。

例 3.13 web.xmlの記述例

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">
  <filter>
    <filter-name>sample-filter</filter-name>
    <filter-class>jp.cssj.cti2.examples.SampleFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>sample-filter</filter-name>
    <url-pattern>/source.jsp</url-pattern>
  </filter-mapping>

  <servlet>
```

```
    <servlet-name>sample-servlet</servlet-name>
    <servlet-class>jp.cssj.cti2.examples.SampleHttpServlet
</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>sample-servlet</servlet-name>
    <url-pattern>/pdf/*</url-pattern>
  </servlet-mapping>

</web-app>
```

次が、サーブレットの実装です。HttpServletRequestのgetPathInfoにより、ユーザーがアクセスしたアドレスの /pdf の後に続くパスを取得し、レスポンスをCTIHttpServletResponseWrapperでラップして、そのパスに転送します。

例えば、ユーザーが http://localhost:8180/webapp/pdf/source.jsp にアクセスすると、source.jsp の出力をPDF変換したものが返されます。

例 3.14 RequestDispatcherにより、他のJSPの出力をキャプチャする

```
package jp.cssj.cti2.examples;

import java.io.IOException;
import java.net.URI;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jp.cssj.cti2.CTIDriverManager;
import jp.cssj.cti2.CTISession;
import jp.cssj.cti2.helpers.CTIHttpServletResponseWrapper;

public class SampleHttpServlet extends HttpServlet {
    private static final long serialVersionUID = 0L;

    /** 接続先。 */
    private static final URI SERVER_URI = URI.create
("ctip://127.0.0.1:8099/");

    /** ユーザー。 */
    private static final String USER = "user";

    /** パスワード。 */
    private static final String PASSWORD = "kappa";

    protected void doGet(HttpServletRequest req,
```

```
HttpServletResponse res)
            throws ServletException, IOException {
            // 出力先をレスポンスに設定
            ServletHelper.setServletResponse(session, res);

            // PATH_INFOのアドレスに転送
            String path = ((HttpServletRequest) req).getPathInfo();

            // 転送先のサーブレットが出力したコンテンツを変換
            CTIHttpServletResponseWrapper ctiRes = new
CTIHttpServletResponseWrapper(
                    res, session, URI.create(path));
            try {
                req.getRequestDispatcher(path).forward(req,
ctiRes);
            } finally {
                ctiRes.close();
            }
    }
}
```

次は、フィルタの実装です。単にレスポンスをCTIHttpServletResponseWrapperでラップして処理を次に渡すだけです。

例えば、ユーザーが http://localhost:8180/webapp/source.jsp にアクセスすると、source.jspの出力をPDF変換したものが返されます。

例 3.15 Filterによる変換

```
package jp.cssj.cti2.examples;

import java.io.IOException;
import java.net.URI;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jp.cssj.cti2.CTIDriverManager;
import jp.cssj.cti2.CTISession;
import jp.cssj.cti2.helpers.CTIHttpServletResponseWrapper;

public class SampleFilter implements Filter {
    /** 接続先。 */
```

```
    private static final URI SERVER_URI = URI.create
("ctip://127.0.0.1:8099/");

    /** ユーザー。 */
    private static final String USER = "user";

    /** パスワード。 */
    private static final String PASSWORD = "kappa";

    private FilterConfig config;

    public void init(FilterConfig config) throws ServletException {
        this.config = config;
    }

    public void doFilter(ServletRequest _req, ServletResponse _res,
            FilterChain chain) throws IOException, ServletException
 {
        HttpServletRequest req = (HttpServletRequest) _req;
        HttpServletResponse res = (HttpServletResponse) _res;
        CTISession session = CTIDriverManager.getSession
(SERVER_URI, USER,
                PASSWORD);
        try {
            // 出力先をレスポンスに設定
            ServletHelper.setServletResponse(session, res);

            // 基底URLとしてコンテキスト以降のパスを使う
            URI uri = URI.create(req.getRequestURI().substring(
                    req.getContextPath().length()));

            // サーブレットが出力したコンテンツを変換
            CTIHttpServletResponseWrapper ctiRes = new
CTIHttpServletResponseWrapper(
                    (HttpServletResponse) res, session, uri);
            try {
                chain.doFilter(req, ctiRes);
            } finally {
                ctiRes.close();
            }
        } finally {
            session.close();
        }
    }

    public void destroy() {
        // ignore
    }
}
```

前記の例では、source.jspと一緒に置かれたCSSや画像が読み込まれません。これらを読み込むようにするには、次のようなSourceResolverを用意します。

例 3.16 リソースを読み込むSourceResolver

```
class ServletContextResolver implements SourceResolver {
    protected final ServletContext context;

    public ServletContextResolver(ServletContext context) {
        this.context = context;
    }

    public Source resolve(URI uri) throws IOException {
        // コンテキストに置かれたファイルを取得する
        URL url = this.context.getResource(uri.toString());
        if (url == null) {
            throw new FileNotFoundException(uri.toString());
        }
        try {
            return new URLSource(url);
        } catch (URISyntaxException e) {
            IOException ioe = new IOException();
            ioe.initCause(e);
            throw ioe;
        }
    }

    public void release(Source source) {
        ((URLSource) source).close();
    }
}
```

次のようにサーブレット内でこのSourceResolverを設定すると、source.jspからの相対パスでCSSや画像にアクセスできるようになります。

例 3.17 SourceResolverを設定する

```
    session.setSourceResolver(new ServletContextResolver(this
            .getServletContext()));
```

ただし、この方法では動的に生成したCSSや画像にはアクセスできませんので、ご注意ください。全てを動的に変換する場合は、Session.transcodeメソッドを呼び出して、ローカルホストのサーブレットコンテナにCopper PDFからアクセスするのがよいでしょう。

3.3.8 ソースコード

ドライバのソースコードはGitHubで公開しています。
https://github.com/zamasoftnet/cti.java

3.3.9 Copper PDFのライブラリに直接アクセスする

Javaでは Copper PDFサーバーを起動せずに、直接Copper PDFのライブラリを使用することができます。このためには、以下のようにドライバの接続先として特別なURIである"copper:direct:"を設定します。

例 3.18 直接Copper PDFのライブラリを使用する

```
import java.io.File;
import java.net.URI;

import jp.cssj.cti2.CTIDriverManager;
import jp.cssj.cti2.CTISession;
import jp.cssj.cti2.helpers.CTISessionHelper;
import jp.cssj.resolver.composite.CompositeSourceResolver;

public class DirectJava {
    public static void main(final String[] args) throws Exception {
        try (CTISession session = CTIDriverManager.getSession(URI
                .create("copper:direct:"))) {
            CTISessionHelper.setResultFile(session, new File
("test.pdf"));
            CompositeSourceResolver resolver =
CompositeSourceResolver
                    .createGenericCompositeSourceResolver();
            session.setSourceResolver(resolver);
            session.transcode(URI
                    .create("https://copper-pdf.com/"));
        };
    }
}
```

こうして得られたCTISessionの使用方法は通常通りドライバを使う場合と変わりありません。また、プログラムのコンパイルも通常通りドライバのjarをクラスパスに加えることで可能です。

しかし、当然ながらプログラムの実行時にはCopper PDFのlibディレクトリ内にあるjarをクラスパスに加える必要があります。また、Javaのシステムプロパティjp.cssj.copper.config, jp.cssj.driver.default にそれぞれ設定ディレクトリとdefault.propertiesファイルへのパスを設定する必要があります。

例えば、Windowsのバッチファイルにより起動する場合は次のようにします。この例では"C:\CopperPDF"に配置したCopper PDFを使用し、カレントディレクトリに置かれたDirectJava.classを実行します。

例 3.19 プログラムの起動(Windows)

```
set COPPER_DIR="C:\CopperPDF"
set CONFIG_DIR="%COPPER_DIR%\conf"
set DEFAULT_FILE="%COPPER_DIR%\conf\profiles/default.properties"
set LIB_DIR="%COPPER_DIR%\lib"

java -cp .;%LIB_DIR%\* -Djp.cssj.copper.config=%CONFIG_DIR% -
Djp.cssj.driver.default=%DEFAULT_FILE% DirectJava
```

Linux等でシェルスクリプトにより起動する場合は次のようにします。この例では、.rpmや.debでインストールしたCopper PDFを使用し、カレントディレクトリに置かれたDirectJava.classを実行します。

例 3.20 プログラムの起動(シェルスクリプト)

```
#!/bin/sh
CONFIG_DIR="/etc/copper-pdf"
DEFAULT_FILE="/etc/copper-pdf/profiles/default.properties"
LIB_DIR="/usr/share/copper-pdf/lib"

java -cp .:$LIB_DIR/* -Djp.cssj.copper.config=$CONFIG_DIR -
Djp.cssj.driver.default=$DEFAULT_FILE DirectJava
```

3.3.10 JRubyを使う場合

Java VMを利用したRuby実行環境であるJRubyでは、RubyからJava用のドライバを利用することができます。JRubyは普通のRuby(CRuby)と同じくらいか、時にはそれ以上の性能を発揮します。また、Javaと併用する手軽なスクリプト言語としても優秀です。ぜひ、JRubyの使用を検討してください。どうしてもCRubyを使う必要がある場合は、HTTP/RESTインターフェースを利用してください。

JRubyを使う場合、まずJava用ドライバをインストールしてください。次にcti-driver-2.x.x.jarを適当な場所(/usr/lib/jruby/libなど)に配置してください。

以下の例では、Copper PDFに接続し、Rubyで出力したHTMLをPDFに変換してファイルに保存します。

例 3.21 Rubyで出力したHTMLを変換する

```
include Java
require "cti-driver.jar" #jarのパスは環境に合わせてください
include_class Java::jp.cssj.cti2.CTIDriverManager
include_class Java::jp.cssj.cti2.CTISession
include_class Java::jp.cssj.cti2.helpers.CTIMessageHelper
include_class Java::jp.cssj.cti2.helpers.CTISessionHelper
include_class Java::jp.cssj.resolver.helpers.MetaSourceImpl
include_class Java::java.io.File
include_class Java::java.net.URI
include_class Java::java.lang.System

# セッションの開始
session = CTIDriverManager.getSession(URI.create
("ctip://localhost:8099/"),
    "user", "kappa")
begin

    # ファイル出力
    CTISessionHelper.setResultFile(session, File.new("test.pdf"))

    # エラーメッセージを標準エラー出力に表示する
    session.setMessageHandler
(CTIMessageHelper.createStreamMessageHandler(System.err))

    # サーバーへの出力をJavaのOutputStreamからRubyのioに変換して取得
    out = session.transcode(MetaSourceImpl.new(URI.create(".),
"text/html", "UTF-8")).to_io;
    begin
        out.puts <<DATA
<html>
<body>
JRubyからCopper PDFを使う。
</body>
</html>
DATA
    ensure
        # クローズを忘れないこと！
        out.close;
    end;

ensure
    # セッションの終了
    session.close
end
```

JRubyではJavaドライバのAPIをそのまま利用することができます。詳細はJavaドライバのAPIのドキュメントを参照してください。

3.3.11 Jythonを使う場合

Java VMを利用したPython実行環境であるJythonでは、PythonからJava用のドライバを利用することができます。Jythonは普通のPython(CPython)と同じくらいか、時にはそれ以上の性能を発揮します。どうしてもCPythonを使う必要がある場合は、HTTP/RESTインターフェースを利用してください。

Jythonを使う場合、まずJava用ドライバをダウンロードしてください。次にcti-driver-2.x.x.jarを適当な場所(/usr/share/java/など)に配置してください。

以下の例では、Copper PDFに接続し、Pythonで出力したHTMLをPDFに変換してファイルに保存します。

例 3.22 Pythonで出力したHTMLを変換する

```
# -*- coding: utf-8 -*-
import sys
sys.path.append("cti-driver.jar");
from jp.cssj.cti2.helpers import CTIMessageHelper
from jp.cssj.cti2.helpers import CTISessionHelper
from jp.cssj.resolver.helpers import MetaSourceImpl
from jp.cssj.driver.ctip import CTIPDriver
from java.io import File
from java.net import URI
from java.lang import System
from java.util import HashMap

driver = CTIPDriver()
params = HashMap()
params.put("user", "user")
params.put("password", "kappa")
session = driver.getSession(URI.create("ctip://localhost:8099/"),
params)
try:
    # ファイル出力
    CTISessionHelper.setResultFile(session, File("test.pdf"))

    # エラーメッセージを標準エラー出力に表示する
    session.setMessageHandler
(CTIMessageHelper.createStreamMessageHandler(System.err))

    # サーバーへの出力を取得
    out = session.transcode(MetaSourceImpl(URI.create(".")),
"text/html", "UTF-8"));
    try:
        out.write("""
<html>
```

```
<body>
JythonからCopper PDFを使う。
</body>
</html>
        """)
    finally:
        # クローズを忘れないこと！
        out.close()
finally:
    # セッションの終了
    session.close()
```

JythonではJavaドライバのAPIをそのまま利用することができます。詳細はJavaドライバのAPIのドキュメントを参照してください。

3.4 Perlドライバ２

3.4.1 概要

Perl用ドライバは、Perlスクリプトによる出力をPDFに変換できることが特徴です。CGI等として作成された他のPerlプログラムの出力を、もとのプログラムを変えないまま変換することができます。また、PDFのもととなるテンプレートをTemplateToolkit 等、Perlベースのテンプレートエンジンにより作成することができます。

3.4.2 ドライバの準備

Perl用ドライバはGitHubの https://github.com/zamasoftnet/cti.perl で公開されています。cpanmを使ってインストールすることができます（推奨）。

例 3.23 cpanmでのインストール

```
cpanm
https://github.com/zamasoftnet/cti.perl/archive/refs/heads/main.tar.gz
```

手動でインストールする場合は、`src/code/CTI/` ディレクトリをプロジェクトにコピーし、`@INC` に追加してください。アプリケーションは `use CTI::DriverManager;` でモジュールをインポートしてください。

Copper PDF 3.0.0, Perlドライババージョン2.1.0ではTLS通信に対応しています。TLS通信には IO::Socket::SSL モジュールが必要です。

例 3.24 copperdへの接続

```
# ドライバモジュールのインポート
use CTI::DriverManager;

# サーバーへの接続
my $uri = 'ctip://localhost:8099/';
my $session = CTI::DriverManager::get_session($uri,
    user => 'user', password => 'kappa');
# 各種操作
...
```

3.4.3 APIの概要

ここではAPIによるアクセスの概要で説明した各手順に対応する関数を列挙します。各関数の詳細は オンラインのAPIドキュメントを参照してください。

サーバーへの接続・認証

- get_driver URI
- get_session URI [OPTIONS]

- CTI__Driver->get_session URI [OPTIONS]

サーバー情報の取得

- CTI__Session->get_server_info FUNCTION

メッセージハンドラ・プログレスリスナの設定

- CTI__Session->set_message_func FUNCTION
- CTI__Session->set_progress_func FUNCTION

出力先の設定

- CTI__Session->set_results RESULTS
- CTI__Session->set_output_as_handle FILEHANDLE
- CTI__Session->set_output_as_file FILENAME
- CTI__Session->set_output_as_directory DIRNAME

プロパティの設定

- CTI__Session->property NAME VALUE

ソースリゾルバの設定

- CTI::Session->set_resolver_func FUNCTION

リソースの送信

- CTI::Session->start_resource FILEHANDLE URI [OPTIONS]
- CTI::Session->end_resource FILEHANDLE

本体の送信・変換

- CTI::Session->transcode URI
- CTI::Session->start_main FILEHANDLE URI [OPTIONS]
- CTI::Session->end_main FILEHANDLE

複数の結果の結合

- CTI::Session->set_continuous MODE
- CTI::Session->join

処理の中断・リセット・通信の終了

- CTI::Session->abort MODE
- CTI::Session->reset
- CTI::Session->close

3.4.4 サンプル

以下は、プログラムによる出力を変換するサンプルです。start_main, end_main関数の間の標準出力への出力をキャプチャして変換します。

例 3.25 プログラムによる出力を変換する

```
#!/usr/bin/perl
use strict;
use lib '../code';
use CTI::DriverManager;

# Copper PDFに接続
my $uri = 'ctip://localhost:8099/';
my $session = CTI::DriverManager::get_session($uri,
    user => 'user', password => 'kappa');

# ファイル出力
$session->set_output_as_file('test.pdf');

# リソースの送信
$session->start_resource(*STDOUT, 'style.css',
    mime_type => 'text/css');
print "p {color: Red;}";
$session->end_resource(*STDOUT);

# 出力の変換を開始
$session->start_main(*STDOUT, '.',
    mime_type => 'text/html');
print "<html>";
print "<head>";
print "<meta http-equiv='Content-Type' content='text/html;
charset=UTF-8'>";
print "<link rel='StyleSheet' type='text/css' href='style.css'>";
print "<title>サンプル</title>";
```

```
print "</head>";
print "<body>";
print "<p>Hello World!</p>";
print "</body>";
print "</html>";
# 出力の変換を終了
$session->end_main(*STDOUT);

# 接続を閉じる
$session->close();
```

次の例は、サーバー側からネットワーク上のウェブページアクセスしてPDFに変換します。

例 3.26 サーバー側ウェブページにアクセスしてPDFに変換

```
#!/usr/bin/perl
use strict;
use lib '../code';
use CTI::DriverManager;

# セッションの開始
my $uri = 'ctip://localhost:8099/';
my $session = CTI::DriverManager::get_session($uri,
    user => 'user', password => 'kappa');

# ファイル出力
$session->set_output_as_file('test.pdf');

# リソースのアクセス許可
$session->property('input.include', 'https://copper-pdf.com/**');

# 文書の送信
$session->transcode('https://copper-pdf.com/');

# セッションの終了
$session->close();
```

他のサンプルはドライバのsrc/testに収められています。

3.4.5 プログラミングのポイント

Content-Type，Content-Length ヘッダの出力

ウェブアプリケーションでは、Content-Type，Content-Lengthヘッダを出力しないと、ブラウザ上でPDFが正常に表示されないことがあります。以下のようにset_output_as_handle関数の2番目の引数に1を設定してください。これらのヘッダが自動的に出力されます。

例 3.27 ヘッダの出力を有効にする

```
...
$session->set_output_as_handle(*STDOUT, 1);
...
```

他のプログラムを呼び出して変換する

他のプログラムによる出力を変換するには、start_main, end_mainの間でrequireしてください。CGIプログラム等は、ヘッダを出力することがありますが、start_mainにignore_headers => 1 オプションを加えて呼び出すと、これを除去します。以下の例では、bbs.cgiというプログラムの出力を変換します。

例 3.28 他のプログラムの出力を変換する

```
...
$session->start_main(*STDOUT, '.',
    mime_type => 'text/html',
    ignore_headers => 1);
require "bbs.cgi";
$session->end_main(*STDOUT);
...
```

start_resource, end_resourceについても、同じことができます。

繰り返し処理

出力先を変え、start_main/end_main または transcodeを繰り返し呼び出すことで、同じセッションで何度もドキュメントを変換することができます。送信済みのリソース、設定済みのプロパティは同じセッションで維持されます。同じセッションのまま初期状態に戻すには $session->reset() を呼び出してください。

出力先の設定

set_output_as_handle, set_output_as_fileにより、単一のファイルハンドルかファイルを出力先として設定することができます。

複数の結果をファイルとして出力する場合は、set_output_as_directory を使用してください。このクラスは、指定したディレクトリに、1から開始する連番の前後に指定した文字列をくっつけたファイル名で結果を出力します。次の例では変換結果の各ページを、resultsディレクトリ内に"image[通し番号].jpeg"という名前で別々のJPEG画像として出力します。

例 3.29 ディレクトリに結果を出力する

```
...
$session->property("output.type", "image/jpeg");
$session->set_output_as_directory("results", "image", ".jpeg");
...
```

サーバーから要求されたリソースの送信

サーバーから要求されたリソースを送る場合、set_resolver_funcに関数を設定してください。関数には、要求されたファイルのURIと、サーバーへのファイルハンドルを得るための関数への参照が渡されます。以下の例では、URIが相対パスであるという前提で、ローカルのファイルがあればサーバーに送信します。

例 3.30 サーバーから要求されたリソースをローカルファイルから送信する

```
...
$session->set_resolver_func(sub {
    my ($uri, $open) = @_;
    if (-e $uri) {
      my $fp = $open->();
      open(my $rfp, "<$uri");
      while (<$rfp>) {print $fp $_};
      close($rfp);
    }
    return undef;
});
...
```

実際に動作するサンプルはドライバの src/test/resolver.pl にあります。

複数の結果の結合

複数の結果を結合したものを得るためには、$session->set_continuous(1) を呼び出した後、transcodeを複数回呼び出し、最後に $session->join() を呼び出してください。

例 3.31 2つの結果の結合

```
...
$session->set_continuous(1);

# 文書の送信
$session->start_main(*STDOUT, '.');
open($rfp, '<data/test.html');
while (<$rfp>) {print};
close($rfp);
$session->end_main(*STDOUT);

# 文書の送信
$session->start_main(*STDOUT, '.');
open($rfp, '<data/test.html');
while (<$rfp>) {print};
close($rfp);
$session->end_main(*STDOUT);

$session->join();
...
```

実際に動作するサンプルはドライバの src/test/continuous.pl にあります。

3.4.6 ソースコード

ドライバのソースコードは GitHub で公開しています。
https://github.com/zamasoftnet/cti.perl

3.5 PHPドライバ2

3.5.1 概要

PHP用ドライバは、PHPによる出力をPDFに変換できることが特徴です。他のPHPプログラムの出力を、もとのプログラムを変えないまま変換することができます。また、PDFのもととなるテンプレートをSmarty等、PHPベースのテンプレートエンジンにより作成することができます。

3.5.2 ドライバの準備

PHP用ドライバはGitHubの https://github.com/zamasoftnet/cti.php で公開されています。Composerを使ってインストールすることができます（推奨）。

例 3.32 Composerでのインストール

```
composer require zamasoft/cti-php
```

またはVCSリポジトリを登録してインストールすることもできます。

例 3.33 VCSリポジトリからのインストール

```
composer config repositories.zamasoftnet-cti-php vcs
https://github.com/zamasoftnet/cti.php
composer require zamasoft/cti-php
```

手動でインストールする場合は、**src/CTI/** ディレクトリをプロジェクトにコピーし、**require_once 'path/to/CTI/DriverManager.php';** でドライバを読み込んでください。

例 3.34 copperdへの接続

```
// ドライバの読み込み
require_once ('CTI/DriverManager.php');

// セッションの開始
$session = cti_get_session('ctip://localhost:8099/',
    array('user' => 'user',
    'password' => 'kappa'));

// 各種操作
...
```

3.5.3 APIの概要

ここではAPIによるアクセスの概要で説明した各手順に対応する関数を列挙します。各関数の詳細は オンラインのAPIドキュメントを参照してください。

サーバーへの接続・認証

- cti_get_driver($uri)
- cti_get_session($uri, $opts)
- Driver->get_session($uri, $opts)

サーバー情報の取得

- Session->get_server_info($uri)

メッセージハンドラ・プログレスリスナの設定

- Session->set_message_func(&$messageFunc)
- Session->set_progress_func(&$progressFunc)

出力先の設定

- Session->set_results(&$results)
- Session->set_output_as_resource(&$fp)
- Session->set_output_as_file($file)
- Session->set_output_as_directory($dir, [$prefix = ''], [$suffix = ''])
- Session->set_output_as_variable(&$var)

プロパティの設定

- Session->property($name, $value)

ソースリゾルバの設定

- Session->set_resolver_func(&$resolverFunc)

リソースの送信

- Session->start_resource($uri, [$opts = array()])
- Session->end_resource()

本体の送信・変換

- Session->transcode($uri)
- Session->start_main($uri, [$opts = array()])
- Session->end_main()

複数の結果の結合

- Session->set_continuous($continuous)
- Session->join()

処理の中断・リセット・通信の終了

- Session->abort($mode)
- Session->reset()
- Session->close()

3.5.4 サンプル

以下は、プログラムによる出力を変換するサンプルです。start_main, end_main関数の間の標準出力への出力をキャプチャして変換します。

例 3.35 プログラムによる出力を変換する

```
<?php
require_once ('CTI/DriverManager.php');

//セッションの開始
$session = cti_get_session('ctip://localhost:8099/',
    array('user' => 'user',
    'password' => 'kappa'));

// ファイル出力
@mkdir($dir, 0777, 'out');
$session->set_output_as_file('test.pdf');

// リソースの送信
$session->start_resource('style.css',
    array('mimeType' => 'text/css'));
?>
p {color: Red;}
<?php
$session->end_resource();

// 出力の変換を開始
$session->start_main('.',
    array('mimeType' => 'text/html'));
?>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="StyleSheet" type="text/css" href="style.css">
<title>サンプル</title>
```

```
</head>
<body>
<p>Hello World!</p>
</body>
</html>
<?php
// 出力の変換を終了
$session->end_main();

// セッションの終了
$session->close();
?>
```

次の例は、サーバー側からネットワーク上のウェブページアクセスしてPDFに変換します。

例 3.36 サーバー側ウェブページにアクセスしてPDFに変換

```
<?php
require_once ('CTI/DriverManager.php');

//セッションの開始
$session = cti_get_session('ctip://localhost:8099/',
    array('user' => 'user',
    'password' => 'kappa'));

//ファイル出力
@mkdir($dir, 0777, 'out');
$session->set_output_as_file('test.pdf');

//リソースのアクセス許可
$session->property('input.include', 'https://copper-pdf.com/**');

//文書の送信
$session->transcode('https://copper-pdf.com/');

//セッションの終了
$session->close();
?>
```

他のサンプルはドライバのsrc/testに収められています。

3.5.5 プログラミングのポイント

Content-Type，Content-Length ヘッダの出力

ウェブアプリケーションでは、Content-Type，Content-Lengthヘッダを出力しないと、ブラウザ上でPDFが正常に表示されないことがあります。set_resultsやset_output_XXX関

数を呼び出さない初期状態では、出力先は標準出力に設定されています。このとき、自動的にContent-Lengthヘッダが出力されます。Content-Type はheader関数により、アプリケーションで出力してください。

他のプログラムを呼び出して変換する

他のプログラムによる出力を変換するには、start_main, end_mainの間でincludeしてください。以下の例では、bbs.phpというプログラムの出力を変換します。

例 3.37 他のプログラムの出力を変換する

```
...
$session->start_main('.',
    array('mimeType' => 'text/html'));
include("bbs.php");
$session->end_main();
...
```

start_resource, end_resourceについても、同じことができます。

繰り返し処理

出力先を変え、start_main/end_main または transcodeを繰り返し呼び出すことで、同じセッションで何度もドキュメントを変換することができます。送信済みのリソース、設定済みのプロパティは同じセッションで維持されます。同じセッションのまま初期状態に戻すには $session->reset() を呼び出してください。

出力先の設定

set_output_as_resource, set_output_as_fileにより、単一のファイルハンドルかファイルを出力先として設定することができます。また、set_output_as_variableに変数の参照を渡すと、変数に結果が出力されます。

複数の結果をファイルとして出力する場合は、set_output_as_directory を使用してください。このクラスは、指定したディレクトリに、1から開始する連番の前後に指定した文字列をくっつけたファイル名で結果を出力します。次の例では変換結果の各ページを、resultsディレクトリ内に"image[通し番号].jpeg"という名前で別々のJPEG画像として出力します。

例 3.38 ディレクトリに結果を出力する

```
...
$session->property("output.type", "image/jpeg");
$session->set_output_as_directory("results", "image", ".jpeg");
...
```

サーバーから要求されたリソースの送信

サーバーから要求されたリソースを送る場合、set_resolver_funcに関数を設定してください。関数には、要求されたファイルのURIと、サーバーへのファイルハンドルを得るための関数への参照が渡されます。以下の例では、URIが相対パスであるという前提で、ローカルのファイルがあればサーバーに送信します。

例 3.39 サーバーから要求されたリソースをローカルファイルから送信する

```
...
function resolver($uri, $r) {
  if (file_exists($uri)) {
    $r->start();
    readfile($uri);
    $r->end();
  }
}
$func = 'resolver';
$session->set_resolver_func($func);
...
```

実際に動作するサンプルはドライバの src/test/resolver.php にあります。

複数の結果の結合

複数の結果を結合したものを得るためには、$session->set_continuous(TRUE) を呼び出した後、transcodeを複数回呼び出し、最後に $session->join() を呼び出してください。

例 3.40 2つの結果の結合

```
...
$session->set_continuous(TRUE);

//文書の送信
$session->start_main('.');
readfile("data/test.html");
$session->end_main();

//文書の送信
$session->start_main('.');
readfile("data/test.html");
$session->end_main();

$session->join();
...
```

実際に動作するサンプルはドライバの src/test/continuous.php にあります。

3.5.6 ソースコード

ドライバのソースコードは GitHub で公開しています。
https://github.com/zamasoftnet/cti.php

3.6 .NETドライバ

3.6.1 概要

.NETドライバはC#で書かれており、C#, VB.NETなどから利用することができます。ストリーム（System.IO.Stream）からストリームへの変換に対応しており、巨大な文書も容易に変換することができます。

また、ASP.NETで利用する際は、ASP.NETからの出力をキャプチャしながら、変換結果を送り出すことができます。これにより、PDFの出力も普通のウェブページと同様にASP.NETにより作ることができます。

3.6.2 ドライバの準備

.NET用ドライバはGitHubの https://github.com/zamasoftnet/cti.net で公開されており、NuGetパッケージ（Zamasoft.CTI）として提供されています。.NET Standard 2.0以降（.NET Framework 4.6.1以降、.NET Core 2.0以降、.NET 5〜9）に対応しています。

以下のコマンドでインストールできます。

例 3.41 NuGetでのインストール

```
dotnet add package Zamasoft.CTI
```

またはプロジェクトファイルに直接追加することもできます（バージョンはNuGetのページで最新版を確認してください）。

例 3.42 プロジェクトファイルへの追加

```
<PackageReference Include="Zamasoft.CTI" Version="X.X.X" />
```

NuGetを使わない場合は、GitHubのリリースページからソースアーカイブをダウンロードし、`CTI/CTI.sln` をVisual Studioまたは `dotnet build` でビルドすることもできます。

https://github.com/zamasoftnet/cti.net/releases

ドライバの窓口となるクラスはCTI.DriverManagerです。例えばlocalhostの8099番ポートで起動しているcopperdに、ユーザーID"user"、パスワード"kappa"で接続するには、以下のようにします。

例 3.43 copperdへの接続

```
using System;
using CTI;
...

using (Session session = DriverManager.getSession(new Uri
("ctip://localhost:8099/"), "user", "kappa"))
{
//各種操作
...
}
...
```

3.6.3 タイムアウトの設定

.NET版ドライバ2.0.1以降から、一定時間通信がない状態で自動的に通信を切断するタイムアウトに対応しています。

以下のようにURLパラメータで、timeoutをミリ秒単位で指定できます。

```
ctip://127.0.0.1:8099/?timeout=10000
```

3.6.4 APIの概要

ここではAPIによるアクセスの概要で説明した各手順に対応する関数を列挙します。各関数の詳細はオンラインのAPIドキュメントを参照してください。

サーバーへの接続・認証

- public static CTIDriver GetDriver(Uri uri)
- public static CTISession GetSession(Uri uri, Hashtable props)
- public static CTISession GetSession(Uri uri, string user, string password)

サーバー情報の取得

- public Stream GetServerInfo(Uri uri)

メッセージハンドラ・プログレスリスナの設定

- MessageHandler MessageHandler { set; }
- ProgressListener ProgressListener { set; }

出力先の設定

- Results Results { set; }

プロパティの設定

- public void Property(string name, string value)

ソースリゾルバの設定

- SourceResolver SourceResolver { set; }

リソースの送信

- public void Resource(SourceInfo info, Stream input)
- public Stream Resource(SourceInfo info)

本体の送信・変換

- public void Transcode(SourceInfo info, Stream input)
- public Stream Transcode(SourceInfo info)
- public void Transcode(string uri)

複数の結果の結合

- bool Continuous { set; }
- public void Join()

処理の中断・リセット・通信の終了

- public void Abort(AbortMode mode)
- public void Reset()
- public void Close()

3.6.5 サンプル

次の例は、サーバー側からネットワーク上のウェブページアクセスしてPDFに変換します。

例 3.44 サーバー側からウェブページにアクセスしてPDFに変換(C#)

```
using System;
using Zamasoft.CTI;

namespace examples
{
    /// <summary>
    /// サーバー側からインターネット上の文書にアクセスして変換します。
    /// </summary>
    class ServerResource
    {
        static void Main(string[] args)
        {
            using (Session session = DriverManager.getSession(new
Uri("ctip://localhost:8099/"), "user", "kappa"))
            {
                // test.pdfに結果を出力する
                Utils.SetResultFile(session, "test.pdf");

                // https://copper-pdf.com/以下にあるリソースへのアクセス
を許可する
                session.Property("input.include", "https://copper-
pdf.com/**");

                // ウェブページを変換
                session.Transcode("https://copper-pdf.com/");
            }
        }
    }
}
```

例 3.45 サーバー側からウェブページにアクセスしてPDFに変換(VB.NET)

```
Imports System
Imports System.IO
Imports Zamasoft.CTI

''' <summary>
''' サーバー側からインターネット上の文書にアクセスして変換します。
''' </summary>
Module ServerResource

    Sub Main()
        Using session As Session = DriverManager.getSession(New Uri
("ctip://localhost:8099/"), "user", "kappa")
            ' test.pdfに結果を出力する
            Utils.SetResultFile(session, "test.pdf")
```

```
            ' https://copper-pdf.com/以下にあるリソースへのアクセスを許可
する
            session.Property("input.include", "https://copper-
pdf.com/**")

            ' ウェブページを変換
            session.Transcode("https://copper-pdf.com/")
        End Using
    End Sub

End Module
```

.NET版ドライバの特徴は、ASP.NETをPDF出力のためのテンプレートとして利用できるようになることです。前準備として、以下のクラスを用意しておきます。

例 3.46 ASP.NETのために用意しておくクラス(C#)

```
    public class CopperPDF
    {
        // 結果を直接ブラウザに返すように設定します。
        static public void SetResponse(Session session,
HttpResponse response)
        {
            session.Results = new SingleResult(new
ContentLengthSender(response));
        }

        // Content-Lengthヘッダを送信するためのビルダー。
        private class ContentLengthSender : Builder
        {
            private readonly HttpResponse response;
            public ContentLengthSender(HttpResponse response)
                : base(response.OutputStream)
            {
                this.response = response;
            }

            public override void Finish()
            {
                this.response.ContentType = this.info.MimeType;
                response.AppendHeader("Content-Length",
this.length.ToString());
                base.Finish();
            }
        }
    }
```

例 3.47 ASP.NETのために用意しておくクラス(VB.NET)

```
Public Class CopperPDF
    ' 結果を直接ブラウザに返すように設定します。
    Public Shared Sub SetResponse(session As Session, response As
HttpResponse)
        session.Results = New SingleResult(New ContentLengthSender
(response))
    End Sub

    ' Content-Lengthヘッダを送信するためのビルダー。
    Private Class ContentLengthSender
        Inherits Builder
        Private ReadOnly response As HttpResponse

        Sub New(response As HttpResponse)
            MyBase.New(response.OutputStream)
            Me.response = response
        End Sub

        Overrides Sub Finish()
            response.ContentType = Info.MimeType
            response.AppendHeader("Content-Length", length.ToString
())
            MyBase.Finish()
        End Sub

    End Class
End Class
```

ASP.NETにより生成したPDFを直接ブラウザに送る場合、HTTPレスポンスのContent-Typeヘッダに"application/pdf"を設定し、Content-LengthヘッダにPDFファイルのサイズを設定する必要があります。上記のContentLengthSenderは、そのためのもので、親クラスのBuilderからPDFのMIME型(Info.MimeType)、ファイルサイズ(length)を得ています。

ASP.NETで、他のページをPDFのテンプレートとして利用するには、以下のようにします。

例 3.48 他のページを実行して、結果をPDFに変換する(C#)

```
    CopperPDF.SetResponse(session, Response);
    string template = Request.ApplicationPath + "PDFTemplate.aspx";
    using (StreamWriter writer = new StreamWriter(session.Transcode
(new SourceInfo(".")))
    {
        Server.Execute(template, writer);
    }
```

例 3.49 他のページを実行して、結果をPDFに変換する(VB.NET)

```
    CopperPDF.SetResponse(session, Response)
    Dim template As String = Request.ApplicationPath +
"PDFTemplate.aspx"
    Using writer As New StreamWriter(session.Transcode(New
SourceInfo(".")))
        Server.Execute(template, writer)
    End Using
```

上記の例ではPDFTemplate.aspxを実行した結果をCopper PDFに送り、変換した結果をブラウザに送信しています。PDFTemplate.aspxは普通のASP.NETページなので、例えばPDF上にカレンダーを印刷するためにカレンダーコントロールを利用するといったことができます。

他のサンプルはドライバのCTIディレクトリに収められています。

3.6.6 プログラミングのポイント

Utilsの利用

結果の出力先、リソースの送信、ファイルの変換等のよく使われる操作が、Zammasoft.CTI.Utils のstatic(Shared)メソッドにまとめられています。

例えば、事前に関連するCSSを送信してHTMLファイルを変換する処理は、次のように簡単に書けます。

例 3.50 ファイルを変換する(C#)

```
...
Utils.SendResourceFile(session, "test.css", "text/css", "UTF-8");
Utils.TranscodeFile(session, "test.html", "text/html", "UTF-8");
...
```

例 3.51 ファイルを変換する(VB.NET)

```
...
Utils.SendResourceFile(session, "test.css", "text/css", "UTF-8")
Utils.TranscodeFile(session, "test.html", "text/html", "UTF-8")
...
```

繰り返し処理

出力先を変え、Transcodeメソッドを繰り返し呼び出すことで、同じセッションで何度もドキュメントを変換することができます。送信済みのリソース、設定済みのプロパティは同じセッションで維持されます。同じセッションのまま初期状態に戻すには Reset を呼び出してください。

出力先(Results)の設定

出力先が単一のファイルやストリームの場合は、Utilsの SetResultFile か、SetResultStream を使ってください。これらのメソッドは内部的にZamasoft.CTI.Result.SingleResultクラスを使用しています。

Zamasoft.CTI.Result.Resultsインターフェースは、複数の出力結果を受け取るためのインターフェースです。SessionのResultsプロパティにセットします。

複数の結果をファイルとして出力する場合は、Zamasoft.CTI.Result.FileResults を使用してください。このクラスは、指定したディレクトリに、1から開始する連番の前後に指定した文字列をくっつけたファイル名で結果を出力します。次の例では変換結果の各ページを、resultsディレクトリ内に"image[通し番号].jpeg"という名前で別々のJPEG画像として出力します。

例 3.52 ディレクトリに結果を出力する(C#)

```
...
session.Property("output.type", "image/jpeg");
session.Results = new FileResults("results/image", ".jpeg"));
...
```

例 3.53 ディレクトリに結果を出力する(VB.NET)

```
...
session.Property("output.type", "image/jpeg")
session.Results = New FileResults("results/image", ".jpeg"))
...
```

さらに複雑な処理が必要な場合は、Resultsインターフェースを実装するクラスを用意する必要があります。Resultsインターフェースを実装したクラスでは、Zamasoft.CTI.Result.Builder を使ってストリームやファイルにデータを出力してください。

サーバーから要求されたリソースの送信(SourceResolver)

Session の SourceResolver プロパティにソースリゾルバ(Zamasoft.CTI.Source.SourceResolver) を設定すると、サーバーから要求されたリソースを都度送信できるようになります。

次の例のように、SourceResolverを実装すると、クライアント側のファイル、あるいはクライアント側からウェブにアクセスして取得したデータをサーバーに送ることができます。

例 3.54 SourceResolverの実装例(C#)

```
    class MySourceResolver : SourceResolver
    {
        public Stream Resolve(string _uri, ref SourceInfo info)
        {
            Uri uri = new Uri(_uri);
            if (uri.IsFile)
            {
                string file = uri.AbsolutePath;
                if (!File.Exists(file))
                {
                    return null;
                }
                info = new SourceInfo(_uri);
                return new FileStream(file, FileMode.Open,
FileAccess.Read);
            }
            else if (uri.Scheme == "http")
            {
                WebRequest req = WebRequest.Create(uri);
                WebResponse resp = req.GetResponse();
                info = new SourceInfo(_uri);
                info.MimeType = resp.Headers.Get("Content-Type");
                return resp.GetResponseStream();
            }
            return null;
        }
    }
```

例 3.55 SourceResolverの実装例(VB.NET)

```
    Class MySourceResolver
        Implements SourceResolver

        Function Resolve(_uri As String, ByRef info As SourceInfo)
As Stream Implements SourceResolver.Resolve
            Dim uri As New Uri(_uri)
            If uri.IsFile Then
                Dim file As String = uri.AbsolutePath
                If Not System.IO.File.Exists(file) Then
                    Return Nothing
                End If
                info = New SourceInfo(_uri)
                Return New FileStream(file, FileMode.Open,
FileAccess.Read)
            ElseIf uri.Scheme = "http" Then
```

```
                Dim req As WebRequest = WebRequest.Create(uri)
                Dim resp As WebResponse = req.GetResponse()
                info = New SourceInfo(_uri)
                info.MimeType = resp.Headers.Get("Content-Type")
                Return resp.GetResponseStream()
            End If

            Return Nothing
        End Function
    End Class
```

SourceInfo

Sessionの Resource, Transcode メソッド等では、データの仮想URI、MIME型、キャラクタ・エンコーディング、予測されるデータサイズ。SourceInfo クラスにより渡します。

複数の結果の結合

複数の結果を結合したものを得るためには、Continuous プロパティにtrueを設定した後、Transcodeを複数回呼び出し、最後に Join を呼び出してください。

例 3.56 2つの結果の結合(C#)

```
...
session.Continuous = true;
Utils.SendResourceFile(session, "test.css", "text/css", "UTF-8");
Utils.TranscodeFile(session, "test.html", "text/html", "UTF-8");
session.Transcode("https://copper-pdf.com/");
session.Join();
...
```

例 3.57 2つの結果の結合(VB.NET)

```
...
session.Continuous = true
Utils.SendResourceFile(session, "test.css", "text/css", "UTF-8")
Utils.TranscodeFile(session, "test.html", "text/html", "UTF-8")
session.Transcode("http://copper-pdf.com/")
session.Join()
...
```

Abortによる中断

SessionのAbort メソッドは文書の変換処理を中断しますが、Transcodeメソッドは処理の間ブロックするため、別スレッドからAbortを呼び出す必要があります。Abortは引数によって、強制的に中断するモードと、きりのよいところまで処理して、一応利用可能な結果を出力するモードの２つがあります。後者のモードは、例えば大きなPDFファイルを出力中に処理を中断して、途中までの出力結果を見たい場合に有用です。ただし、1ページ目を出力される前に中断してしまった場合など、読み込み可能なデータが出力できないことも起こり得ます。

3.6.7 ソースコード

ドライバのソースコードは GitHub で公開しています。
https://github.com/zamasoftnet/cti.net

3.7 Rubyドライバ

3.7.1 概要

Rubyドライバは、ストリーム(IO)への出力をPDFに変換することができます。例えば、ERBでPDFの元となるHTMLのテンプレートを作ることができます。

3.7.2 ドライバの準備

Ruby用ドライバはGitHubの https://github.com/zamasoftnet/cti.ruby で公開されています。Bundlerを使ってインストールすることができます（推奨）。

`Gemfile` に以下を追加してください。

例 3.58 Bundlerでのインストール

```
gem 'copper-cti', git:
'https://github.com/zamasoftnet/cti.ruby.git'
```

その後 `bundle install` を実行してください。インストール後は `require 'CTI'` でドライバを読み込んでください。

例 3.59 copperdへの接続

```
# ドライバの読み込み
require 'CTI'
include CTI

# セッションの開始
get_session('ctip://localhost:8099/',
{
  'user' => 'user',
  'password' => 'kappa'
}
) do |session|
    # 各種操作
    ...
end
```

3.7.3 APIの概要

ここではAPIによるアクセスの概要で説明した各手順に対応する関数を列挙します。各関数の詳細は オンラインのAPIドキュメントを参照してください。

サーバーへの接続・認証

- CTI#get_driver(uri)
- CTI#get_session(uri, options = {}, &block)

ユーティリティ

- CTI#copy_stream(inp, out)

サーバー情報の取得

- CTI::Session#get_server_info(uri)

メッセージハンドラ・プログレスリスナの設定

- CTI::Session#receive_message(&messageFunc)
- CTI::Session#set_progress_func(&progressFunc)

出力先の設定

- CTI::Session#set_results(results)
- CTI::Session#set_output_as_stream(out)
- CTI::Session#set_output_as_file(file)
- CTI::Session#set_output_as_directory(dir, prefix = '', suffix = '')

プロパティの設定

- CTI::Session#property(name, value)

ソースリゾルバの設定

- CTI::Session#resolver(&resolverFunc)

リソースの送信

- CTI::Session#resource(uri, opts = {}, &block)

本体の送信・変換

- CTI::Session#transcode(uri = '.', opts = {}, &block)
- CTI::Session#transcodeServer(uri)

複数の結果の結合

- CTI::Session#set_continuous(continuous)
- CTI::Session#join

処理の中断・リセット・通信の終了

- CTI::Session#abort(mode)
- CTI::Session#reset
- CTI::Session#close

3.7.4 サンプル

以下は、ERBによる出力を変換するサンプルです。$stdoutを途中でCopper PDFへの出力に切り替えています。

例 3.60 プログラムによる出力を変換する

```
require 'CTI'
include CTI
require 'erb'

# セッションの開始
get_session('ctip://localhost:8099/',
{
  'user' => 'user',
  'password' => 'kappa'
}
) do |session|
  # ファイル出力
  dir = 'out';
  Dir::mkdir(dir, 0777) unless File.exist?(dir)
  session.set_output_as_file('out/erb.pdf')

  # テンプレートを変換
  session.transcode do |out|
    begin
      $stdout = out
      ERB.new(DATA.read).run
    ensure
      $stdout = STDOUT
    end
  end
end
```

```
__END__
<html>
  <head>
    <title>ERB</title>
  </head>
  <body>
  <p>Hello ERB</p>
  <p>ただいまの時刻は <%= Time.now %></p>
  </body>
</html>
```

次の例は、サーバー側からネットワーク上のウェブページアクセスしてPDFに変換します。

例 3.61 サーバー側ウェブページにアクセスしてPDFに変換

```
require 'CTI'
include CTI

# セッションの開始
get_session('ctip://localhost:8099/',
{
  'user' => 'user',
  'password' => 'kappa'
}
) do |session|
  # ファイル出力
  dir = 'out';
  Dir::mkdir(dir, 0777) unless File.exist?(dir)
  session.set_output_as_file('out/server-resource.pdf')

  #リソースのアクセス許可
  session.property('input.include', 'https://copper-pdf.com/**')

  #文書の変換
  session.transcodeServer('https://copper-pdf.com/');
end
```

他のサンプルはドライバのsrc/testに収められています。

3.7.5 プログラミングのポイント

Content-Type，Content-Length ヘッダの出力

ウェブアプリケーションでは、Content-Type，Content-Lengthヘッダを出力しないと、ブラウザ上でPDFが正常に表示されないことがあります。set_resultsやset_output_XXXメソッドを呼び出さない初期状態では、出力先は標準出力(STDOUT)に設定されています。このとき、自動的にContent-Type，Content-Lengthヘッダが出力されます。

これはSingleResultとStreamBuilderのコンストラクタに渡すことができるブロックを利用しています。同じ状態にするためには、以下のようにSession#set_resultsメソッドを呼び出してください。

例 3.62 ヘッダの出力

```
session.set_results(SingleResult.new(StreamBuilder.new(STDOUT) do
|length|
  print "Content-Length: #{length}\r\n\r\n"
end) do |opts|
  print "Content-Type: #{opts['mime_type']}\r\n"
end)
```

繰り返し処理

出力先を変え、transcode または transcodeServerを繰り返し呼び出すことで、同じセッションで何度もドキュメントを変換することができます。送信済みのリソース、設定済みのプロパティは同じセッションで維持されます。同じセッションのまま初期状態に戻すには session.reset を呼び出してください。

出力先の設定

set_output_as_stream, set_output_as_fileにより、単一のファイルハンドルかファイルを出力先として設定することができます。

複数の結果をファイルとして出力する場合は、set_output_as_directory を使用してください。このクラスは、指定したディレクトリに、1から開始する連番の前後に指定した文字列をくっつけたファイル名で結果を出力します。次の例では変換結果の各ページを、resultsディレクトリ内に"image[通し番号].jpeg"という名前で別々のJPEG画像として出力します。

例 3.63 ディレクトリに結果を出力する

```
session.property("output.type", "image/jpeg")
session.set_output_as_directory("results", "image", ".jpeg")
```

サーバーから要求されたリソースの送信

サーバーから要求されたリソースを送る場合、resolverを呼び出してください。このメソッドには、要求されたファイルのURIと、サーバーへのファイルハンドルを得るためのブロックを渡します 以下の例では、URIが相対パスであるという前提で、ローカルのファイルがあればサーバーに送信します。RubyのFileUtils::copy_stream, IO::copy_streamは使用できないため、CTI::copy_streamを使ってください。

例 3.64 サーバーから要求されたリソースをローカルファイルから送信する

```
session.resolver do |uri, r|
  if File.exist?(uri)
    r.found do |out|
      copy_stream(File.open(uri), out)
    end
  end
end
```

実際に動作するサンプルはドライバの src/test/resolver.rb にあります。

複数の結果の結合

複数の結果を結合したものを得るためには、session.set_continuous(true) を呼び出した後、transcodeを複数回呼び出し、最後に session.join を呼び出してください。

例 3.65 2つの結果の結合

```
session.set_continuous(true)

# 文書の送信
session.transcode do |out|
  copy_stream(File.open('data/test.html'), out)
end

#リソースのアクセス許可
session.property('input.include', 'https://copper-pdf.com/**')

#文書の変換
session.transcodeServer('https://copper-pdf.com/')

# 結合
session.join
```

実際に動作するサンプルはドライバの src/test/continuous.rb にあります。

3.7.6 ソースコード

ドライバのソースコードは GitHub で公開しています。
https://github.com/zamasoftnet/cti.ruby

3.8 Pythonドライバ

3.8.1 概要

Pythonドライバは、出力をPDFに変換することができます。

3.8.2 ドライバの準備

Python用ドライバ（Python 3対応）はGitHubの https://github.com/zamasoftnet/cti.python で公開されています。pipを使ってインストールすることができます。

例 3.66 pipでのインストール

```
pip install
"git+https://github.com/zamasoftnet/cti.python.git#subdirectory=python3"
```

インストール後は `from cti import get_session` でドライバを読み込んでください。

例 3.67 copperdへの接続

```
# ドライバの読み込み
from cti import get_session

# セッションの開始（with文を使用）
with get_session('ctip://localhost:8099/',{
    'user'  :'user',
    'password'  :'kappa'
}) as session:
    # 各種操作
    ...
```

3.8.3 APIの概要

ここではAPIによるアクセスの概要で説明した各手順に対応する関数を列挙します。各関数の詳細は オンラインのAPIドキュメントを参照してください。

サーバーへの接続・認証

- get_driver(uri)
- get_session(uri, options = {})

サーバー情報の取得

- Session#get_server_info(uri)

メッセージハンドラ・プログレスリスナの設定

- Session#set_message_func(&message_func)
- Session#set_progress_func(&progress_func)

出力先の設定

- Session#set_results(results)
- Session#set_output_as_stream(out)
- Session#set_output_as_file(file)
- Session#set_output_as_directory(dir, prefix = '', suffix = '')

プロパティの設定

- Session#property(name, value)

ソースリゾルバの設定

- Session#set_resolver_func(&resolver_func)

リソースの送信

- Session#resource(uri, opts = {},)

本体の送信・変換

- Session#transcode(uri = '.', opts = {})
- Session#transcode_server(uri)

複数の結果の結合

- Session#set_continuous(continuous)
- Session#join()

処理の中断・リセット・通信の終了

- Session#abort(mode)
- Session#reset()
- Session#close()

3.8.4 サンプル

以下は、プログラムによる出力を変換するサンプルです。sys.stdoutを途中でCopper PDFへの出力に切り替えています。

例 3.68 プログラムによる出力を変換する

```
# -*- coding: utf-8 -*-
import sys

import os
import os.path
import time
from cti import *

# セッションの開始
session = get_session('ctip://localhost:8099/',{
    'user'  :'user',
    'password'  :'kappa'
})
try:
    # ファイル出力
    dir = 'out';
    if not os.path.exists(dir):
        os.mkdir(dir)
    session.set_output_as_file('out/stdin.pdf')

    # 文書の送信
    sys.stdout = session.transcode()
    try:
        print """
<html>
  <head>
    <title>Python Test</title>
  </head>
  <body>
<h1>Hello Python</h1>
<p>只今の時刻は: %s</p>
  </body>
</html>""" % time.strftime("%Y/%m/%d %H:%M:%S")
    finally:
        sys.stdout.close()

    sys.stdout = sys.__stdout__
finally:
    session.close()
```

次の例は、サーバー側からネットワーク上のウェブページアクセスしてPDFに変換します。

例 3.69 サーバー側ウェブページにアクセスしてPDFに変換

```
# -*- coding: utf-8 -*-
import sys

import os
import os.path
from cti import *

# セッションの開始
session = get_session('ctip://localhost:8099/',{
    'user'  :'user',
    'password'  :'kappa'
})
try:
    # ファイル出力
    dir = 'out';
    if not os.path.exists(dir):
        os.mkdir(dir)
    session.set_output_as_file('out/server-resource.pdf')

    #リソースのアクセス許可
    session.property('input.include', 'https://copper-pdf.com/**')

    #文書の変換
    session.transcode_server('https://copper-pdf.com/');
finally:
    session.close()
```

他のサンプルはドライバのsrc/testに収められています。

3.8.5 プログラミングのポイント

Content-Type，Content-Length ヘッダの出力

ウェブアプリケーションでは、Content-Type，Content-Lengthヘッダを出力しないと、ブラウザ上でPDFが正常に表示されないことがあります。set_resultsやset_output_XXXメソッドを呼び出さない初期状態では、出力先は標準出力(sys.__stdout__)に設定されています。このとき、自動的にContent-Type，Content-Lengthヘッダが出力されます。

これはSingleResultとStreamBuilderのコンストラクタに渡すことができるブロックを利用しています。同じ状態にするためには、以下のようにSession#set_resultsメソッドを呼び出してください。

例 3.70 ヘッダの出力

```
def content_type(opts):
    print >> sys.__stdout__, "Content-Type: "+opts['mime_type']
def content_length(length):
    print >> sys.__stdout__, "Content-Length: "+str(length)
    print >> sys.__stdout__
results = SingleResult(StreamBuilder(sys.__stdout__,
content_length), content_type)
session.set_results(results)
```

繰り返し処理

出力先を変え、transcode または transcode_serverを繰り返し呼び出すことで、同じセッションで何度もドキュメントを変換することができます。送信済みのリソース、設定済みのプロパティは同じセッションで維持されます。同じセッションのまま初期状態に戻すには session.reset() を呼び出してください。

出力先の設定

set_output_as_stream, set_output_as_fileにより、単一のファイルハンドルかファイルを出力先として設定することができます。

複数の結果をファイルとして出力する場合は、set_output_as_directory を使用してください。このクラスは、指定したディレクトリに、1から開始する連番の前後に指定した文字列をくっつけたファイル名で結果を出力します。次の例では変換結果の各ページを、resultsディレクトリ内に"image[通し番号].jpeg"という名前で別々のJPEG画像として出力します。

例 3.71 ディレクトリに結果を出力する

```
session.property("output.type", "image/jpeg")
session.set_output_as_directory("results", "image", ".jpeg")
```

サーバーから要求されたリソースの送信

サーバーから要求されたリソースを送る場合、resolverを呼び出してください。このメソッドには、要求されたファイルのURIと、サーバーへのファイルハンドルを得るためのブロックを渡します 以下の例では、URIが相対パスであるという前提で、ローカルのファイルがあればサーバーに送信します。

例 3.72 サーバーから要求されたリソースをローカルファイルから送信する

```
def resolver(uri, r):
    if os.path.exists(uri):
        out = r.found()
        try:
            file = open(uri)
            try:
                out.write(file.read())
            finally:
                file.close()
        finally:
            out.close()
session.set_resolver_func(resolver)
```

実際に動作するサンプルはドライバの python3/src/test/ にあります。

複数の結果の結合

複数の結果を結合したものを得るためには、session.set_continuous(True) を呼び出した後、transcodeを複数回呼び出し、最後に session.join() を呼び出してください。

例 3.73 2つの結果の結合

```
session.set_continuous(True)

# 文書の送信
out = session.transcode()
try:
    file = open('data/test.html')
    try:
        out.write(file.read())
    finally:
        file.close()
finally:
    out.close()

#リソースのアクセス許可
session.property('input.include', 'https://copper-pdf.com/**')

#文書の変換
session.transcode_server('https://copper-pdf.com/')

# 結合
session.join()
```

実際に動作するサンプルはドライバの python3/src/test/ にあります。

3.8.6 ソースコード

ドライバのソースコードは GitHub で公開しています。
https://github.com/zamasoftnet/cti.python

3.9 Node.jsドライバ

3.9.1 概要

Node.jsドライバは、ストリーム(Writable)への出力をPDFに変換することができます。TypeScriptで実装されており、TypeScriptおよびJavaScript(CommonJS/ESM)から利用できます。

3.9.2 ドライバの準備

Node.js用ドライバはCopper PDF本体とは別に配布されています。https://github.com/zamasoftnet/cti.nodejs から取得してください。npmを使ってインストールするには、次のコマンドを実行してください。

例 3.74 インストール

```
npm install https://github.com/zamasoftnet/cti.nodejs.git
```

動作環境はNode.js 14以降です。インストール後、TypeScriptでは `import { get_session } from 'copper-cti'`、CommonJSでは `const { get_session } = require('copper-cti')` でドライバを読み込んでください。

例 3.75 copperdへの接続 (TypeScript/ESM)

```
import { get_session } from 'copper-cti';

// セッションの開始
const session = get_session('ctip://localhost:8099/', {
    user: 'user',
    password: 'kappa'
});
try {
    // 各種操作
    ...
    await session.waitForCompletion();
} finally {
    session.close();
}
```

例 3.76 copperdへの接続 (JavaScript/CommonJS)

```
const { get_session } = require('copper-cti');

// セッションの開始
const session = get_session('ctip://localhost:8099/', {
    user: 'user',
    password: 'kappa'
});
try {
    // 各種操作
    ...
    await session.waitForCompletion();
} finally {
    session.close();
}
```

3.9.3 APIの概要

ここではAPIによるアクセスの概要で説明した各手順に対応する関数を列挙します。各関数の詳細はオンラインのAPIドキュメントを参照してください。

サーバーへの接続・認証

- get_driver()
- get_session(uri, options?)

サーバー情報の取得

- Session#getServerInfo(uri)

メッセージハンドラ・プログレスリスナの設定

- Session#setMessageFunc(callback)
- Session#setProgressFunc(callback)

出力先の設定

- Session#setResults(results)
- Session#setOutputAsStream(stream)
- Session#setOutputAsFile(file)
- Session#setOutputAsDirectory(dir, prefix?, suffix?)

プロパティの設定

- Session#setProperty(name, value)

ソースリゾルバの設定

- Session#setResolverFunc(callback)

リソースの送信

- Session#resource(uri, opts?)

本体の送信・変換

- Session#transcode(uri?, opts?)
- Session#transcodeServer(uri)
- Session#waitForCompletion()

複数の結果の結合

- Session#setContinuous(continuous)
- Session#join()

処理の中断・リセット・通信の終了

- Session#abort(mode)
- Session#reset()
- Session#close()

3.9.4 サンプル

以下は、プログラムによる出力を変換するサンプルです。`session.transcode()`が返すWritableストリームにHTMLを書き込んでいます。

例 3.77 プログラムによる出力を変換する

```
import { get_session } from 'copper-cti';
import * as fs from 'fs';
import * as path from 'path';

async function main() {
    // セッションの開始
    const session = get_session('ctip://localhost:8099/', {
        user: 'user',
        password: 'kappa'
    });
    try {
        // ファイル出力
        const outDir = 'out';
        if (!fs.existsSync(outDir)) {
```

```
            fs.mkdirSync(outDir, { recursive: true });
        }
        session.setOutputAsFile('out/hello.pdf');

        // 変換中のメッセージを表示
        session.setMessageFunc((code, msg, args) => {
            console.log(`Message [${code}]: ${msg}`);
        });

        // 文書の送信
        const writer = session.transcode('.');
        writer.write(`<html>
  <head><title>Node.js Test</title></head>
  <body>
    <h1>Hello Node.js</h1>
    <p>只今の時刻は: ${new Date().toLocaleString('ja-JP')}</p>
  </body>
</html>`);
        writer.end();

        // 変換完了を待機
        await session.waitForCompletion();
        console.log('PDF generated: out/hello.pdf');
    } finally {
        session.close();
    }
}

main();
```

次の例は、ファイルをストリームとして読み込み変換します。

例 3.78 HTMLファイルをPDFに変換する

```
import { get_session } from 'copper-cti';
import * as fs from 'fs';

async function main() {
    const session = get_session('ctip://localhost:8099/', {
        user: 'user',
        password: 'kappa'
    });
    try {
        session.setOutputAsFile('out/file.pdf');

        // HTMLファイルをストリームとして流し込む
        const writer = session.transcode('.');
```

```
        fs.createReadStream('index.html').pipe(writer);

        await session.waitForCompletion();
        console.log('PDF generated: out/file.pdf');
    } finally {
        session.close();
    }
}

main();
```

次の例は、サーバー側からネットワーク上のウェブページにアクセスしてPDFに変換します。

例 3.79 サーバー側ウェブページにアクセスしてPDFに変換

```
import { get_session } from 'copper-cti';
import * as fs from 'fs';

async function main() {
    const session = get_session('ctip://localhost:8099/', {
        user: 'user',
        password: 'kappa'
    });
    try {
        const outDir = 'out';
        if (!fs.existsSync(outDir)) {
            fs.mkdirSync(outDir, { recursive: true });
        }
        session.setOutputAsFile('out/server-resource.pdf');

        // リソースのアクセス許可
        session.setProperty('input.include', 'https://copper-
pdf.com/**');

        // 文書の変換
        session.transcodeServer('https://copper-pdf.com/');

        await session.waitForCompletion();
        console.log('PDF generated: out/server-resource.pdf');
    } finally {
        session.close();
    }
}

main();
```

他のサンプルはドライバの examples ディレクトリに収められています。

3.9.5 プログラミングのポイント

async/awaitによる完了待機

Node.jsドライバでは、変換処理は非同期で行われます。session.transcode()でストリームへの書き込みを開始し、書き込みが完了したらawait session.waitForCompletion()で変換完了を待機してください。waitForCompletion()を呼び出さないと、変換が完了する前に処理が終了してしまいます。

繰り返し処理

出力先を変え、transcode()またはtranscodeServer()を繰り返し呼び出すことで、同じセッションで何度もドキュメントを変換することができます。送信済みのリソース、設定済みのプロパティは同じセッションで維持されます。同じセッションのまま初期状態に戻すにはsession.reset()を呼び出してください。

出力先の設定

setOutputAsStream(), setOutputAsFile()により、単一の書き込みストリームかファイルを出力先として設定することができます。

複数の結果をファイルとして出力する場合は、setOutputAsDirectory()を使用してください。このメソッドは、指定したディレクトリに、1から開始する連番の前後に指定した文字列をくっつけたファイル名で結果を出力します。次の例では変換結果の各ページを、resultsディレクトリ内に"image[通し番号].jpeg"という名前で別々のJPEG画像として出力します。

例 3.80 ディレクトリに結果を出力する

```
session.setProperty('output.type', 'image/jpeg');
session.setOutputAsDirectory('results', 'image', '.jpeg');
```

サーバーから要求されたリソースの送信

サーバーから要求されたリソースを送る場合、setResolverFunc()でコールバック関数を設定してください。コールバックには要求されたリソースのURIとResourceオブジェクトが渡されます。リソースが存在する場合はresource.found()を呼び出して取得した書き込みストリームにデータを流し込んでください。リソースが存在しない場合はresource.found()を呼び出さないでください。

例 3.81 サーバーから要求されたリソースをローカルファイルから送信する

```
import { get_session, Resource } from 'copper-cti';
import * as fs from 'fs';
import * as path from 'path';

session.setResolverFunc((uri: string, resource: Resource) => {
    const localPath = path.join(__dirname, uri);
    if (fs.existsSync(localPath)) {
        const out = resource.found({ mime_type: 'text/css' });
        fs.createReadStream(localPath).pipe(out);
    }
    // リソースが見つからない場合は resource.found() を呼ばない
});
```

複数の結果の結合

複数の結果を結合したものを得るためには、**session.setContinuous(true)** を呼び出した後、**transcode()**を複数回呼び出し、最後に **session.join()** を呼び出してください。

例 3.82 2つの結果の結合

```
import { get_session } from 'copper-cti';
import * as fs from 'fs';

async function main() {
    const session = get_session('ctip://localhost:8099/', {
        user: 'user',
        password: 'kappa'
    });
    try {
        session.setOutputAsFile('out/combined.pdf');
        session.setContinuous(true);

        // 1つ目の文書を送信
        const writer1 = session.transcode('.');
        fs.createReadStream('data/doc1.html').pipe(writer1);
        await session.waitForCompletion();

        // 2つ目の文書を送信
        const writer2 = session.transcode('.');
        fs.createReadStream('data/doc2.html').pipe(writer2);
```

```
        await session.waitForCompletion();

        // 結合
        session.join();
    } finally {
        session.close();
    }
}

main();
```

3.9.6 ソースコード

ドライバのソースコードは GitHub に公開しています。
https://github.com/zamasoftnet/cti.nodejs

3.10 transcode Antタスク

3.10.1 Antタスクの概要

頻繁に更新されるドキュメントを素早くまとめて変換するために、Apache Antによるバッチ処理をサポートしています。AntはJavaで開発されたフリーのビルド・ツールです。Antによるバッチ処理は、ソースが変更されたファイルだけを変換するため、効率的です。Antについての詳細は書籍などをご参照下さい。

transcode Antタスクは、Copper PDF本体と、Java版CTIP 2.0ドライバに含まれています。Copper PDF本体のライブラリを使用する場合は、直接ローカルマシンのライブラリを使用する方法と、CTIP 2.0またはHTTP/RESTプロトコルで接続する方法を使うことができます。CTIP 2.0ドライバを使用する場合は、後者の方法だけです。

3.10.2 transcode タスクの使用方法

ローカルマシン上のCopper PDFを直接利用して、transcode タスクを使用するためには、Antのbuild.xml中で次のように宣言する必要があります。

例 3.83 transcode タスクの宣言（ローカルマシン）

```
<path id="lib.path">
  <fileset dir="
            Copper PDFのlibディレクトリのパス
            " includes="*.jar"/>
</path>
<typedef classpathref="lib.path"
resource="jp/cssj/driver/ant/tasks.properties"/>
```

Java版のCTIP 2.0ライブラリを使う場合は、同様に次のように宣言します。

例 3.84 transcode タスクの宣言（CTIP 2.0ドライバ）

```
<typedef classpath="cti-driver.jarへのパス"
resource="jp/cssj/driver/ant/tasks.properties"/>
```

以降、transcodeという要素名でタスクを使えるようになります。

transcodeタスクに指定することができる属性は次の通りです。

表 3.1 transcode タスクの属性

属性名	説明
srcDir	変換前のXML,HTMLファイルなどが格納されたディレクトリです。省略した場合、カレントディレクトリとなります。

属性名	説明
includes	srcDir中の変換対象となるファイルのパターンです。
excludes	srcDir中の変換対象外となるファイルのパターンです。
destDir	出力先のディレクトリです。省略するとsrcDirと同じディレクトリになります。
suffix	出力結果ファイルの拡張子です。省略した場合は.pdfとなります。

接続先はtranscode要素内で、connection要素を使って設定します。ただし、ローカルマシンに接続する場合は設定は不要です。

表 3.2 connection要素の属性

属性名	説明
uri	接続先URI。
user	ユーザーID。
password	パスワード。

transcode要素内で、property要素を使って入出力プロパティを設定することができます。

表 3.3 property要素の属性

属性名	説明
name	プロパティの名前。
value	プロパティの値。

以下の例では、ローカルホストで動作しているCopper PDFを使って、docs/manual.htmlからdocs/manual.pdfを出力します。

例 3.85 transcode タスクの使用例

```
<transcode includes="docs/manual.html" suffix=".pdf">
    <connection uri="ctip://localhost:8099/" user="user"
password="kappa" />
    <property name="input.include" value="**" />
    <property name="output.pdf.bookmarks" value="true" />
    <property name="output.pdf.hyperlinks" value="true" />
    <property name="output.pdf.fonts.policy" value="cid-keyed" />
</transcode>
```

transcode Antタスクはhttp/restによる接続にも対応しています。HTTPを使う場合は "http://127.0.0.1:8097/" のようにURIを設定してください。

3.10.3 うまく動かない場合（ローカルマシンで実行する場合）

ライセンスが認証されない場合

出力結果に「ライセンスファイルが存在しないか期限切れです」と表示される場合や、一部の機能が使用できない場合は、ライセンスキーのインストールか、confディレクトリ(license-keyファイルが置かれたディレクトリ)のパスを指定する必要があります。

confディレクトリはcopperタスクのconfigDir属性で指定するか、ANT_OPTS環境変数によりjp.cssj.copper.configシステムプロパティで指定することもできます。

例 3.86 設定ディレクトリの指定(ANT_OPTS環境変数)

```
export ANT_OPTS=-Djp.cssj.copper.config=/etc/copper-pdf
```

例 3.87 設定ディレクトリの指定(configDir属性)

```
<copper includes="docs/manual.html" suffix=".pdf"
    configFile="conf/profiles/default.properties"
    configDir="/etc/copper-pdf">
    <includeresource pattern="**" />
    <property name="output.pdf.bookmarks" value="true" />
    <property name="output.pdf.hyperlinks" value="true" />
    <property name="output.pdf.fonts.policy" value="cid-keyed" />
</copper>
```

3.11 HTTP/RESTインターフェース

3.11.1 概要

HTTP/RESTインターフェースはCopper PDF 2.1.0からサポートされた、HTTPベースのインターフェースです。高速・高機能なCTIPに比べて冗長なプロトコルですが、HTTPをベースとしているため、普通のウェブブラウザやHTTPクライアントライブラリを利用できる利点があります。

HTTP通信による処理速度の損失は、変換処理の開始・終了時に発生するものなので、数十ページ以上の文書の出力ではほとんど問題になりません。数ページ程度の文書を繰り返し出力する場合は、処理速度はおおむね70%程度に低下します。

HTTP/RESTインターフェースはCTIP 2.0と同等の機能を備えており、実際にJava版のCTIP 2.0ドライバは、CTIP, HTTP接続の両方で、同一のインターフェースを利用することができます。

このドキュメントでは、HTTP/RESTインターフェースの基本的な使用方法だけを解説します。HTTP/RESTインターフェースの完全な仕様は、以下のアドレスで公開している仕様書を参照してください。
https://htmlpreview.github.io/?https://github.com/zamasoftnet/cti.java/blob/main/docs/cti-rest-v1.xhtml

3.11.2 アクションの実行

HTTP/RESTインターフェースにより、Copper PDFに対して何らかの動作(アクション)の実行を要求するためには、HTTPのGETまたはPOSTメソッドでクライアントからアクセスします。アクションの種類は、ファイルパスにより識別されます。処理対象文書や認証情報など、必要な情報はリクエストパラメータとして送ります。アクションとファイルパスの対応と、使用可能なリクエストパラメータの表は以下の通りです。

パス	リクエストパラメータ	説明
/open	rest.user rest.password rest.timeout rest.httpSession	セッションの開始
/info	rest.id rest.user rest.password rest.uri	サーバーの情報(セッションの開始と終了を兼ねる)
/properties	rest.id "rest." で始まらないパラメータ	プロパティ設定

パス	リクエストパラメータ	説明
/resources	rest.id rest.uri rest.mimeType rest.encoding rest.resource rest.notFound "rest."で始まらないパラメータ	リソース送信(プロパティ設定を兼ねる)
/transcode	rest.id rest.user rest.password rest.async rest.requestResource "rest."で始まらないパラメータ rest.uri rest.mimeType rest.encoding rest.resource rest.main rest.mainURI	ドキュメント変換処理(セッションの開始と終了、プロパティ設定、リソース送信を兼ねる)
/messages	rest.id rest.wait	メッセージ受信
/result	rest.id rest.uri	ドキュメント変換結果受信
/abort	rest.id rest.mode	ドキュメント変換処理の中断
/reset	rest.id	リセット
/close	rest.id	セッションの終了

例えばlocalhostの8097ポートでドキュメント変換サーバーが動作している場合、変換処理を実行するには、`http://localhost:8097/transcode` にアクセスします。

HTTP/RESTインターフェースの機能をフルに活用する場合は、openによりセッションを開始し、各種操作をした後、closeによりセッションを終了することが基本となります。ただし、transcodeアクションは簡単な変換処理であれば1回のリクエストだけで完結できるようになっています。

リクエストパラメータは、クエリパラメータで送る方法、POSTメソッドのフォームパラメータとして送る方法(application/x-www-form-urlencoded)、そしてmultipart/form-data形

式のファイルとフォームフィールドとして送る方法があります (さらに、ファイルそのものをリクエストボディとして送る方法があります。詳細は仕様書を参照してください)。

application/x-www-form-urlencoded を使用する場合、送信できるデータの最大サイズに制限があります（190KB程度）。そのため、アプリケーション側から変換対象のドキュメントをCopper PDFに送る場合は、なるべくmultipart/form-dataの使用を推奨します。

リクエストパラメータの意味は次の表の通りです。

パラメータ名	説明
rest.user	認証のためのユーザー名です。
rest.password	認証のためのパスワードです。
rest.timeout	セッションの最小持続時間(ミリ秒)です。
rest.httpSession	trueを設定すると、クッキーによるHTTPセッション使用します。
rest.id	セッションを識別するIDです。クッキーによるHTTPセッション使用する場合は不要です。
rest.uri	infoアクションではサーバー情報の識別URI、resultアクションでは結果の識別URI、他のアクションでは次に送るデータの仮想URIを表します。
rest.mimeType	次に送るデータのMIME型です。
rest.encoding	次に送るデータのキャラクタ・エンコーディングです。
rest.resource	リソースデータです。これはmaltipart/form-dataのファイルとして送ることもできます。
rest.notFound	リソースデータの代わりに、このパラメータにtrueを設定すると、リソースが存在しないことを示します。
rest.async	transcodeアクションは通常、変換結果をレスポンスとして返しますが、このパラメータにtrueを設定すると変換結果をresultアクションにより得ることを前提に、非同期的にドキュメント変換処理を実行します。
rest.requestResource	trueを設定すると、サーバーが必要なリソースをクライアントに要求するようになります。クライアントはリソースの送信要求をmessagesアクションで確認する必要があります。
rest.main	変換対象のメインドキュメントです。これはmaltipart/form-dataのファイルとして送ることもできます。
rest.mainURI	サーバーに変換対象のメインドキュメントを取得させる場合のURIです。
rest.wait	メッセージの変化があるまでmessagesアクションのレスポンスを保留する場合の最大待ち時間(ミリ秒)です。

パラメータ名	説明
rest.mode	1であればなるべく有効なデータを出力して処理を中断し、2であればなるべく直ちに処理を中断します。

なお、各アクションのレスポンスの詳細は仕様書を参照してください。

3.11.3 ウェブブラウザからのアクセス

ウェブブラウザからデータの変換を行う場合、最も簡単な方法は以下のフォームをHTMLファイルとして保存して **(キャラクタ・エンコーディングはUTF-8にしてください)**、ブラウザで表示し、「変換」ボタンを押すことです。テキストエリアの内容(Hello world!)がPDF化されます。

例 3.88 フォームからCopper PDFを使う

```
<form action="http://localhost:8097/transcode">
  <input type="hidden" name="rest.user" value="user"/>
  <input type="hidden" name="rest.password" value="kappa"/>
  <textarea name="rest.main">
<html><body>Hello world!</body></html>
  </textarea>
  <button type="submit">変換</button>
</form>
```

アップロードしたファイルを変換することもできます。

例 3.89 アップロードしたファイルを変換

```
<form action="http://localhost:8097/transcode" method="post"
      enctype="multipart/form-data">
  <input type="hidden" name="rest.user" value="user"/>
  <input type="hidden" name="rest.password" value="kappa"/>
  <input type="file" name="rest.main" />
  <button type="submit">変換</button>
</form>
```

3.11.4 Ruby (httpclient)

※新しく開発されたRubyインターフェースか、実行環境としてJRubyを使う場合はJavaインターフェースを利用した方がより高速です。

Rubyではhttpclientモジュールを使うと簡単です。以下のサンプルでは、ヒアドキュメントとして記述したHTMLを変換します。

例 3.90 Rubyでドキュメントを変換

```
require 'httpclient'

# 変換対象のHTML
data = <<DATA
<html>
<body>
Rubyからを使う。
</body>
</html>
DATA

# POSTの準備
client = HTTPClient.new
postdata = {
  "rest.user" => "user",
  "rest.password" => "kappa",
  "rest.main" => data,
}

# 大きなデータを扱えるようにmultipart/formdataで送信(boundaryは適当な文字
列)
boundary = "3w48588hfwfdwed2332hdiuj2d3jiuhd32"
puts client.post_content("http://localhost:8097/transcode",
postdata,
"content-type" => "multipart/form-data, boundary=#{boundary}")
```

以下の例では https://copper-pdf.com/ をPDF化します。前の例では大きなデータをPOSTするため multipart/form-data を使用しましたが、POSTするデータが小さい場合は普通のPOST(application/x-www-form-urlencoded)で構いません。パラメータの input.include画像等の取得のためにアクセスを許可するアドレスのパターンです。

例 3.91 Rubyでウェブサイトを変換

```
require 'httpclient'

# POSTの準備
client = HTTPClient.new
postdata = {
  "rest.user" => "user",
  "rest.password" => "kappa",
  "input.include" => "https://copper-pdf.com/**",
  "rest.mainURI" => "https://copper-pdf.com/",
}

# POSTを実行
puts client.post_content("http://localhost:8097/transcode",
postdata)
```

3.11.5 Python (urllib2, urllib)

※新しく開発されたPythonインターフェースか、実行環境としてJythonを使う場合はJavaインターフェースを利用した方がより高速です。

Pythonではurllib2モジュールを使うことができます。以下のサンプルでは、三重クォートで囲った文字列リテラルとして記述したHTMLを変換します。

例 3.92 Pythonでドキュメントを変換

```
# -*- coding: utf_8 -*-
import urllib2

# multipart/form-dataの出力(boundaryは適当な文字列)
boundary = '3w48588hfwfdwed2332hdiuj2d3jiuhd32'
def multipart_formdata(form_dict):
    disposition = 'Content-Disposition: form-data; name="%s"'
    lines = []
    for k, v in form_dict.iteritems():
        lines.append('--' + boundary)
        lines.append(disposition % k)
        lines.append('')
        lines.append(v)
    lines.append("--" + boundary + "--")
    lines.append('')
    value = "\r\n".join(lines)
    return value

# 変換対象のHTML
data = """
<html>
<body>
Pythonからの Copper PDFを使う。
</body>
</html>
"""

# POSTの実行
params = {'rest.user':'user',
    'rest.password':'kappa',
    'rest.main':data}
```

```
url = 'http://localhost:8097/transcode'
req = urllib2.Request(url)
req.add_header("Content-Type",
               "multipart/form-data; boundary=" + boundary)
data = multipart_formdata(params)
f = urllib2.urlopen(req, data)

# 結果表示
print f.read()
```

以下の例では https://copper-pdf.com をPDF化します。前の例では大きなデータをPOSTするため multipart/form-data を使用しましたが、POSTするデータが小さい場合は普通のPOST(application/x-www-form-urlencoded)で構いません。こちらはurllibを使っています。パラメータの input.include画像等の取得のためにアクセスを許可するアドレスのパターンです。

例 3.93 Pythonでウェブサイトを変換

```
# -*- coding: utf_8 -*-
import urllib

url = 'http://localhost:8097/transcode'
params = urllib.urlencode({'rest.user':'user',
                           'rest.password':'kappa',
                           'input.include':'https://copper-
pdf.com**',
                           'rest.mainURI':'https://copper-pdf.com',
                           })
f = urllib.urlopen(url, params)
print f.read()
```

3.11.6 C# (.NET WebClient)

※C# / VB.NETではRESTインターフェースを使用する必要はなくなりました。より高速で高機能な.NETドライバの使用を推奨します。

C#では.NETのWebClientを使うことができます。以下のサンプルでは、ヒアドキュメントとして記述したHTMLを変換します。

例 3.94 C#でドキュメントを変換

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Collections.Specialized;

namespace cti.net
{
    class Program
    {
        static void Main(string[] args)
        {
            string data = @"
<html>
<body>
C#からCopper PDFを使う。
</body>
</html>
            ";

            WebClient client = new WebClient();
            NameValueCollection par = new NameValueCollection();
            par.Add("rest.user", "user");
            par.Add("rest.password", "kappa");
            par.Add("rest.main", data);
            byte[] res = client.UploadValues
("http://localhost:8097/transcode", par);
            Console.OpenStandardOutput().Write(res, 0, res.Length);
            client.Dispose();
        }
    }
}
```

以下の例では https://copper-pdf.com をPDF化します。パラメータの input.include 画像等の取得のためにアクセスを許可するアドレスのパターンです。

この例では、変換結果をストリームに逐次送り出すため、メモリを節約することができ、巨大なPDFの出力にも対応できます。

例 3.95 C#でウェブサイトを変換

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Collections.Specialized;

namespace cti.net
{
    class ServerDocument
    {
        static void Main(string[] args)
        {
            Response.ClearContent();
            Response.ContentType = "application/pdf";
            WebClient wc = new WebClient();
            String uri = "http://localhost:8097/transcode" + // ローカルマシンのCopper PDF
                "?rest.user=user" +
                "&rest.password=kappa" +
                "&input.include=https://copper-pdf.com/**" + // 画像等へのアクセス許可
                "&rest.mainURI=https://copper-pdf.com/"; // 変換対象アドレス

            Stream input = wc.OpenRead(uri);
            Stream output = Console.OpenStandardOutput();
            byte[] buff = new byte[4096];
            int len;
            while ((len = input.Read(buff, 0, buff.Length)) > 0)
                output.Write(buff, 0, len);

            input.Close();
            wc.Dispose();
        }
    }
}
```

3.11.7 ASP.NET (C#, VisualBasic)

※C# / VB.NETではRESTインターフェースを使用する必要はなくなりました。より高速で高機能な.NETドライバの使用を推奨します。

以下の例では https://copper-pdf.com/ をPDF化します。それぞれC#とVisualBasicのプログラム例です。パラメータの input.include 画像等の取得のためにアクセスを許可するアドレスのパターンです。

例 3.96 ASP.NET(C#) でウェブサイトを変換

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.IO" %>
<%
    Response.ClearContent();
    Response.ContentType = "application/pdf";

    // ファイル名
    String filename = "FILENAME.pdf";
    Response.AddHeader("Content-Disposition", "attachment;
filename=" + filename);

    Response.ContentType = "application/pdf";
    WebClient wc = new WebClient();
    String uri = "http://localhost:8097/transcode" + // ローカルマシ
ンのCopper PDF
        "?rest.user=user" +
        "&rest.password=kappa" +
        "&input.include=https://copper-pdf.com/**" + // 画像等へのア
クセス許可
        "&rest.mainURI=https://copper-pdf.com/"; // 変換対象アドレス

    Stream input = wc.OpenRead(uri);
    Stream output = Response.OutputStream;
    byte[] buff = new byte[4096];
    int len;
    while ((len = input.Read(buff, 0, buff.Length)) > 0)
        output.Write(buff, 0, len);

    input.Close();
    wc.Dispose();
%>
```

例 3.97 ASP.NET(VisualBasic) でウェブサイトを変換

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.IO" %>
<%
    Response.ClearContent()
    Response.ContentType = "application/pdf"

    ' ファイル名
    Dim filename As String = "FILENAME.pdf"
    Response.AddHeader("Content-Disposition", "attachment;
filename=" + filename)

    Response.ContentType = "application/pdf"
    Dim wc As WebClient = New WebClient()
```

```
    ' ローカルマシンのCopper PDF
    Dim uri As String = "http://localhost:8097/transcode"
    uri &= "?rest.user=user"
    uri &= "&rest.password=kappa"
    ' 画像等へのアクセス許可
    uri &= "&input.include=hhttps://copper-pdf.com/**"
    ' 変換対象アドレス
    uri &= "&rest.mainURI=https://copper-pdf.com/"

    Dim input As Stream = wc.OpenRead(uri)
    Dim output As Stream = Response.OutputStream
    Dim buff(4096) As Byte
    Dim len As Integer
    len = input.Read(buff, 0, buff.Length)
    While (len > 0)
        output.Write(buff, 0, len)
        len = input.Read(buff, 0, buff.Length)
    End While
    input.Close()
    wc.Dispose()
%>
```

3.11.8 4th Dimension (Internet Commands)

4th Dimension(4D)ではInternet CommandsのTCP/IPコマンドを使うことができます。以下のサンプルでは、BLOBに格納したHTMLを変換します。

例 3.98 4Dでドキュメントを変換

```
C_TEXT
($host;$user;$password;$boundary;$sourceType;$source;$beforeContent;$afterCon

C_LONGINT($port;$length;$err;$status;$socket;$pos)
C_BLOB($content;$buffer;$data)
C_TIME($doc)

  `Copper PDFサーバー
$host:="localhost"
$port:=8097
$user:="user"
$password:="kappa"

  `変換対象HTML
$sourceType:="text/html"
$source:="<html><head><title>サンプル</title></head><body>4Dから
Copper PDFを使う。</body></html>"
TEXT TO BLOB($source;$content)  `BLOBに変換

  `結果ファイル名
$resultFile:="Result.pdf"
```

```
  `通信開始
$err:=TCP_Open ($host;$port;$socket)
$err:=TCP_State ($socket;$status)

  `リクエスト送信
$boundary:=String(Random)+String(Random)+String(Random)  `マルチパートの境界(ランダムな文字列)
$beforeContent:="---"+$boundary+"\r\n"
$beforeContent:=$beforeContent+"Content-Disposition: form-data;
name=\"rest.main\"; filename=\"data.html\"\r\n"
$beforeContent:=$beforeContent+"Content-Type:
"+$sourceType+"\r\n\r\n"
$afterContent:="\r\n---"+$boundary+"--\r\n"
$length:=Length($beforeContent)+BLOB size($content)+Length
($afterContent)

$err:=TCP_Send ($socket;"POST /transcode?
rest.user="+$user+"&rest.password="+$password+" HTTP/1.0\r\n")
`keep-aliveとchunkedを使わないためHTTP/1.0で接続する
$err:=TCP_Send ($socket;"Host: "+$host+"\r\n")
$err:=TCP_Send ($socket;"Content-Length: "+String($length)+"\r\n")
$err:=TCP_Send ($socket;"Content-Type: multipart/form-data;
boundary=-"+$boundary+"\r\n")
$err:=TCP_Send ($socket;"\r\n")

$err:=TCP_Send ($socket;$beforeContent)
$err:=TCP_SendBLOB ($socket;$content)
$err:=TCP_Send ($socket;$afterContent)

  `レスポンス受信
Repeat
    SET BLOB SIZE($buffer;0)
    $err:=TCP_ReceiveBLOB ($socket;$buffer)
    $err:=TCP_State ($socket;$status)
    $bufferLen:=BLOB size($buffer)
    $dataLen:=BLOB size($data)
    INSERT IN BLOB($data;$dataLen;$bufferLen)
    COPY BLOB($buffer;$data;0;$dataLen;$bufferLen)
Until (($status=0) | ($err#0))

$err:=TCP_State ($socket;$state)
$err:=TCP_Close ($socket)
```

```
  `ヘッダの除去
$text:=BLOB to text($data;Mac C string )
$pos:=Position("\r\n\r\n";$text)
DELETE FROM BLOB($data;0;$pos+3)  `先頭から最初の空行までの間を除去する

  `ファイルの出力
$doc:=Create document($resultFile)
CLOSE DOCUMENT($doc)
BLOB TO DOCUMENT($resultFile;$data)
```

以下の例では http://print.cssj.jp/ をPDF化します。パラメータの input.include 画像等の取得のためにアクセスを許可するアドレスのパターンです。

例 3.99 4Dでウェブサイトを変換

```
C_TEXT
($host;$user;$password;$include;$uri;$query;$resultFile;$text)
C_LONGINT($port;$err;$status;$socket;$pos)
C_BLOB($content;$buffer;$data)
C_TIME($doc)

  `Copper PDFサーバー
$host:="neko"
$port:=8097
$user:="user"
$password:="localhost"

  `変換対象アドレス
$include:="hhttps://copper-pdf.com/**"
$uri:="https://copper-pdf.com/"

  `結果ファイル名
$resultFile:="Result.pdf"

  `通信開始
$err:=TCP_Open ($host;$port;$socket)
$err:=TCP_State ($socket;$status)

  `リクエスト送信
$query:="rest.user="+$user+"&rest.password="+$password+
"&input.include="+$include+"&rest.mainURI="+$uri
$err:=TCP_Send ($socket;"GET /transcode?"+$query+" HTTP/1.0\r\n")
`keep-aliveとchunkedを使わないためHTTP/1.0で接続する
$err:=TCP_Send ($socket;"Host: "+$host+"\r\n")
$err:=TCP_Send ($socket;"\r\n")

$err:=TCP_Send ($socket;$beforeContent)
$err:=TCP_SendBLOB ($socket;$content)
$err:=TCP_Send ($socket;$afterContent)
```

```
  `レスポンス受信
Repeat
    SET BLOB SIZE($buffer;0)
    $err:=TCP_ReceiveBLOB ($socket;$buffer)
    $err:=TCP_State ($socket;$status)
    $bufferLen:=BLOB size($buffer)
    $dataLen:=BLOB size($data)
    INSERT IN BLOB($data;$dataLen;$bufferLen)
    COPY BLOB($buffer;$data;0;$dataLen;$bufferLen)
Until (($status=0) | ($err#0))

$err:=TCP_State ($socket;$state)
$err:=TCP_Close ($socket)

  `ヘッダの除去
$text:=BLOB to text($data;Mac C string )
$pos:=Position("\r\n\r\n";$text)
DELETE FROM BLOB($data;0;$pos+3)  `先頭から最初の空行までの間を除去する

  `ファイルの出力
$doc:=Create document($resultFile)
CLOSE DOCUMENT($doc)
BLOB TO DOCUMENT($resultFile;$data)
```

3.11.9 その他のサンプルについて

HTTP/RESTインターフェースは、さらに多くの種類の開発環境から、様々な方法で利用することできます。最新情報はCopper PDFのサイト(https://copper-pdf.com/)で提供しています。

3.12 HTTPクライアント機能

Copper PDFには、ウェブサーバーから画像、スタイルシート、文書等を取得するためのHTTPクライアントが入っています。公開されているウェブコンテンツにアクセスする場合は、特に設定は必要ありませんが、プロクシを通してのアクセスや、認証(BASICまたはDigest)が必要なウェブサイトへのアクセスもサポートしています。

以下の説明では、入出力プロパティの設定例をJavaで記述しています。他の言語で実装する場合は、各プログラミング言語のプロパティ設定関数に書き換えてください。

3.12.1 BASIC認証またはDigest認証

BASIC認証が必要なウェブサーバーに接続する場合、

- `input.http.authentication.n.host` [io]
- `input.http.authentication.n.port` [io]
- `input.http.authentication.n.user` [io]
- `input.http.authentication.n.password` [io]

にそれぞれ対象のウェブサーバーのホスト名またはIPアドレス、ポート番号、ユーザー名、パスワードを設定します。ポート番号を省略した場合は、ウェブサーバーのポート番号は任意となります。パスワードを省略した場合は、空のパスワードが使われます。ホスト名とユーザー名を省略することはできません。*n*は0から始まる整数で、連番にすることで、複数のサイトやレルム(認証領域)に対応することができます。

サーバーに複数のレルムが存在する場合や、Digest認証を行う場合は、実際の認証を行う前に、サーバーから認証情報を取得する必要があります。

`input.http.authentication.preemptive` [io] にtrueを設定することで、サーバーから認証情報を取得できるようになります。レルムを明示する場合は、`input.http.n.authentication.realm` [io] にレルム名を設定します。

`input.http.n.authentication.schema` [io] に BASIC 認証(basic)か、Digest 認証(digest)を設定することで、認証方法を限定することができます。

デフォルトの設定で、`input.http.authentication.preemptive` [io] がtrue の場合は、認証方法が自動判別されます。

以下の例ではwww.foo.com とwww.bar.com にそれぞれ別のユーザーアカウントで接続し、BASIC認証かDigest認証かを自動判別します。

例 3.100 認証の設定

```
session.property("input.http.authentication.preemptive", "true");
session.property("input.http.0.authentication.host",
"www.foo.com");
session.property("input.http.0.authentication.user", "foouser");
session.property("input.http.0.authentication.password",
"foopass");
session.property("input.http.1.authentication.host",
"www.bar.com");
session.property("input.http.1.authentication.user", "baruser");
session.property("input.http.1.authentication.password",
"barpass");
```

3.12.2 プロクシの設定

ウェブブラウザ等と同様に、HTTP接続のためのプロクシを設定することができます。

`input.http.proxy.host`[io]に、プロクシ・サーバーのホスト名またはIPアドレスを設定することにより、プロクシを通して接続するようになります。プロクシ・サーバーのデフォルトのポート番号は8080ですが、`input.http.proxy.port`[io]により、任意のポート番号を設定することができます。

認証が必要なプロクシ・サーバーを使用する場合、

- `input.http.proxy.authentication.user`[io]
- `input.http.proxy.authentication.password`[io]

にそれぞれユーザー名とパスワードを設定してください。

次の例では、認証が必要なプロクシ・サーバー proxy.foo.com に、"mei", "pass" というユーザ名とパスワードで接続します。

例 3.101 プロクシサーバーの設定

```
session.property("input.http.proxy.host", "proxy.foo.com");
session.property("input.http.proxy.authentication.user", "mei");
session.property("input.http.proxy.authentication.password",
"pass");
```

3.12.3 HTTPヘッダの送信

2つで1組の入出力プロパティ、

- `input.http.header.`*`n`*`.name` [io]
- `input.http.header.`*`n`*`.value` [io]

でHTTPのヘッダを設定することができます。*n*は0から始まる整数で、連番にすることで、複数のHTTPヘッダを送ることができます。

次の例では、クライアントの使用言語を韓国語、ブラウザの種類をInternet Explorer7であるとウェブサーバーに申告するようにHTTPヘッダを設定しています。

例 3.102 HTTPヘッダの設定

```
session.property("input.http.header.0.name", "Accept-Language");
session.property("input.http.header.0.value", "ko");
session.property("input.http.header.1.name", "User-Agent");
session.property("input.http.header.1.value", "Mozilla/4.0
(compatible; MSIE 7.0; Windows NT 5.1)");
```

3.12.4 参照元(Referer)の送信

デフォルトでは、変換対象の文書のURIがRefererヘッダの値として送信されます。この機能は、`input.http.referer` [io]をfalseに設定することにより無効化することができます。

3.12.5 クッキーの送信

Copper PDFのHTTPクライアントには、ウェブサーバーからクッキーを取得し、保存する機能はありません。

ただし、アプリケーション側で設定したクッキーをウェブサーバーに送信する機能があります。これは、クッキーを使ったセッション認証を行うウェブアプリケーションでCopper PDFを使用する場合に、ウェブアプリケーションがユーザーのセッションIDを知っていて、Copper PDFから自分自身のウェブサーバーに接続する場合には有効です。

クッキーは4つで1組となっている、

- `input.http.cookie.n.domain` [io]
- `input.http.cookie.n.name` [io]
- `input.http.cookie.n.value` [io]
- `input.http.cookie.n.path` [io]

という入出力プロパティで設定します。それぞれクッキーのドメイン、名前、値、パスです。パスを省略した場合はルートパス("/")となります。*n*は0から始まる整数で、連番にすることで、複数のクッキーを送ることができます。

次は、Javaサーブレットで現在のクライアントのセッションIDをCopper PDFのHTTPクライアントに引き継ぐ例です。

例 3.103 クッキーの設定

```
String sessionId = request.getSession().getId();
session.property("input.http.cookie.0.domain", "www.foo.com");
session.property("input.http.cookie.0.name", "JSESSIONID");
session.property("input.http.cookie.0.value", sessionId);
session.property("input.http.cookie.0.path", "/");
```

3.12.6 タイムアウトの設定

相手先サーバとの接続に時間がかかる場合、あるいは接続後一定時間データがやりとりされない場合、接続を切断してコンテンツの取得をあきらめる(タイムアウトする)ように設定することができます。*[2.0.7]* デフォルトではタイムアウトしないため、ずっと待ち続けます。

タイムアウトは

- `input.http.connection.timeout` [io]
- `input.http.socket.timeout` [io]

により設定します。数値で設定し、単位はms(ミリ秒)です。

次の例では、接続が確立するまで30秒以上かかった場合または接続語10秒間データがやりとりされなかった場合にタイムアウトするように設定しています。

例 3.104 タイムアウトの設定

```
session.property("input.http.connection.timeout", "30000");
session.property("input.http.socket.timeout", "10000");
```

3.13 出力制限機能

一般に解放するウェブアプリケーション等で、極端にページ数の大きな結果、あるいは物理的なサイズが極端に大きな結果が出力されてしまうような文書が入力される可能性がある場合があります。このとき、意図的であるかどうかに関わらずサーバーに極端に負荷がかけられるような状態に陥ることがあります。Copper PDFには、そういった危険を防ぐための動作制限の機能があります。

動作制限が発生した場合、ドライバは例外を投げるか、文書変換のための関数が戻り値としてエラーを返します。その際、ドライバのエラーハンドラには(警告や致命的エラーではなく)エラーが通知されます。

3.13.1 ページ数の制限

出力結果が一定のページ数に達した場合に、強制的に処理を中断する機能です。

`output.page-limit`[io]に、出力する最大ページ数を設定してください。デフォルトでは、ページ数の制限はありません。

ページ数の制限を越えた場合、途中までのページを結果として出力するか、結果を破棄するかどうかを選ぶことができます。

`output.page-limit.abort`[io]に、"normal"を設定すると、途中までのページを結果として出力します。"force"を設定すると、結果を破棄します。デフォルトは"force"です。

ただし、2パス以上の処理を行う場合は、最終パス以外でページ数が限界に達すると結果が破棄されます。また、（PDFではなく）画像として結果を出力する場合は、（最終パスの場合）いずれの場合も途中までページが出力されます。

3.13.2 データサイズの制限

出力結果が一定の物理的なサイズに達した場合に、強制的に処理を中断する機能です。

`output.size-limit`[io]に、最大データサイズをバイト単位で設定してください。デフォルトではデータサイズの制限はありません。

4.デザイナーガイド

4.1 Copper PDFによる文書のレイアウト

Copper PDFはHTMLやCSS等、ウェブコンテンツの作成で標準となっている方法を使用するため、ウェブコンテンツの作成の知識があれば、PDFの作成も同様にできるのが特徴です。このドキュメントでは、Copper PDF独自の機能や、注意点について解説します。

4.1.1 Copper PDFで文書をレイアウトするには

ドキュメントをレイアウトする方法は、通常のウェブアプリケーションの作成と大差ありません。HTMLエディタ等を使ってHTMLや、プログラムで処理するためのテンプレートを作成し、それをプログラムによりPDFに変換するという手順となります。

Copper PDFによる出力結果を手軽に確認するために、Copper PDFにはcopper-webappというツールが用意されています。このツールを起動すると、ローカルマシン上にあるHTMLファイルをウェブベースのツールでPDFに変換することができます。

4.2 入出力プロパティ

Copper PDF独自の機能や、一般的なブラウザの環境設定やオプションに相当する部分を設定するためには、名前と値の組み合わせである「入出力プロパティ」を設定します。入出力プロパティの一覧は資料集の入出力プロパティ (268ページ)を参照してください。

入出力プロパティは、プログラマにより プログラムにより設定 (66ページ) されるか、システム管理者により デフォルトの入出力プロパティの設定ファイル (42ページ) により設定されます。

また入出力プロパティはHTMLやXML文書中でjp.cssj.property処理命令により設定することもできます。ただし、プログラムや設定ファイルにより、あらかじめ `input.property-pi`[io] がtrueに設定されている必要があります。次のように処理命令のname属性にプロパティ名、value属性に値を記述してください。また、必ずドキュメントの先頭から最初の要素の間に記述してください。

例 4.1 jp.cssj.property処理命令による入出力プロパティの設定

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<?jp.cssj.property name="output.pdf.encryption" value="v2"?>
<?jp.cssj.property name="output.pdf.encryption.user-password"
value="user"?>
<?jp.cssj.property name="output.pdf.encryption.owner-password"
value="owner"?>
<?jp.cssj.property name="output.pdf.encryption.permissions.print"
value="false"?>
<!--
上記の設定を有効にするには、プログラムか設定ファイルで input.property-pi が
true にされている必要があります。
-->
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <title>暗号化1</title>
    <style type="text/css">
    </style>
```

```
  </head>

  <body>
    <h1>暗号化1</h1>
    <p>
      この文書のユーザーパスワードは"user" オーナーパスワードは"owner"です。
      印刷が禁止されています。
    </p>
  </body>
</html>
```

なお、jp.cssj.property処理命令により設定できないプロパティがあります。詳細は資料集(282ページ)を参照してください。

4.3 対応する入力ファイル

4.3.1 HTML/XMLの処理

ドキュメントの判別

Copper PDFはドキュメントがHTMLかXMLであるかを、以下の情報をもとに判別します。

1. プログラマが指定したドキュメントのMIMEタイプ
2. HTTPのContent-Type ヘッダ

1の情報は、2より優先されます。上記の情報以外によりCopper PDFがドキュメントの型を判別することはありません。HTMLとして判別されたドキュメントは、`<?xml～` で開始していても、HTMLとして認識します。ただし、HTMLと認識されても、XMLやXHTMLがサポートしている名前空間は認識されます。

HTMLと認識された文書は、ゆるやかに解釈されるため、文法ミスが許容されます。デフォルトの名前空間（xmlns="～"で指定される名前空間）はXHTMLの名前空間（ http://www.w3.org/1999/xhtml ）に常に固定されます。`input.html.change-default-namespace`[io]をtrueに設定するとデフォルトの名前空間を変更できるようになります。*[3.2.12]*。

一方、XMLと認識された文書は厳密に解釈されます。XHTMLを記述する場合、全ての要素名と属性名は小文字で記述してください。文法エラーがあった場合、処理が停止します。

キャラクタ・エンコーディング

Copper PDFが認識できるキャラクタ・エンコーディングはJava実行環境に依存します。エンコーディング名のリストは Java実行環境の「サポートされているエンコーディング」ドキュメント を参照してください。

XMLドキュメントのキャラクタ・エンコーディングは、次の優先順位で判別します。

1. プログラマが指定したドキュメントのキャラクタ・エンコーディング
2. BOM(Byte Order Mark)
3. XML宣言のencoding属性

デフォルトのエンコーディングは、XMLの仕様に従い、UTF-8またはUTF-16が自動判別されます。

HTMLドキュメントのキャラクタ・エンコーディングは、次の優先順位で判別します。

1. プログラマが指定したドキュメントのキャラクタ・エンコーディング

2. BOM(Byte Order Mark)
3. `<meta http-equiv="Content-Type" content="text/html; charset=エンコーディング名">`
4. XML宣言のencoding属性
5. `input.default-encoding`[io]によるエンコーディング

`input.default-encoding`[io]はデフォルトではJISAutoDetect(ISO-2022-JP, Shift_JIS, EUC_JPの自動判別)です。Copper PDF 3.0.1 からはデフォルトがJISUniAutoDetect(ISO-2022-JP, UTF-8, Windows-31J, EUC_JP_Solarisの自動判別) という Copper PDF 独自のエンコーディング名になりました。

文書情報

Copper PDFはHTMLのmeta要素から取得した情報を、PDFの文書情報として使用します。文書情報はHTMLの `<meta name="名前" content="値">` 要素によって設定することができます。ただし、TITLEはHTMLのtitle要素の内容も使われます。

表 4.1 meta要素による文書情報の設定

名前	PDFの属性名	説明
TITLE	Title/タイトル	文書の表題。
DESCRIPTION SUBJECT	Subject/サブタイトル	文書に内容についての簡潔な説明。
KEYWORDS	Keywords/キーワード	スペースまたはカンマ区切りで羅列した、文書の内容に関連するキーワード。
AUTHOR	Author/作成者	文書の作成者。
PRODUCER	Producer/PDF変換	PDFを生成したプログラム。省略した場合はCopper PDFの名前とバージョンが入ります。
GENERATOR CREATOR	Creator/アプリケーション	HTML文書を生成したプログラム、エディタ、オーサリングツールなど。

なお、meta要素のname属性は大文字小文字を区別しません。
`<meta name="AUTHOR" content="作者名">` としても、
`<meta name="Author" content="作者名">` としても、
PDF文書に作者名が設定されます。

HTMLのmeta, title要素による文書情報の設定は、`output.use-meta-info`[io]に"false"を設定することにより無効化することができます *[3.1.8]*。その場合、次の方法が唯一の文書情報の設定手段となります。

文書情報は、入出力プロパティによっても設定することができます *[2.0.3]*。

- `output.meta.n.name` [io]

- output.meta.*n*.value [io]

により、名前と値を設定することができます。*n*は0から始まる連番で、複数の文書情報を設定することができます。この設定は、前記のmeta要素で上書きされます。

XSLTスタイルシートの適用

XSLTスタイルシートはCSS同様にxml-stylesheet処理命令により適用されます。xml-stylesheet処理命令についての詳細はCSSのxml-stylesheet処理命令の節を参照してください。

CSS同様に、デフォルトのXSLTスタイルシートを設定することができます。input.xslt.default-stylesheet [io]により指定されたスタイルシートが最初に適用されます。

input.xslt.default-stylesheet [io]またはxml-stylesheet処理命令によって、ドキュメントに複数のXSLTスタイルシートが指定されているとき、実際に適用するのは最初に指定されたスタイルシートです。他のスタイルシートは無視されます。

4.3.2 画像

Copper PDFがサポートする画像

Copper PDFはJPEG, GIF, PNG, SVG形式の画像を標準でサポートしています。各画像は通常のブラウザ同様にimg, object, embed要素によってドキュメント中に含めることができます。CSSのbackground-image [css]プロパティ、あるいはcontent [css]プロパティによっても画像の表示が可能です。

また、画像を直接読み込んでPDFに変換することができます。この場合、全体が画像となっている1ページだけのPDFが生成されます。

他の画像形式の利用

より多種多様な画像フォーマットを使用する場合は Java Advanced Imaging Image I/O Tools(JAI-ImageI/O) をCopper PDFを動作させるJava環境に管理者がインストールする必要があります。JAI-ImageI/Oをインストールすることにより、BMP, JPEG 2000, PNM, TIFF, WBMPの各画像フォーマットが利用可能になります。また、JAI-ImageI/O以外にもJava Image I/Oに対応した拡張ライブラリをインストールすることで利用可能な画像を追加することができます。

JAI-ImageI/Oは画像形式での出力 (180ページ)にも利用することができます*[2.0.3]*。

ラスター(ビットマップ/ピクセルマップ)画像

GIF / PNG画像

GIFおよびPNG画像の透明化効果はPDF内でも有効です。ただし、PDF 1.3以前のバージョンのPDFを出力する場合、PNG画像の半透明化効果が正確に表現されず、透明・不透明だけとなります。

アニメーションGIFの場合、最初のコマだけが使われ、アニメーションしません。

JPEG / JPEG 2000画像

JPEG画像は処理を加えずに、そのままPDFに含めることができます。入出力プロパティ `output.pdf.jpeg-image`[io] にto-flateを設定することで画像を展開してからPDFに含むこともできますが、PDFのサイズは大きくなります。

JAI-ImageI/Oがインストールされている場合は、JPEG 2000にも同様の処理が適用されます。JAI-ImageI/Oがない場合はJPEG 2000は全く利用できませんのでご注意ください。

その他の画像

JAI-ImageI/O等のライブラリで他の画像形式が利用可能な場合、それらの扱いはGIF/PNG画像に準じます。半透明効果をサポートする画像形式の場合はPDF内でも再現され、アニメーションする画像では最初のコマだけが使われます。

画像の解像度

レイアウトされるときの、埋め込まれた画像の大きさは `output.resolution`[io] によります。デフォルトでは96dpiです。つまりデフォルトでは、例えばimgタグでheightが100と設定された場合、あるいはimgタグでサイズが指定されずに、元の画像の大きさが100の場合、実際のレイアウト結果では75ptの高さになります。

SVG画像

SVG画像ファイルの参照

Copper PDFはSVG画像ファイル(.svg)またはGZIPで圧縮されたSVG画像ファイル(.svgz)をサポートしています。ただし、**PDF出力の場合、SVG埋め込みフォントと、透明度を含むグラデーション塗り(linearGradient, radialGradientでstop-opacityの使用) には対応していません。**

一般的なブラウザではobject要素によりSVG画像サポートされていますので、Copper PDFでも同様の方法を推奨します。

例 4.2 object要素によるSVG画像の埋め込み

```
<object data="image.svg" type="image/svg+xml">
SVGをサポートしないブラウザで表示されるテキスト。
</object>
```

Copper PDFでは、SVG画像を他の画像と同様に扱うことができます。img要素を利用することができ、`background-image` [css]プロパティにより背景に設定することもできます。

インラインSVG

SVGを別ファイルに分けずに、文章中に直接記述することができます。グラフ等、動的に変化する画像をドキュメントに含める場合に適しています。インラインSVGは、ドキュメント中でhttp://www.w3.org/2000/svg と名前空間宣言したSVGを記述するだけです。

例 4.3 インラインSVG

```
<html>
<head>
  <title>インラインSVGを含むドキュメント</title>
</head>
<body>
<p>すぐ下に○が表示されます。</p>
<svg:svg xmlns:svg="http://www.w3.org/2000/svg"
         xmlns:xlink="http://www.w3.org/1999/xlink"
         preserveAspectRatio="none"
         width="100" height="100"
         viewBox="0 0 100 100"
         xml:space="preserve">
    <svg:g>
        <svg:circle cx="50" cy="50" r="45" stroke="Blue"
fill="White" stroke-width="10"/>
    </svg:g>
</svg:svg>
</body>
</html>
```

画像を読み込めない場合

画像ファイルが存在しない、データの破損、Copper PDFがサポートしない画像形式といった原因で 画像を読み込めない場合は、通常はalt属性で指定された代替テキストが表示されます。入出力プロパティ`output.broken-image` [io] の設定により、画像の形に×印を表示する(cross)か、空白を空ける(hidden)ことができます。

4.3.3 EPUB電子書籍

Copper PDFはEPUB 2.0またはEPUB 3.0ファイルを変換することができます*[3.1.0]*。

EPUBファイルは、コンテンツのMIME方が application/epub+zip であるか、拡張子が.epub であることで判別されます。

4.4 出力するファイル形式

Copper PDFは長らくPDFを唯一の出力形式としていましたが、Copper PDF 2.0.3から画像の出力をサポートしました。

output.type[io]にMIMEタイプを設定することにより、出力形式を切り替えることができます。デフォルトではPDF("application/pdf")です。

4.4.1 PDFの出力

Copper PDFはデフォルトの設定でPDFを出力します。PDF出力機能の詳細はPDFの機能 (ページ)を参照してください。

PDFのバージョンと機能

出力するPDFのバージョンは output.pdf.version[io] で設定することができます。デフォルトではバージョン1.5のPDFが出力されます。PDF 1.5以降であれば、Copper PDFの全ての機能を利用することができます。

PDF 1.4以前では、使用できる機能に制限があります。詳細は資料集の output.pdf.version[io] の説明を参照してください。

Copper PDF 2.1.0からは、PDF/A-1(ISO 19005-1)に対応したファイルを出力することができます (179ページ)。

暗号化

Copper PDFは暗号化したPDFを出力することができます。パスワードを設定して暗号化したPDFは、パスワードがなければ閲覧不可能になります。また、内容のコピーなどPDFの利用制限を設定するためにも暗号化が必要で、広く配布するPDFのためにパスワードを設定せず、暗号化だけしたPDFを出力することができます。

暗号化は、output.pdf.encryption[io] の設定で有効となります。暗号化方式はv1とv2の2種類がありますが、通常はPDF 1.3以降でサポートされているv2暗号化を使用してください。

暗号強度(暗号化キーのビット数)は output.pdf.encryption.length[io] で設定できます。デフォルトでは最も強い暗号化(v1では40, v2では128)がされるため、通常は設定の必要はありません。

閲覧のためのパスワードは output.pdf.encryption.user-password[io] で設定することができます。このプロパティを設定しなかった場合は、暗号化だけされて誰で

も開くことができるPDFが生成されます。さらに output.pdf.encryption.owner-password[io] でAdobe Acrobat等で文書の編集をするためのパスワードを設定することができます。

PDFを暗号化する場合は、パーミッションを設定することができます。パーミッションはoutput.pdf.encryption.permissionsで開始する、次の8種類の入出力プロパティで設定することができます。true(有効)またはfalse(無効)で設定してください。デフォルトでは全て有効です。

output.pdf.encryption.permissions.print[io]
印刷

output.pdf.encryption.permissions.modify[io]
内容の変更

output.pdf.encryption.permissions.copy[io]
テキストや画像のコピー

output.pdf.encryption.permissions.add[io]
注釈の追加、変更とPDFフォームへの入力

次の4つはv2暗号化でのみ有効です。

output.pdf.encryption.permissions.fill[io]
PDFフォームへの入力

output.pdf.encryption.permissions.extract[io]
障害を持つユーザーのための文書中のテキストや画像の抽出

output.pdf.encryption.permissions.assemble[io]
文書中に新しいページ、ブックマーク、サムネイル画像を追加する

output.pdf.encryption.permissions.print-high[io]
文書を高画質で印刷する

なお、Copper PDFはPDFフォームをサポートしていないため、フォームに対する権限は通常は無意味です。



パーミッションを設定することにより、Adobe Reader等のPDF閲覧ソフトは設定に沿った動作をしますが、他のツール等でPDFに対する該当する操作が行われないことを保証するものではありません。

ファイルの添付

PDF 1.4以降では、PDFにファイルを添付することができます。ファイルの添付は

- output.pdf.attachments.*n*.name [io]
- output.pdf.attachments.*n*.description [io]
- output.pdf.attachments.*n*.mime-type [io]
- output.pdf.attachments.*n*.uri [io]

の4つで1組のプロパティを使います。*n*は0から始まる通し番号で、複数のファイルを添付することができます。

このうち必須なのはoutput.pdf.attachments.*n*.uri [io]です。あらかじめドライバにより送られてきたファイルのURIか、アクセス可能なファイルのURLを設定してください。

output.pdf.attachments.*n*.name [io]はファイルの名前です。ASCII文字以外も使用することができますが、文字化けが発生するおそれがあります。日本語ファイル名を使用する場合は、output.pdf.attachments.*n*.name [io]にはなるべくASCII文字を使い、output.pdf.attachments.*n*.description [io] に実際のファイル名を設定してください。

output.pdf.attachments.*n*.mime-type [io]は、ファイルのMIMEタイプです。

PDFの圧縮形式

Copper PDFのデフォルトの設定では、出力されるPDFはなるべくサイズが小さくなるように圧縮されます。結果、出力されるPDFはバイナリデータとなります。

output.pdf.compression[io]の設定により、テキスト形式か、圧縮されないPDFを生成することができます。asciiという設定では、PDFを圧縮しますが、出力されるPDFはASCIIテキストになります。若干サイズは大きくなりますが、テキストとしてそのままコピー&ペーストできるようなファイルができ上がります。noneではPDFを全く圧縮せず、描画命令などがそのままの形のテキストで出てきます。画像はHEX形式となり、非常にファイルサイズが大きくなりますが、出力されたPDFの内容をテキストエディタ等で確認するのに便利です。

PDF中の画像の圧縮形式

容量の大きなPDFの場合、画像が容量のほとんどを占めていることがよくあります。そのため、画像の圧縮形式を適切に設定することで、PDFのサイズを節約することができます。

デフォルトでは、JPEG画像は加工されずにPDF内で使用され、他の画像はFlateDecode形式(可逆圧縮)で圧縮されます。

画像の圧縮形式は `output.pdf.image.compression`[io] で指定することができます*[2.0.3]*。デフォルトはflateですが、jpegにするとJPEGで圧縮します。`JAI-ImageI/O`[io] がインストールされた環境では、jpeg2000の指定が可能です。画像の圧縮は、デフォルトではJPEG / JPEG2000には適用されませんが `output.pdf.jpeg-image`[io] をrecompressにすると、JPEG / JPEG2000を再圧縮するようになります*[2.0.3]* (ただしJPEGをJPEGに、JPEG2000をJPEG2000に圧縮することはしません、JPEGとJPEG2000を互いに変換することはします)。

JPEG / JPEG2000による再圧縮は不可逆であり、画像を劣化させるためアイコンのような小さな画像には適用したくないことがあります。そのため、Copper PDFは一定の大きさより小さな画像を常にFlateDecode形式で圧縮するように指定することができます*[2.0.3]*。閾値は`output.pdf.image.compression.lossless`[io]に、画像の縦のピクセル数と横のピクセル数を足した値で設定します。デフォルトは200です。この場合、例えば縦が90ピクセル、横が110ピクセルの画像はJPEG / JPEG2000に再圧縮されますが、縦が80ピクセル、横が100ピクセルの画像はFlateDecodeで可逆圧縮されます。ただし、元の画像がJPEG / JPEG2000で、`output.pdf.image.compression`[io]で指定した形式と同じ場合は、そのままの形式でPDFに埋め込みます。

PDFの表示環境のキャラクタ・エンコーディング

PDF 1.2以前ではフォント名、PDF 1.6以前では添付ファイル名が、PDFの表示環境のキャラクタ・エンコーディングに依存していました。後のバージョンのPDFではユニコードを使用するため、特に表示環境のキャラクタ・エンコーディングを気にする必要はありません。

古いバージョンのPDFを生成する場合は、文字化けを防ぐために、`output.pdf.platform-encoding`[io] に想定される表示環境のエンコーディングを設定する必要があります。デフォルトではMS932(Windows版Shift_JIS)が設定されているため、日本語環境ではおおよそ問題は起きません。ただし、該当する箇所に設定されたエンコーディングがサポートしない文字が使われた場合や、設定されたエンコーディング以外の表示環境では文字化けが発生する可能性があります。

作成・更新時刻、ファイルIDの設定

デフォルトでは、PDFの作成・更新時刻(CreationDate, ModDate)にはCopper PDFが動作している環境の時計の現在時刻が使われます。また、ファイルIDは乱数が使用されます。

作成・更新時刻とファイルIDは、明示的に設定することもできます*[2.0.9]*。

作成・更新時刻はそれぞれ

- output.pdf.meta.creation-date[io]
- output.pdf.meta.mod-date[io]

を設定してください。日付の形式は"2009-05-22 21:10:14"または"2009-06-04 15:53:02 +09:00" (タイムゾーンを明示する場合)といった形式です。

ファイルIDは output.pdf.file-id[io] で設定してください。これは必ず32桁固定の16進数で、"000067A36902BF8D2A0617B9CD02BCFA" のような値です。

すかし

PDFの前面または背面に、すかし画像を出力することができます *[2.1.8]*。すかしを利用できるのは、PDF 1.4以降です。

同様のことは、CSSのbackground-image[css]等をつかって実現することもできますが、PDFの場合は、画面では見えず印刷時だけすかしを表示する機能があります。

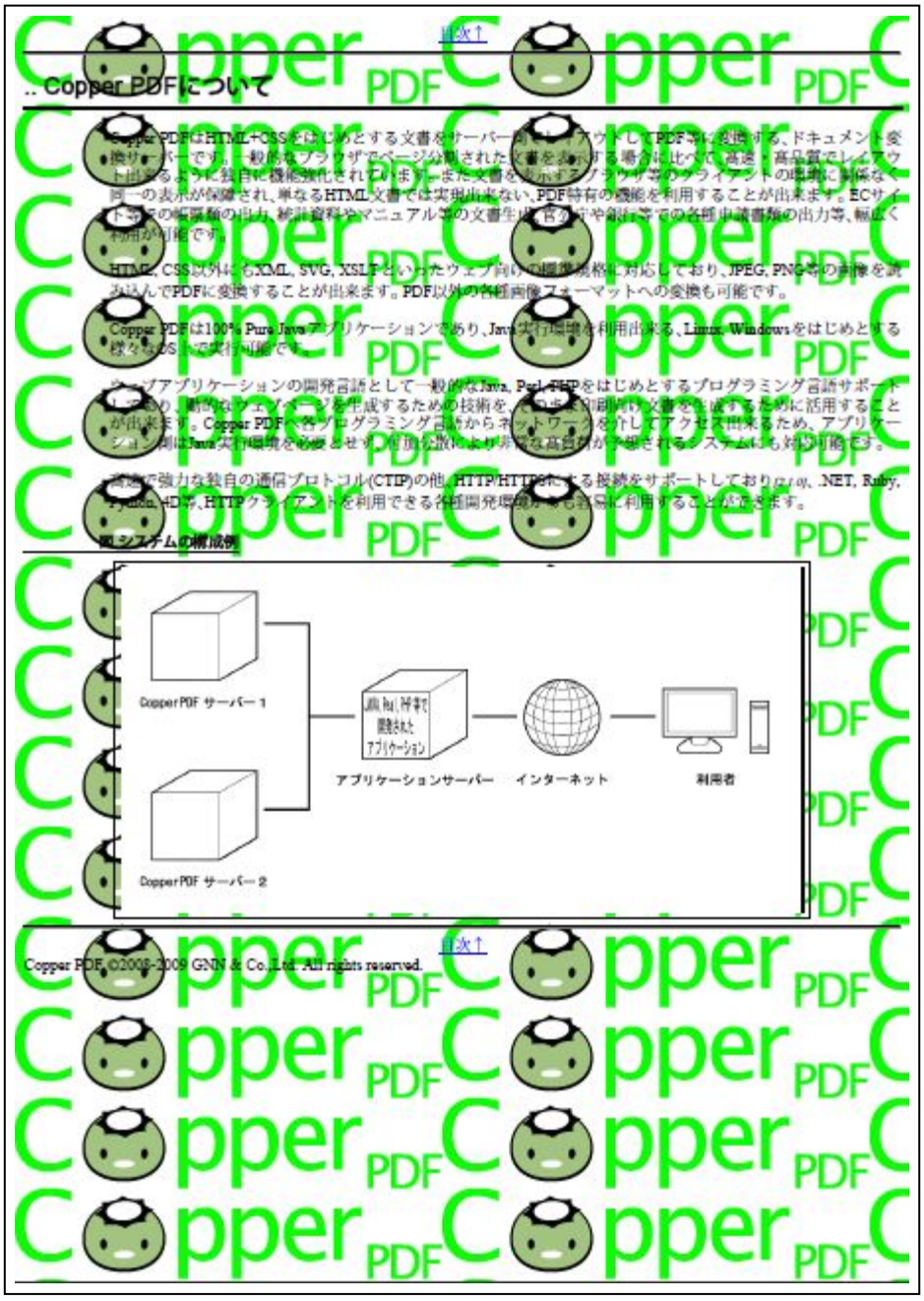
output.pdf.watermark.uri[io]に、すかしに使う画像を、絶対アドレスで指定すると、繰り返しパターンとして、画像がPDFの全ページに表示されるようになります。

デフォルトでは背景は背面に表示されますが、output.pdf.watermark.mode[io]に"front"を設定すると、前面に表示されます。

output.pdf.watermark.opacity[io]により、すかしの不透明度を設定することができます。例えば、"0.5"という値を設定すると、すかしが半透明になります。

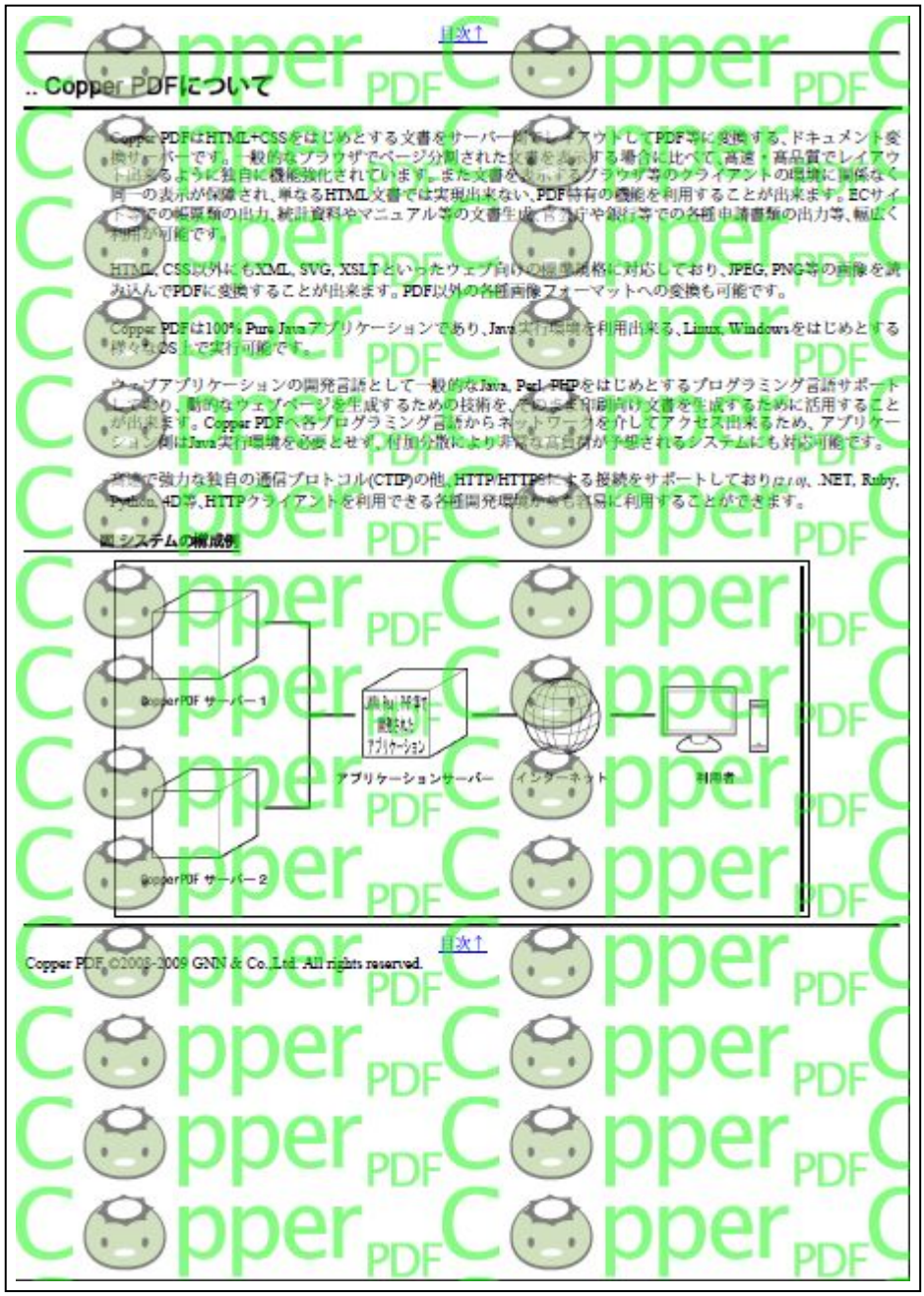
以下の出力例は、output.pdf.watermark.uri[io]にSVG画像を指定し、output.pdf.watermark.mode[io]に"back"を指定しています。

図 4.17 背面にすかしを配置



以下の出力例は、output.pdf.watermark.uri[io] にSVG画像を指定し、output.pdf.watermark.mode[io] に"front"を指定し、output.pdf.watermark.opacity[io]に0.5を指定しています。

図 4.18 前面にすかしを配置



印刷時だけ、または画面表示だけすかしを表示する

`output.pdf.watermark.view`[io]、`output.pdf.watermark.print`[io]は、それぞれすかしを画面表示時と印刷時に表示するかどうかを設定するものです。デフォルトでは両方とも表示しますが、例えば`output.pdf.watermark.view`[io]に"false"を設定すると、印刷時だけすかしを表示します。

すかしを背面に表示する場合、これらの設定はPDF 1.4以前では有効になりません。

PDF/A-1bに準拠したファイルの出力

PDF/A (ISO 19005-1)はPDFの長期保存のための国際規格です。出力されるPDFの形式に一定の制約を加えることにより、様々なPDF表示ソフトウェアの互換性の問題が起きにくくなります。PDF/AにはPDF/A-1bと、より制約の強いPDF/A-1aがありますが、Copper PDFは現在のところPDF/A-1bをサポートします。

output.pdf.version[io]に"1.4A-1"を設定すると、PDF/A-1bに準拠したファイルが生成されます*[2.1.0]*。このモードで生成されるPDFのバージョンは1.4ですが、通常のPDF 1.4の以下の機能が使用できなくなります。

- 暗号化
- 添付ファイル
- 半透明表示(半透明すかし、半透明PNG、SVGの透明度指定など)

フォントは常に埋め込みフォントだけが使用されます。CID-Keyedフォント、外部フォント、コア14フォントも使用できません。

PDFビューワの表示設定*[3.0.2/2.1.11]*

CopperPDFは、PDFをビューワで開いた際の表示(ViewerPreferences)を設定することができます。ViewerPreferencesの設定がどのように影響するかは、そのPDFを開くビューワー（Adobe Readerなど）によります。必ずしもユーザーの環境で設定通りに表示されるものではありませんが、組織内での事務処理や印刷作業のためには非常に有効です。

ViewerPreferencesはoutput.pdf.viewer-preferences. で始まる名前の入出力プロパティにより設定します。たとえば、文書を印刷する場合の印刷部数をあらかじめ3に設定したい場合は、output.pdf.viewer-preferences.num-copies[io] を3に設定します。

設定の一覧は、資料集の入出力プロパティ一覧を参照してください。

PDFの表示の際に実行されるJavaScript*[3.0.2/2.1.11]*

output.pdf.open-action.java-script[io]により、PDFをビューワで開いたタイミングで実行されるJavaScriptを設定することができます。

例えば、PDFを開いた際に印刷ダイアログを表示する場合は "print();" を設定します。

PDF の JavaScript の仕様については Adobe 社が公開しているドキュメント (http://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/js_api_reference.pdf)を参照してください。

4.4.2 画像の出力

Copper PDFはPDFだけではなく、JPEG等のラスタ(ピクセルマップ)画像を出力することができます。*[2.0.3]* output.type[io]に"image/jpeg"のように、画像のMIMEタイプを指定してください。

画像のエンコーディングにはJava実行環境の機能を利用しており、大抵のJava実行環境ではJPEG("image/jpeg"), PNG("image/png")画像を出力することができます。Java実行環境に拡張ライブラリを導入することにより、出力できる画像を増やすことができます。拡張方法については他の画像形式の利用を参照してください。

画像出力の制約

コアフォント (47ページ) とCID-Keyedフォント (52ページ)を描画できないという制約があります。フォントを正しく表示するためには、埋め込みフォント (52ページ)を利用する必要があります。フォントの設定方法はフォントの種類 (46ページ)を参照してください。

画像出力の解像度

出力される画像の解像度は output.image.resolution[io] により設定することができます*[2.0.4]*。値の単位はdpiで、CSSで1inの長さのオブジェクトを描画するときに並ぶピクセル数です。デフォルトの画像の解像度は96dpiです。1ptは1/72inであるため、デフォルトでは1ptは1ピクセルより若干大きくなります。

1pxが実際に出力される画像の1ピクセルと一致するようにするためには output.resolution[io] と output.image.resolution[io] が同じ値になるようにしてください。デフォルトでは両方とも96です。

なお、2.0.8以前ではoutput.image.resolution[io]のデフォルト値が72となっており、解像度が正しく反映されないバグがありました。2.0.9以降では (以前の設定 × output.resolution[io] / 72) で換算した値を設定してください。

4.4.3 SVGの出力

Copper PDF 3.0.1 からSVGの出力をサポートしました。output.type[io] に"image/svg+xml"を設定してください。

SVG出力では、文字が全てアウトライン化されます。出力されたSVGはAdobe Illustrator CSなどのドローソフトで読み込み、加工することができます。

4.5 一般的なブラウザとの互換性



Copper PDF 3.1.0以降では、Internet Explorer 7互換モードは廃止されました。

4.5.1 互換性モードの切り替え

Copper PDFは、デフォルト(標準モード)ではCSS 2.1に準拠したレンダリングをしますが、実用上の問題から一般的なブラウザとの互換モードが用意されています。

`output.compatible_mode`[io] にmsieを指定することにより、Copper PDFはInternet Explorer 7に近いレイアウトをするように試みます。

ただし、ドキュメントに、次のいずれかのpublic idを持つDOCTYPE宣言がある場合は、常に通常のモードでレイアウトします。

互換モードはCopper PDFのバージョンアップごとに一般的なブラウザと互換性を持つように改善が試みられますが、完全な互換性を保証するものではありません。また、制限事項があり、デフォルトのモードより非効率な処理が行われることがあるため、若干パフォーマンスが劣ります。

- -//W3C//DTD HTML 4.01//EN
- -//W3C//DTD XHTML 1.0 Transitional//EN
- -//W3C//DTD XHTML 1.0 Strict//EN
- -//W3C//DTD XHTML 1.1//EN

4.5.2 標準モードとmsieモードの違い

CSSで数字のクラス名が認識される

標準モードでは数字のクラス名はエラーとなり無視されますが、msieモードでは有効です。

例 4.4 数字のクラス名を使用する

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <title>数字のクラス名</title>
    <style type="text/css">
.A {
    color: Red;
}
.1 {
    color: Red;
}
    </style>
  </head>

  <body>
    <p class="a">このテキストは赤字です</p>
    <p class="1">msieモードではここも赤くなります</p>
  </body>
</html>
```

CSSで':'の代わりに'='が使用できる

CSSでプロパティ名と値の区切り文字は':'ですが、msieモードでは'='も使用することができます。

例 4.5 ':'の代わりに'='を使用する

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <title>':'の代わりに'='を使用する</title>
    <style type="text/css">
.A {
    color: Red;
}
.B {
    color=Red;
}
    </style>
  </head>

  <body>
    <p class="a">このテキストは赤字です</p>
    <p class="b">msieモードではここも赤くなります</p>
  </body>
</html>
```

CSSの色指定で#を省略しても認識される

msieモードではRGBコードで色を指定する場合、#を省略できます。

例 4.6 色指定で#を省略する

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <title>色指定で#を省略する</title>
    <style type="text/css">
.A {
    color: #FF0000;
}
.B {
    color: FF0000;
}
    </style>
  </head>

  <body>
    <p class="a">このテキストは赤字です</p>
    <p class="b">msieモードではここも赤くなります</p>
  </body>
</html>
```

CSSの長さ指定で単位を省略しても認識される

msieモードでは長さ指定で単位を省略した場合、px単位による指定と認識されます。

フォームの前後にマージンが設定される

msieモードでは、デフォルトでフォーム(form要素)の前後に1.12emのマージンが設定されます。

段落と見出しのマージン上下のマージンをなくす場合がある

msieモードでは、段落または見出しが(p, h1～h6要素)の、ページまたはテーブルセルの先頭にある場合、前のマージンがなくなります。同様に末尾にある場合は後のマージンがなくなります。

ボックスの幅または高さに100%が指定された場合、外側のボックスをはみ出さない

`width` [css] または `height` [css] のパーセント指定は、標準モードでは外側のボックスの幅に対して、ボックスの内側の幅を指定するため、境界のあるボックスに対して100%を

指定すると、ボックスが外側のボックスの内部からはみ出します。msieモードでは100%指定された場合にボックスの外側の幅が外側のボックスの内側の幅に一致し、0%指定された場合にボックスの内側の幅がゼロになるように計算します。

絶対位置指定ボックス、浮動ボックスの大きさが内容により拡張される

絶対位置指定ボックス、浮動ボックスに{overflow: visible;}が指定され、かつボックスの幅と高さが設定されている場合、標準モードではボックスのサイズは常に固定で、内容がはみ出した部分はボックスの境界にかかりますが、msieモードではボックスの内容によってボックスの幅と高さが拡張されます。

通常のフローのボックスが、幅が指定されたボックスにより拡張される

幅指定されたボックスに、より大きな幅のボックスが含まれている場合、標準モードでは内部のボックスは外側のボックスをはみ出しますが、msieモードでは外側のボックスの幅が、内側のボックスが収まるように拡張されます。

幅がautoの固定レイアウトテーブルが固定レイアウトのまま処理される

標準モードでは、{table-layout: fixed;}が指定されているテーブルであっても、width [css]がautoであれば{table-layout: auto;} が指定されたのと同様に扱われます。

msieモードでは、width [css]がautoの場合は、テーブルの外側の幅を、外側のボックスの内側の幅に合わせて固定し、固定レイアウトとして処理します。ただし、全てのカラムに幅が明示されている場合は、テーブルの幅はカラムの指定幅の合計となります。

テーブル行に対するmax-height [css]が適用されない

msieモードではテーブル行に対するmax-height [css]の設定は無視されます。

ブロックに対するline-height [css]による高さが確保されない

ブロックに対してline-height [css]が指定されている場合、標準モードではブロック内の行に対して指定した高さが必ず確保されますが、msieモードでは行内にインライン画像、インラインボックスまたはインラインテーブルだけが含まれ、かつそれらの高さがline-height [css]より小さい場合は、line-height [css]で指定された高さは確保されません。

全角スペースの間で折り返しされない

msieモードでは、全角スペースが連続している場合、その中での折り返しは発生しません。また、全角スペースにより外側のボックスが拡張されることはありません。

input要素の高さが強制される

msieモードでは、input要素で配置されるフォームの高さが1.12emに強制されます。height [css]は適用されません。

マージンの設定に関わらず中央寄せされる

img, input, applet, object, iframe, tableのalign属性による中央寄せは、標準モードでは左右のマージンをautoにした状態{margin: 0 auto;}と同等ですが、msieモードでは左右のマージンが設定された場合も中央寄せになります。

例えば、標準モードでは{margin-left: 100pt;}が指定されれば、中央寄せが設定された要素でも常に左から100ptの位置に配置されますが、msieモードでは中央寄せから左に100ptの1/2(つまり50pt)ずれた位置に配置されます。

匿名のテーブルセルにより補完されない

テーブル内で、かつテーブルセルの外にインライン、インラインブロックまたはインラインテーブルが配置された場合、標準モードでは自動的に上位に匿名のセル要素が挿入されますが、msieモードでは全てテーブルの前に表示されます。

インラインボックスにwidth [css]が適用される

標準モードではwidth [css]はインラインボックスには適用されませんが、msieモードではwidth [css]の設定によりインラインボックスの幅が変化します。

また、width [css]がauto以外に設定されたインラインボックスはインラインボックスと同様に配置されます。すなわち、{display: inline;}が指定されている要素であっても、width [css]がauto以外であれば {display: inline-block;}が適用されているのと同じことになります。

text-align [css]がブロックの配置にも適用される

標準モードではtext-align [css]はテキストの配置にだけ適用されますが、msieモードでは内部にある通常のフローのボックスの配置にも適用されます。これはp要素やdiv要素等のalign属性と同様の挙動をします。

テーブルカラムのプロパティがセルに継承される

msieモードではテーブルカラムに設定された以下のプロパティがテーブルセルに継承されます。

- `text-align` [css]
- `color` [css]
- `font-style` [css]
- `font-weight` [css]
- `font-family` [css]
- `font-size` [css]
- `text-transform` [css]
- `letter-spacing` [css]
- `word-spacing` [css]

上記のプロパティがテーブルセルと、テーブル行の両方に設定されている場合は、テーブル行のプロパティの方が継承されます。

`width` [css]が設定されたテーブルセルには`{white-space: nowrap;}`が適用されない

標準モードでは`{white-space: nowrap;}`が指定された領域ではテキストの折り返しが行われないため、`width` [css]で幅が指定されたテーブルセルでは、内容が幅をはみ出すことがありますが、msieモードでは`width` [css]で幅が指定されたテーブルセルでは`{white-space: nowrap;}`が適用されず、内容がはみ出さないようにテキストの折り返しが発生します。

4.5.3 既知の制限事項

ＭＳ明朝系フォントの文字幅について

ＭＳ明朝、ＭＳゴシックのフォントを使用する場合、一般的なブラウザでは小さなフォントサイズが指定された場合、一番近いビットマップフォントが使用されるため、指定された文字サイズと実際に使われる文字サイズが異なることがありますが、Copper PDFでは全てアウトラインフォントを使用するため、指定したとおりの文字サイズとなります。そのため、文字幅が一般的なブラウザで表示する場合と異なり、文字列の折り返し位置等が異なることがあります。

サポートしていないCSSプロパティ

Internet Explorer独自のCSSプロパティで、以下のものはサポートしていません。

- `accelerator` [css]
- `behavior` [css]
- `block-progression` [css]
- `filter` [css]
- `layout-grid` [css]
- `layout-grid-*` [css]
- `interpolation-mode` [css]
- `overflow-*` [css]
- `scrollbar-*` [css]
- `text-autospace` [css]
- `text-justify` [css]
- `text-kashida-space` [css]
- `text-overflow` [css]
- `text-underline-position` [css]
- `word-break` [css]
- `word-wrap` [css]
- `writing-mode` [css]
- `zoom` [css]

4.5.4 自動レイアウトテーブル

CSS 2.1では自動レイアウトテーブル (デフォルト、あるいは`{table-layout: auto;}`が指定されたテーブル) のレイアウトの調整方法について、明確な仕様が定められていないため、どうしても一般的なブラウザと若干のレイアウトの違いが生じます。Copper PDF 2.0.7以降では、ほぼ一般的なブラウザに近いレイアウトとなりますが、なるべく以下のいずれかの条件の下で使用してください。

- `{table-layout: fixed;}`が指定されたーブル)を使用する
- 自動レイアウトテーブルで、横方向に連結されたセルが存在する場合は、%による幅指定をしない。

4.6 XML/HTMLの拡張機能

4.6.1 見出し

HTMLのh1〜h6要素は、見出しとして特別な意味を持ちます。h1〜h6要素の数字は見出しのレベルとして認識され、数字が大きいものほど深い階層にあるものとして処理されます。

また、任意の要素に `html:header="レベル"` という属性を付与することで、任意の要素をHTMLのh1〜h6要素と同様の意味を持たせることができます。

ブックマーク

`output.pdf.bookmarks` [io] をtrueにすると、PDFのブックマーク(しおり)が生成されます。ブックマークは、見出しのレベルによって自動的に階層化されます。

現在ページのセクション

ページの内容として出力済みの見出しを、`content` [css] プロパティ内で-cssj-heading関数によって生成することができます。

例 4.7 見出しの表示

```
content: -cssj-heading(1) ' - ' -cssj-heading(2);
```

上の例はレベル1とレベル2の見出しを表示します。すなわち、ドキュメントの先頭から、前ページの最後までの間で、一番最後のh1とh2の内容を表示します。

-cssj-heading関数を `-cssj-page-content` [css] が設定されたページで使用すると、ドキュメントの先頭から現在のページの最後までの間で、一番最後の見出しを表示することができます。これは、ノンブルに見出しを表示する場合に便利です。

4.6.2 目次の生成

見出しは、目次の生成に利用することができます。目次を生成するためには、Copper PDFが文書全体の見出しと、見出しのあるページ番号を収集している必要があります。

例 4.8 目次の生成

```
<cssj:make-toc xmlns:cssj="http://www.cssj.jp/ns/cssjml"
               counter="page-number" type="decimal"/>
```

ドキュメントの末尾に目次を生成する場合は、目次の生成の開始時点で本文が全て処理されていますが、ドキュメントの先頭や途中に目次を生成する場合は、2パス以上の処理が必要です。

counter属性はページ番号を振るために使用するページカウンタの名前です。typeは、ページ番号のタイプです。CSSのlist-style-type [css] で使われるのと同じタイプ名が利用可能です。

生成される目次は、次のようなリストの形式をしています。

例 4.9 生成される目次の例

```
  <ul class="cssj-toc">
    <li><a href="#cssj-heading-1"><span class="cssj-title">1. タイトル1</span><span class="cssj-page">1</span></a></li>
    <li><a href="#cssj-heading-2"><span class="cssj-title">2. タイトル2</span><span class="cssj-page">5</span></a></li>
    <li><a href="#cssj-heading-3"><span class="cssj-title">3. タイトル3</span><span class="cssj-page">8</span></a></li>
    <ul>
      <li><a href="#cssj-heading-4"><span class="cssj-title">3.1. タイトル4</span><span class="cssj-page">8</span></a></li>
      <li><a href="#cssj-heading-5"><span class="cssj-title">3.2. タイトル5</span><span class="cssj-page">9</span></a></li>
    </ul>
  </ul>
```

例として、以下のようなスタイルシートを使うことで目次を整形してください。

例 4.10 目次を整形するCSSの例

```
ul.cssj-toc, ul.cssj-toc ul {
  list-style: none;
}
ul.cssj-toc {
  margin: 0;
}
ul.cssj-toc ul {
  margin: 1em;
}
ul.cssj-toc li {
  margin: 1em 1em 1.5em 0;
  font-family: sans-serif;
  height: 0.5em;
  border-bottom: 1pt dotted;
}
ul.cssj-toc ul li {
  margin: 0 0 0.5em 1em;
```

```
  font-family: serif;
}
ul.cssj-toc span {
  background-color: White;
  padding: 0 0.5em;
}
ul.cssj-toc span.cssj-page {
  position: absolute;
  right: 0;
}
```

4.6.3 リンクとフラグメント

フラグメント識別子によるリンク

output.pdf.hyperlinks[io] をtrueに設定すると、文書内のハイパーリンクがPDFに反映されます。

デフォルトでは、ハイパーリンクは文書のURIに対する相対アドレスに変換されます。ハイパーリンクのベースとなるURIを文書のURI以外にする場合は、output.pdf.hyperlinks.base[io] にURIを設定することで、変更することができます。これらの場合、PDFファイルの位置が適切でないと、リンクは意味を持ちません。

output.pdf.hyperlinks.href[io] にabsoluteを指定すると、文書内リンクを除いて、全て絶対リンクになります。この場合、PDFファイルの配置場所を変えてもリンクが切れることはありません。

output.pdf.hyperlinks.fragment[io] をtrueに設定すると、ドキュメントの一部に対してフラグメント識別子によるリンクが可能になります。フラグメントの作成方法は通常のHTMLと同じで、任意の要素のid属性と、a要素のname属性をサポートしています。

-cssj-page-ref関数

Copper PDFは、フラグメントが存在するページを表示するために、content[css]内で使用できる-cssj-page-ref関数を用意しています。

例 4.11 フラグメントのページ番号の表示

```
@page {
    counter-increment: page;
}
span:before {
    content: 'p' -cssj-page-ref(frag, page, decimal, ', p') ' ';
}
a:after {
    content: ' (' -cssj-page-ref(attr(href), page, decimal, ', ') '
ページ)';
}
```

-cssj-page-ref関数の最初の引数は、参照するフラグメント識別子です。2番目の引数はページ番号の表示に使うページカウンタです。3番目の引数は数字のスタイルで、`list-style-type` [css]で使われるのと同じスタイル名です。3番目の引数を省略するとdecimalスタイルで表示されます。4番目の引数(Copper PDF 2.0.2以降)は同じ名前のフラグメントが複数存在する場合、複数のページ番号を表示するための区切り記号です。4番目の引数を省略すると、最初のフラグメントのページ番号だけが表示されます。

フラグメント識別子の部分はattr関数を使うことができます。上の例のように、HTMLのa要素でhref属性の値を使うと、フラグメントへリンクすると同時に、フラグメントが存在知るページの番号を表示することができます。なお、フラグメント識別子の最初の'#'はあってもなくても構いません。

-cssj-page-ref関数を使うためには、一般的に2パス以上の処理の処理が必要です。ドキュメント全体の処理が一度終わらないと、ドキュメントの後の方にあるフラグメントのページが分からないためです。

ドライバで複数の結果を結合する機能を用いる場合、他のドキュメントのページ番号を得るには、-cssj-page-refの最初の引数に'document.html#frag'のようなURIを指定してください。*[3.0.14][3.1.0]*

4.6.4 注釈

cssj:annot属性は、Copper PDFを利用するアプリケーションに、メッセージを送るものです。例えば、日付順に内容が並んでいる文書を処理する場合に、cssj:annot属性に日付を入れることで、ドキュメント中でCopper PDFが現在処理中の部分をアプリケーションが知ることができます。

4.6.5 XML中でHTMLの要素と属性を使用する

名前空間 http://www.w3.org/1999/xhtml を使用することで、画像(img要素)、テーブルセルの結合(colspan, rowspan属性)といった HTMLの一部の要素や属性はXML内でも使用することができます。

利用可能な要素や属性のリストは資料集のXMLの拡張の章を参照してください。

4.6.6 ルビ*[3.0.0]*

ルビはHTMLのruby, rb, rt, rp要素により表示可能です。rbがルビが振られる部分、rtがルビです。rpにはルビをサポートしない環境のためのカッコなどを記述します。

例 4.12 ルビ（ソース）

```
<ruby>
<rb>東海林</rb>
<rp>（</rp>
<rt>しょうじ</rt>
<rp>）</rp>
<rb>太郎</rb>
<rp>（</rp>
<rt>たろう</rt>
<rp>）</rp>
</ruby>
```

図 4.19 ルビ(表示結果)

東海林(しょうじ)太郎(たろう)

前記の記述は、さらに簡略化して次のように書くことができます。ruby中に直接テキストを記述した場合、自動的にrbが上に挿入されます。またrpを省略した場合、ルビをサポートしない環境ではカッコは表示されません。

例 4.13 簡単なルビ（ソース）

```
<ruby>
東海林
<rt>しょうじ</rt>
太郎
<rt>たろう</rt>
</ruby>
```

ルビは本文の半分のサイズで表示されます。また、ルビのある行ではルビの配置のために行間が拡張されます。

ルビのために拡張される行幅は、ruby要素の`line-height` [css]によって決まります。明示しなければ、横書きでは1.414（白銀比）、縦書きでは1.618（黄金比）という値になります。行幅を一定にするためには、本文の`line-height` [css]をこれより大きくとって、あらかじめ行幅を確保しておいてください。あるいは、次のようにruby要素に対する`line-height` [css]を明示して、行幅に合わせてください。

例 4.14 ルビの幅を行幅に合わせるCSS

```
body {
    writing-mode: vertical-rl;
    line-height: 1.5;
}
ruby {
    line-height: 1.5;
}
```

4.7 CSSによるドキュメントのレイアウト

Copper PDFはHTMLとXMLをCSSによってスタイル付けすることができます。

Copper PDFは CSS Level 2 Revision 1 に準拠しています。Copper PDFがサポートするCSSプロパティの一覧は資料集を参照してください。

4.7.1 スタイルシートの型

Copper PDFは、特に型が指定されていないスタイルシートをCSS(MIME型text/css)として認識します。Content-Style-Type ヘッダ (HTML中では <meta http-equiv="Content-Style-Type" content="... といった記述) やstyle要素のtype属性等によってtext/css以外がスタイルシートの型として指定されている場合は、スタイルシートをCSSとして認識しません。

4.7.2 メディアタイプ

style要素のmedia属性、あるいはCSSの@media指示子等でスタイルシートが適用されるメディアタイプが限定されている場合、通常Copper PDFは印刷メディア向けのスタイルだけを適用します。すなわち、Copper PDFのメディアタイプはall print paged visual bitmap staticのいずれかです。

なお、Copper PDFのユーザーエージェントのメディアタイプは `output.media_types`[io] 入出力プロパティによって変更することができます。適用するCSS 2.1のメディアタイプ名またはメディアグループ名を列挙してください。例えば、ブラウザの画面表示用のスタイルを適用する場合は"all visual bitmap static screen continuous"を設定してください。

4.7.3 ドキュメント中にスタイルシートを記述する

ドキュメント中にスタイルおよびスタイルシートを記述する方法は以下の3通りがあります。

1. HTMLのstyle属性
2. HTMLのstyle要素
3. jp.cssj.stylesheet処理命令

HTMLのstyle属性

style属性により、各要素に直接スタイルを指定する方法です。style属性には、適用したいCSSのプロパティを記述してください。

例 4.15 style属性によるスタイルの指定(ソース)

```
<html>
<body style="border: 1px Black dashed;">
<h1 style="font-style: italic;">題名</h1>
<p style="text-decoration: underline;">本文<span style="font-size:
x-large;">大きなテキスト</span>本文</p>
</body>
</html>
```

上記の例は以下のように表示されます。

例 4.16 style属性によるスタイルの指定(表示結果)

題名

本文大きなテキスト本文

style要素はHTMLだけではなく、XML文書内でも利用可能です。ただし、style要素が名前空間"http://www.w3.org/1999/xhtml" に属するように、名前空間宣言を行ってください。

例 4.17 style属性によるスタイルの指定(XML)

```
<?xml version="1.0">
<body xmlns:html="http://www.w3.org/1999/xhtml" html:style="border:
 1px Black dashed;">
<h1 html:style="font-style: italic;">題名</h1>
<p html:style="text-decoration: underline;">本文<span
html:style="font-size: x-large;">大きなテキスト</span>本文</p>
</body>
```

HTMLのstyle要素

一般的なブラウザと同じくstyle要素内にスタイルシートを記述することができます。

例 4.18 style要素によるスタイルシートの記述

```
<html>
<head>
<style type="text/css" media="print">
    body {
        border: 1px Black dashed;
    }
    h1 {
        font-style: italic;
    }
    p {
        text-decoration: underline;
    }
```

```
    span {
        font-size: x-large;
    }
</style>
</head>
<body>
<h1>題名</h1>
<p>本文<span>大きなテキスト</span>本文</p>
</body>
</html>
```

style要素はなるべくhead要素内に記述してください。Copper PDFは他の場所のstyle要素も認識しますが、ドキュメント中でstyle要素が現れる以前の部分には、記述されたスタイルが適用されなくなります。

また、XMLドキュメント中ではstyle要素によるスタイル指定は行わず、次に説明するjp.cssj.stylesheet処理命令を使用するか、あるいは一般的なブラウザでもサポートされているxml-stylesheet処理命令を使用してください。

jp.cssj.stylesheet処理命令

jp.cssj.stylesheet処理命令はHTMLのstyle要素に相当する機能を処理命令 (<?で始まり?>で終わるHTMLやXML中の特別な記述。) によって実現したものです。HTML, XMLの両方のドキュメントの先頭で使用することができます。

style要素のtype, media属性に相当する部分はHTMLの属性と同様の形式で記述し、スタイルシートは'[]'で囲みます。

例 4.19 jp.cssj.stylesheet処理命令によるスタイルシート記述

```
<?jp.cssj.stylesheet type="text/css" media="print" [
    body {
        border: 1px Black dashed;
    }
    h1 {
        font-style: italic;
    }
    p {
        text-decoration: underline;
    }
```

```
    span {
        font-size: x-large;
    }
] ?>
<html>
<body>
<h1>題名</h1>
<p>本文<span>大きなテキスト</span>本文</p>
</body>
</html>
```

4.7.4 外部のCSSの使用

ドキュメントと外部のスタイルシートを結びつける方法は以下の3通りがあります。

1. HTMLのlink要素
2. CSSの@import指示子
3. xml-stylesheet処理命令

HTMLのlink要素

一般的なブラウザと同様に、link要素によってスタイルシートをドキュメントに関連付けることができます。

例 4.20 linkタグによるスタイルシートの指定

```
<link rel="stylesheet" type="text/css" media="print" href="スタイルシートのURL">
```

rel属性にキーワードalternateを加えることで、1つのドキュメントに対して、複数の代替スタイルシートを用意することができます。title属性は代替スタイルシートの名前です。

例 4.21 alternateスタイル

```
<link rel="alternate stylesheet" title="a" type="text/css"
media="print" href="a.css">
<link rel="alternate stylesheet" title="b" type="text/css"
media="print" href="b.css">
```

代替スタイルシートは通常は適用されませんが、input.stylesheet.titles[io] に代替スタイルシートのtitleを設定することで、適用するスタイルシートを選択することができます。

CSSの@import指示子

これは、HTMLのstyle要素とCSSの@importを組み合わせる方法です。

例 4.22 @import指示子によるスタイルシートの指定

```
<style type="text/css" media="print">
    @import url(スタイルシートのURL);
</style>
```

HTMLのstyle要素の代わりに、jp.cssj.stylesheet処理命令を使うこともできます。

例 4.23 @import指示子によるスタイルシートの指定(jp.cssj.stylesheet処理命令)

```
<?jp.cssj.stylesheet type="text/css" media="print" [
    @import url(スタイルシートのURL);
] ?>
```

xml-stylesheet処理命令

xml-stylesheet処理命令を使う方法は、一般的なブラウザでXMLとスタイルシートを結びつけるために広くサポートされている方法です。

スタイルシートを適用したいドキュメントの冒頭で、以下のように記述してください。

例 4.24 xml-stylesheet処理命令によるスタイルシートの指定

```
<?xml-stylesheet type="text/css" media="print" href="スタイルシートの
URL"?>
```

alternate属性はlink要素のrel属性に対するalternateの指定に相当するものです。xml-stylesheet処理命令でalternate="yes"を指定することは、link要素にrel="alternate stylesheet"を設定するのと同じ意味です。

例 4.25 xml-stylesheet処理命令による代替スタイルシート

```
<?xml-stylesheet alternate="yes" title="a" type="text/css"
media="print" href="a.css"?>
<?xml-stylesheet alternate="yes" title="b" type="text/css"
media="print" href="b.css"?>
```

xml-stylesheetのtype属性にはCSSだけでなくXSLTスタイルシートを指定することもできます。この場合、type属性にtext/xslを設定し、href属性にはXSLTファイルの指定してください。

4.7.5 デフォルトのスタイルシート

Copper PDFでは、ドキュメント内の記述に関係なく、特定のスタイルシートを適用する機能が用意されています。`input.default-stylesheet`[io] 入出力プロパティで指定されたスタイルシートが最初に適用されます。

4.7.6 長さの単位

CSSでは、絶対単位としてmm, cm, in, pt, pcが使われます。各絶対単位は、Copper PDFはCSSの仕様どおり次の一定の長さでレイアウトします。PDFではptが長さの単位の基準となり、Copper PDFでも内部的な単位としてptを使用しています。以下は各単位の関係です。

mm

1mm = 1/25.4in　2.835pt

cm

1cm = 10mm　28.35pt

in

1in = 72pt = 25.4mm

pt

1pt = 1/72in　0.353mm

pc

1pc = 1/6in = 12pt　4.233mm

相対単位としてem, ex, pxがあり、これは状況によって変化します。

em, exはそれぞれフォントの高さとxの高さを基準とした単位です。正確には、文書中の基準となる位置で利用可能なフォントの高さの最大値と、xの高さの最大値を基準とします。xという文字が存在しないフォントでは、exはxの高さではない「適当な高さ」となります。

pxは、`output.resolution`[io]で設定される解像度が基準となります。デフォルトでは96dpiで、この状況では1pt　1.33pxとなり、1pxは1ptより若干小さくなります。この値は、一般的なブラウザに合わせたものです。

ラスタ画像として結果を出力する場合、デフォルトでは出力結果の1ピクセルは常にCSSの1pxに一致します。詳細は画像の出力 (180ページ)を参照してください。

4.8 CSSの拡張機能

4.8.1 名前空間

Copper PDFはCSSの名前空間のための拡張機能 (CSS Namespace Enhancements) をサポートしており、複数の名前空間が混在するXMLをスタイル付けすることができます。

CSSスタイルシート中で名前空間の接頭辞(prefix)とURIを指定するためには、@namespace指示子を使って以下のように宣言してください。

例 4.26 名前空間の宣言

```
/* デフォルトの名前空間のURIをhttp://www.w3.org/1999/xhtmlとする。 */
@namespace "http://www.w3.org/1999/xhtml";

/* 接頭辞rdfのURIをhttp://www.w3.org/1999/02/22-rdf-syntax-ns#とする。
*/
@namespace rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#";
```

なお、互換性のためにURIの部分はurl(http://www.w3.org/1999/xhtml)という書き方も許されています。

スタイルシートの選択子(selector)で接頭辞を使う場合は、'|'で区切ります。(':'でないことに注意してください。)

例 4.27 接頭辞を含む選択子

```
/* <pdf:Description>要素のスタイルを指定する。 */
rdf|Description { display: block; }

/* ref:about属性がhttp://foo.com/barであるitem要素のスタイルを指定する。
*/
item[rdf|about=http://foo.com/bar] { color: Red; }
```

選択子の記述方法と、意味は次のとおりです。

prefix|ELEMENT
: prefixが指す名前空間に属するELEMENT

|ELEMENT
: どの名前空間にも属さないELEMENT

***|ELEMENT**
: 任意の名前空間にも属すか、どの名前空間にも属さないELEMENT

ELEMENT
: デフォルトの名前空間に属するELEMENT

4.8.2 ページカウンタ

Copper PDFは、ページの番号付けのために ページの生成ごとに処理されるページカウンタを用意しています。ページカウンタは、以下のように@page ルール内の counter-increment [css] プロパティにより宣言します。

例 4.28 ページカウンタ

```
@page {
    counter-increment: page;
}
```

ページカウンタの処理は、ページの内容が処理される直前に行われます。また、ページカウンタは通常のカウンタと同様に、content [css] プロパティ内でcounter関数により参照可能です。従って、上記の宣言を行った場合、最初のページで {content: counter(page);} という宣言が処理されるとき、1が出力されます。

途中でページカウンタをリセットする場合 (例えば、目次が終わった後、本文で改めて番号を振りなおすなど) は、通常のカウンタと同様にcounter-reset [css] を使うことができます。例えば、ある要素が表示されるページでpageという名前のカウンタを1に設定しなおす場合は、その要素で {counter-reset: page 1;} と宣言します。

4.8.3 全角数字と漢数字による箇条書き番号

Copper PDFは、箇条書きの先頭に付けるマーカー文字の形式を拡張しています。list-style-type [css] で、次のキーワードを利用可能です。

-cssj-full-width-decimal *[3.0.0]*
: マーカーに全角数字を使います。

-cssj-cjk-decimal *[3.0.0]*
: マーカーに位取り漢数字を使います。

-cssj-decimal-full-width *[2.1.2]*
: -cssj-full-width-decimalと同じです。

例 4.29 マーカーの形式（ソース）

```
<html>
  <head>
    <style type="text/css">
    #a {
      list-style-type: -cssj-full-width-decimal;
    }
    #b {
      list-style-type: -cssj-cjk-decimal;
    }
    </style>
  </head>
  <body>
    <ol id="a">
      <li>田作り</li>
      <li>黒豆</li>
      <li>栗きんとん</li>
    </ol>
    <ol id="b">
      <li>かまぼこ</li>
      <li>伊達巻</li>
      <li>数の子</li>
    </ol>
  </body>
</html>
```

例 4.30 マーカーの形式（表示結果）

1. 田作り
2. 黒豆
3. 栗きんとん

一、かまぼこ
二、伊達巻
三、数の子

`content` [css]のcounter関数でも同じキーワードを使うことができます。

例 4.31 counterの形式（ソース）

```
<html>
  <head>
    <style type="text/css">
    #a:before {
      counter-increment: a;
      content: counter(a, -cssj-full-width-decimal);
    }
    #b:after {
      counter-increment: b;
      content: counter(b, -cssj-cjk-decimal);
    }
    </style>
  </head>
  <body>
    <div id="a">田作り</div>
    <div id="a">黒豆</div>
    <div id="a">栗きんとん</div>
    <div id="b">かまぼこ</div>
    <div id="b">伊達巻</div>
    <div id="b">数の子</div>
  </body>
</html>
```

例 4.32 counterの形式（表示結果）

１田作り
２黒豆
３栗きんとん
かまぼこ一
伊達巻二
数の子三

4.8.4 禁則処理

行頭禁則文字（直前での折り返しをしない文字）は、全角スペースと次の文字です。

```
〜〜ヽゞゝゞ々ーぁぃぅぇぉっゃゅょゎァィゥェォッャュョヮカケ・,
) ] } 、 〕 〉 》 」 』 】 ゛ ゜ ′ ″ » ‐ ゠ – 〜 ? ! ‼ ⁇ ⁈ ⁉ ・ , . : ; / ％
```

さらに、UnicodeのEND_PUNCTUATION （閉じ括弧類）、OTHER_PUNCTUATION （その他の括弧類）、MODIFIER_LETTER（修飾文字）、MODIFIER_SYMBOL（修飾記号）が行頭禁則文字とされます。

行末禁則文字（直後での折り返しをしない文字）は次の文字です。

```
( [ { 〔 〈 《 「 『 【 ゛ ゜ ′ ″ «
```

さらに、UnicodeのSTART_PUNCTUATION （開き括弧類）が行末禁則文字とされます。

半角英数字の間でも折り返しが禁止されます。ただし、半角スペースと、次の文字の間では折り返しされます。

```
-!?
```

また、禁則処理により折り返しをできない区間は48文字を超えることはできません(Copper PDF 3.0.0からは、この仕様はなくなりました。次のword-wrap[css]を使ってください)。

word-wrap *[3.0.0]*

word-wrap[css] はCSS3 Text の先行実装です。仕様は次のとおりです。

値

normal | break-word

初期値

normal

適用対象

すべて

値の継承

する

break-wordを設定すると、内容が行幅の限界をはみ出さないように、必要に応じて禁則処理されている部分での折り返しをします。

例 4.33 word-wrapの使用例（ソース）

```
<html>
  <head>
    <style type="text/css">
    div {
      width: 6ex;
      border: 1pt solid Red;
    }
    #a {
      word-wrap: normal;
    }
```

```
    #b {
      word-wrap: break-word;
    }
    </style>
  </head>
  <body>
    <div id="a">Distance lends enchantment to the view.</div>
    <div id="b">Distance lends enchantment to the view.</div>
  </body>
</html>
```

例 4.34 word-wrapの使用例（表示結果）

Distance lends enchantment to the view.

Distance lends enchantment to the view.

禁則文字を新たに追加・除外するために、`-cssj-no-break-characters`[css], `-cssj-break-characters`[css]という独自CSSプロパティを用意しています。

-cssj-no-break-characters*[3.0.6]*

値

none | <string>{1,2}

初期値

none

適用対象

すべて

値の継承

する

禁則文字を追加します。１つめの<string>は行頭禁則文字、２つめの<string>行末禁則文字を指定します。<string>が１つだけの場合は行頭禁則文字だけが追加されます。

例えば、次の文章があるとします。

例 4.35 -cssj-no-break-characters適用前（ソース）

```
<div style="border:1px solid; width: 7em;">
今日の相場は１＄がロンドンで９８円５５銭－先月と比べて２㌫上昇しました。
</div>
```

例 4.36 -cssj-no-break-characters適用前（表示結果）

今日の相場は１＄がロンドンで９８円５５銭－先月と比べて２㌫上昇しました。

'㌫'と'＄'を行頭に表示させたくなく、'－'を行末に表示させたくないという場合は、次のように指定してください。

例 4.37 -cssj-no-break-characters適用後（ソース）

```
<div style="border:1px solid; width: 7em; -cssj-no-break-
characters: '㌫＄' '－';">
今日の相場は１＄がロンドンで９８円５５銭－先月と比べて２㌫上昇しました。
</div>
```

例 4.38 -cssj-no-break-characters適用後（表示結果）

今日の相場は１＄がロンドンで９８円５５銭－先月と比べて２㌫上昇しました。

-cssj-break-characters*[3.0.6]*

値

none | <string>{1,2}

初期値

none

適用対象

すべて

値の継承

する

禁則文字を除外します。１つめの<string>は行頭禁則文字、２つめの<string>行末禁則文字を指定します。<string>が１つだけの場合は行頭禁則文字だけが除外されます。

例えば、Copper PDFは次のように、拗音が行頭に来ないように調整します。

例 4.39 -cssj-break-characters適用前（ソース）

```
<div style="border:1px solid; width: 10em;">
「トロメライ、ロマチックシューマン作曲。」猫は口を拭いて済まして云いました。
</div>
```

例 4.40 -cssj-break-characters適用前（表示結果）

「トロメライ、ロマチックシューマン作曲。」猫は口を拭いて済まして云いました。

しかし、実際に書籍は拗音を行頭禁則しないことが多いため、それに従うには次のようにします。

例 4.41 -cssj-break-characters適用後（ソース）

```
<div style="border:1px solid; width: 10em; -cssj-break-characters:
'ア イ ウ エ オ ツ ヤ ユ ヨ ワ カ ケ あ い う え お つ や ゆ よ わ 309F 30FF 31F0
31F1 31F3 31F2 31F3 31F4 31F5 31F6 31F7 31F8 31F9 31FA 31FB 31FC 31FD 31FE 31FF ';">
「トロメライ、ロマチックシューマン作曲。」猫は口を拭いて済まして云いました。
</div>
```

例 4.42 -cssj-break-characters適用後（表示結果）

「トロメライ、ロマチックシューマン作曲。」猫は口を拭いて済まして云いました。

4.8.5 圏点*[3.0.4]*

主に日本語の文章の一部を強調するために使われる、圏点を打つことができます。圏点のためには、CSS3 Text の先行実装である、`-cssj-text-emphasis-style` [css], `-cssj-text-emphasis-color` [css], `-cssj-text-emphasis` [css] というCSSプロパティが使われます。

EPUB との互換性のため、これらのプロパティはそれぞれ、`-epub-text-emphasis-style` [css], `-epub-text-emphasis-color` [css], `-epub-text-emphasis` [css] という名前も使うことができます。

圏点の記号は本文のフォントを使って表示されます。より美しい圏点を出力するためには Kenten Generic OpenType Font(https://github.com/adobe-fonts/kenten-generic) を埋め込みフォントとして使用することを推奨します。本文のフォントファミリを `{font-family: 'Kenten Generic' 本文フォント...;}` のように指定してください。

-cssj-text-emphasis-style

値

none | [[filled | open] || [dot | circle | double-circle | triangle | sesame]] | <string>

初期値

none

適用対象

すべて

値の継承

する

noneが設定された場合、圏点を打ちません。

その他の値が設定された場合、圏点の種類（ユニコード 文字）はそれぞれ次のとおりになります。

filled dot

U+2022 ‘•’

open dot

U+25E6 ‘◦’

filled circle

U+25CF ‘●’

open circle

U+25CB ‘○’

filled double-circle

U+25C9 ‘◉’

open double-circle

U+25CE ‘◎’

filled triangle

U+25B2 ‘▲’

open triangle

U+25B3 ‘△’

filled sesame

U+FE45 ‘﹅’

open sesame

U+FE46 ‘﹆’

filledかopenだけが指定された場合は、横書きではそれぞれ filled circle, open circle が指定されるのと同じになり、縦書きではそれぞれ filled sesame, open sesame が指定されるのと同じになります。

文字列が指定された場合、文字列の最初の文字が圏点になります。

例 4.43 -cssj-text-emphasis-styleの使用例（ソース）

```
<html>
  <head>
    <style type="text/css">
    #a {
      -cssj-text-emphasis-style: filled;
    }
    #b {
      -cssj-text-emphasis-style: open triangle;
    }
    #c {
      -cssj-text-emphasis-style: '※';
    }
    </style>
  </head>
  <body>
<p><span id="a">ここ</span>に丸い圏点を打ちます</p>
<p><span id="b">ここ</span>に三角の圏点を打ちます</p>
<p><span id="c">ここ</span>に米印の圏点を打ちます</p>
  </body>
</html>
```

例 4.44 -cssj-text-emphasis-styleの使用例（表示結果）

ここに丸い圏点を打ちます

ここに三角の圏点を打ちます

ここに米印の圏点を打ちます

-cssj-text-emphasis-color

値

<color>

初期値

文字の色と同じ

適用対象

すべて

値の継承

する

圏点の色を指定します。色の指定方法はcolor[css]プロパティなどの場合と同じです。

例 4.45 -cssj-text-emphasis-colorの使用例（ソース）

```
<html>
  <head>
    <style type="text/css">
    span {
      -cssj-text-emphasis-style: filled;
    }
    #a {
      -cssj-text-emphasis-color: Red;
    }
    #b {
      color: Red;
    }
    </style>
  </head>
  <body>
<p><span id="a">この</span>圏点は赤いです</p>
<p><span id="b">この</span>圏点も文字も赤です</p>
  </body>
</html>
```

例 4.46 -cssj-text-emphasis-colorの使用例（表示結果）

この圏点は赤いです

この圏点も文字も赤です

-cssj-text-emphasis

-cssj-text-emphasis-style[css], -cssj-text-emphasis-color[css] をまとめて指定するプロパティです。圏点のスタイル、色の順に指定します。

例 4.47 -cssj-text-emphasisの使用例（ソース）

```
<html>
  <head>
    <style type="text/css">
    #a {
      -cssj-text-emphasis: filled triangle Red;
    }
    #b {
      -cssj-text-emphasis: '※' Pink;
    }
    </style>
  </head>
  <body>
<p><span id="a">この</span>圏点は赤い三角です</p>
<p><span id="b">この</span>圏点はピンクの米印です</p>
  </body>
</html>
```

例 4.48 -cssj-text-emphasisの使用例（表示結果）

この圏点は赤い三角です

この圏点はピンクの米印です

4.8.6 文字の影*[3.0.8]*

Copper PDFは、一般的なブラウザがサポートしている「文字の影落とし」をサポートしています。ただし、制約として影の「ぼかし」には対応していません。常にぼかしの半径にゼロを設定したのと同じになります。

text-shadow

値

none | [<length>{2} && <color>?]#

初期値

none

適用対象

すべて

値の継承

する

`text-shadow`[css]の値は、影のx方向の位置、y方向の位置、影の色の順に指定します。また、カンマで区切ることにより、複数の影を落とすことができます。1つ目の影は文字の後ろに作られ、順に背面へと作られます。

例 4.49 text-shadowの使用例（ソース）

```
<html>
  <head>
    <style type="text/css">
    * {
      font-size: 32pt;
    }
    #a {
      text-shadow: 4pt 8pt Gray;
    }
    #b {
      text-shadow: 8pt 8pt Gray, 16pt 16pt LightGray;
    }
    </style>
  </head>
  <body>
<p id="a">灰色の影のある文字</p>
<p id="b">灰色の影の後ろにさらに薄い灰色の影</p>
  </body>
</html>
```

例 4.50 text-shadowの使用例（表示結果）

灰色の影のある文字

灰色の影の後ろにさらに薄い灰色の影

4.8.7 袋文字*[3.0.8]*

レンダリングエンジンとしてWebKitを利用しているブラウザ（Google Chrome, Safari等）との互換性のために、テキストの輪郭と塗りを別々に指定するプロパティを用意しています。これは袋文字の効果を実現するために使うことができます。

`-cssj-text-fill-color`[css], `-cssj-text-stroke-width`[css], `-cssj-text-stroke-color`[css], `-cssj-text-stroke`[css] という４つの独自プロ

パティを用意しています。これらは、WebKit との互換性のために、それぞれ `-webkit-text-fill-color` [css], `-webkit-text-stroke-width` [css], `-webkit-text-stroke-color` [css], `-webkit-text-stroke` [css] というプロパティ名でも利用することができます。

-cssj-text-fill-color

値

<color> | currentcolor

初期値

currentcolor

適用対象

すべて

値の継承

する

テキストの塗りつぶし色を設定します。なにも指定しない場合は、`color` [css]による指定と同じになります。

-cssj-text-stroke-width

値

<width> | medium | thick | thin

初期値

0

適用対象

すべて

値の継承

する

テキストの枠の幅をしていします。0以外の値を設定すると、枠が描画されるようになります。

-cssj-text-stroke-color

値

<color> | currentcolor

初期値

currentcolor

適用対象

すべて

値の継承

する

テキストの枠の色を設定します。なにも指定しない場合は、color [css]による指定と同じになります。

-cssj-text-stroke

値

<width> <color>

初期値

none

適用対象

すべて

値の継承

する

テキストの枠の幅と色を一度に設定します。

例 4.51 袋文字（ソース）

```
<html>
  <head>
    <style type="text/css">
    * {
      font-size: 32pt;
    }
    #a {
      -cssj-text-stroke-width: 2pt;
    }
    #b {
      -cssj-text-stroke: 1pt Black;
      -cssj-text-fill: White;
    }
    </style>
  </head>
  <body>
<p id="a">輪郭を太らせた文字</p>
<p>テキストの<span id="b">この部分</span>が白抜き</p>
  </body>
</html>
```

例 4.52 袋文字（表示結果）

輪郭を太らせた文字

文章のこの部分が白抜き

4.8.8 透明化*[3.0.6]*

opacity

値

<alphavalue>

初期値

1

適用対象

すべて

値の継承

しない

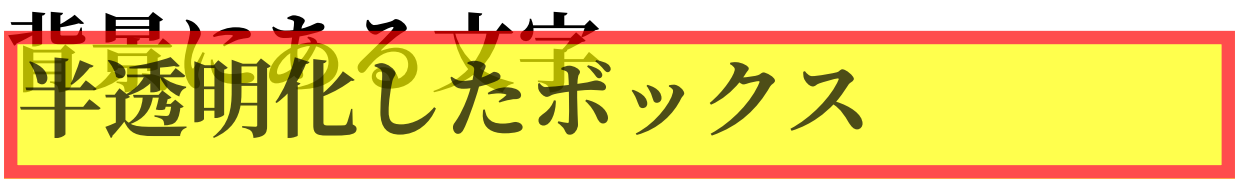
要素の不透明度を設定します。値は0(透明)から1(不透明)までの小数値です。例えば0.5に設定すると、その要素は50%の透明度を持つことになります。

例 4.53 opacityの使用例（ソース）

```
<html>
  <head>
    <style type="text/css">
    * {
      font-size: 32pt;
    }
    #a {
      position: relative;
      top: -32pt;
      font-size: 32pt;
      border: 5pt solid Red;
```

```
      background-color: Yellow;
      opacity: 0.7;
    }
    </style>
  </head>
  <body>
<div>輪郭を太らせた文字</div>
<div id="a">透明化したボックス</div>
  </body>
</html>
```

例 4.54 opacityの使用例（表示結果）



4.8.9 透明色*[3.0.8]*

前記のopacity[css]は要素全体を半透明にするものでしたが、これとは別に、rgba関数により各指定色に透明度を加えることができます。これにより、例えば文字、境界線に別々の透明度を指定することができます。rgba関数は色指定をするcolor[css], background-color[css], border-color[css]等のプロパティで使用することができます。rgba関数内の数値は、それぞれ赤、緑、青、不透明度の順です。不透明度は0から1までの小数で指定します。

例 4.55 rgbaの使用例（ソース）

```
<html>
  <head>
    <style type="text/css">
    * {
      font-size: 32pt;
    }
    #a {
      position: relative;
      top: -32pt;
      font-size: 32pt;
```

```
      border: 5pt solid Red;
      background-color: rgba(255,255,0,0.5);
    }
    </style>
  </head>
  <body>
<div>輪郭を太らせた文字</div>
<div id="a">透明化したボックス</div>
  </body>
</html>
```

例 4.56 rgbaの使用例（表示結果）



4.8.10 角丸境界*[3.0.6]*

ボックスの境界線の四隅を丸くすることができます。

Copper PDF 3.0.8からは、背景の四隅も丸くなります。また、既知の問題として、４つの境界線の色・太さ・スタイル・角の半径が異なる場合、背景が境界をはみ出すことがあります。

角の半径を指定するために、`border-*-*-radius` [css]という形式の４つのプロパティを使います。これらのプロパティは１つまたは２つの長さ、%値を指定し、１つの場合は角の半径、２つの場合は角を楕円形の弧としそれぞれ左右方向と上下方向の半径となります。

border-top-left-radius

値

[<length> | <percentage>]{1,2}

初期値

0

適用対象

すべて

値の継承

しない

border-top-right-radius

値

[<length> | <percentage>]{1,2}

初期値

0

適用対象

すべて

値の継承

しない

border-bottom-left-radius

値

[<length> | <percentage>]{1,2}

初期値

0

適用対象

すべて

値の継承

しない

border-bottom-right-radius

値

[<length> | <percentage>]{1,2}

初期値

0

適用対象

すべて

値の継承

しない

`border-radius`[css]は前記のプロパティをまとめて指定するためのものです。値はそれぞれ、`border-top-left-radius`[css]、`border-top-right-radius`[css]、`border-bottom-right-radius`[css]、`border-bottom-left-radius`[css] の順に指定します。４番目の値が省略された場合、`border-bottom-left-radius`[css]は

border-top-right-radius[css]と同じになります。３番目の値が省略された場合、border-bottom-right-radius[css]はborder-top-left-radius[css]と同じになります。２番めの値が省略された場合、全ての隅が同じ指定になります。

左右方向と上下方向の半径は'/'(スラッシュ)記号で区切り、それぞれをまとめて指定します。

border-radius

値

[<length> | <percentage>]{1,4} [/ [<length> | <percentage>]{1,4}]?

適用対象

すべて

値の継承

しない

例 4.57 角丸境界（ソース）

```
<html>
  <head>
    <style type="text/css">
    div {
      border: 1pt solid Red;
      background-color: Yellow;
      height: 50pt;
    }
    #a {
      border-radius: 5pt;
    }
    #b {
      border-radius: 10pt 20pt / 20pt 10pt;
      border-top-left-radius: 30pt;
    }
    </style>
  </head>
  <body>
<div id="a">四隅が丸いボックス</div>
<div id="b">四隅がいびつなボックス</div>
  </body>
</html>
```

例 4.58 角丸境界（表示結果）

四隅が丸いボックス

四隅がいびつなボックス

4.8.11 回転・拡大・変形*[3.0.8]*

ボックスに対して二次元変換により回転, 拡大, 変形などを行うことができます。なお、三次元変換には対応していません。

`-cssj-transfrom`[css], `-cssj-transform-origin`[css] というプロパティで変換を行います。WebKit, Firefox との互換性のために、`-webkit-transfrom`[css], `-webkit-transform-origin`[css], `-moz-transfrom`[css], `-moz-transform-origin`[css] という名前でも使用することができます。

-cssj-transform

値

none | <transform-function> [<transform-function>]*

初期値

none

適用対象

ブロックレベル要素

値の継承

しない

ボックスに対する変換行列を指定します。値には, noneまたは複数の変換関数を指定することができます。利用できる変換関数は次のとおりです。

matrix(a,b,c,d,e,f)

変換行列を直接指定します。

変換行列は次のとおりに指定されます。

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

aは左右方向の拡大率、bは上下方向の傾斜率、cは左右方向の傾斜率、dは上下方向の拡大率、eは左右方向の移動距離、fは上下方向の移動距離に対応します。e, fにはpt, em等の長さの単位を使用することができます。単位がない場合はpt単位です。なお、%値による指定には対応していません。

translate(x, y)
: 左右方向と上下方向の移動距離を指定します。yを省略した場合は、左右方向のみが指定されます。長さの単位を使用することができます。単位がない場合はpt単位です。

translateX(x)
: 左右方向の移動距離を指定します。長さの単位を使用することができます。単位がない場合はpt単位です。

translateY(y)
: 上下方向の移動距離を指定します。長さの単位を使用することができます。単位がない場合はpt単位です。

scale(x, y)
: 左右方向と上下方向の拡大率を指定します。

scaleX(x)
: 左右方向の拡大率を指定します。

scaleY(y)
: 上下方向の拡大率を指定します。

rotate(theta)
: 回転させます。thetaは角度で、degを付けると度単位、単位を省略するとラジアン単位となります。

skew(xtheta, ytheta)
: 左右方向と上下方向の傾斜角度を指定します。それぞれの値は角度で、degを付けると度単位、単位を省略するとラジアン単位となります。

skewX(xtheta)
: 左右方向の傾斜角度を指定します。xthetaは角度で、degを付けると度単位、単位を省略するとラジアン単位となります。

skewY(ytheta)
: 上下方向の傾斜角度を指定します。ythetaは角度で、degを付けると度単位、単位を省略するとラジアン単位となります。

-cssj-transform-origin

値
: <position> [, <position>]*

初期値

50% 50%

適用対象

ブロックレベル要素

値の継承

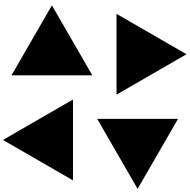
しない

変換の中央点を指定します。デフォルトではボックスの中心になっており、例えばrotate関数を使用するとボックスの中心を回転の中心とします。

例 4.59 transform, transform-originの使用例（ソース）

```
<html>
  <head>
    <style type="text/css">
    body {
      margin: 40pt;
    }
    div {
      font-size: 32pt;
      position: absolute;
      -cssj-transform-origin: 5pt 5pt;
    }
    #a {
      -cssj-transform: rotate(0deg);
    }
    #b {
      -cssj-transform: rotate(90deg);
    }
    #c {
      -cssj-transform: rotate(180deg);
    }
    #d {
      -cssj-transform: rotate(270deg);
    }
    </style>
  </head>
  <body>
<div id="a">▼</div>
<div id="b">▼</div>
<div id="c">▼</div>
<div id="d">▼</div>
  </body>
</html>
```

例 4.60 transform, transform-originの使用例（表示結果）



4.9 ページ処理機能

4.9.1 ページのレイアウト

生成されるページは以下の部分から構成されています。

用紙
ドキュメントが印刷される用紙です。

印刷面
ドキュメントの内容が印刷される部分です。

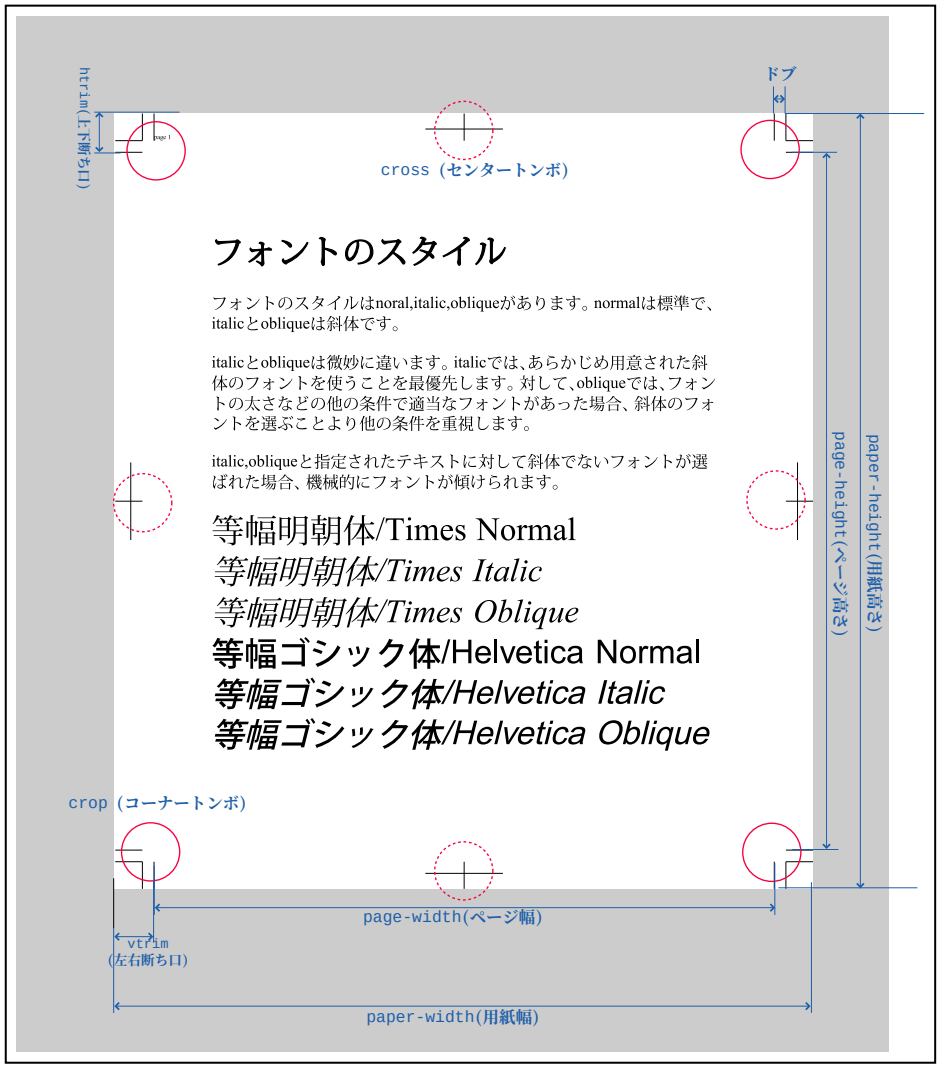
マージン
通常は内容がみだすことのない、ページの余白部分です。

トンボ
製本する際、断裁の目印となる印です。

裁ち口
断裁されて切り落とされる部分です。

ドブ
断裁される可能性のある範囲です。

図 4.20 ページのレイアウト



ページのレイアウトは入出力プロパティにより設定されます。対応する入出力プロパティは次の通りです。

用紙のサイズ

output.paper-width[io], output.paper-height[io]

印刷面のサイズ

output.page-width[io], output.page-height[io]

マージン

output.page-margins[io]

トンボ

output.marks[io]

裁ち口

output.htrim[io], output.vtrim[io]

用紙のサイズの指定がない場合、用紙の幅と高さは、それぞれページの幅と高さに断ち口の幅を足したものに自動的に設定されます。用紙のサイズを指定した場合、output.fit-to-paper[io] がtrueの場合は用紙に合わせて内容が拡大され、falseの場合は中央に寄せられます。

output.page-margins[io] の設定はページマージンのデフォルト値として使用されるものです。ページのマージンは、CSSの margin-top[css], margin-right[css], margin-bottom[css], margin-left[css] プロパティにより@pageルール内で上書きすることができます。

4.9.2 ページの大きさの制約と切り落とし

Copper PDFのデフォルトの設定では、ページの高さは297mm、ページの幅は210mm(A4サイズ)です。上下左右のマージンはいずれも12.7mm(3pc)です。

用紙の縦横の長さはいずれも1ptから14400pt(5080mm)の 間である必要があります。用紙サイズがこの範囲を超えた場合は、超えた部分は切り落とされます。このサイズ制限はPDFの仕様によるものです。

また、output.auto-height[io]をtrueに設定することで、用紙の高さは固定されずに、内容に合わせて拡張されるようになります。この場合、改ページが発生することはありませんが、前記の用紙サイズの制限により切り落とされることがあります。

用紙サイズを固定し、かつ改ページが行われないようにするためには output.no-page-break[io]をtrueに設定してください *[2.0.3]*。この場合、固定された用紙サイズをはみ出した部分は切り落とされます。

印刷内容の切り落としは、トンボがある場合、デフォルトではドブの外側の境界線上で行われます。トンボのさらに外側まで内容を印刷する場合は、output.clip[io]をfalseに設定してください *[2.0.3]*。

4.9.3 2パス以上の変換処理

Copper PDFには、ページ番号による参照を可能にするために、同じドキュメントを複数回処理する(それぞれの処理を「パス」と呼びます)機能が用意されています。複数パスが必要になるのは、ドキュメントの最後以外の場所に目次を入れる場合と、-cssj-page-ref関数を使用する場合です。

文書中にページ番号を表示する場合、1度目のパスでアンカーや見出し等が表示されるページの番号を求めて、2度目のパスで実際に参照のためのページ番号を挿入します。また、目次と本文を通しでページ番号を振る場合や、非常に大きなドキュメントで、ページ番号の挿入のためにページ数が増加する場合は、3パス以上必要になることがあります。

パスの数は入出力プロパティ processing.pass-count[io] により設定可能です。

2パス以上の変換処理は、processing.middle-pass[io] *[3.0.4]* により行うこともできます。この場合、文字通りドキュメントを複数回Copper PDFに送り出す必要があります。実際に結果を出力しない最初と中間のパスの時はprocessing.middle-pass[io]を"true"にしておき、最後にprocessing.middle-pass[io]を"false"にしてからドキュメントを処理してください。

4.9.4 ページの参照

Copper PDFには、目次をつくる機能(cssj:make-toc要素)と、ある内容が印刷されるページ番号を表示する機能(-cssj-page-ref関数)があります。

これらの機能を利用するためには、processing.page-references[io] をtrueに設定し、ページ参照情報を収集する機能を有効にしてください。また、必要に応じて2パス以上の変換処理を行ってください。



CSSJ 1.xのcssj:toc要素、processing.make-toc-before[io]は廃止されました。

4.9.5 グレイスケール印刷

Copper PDFは原則として、結果をカラーで出力しますが、グレイスケールで印刷した状態をプレビューするために、グレイスケールに変換する機能を持っています。

グレイスケールの出力結果を得るためには、output.color[io]をgrayに設定してください

4.9.6 片面印刷と両面印刷

デフォルトではCopper PDFによるページ生成は横書き・両面印刷として行われます。従って、CSSの@pageルールにおいて、最初のページは:firstまたは:right擬似要素として扱われ、以降は:left, :right擬似要素のページが交互に現れます。

output.print-mode[io] にsingle-sideを設定することにより、片面印刷に切り替えることができます。片面印刷では最初のページは:first擬似要素として扱われ、以降はどの擬似要素にも属さないページが生成されます。

4.9.7 ページごとに生成されるコンテンツ

Copper PDFは「ノンブル」のようなページごとに生成されるコンテンツをサポートするために、独自の`-cssj-page-content`[css]プロパティを用意しています。

`-cssj-page-content`[css]は要素をページ毎に生成するボックスとして配置するものです。ボックスの配置方法は`{position: fixed;}`と同じですが、ページ毎にボックスが再生成され、`content`[css]プロパティが処理される点が異なります。

`-cssj-page-content`[css] 最初の値は、生成されるコンテンツの識別名です。2つめの以降の値は、コンテンツを最初のどちらのページに再生成するかの指定です。first(最初), left(左ページ), right(右ページ), single(片面印刷のページ)のいずれかです。2つめの値を省略すると、全てのページで再生成されます。2つめ以降に複数の値を指定すると、いずれかの条件にマッチする場合にページを再生成します。例えばleft singleでは両面印刷の左か、片面印刷のページで再生成します。

`-cssj-page-content`[css]が指定されたボックスは、それが空のインラインボックスであると仮定した場合に表示されるページから表示されます。*[2.0.8]*

識別名は再生成の停止や置き換えのためのものです。`-cssj-page-content-clear`[css] により、ボックスの再生成を止めることができます。また、`-cssj-page-content`[css]で指定した名前が既に使われている場合、古いボックスの再生成を止め、新しいボックスに置き換えられます。

`-cssj-page-content-clear`[css]は、プロパティを設定したオブジェクトが表示されたページから適用されます。*[2.0.8]*

次の例は再生成ボックスとページカウンタ (201ページ)を利用して、ページごとに「ノンブル」を振ります。各ページの下部中央にページ番号が表示されます。また、本文の前でページカウンタをリセットし、目次と本文は別にページを振りなおしています。

例 4.61 ページ番号の生成

```
<style type="text/css">
  @page {
    margin: 2cm 2cm 5cm 2cm;
    counter-increment: page;
  }
  #page-number {
    -cssj-page-content: footer;
    bottom: -1cm;
    text-align: center;
    width: 100%;
  }
  #page-number:before {
```

```
    content: counter(page);
  }
  #body {
    page-break-before: right;
    counter-reset: page 1;
  }
</style>
<div id="page-number"></div>
<div id="toc">
... 目次 ...
</div>
<div id="body">
... 本文 ...
</div>
```

Copper PDF 3.1.4 以降では、@-cssj-page-content ルールによって、CSSだけでページ番号を表示できるようになりました。例えば、上記の処理は、以下のように記述することもできます。

例 4.62 ページ番号の生成（@-cssj-page-contentを使う）

```
<style type="text/css">
  @page {
    margin: 2cm 2cm 5cm 2cm;
    counter-increment: page;
    @-cssj-page-content footer {
        bottom: -1cm;
        text-align: center;
        width: 100%;
        content: counter(page);
    }
  }
  #body {
    page-break-before: right;
    counter-reset: page 1;
  }
</style>
<div id="toc">
... 目次 ...
</div>
<div id="body">
... 本文 ...
</div>
```

再生成ボックスどのページに表示するかを指定するには、@page :left のように擬似クラスを記述してから、その中で@-cssj-page-contentを記述してください。擬似クラスは:first, :left, :right, :singleのいずれかです。例えば、以下の例では左ページだけにページ番号を表示します。

例 4.63 ページ番号の生成（左ページだけ）

```
<style type="text/css">
  @page {
    margin: 2cm 2cm 5cm 2cm;
    counter-increment: page;
  }
  @page :left {
    @-cssj-page-content footer {
        bottom: -1cm;
        text-align: center;
        width: 100%;
        content: counter(page);
    }
  }
  #body {
    page-break-before: right;
    counter-reset: page 1;
  }
</style>
<div id="toc">
... 目次 ...
</div>
<div id="body">
... 本文 ...
</div>
```

「現在ページ / 総ページ数」のように表示するには、入出力プロパティ processing.pass-count[io]を2以上の値にし、総ページ数を記録する特殊なカウンタ名 pages を用いてください。*[3.1.4]*

例 4.64 総ページ数（Copper PDF 3.1.4以降）

```
<style type="text/css">
  @page {
    margin: 2cm 2cm 5cm 2cm;
    counter-increment: page;
  }
  #page-number {
    -cssj-page-content: footer;
    bottom: -1cm;
    text-align: center;
    width: 100%;
  }
  #page-number:before {
    content: counter(page) '/' counter(pages);
  }
  #body {
    page-break-before: right;
```

```
    counter-reset: page 1;
  }
</style>
<div id="page-number"></div>
<div id="toc">
... 目次 ...
</div>
<div id="body">
... 本文 ...
</div>
```

なお、バージョン3.1.3以前で「現在ページ / 総ページ数」のように表示するには、以下のように文書の末尾の要素のページ番号を-cssj-page-ref関数で参照してください。この場合は、入出力プロパティ processing.pass-count[io]を2以上の値にした上、processing.page-references[io]をtrueに設定する必要があります。

例 4.65 総ページ数（Copper PDF 3.1.3以前）

```
<style type="text/css">
  @page {
    margin: 2cm 2cm 5cm 2cm;
    counter-increment: page;
  }
  #page-number {
    -cssj-page-content: footer;
    bottom: -1cm;
    text-align: center;
    width: 100%;
  }
  #page-number:before {
    content: counter(page) '/' -cssj-page-ref(last,page);
  }
  #body {
    page-break-before: right;
    counter-reset: page 1;
  }
</style>
<div id="page-number"></div>
<div id="toc">
... 目次 ...
</div>
<div id="body">
... 本文 ...
</div>
<div id="last"></div>
```

左右のページでアラインメントを変えることにより、ページの外側にノンブルを振ることができます。また、-cssj-heading関数を利用することで、現在ページのセクションの見出しを一緒に表示することができます。Copper PDF 3.1.4 以降では-cssj-title 識別子により文書のタイトルを表示することができます。

例 4.66 タイトルと見出しの表示

```
<style type="text/css">
  @page {
    margin: 2cm 2cm 5cm 2cm;
    counter-increment: page;
  }
  #nombre-left, #nombre-right {
    bottom: -1cm;
    width: 100%;
  }
  #nombre-left {
    -cssj-page-content: nombre-left left;
    text-align: left;
  }
  #nombre-right {
    -cssj-page-content: nombre-right right;
    text-align: right;
  }
  #nombre-left:before {
    content:  -cssj-title counter(page) ' - ' -cssj-heading(1);
  }
  #nombre-right:after {
    content: -cssj-heading(2) ' - ' counter(page);
  }
</style>
<div id="nombre-left"></div>
<div id="nombre-right"></div>
... 本文 ...
```

CSSJ 1.x系の`-cssj-regeneratable`[css]も互換性のためサポートされていますが、パフォーマンスの面から`-cssj-page-content`[css]の使用を推奨します。

4.10 改ページ制御

4.10.1 用語の定義

絶対配置ボックス
{position: absolute;}が指定された要素です。{position: fixed;}が指定された要素やページごとに生成されるコンテンツも 広義の絶対配置ボックスです。

浮動ボックス
{float: left;}または{float: right;} が指定された要素です。

通常のフローのブロック
何も指定されていない<p>要素や 要素や{display: block;}が 指定された要素で絶対配置ボックスでも浮動ボックスでもないものです。

テーブルは次の部分からなっています。

テーブルキャプション
HTMLのcaptionタグ、あるいはdisplayがtable-captionと指定された部分。

テーブルヘッダ
HTMLのtheadタグ、あるいはdisplayがtable-header-groupと指定された部分。

テーブルフッタ
HTMLのtfootタグ、あるいはdisplayがtable-footer-groupと指定された部分。

テーブル行グループ
HTMLのtbodyタグ、あるいはdisplayがtable-row-groupと指定された部分。ただし、tbodyやtable-row-groupを省略して、テーブルの中に直接存在する行も行グループに属すると見なされます。

4.10.2 強制改ページ

強制改ページは、指定した場所で強制的に改ページを発生させる機能です。強制改ページを指定できるのは次の場所です。

- 通常のフローのブロックの直前
- 通常のフローのブロックの直後
- 浮動ボックスの直前
- 浮動ボックスの直後
- テーブルの直前
- テーブルの直後
- テーブル行グループの直前
- テーブル行グループの直後
- テーブル行の直前

- テーブル行の直後
- テーブルセルの直前
- テーブルセルの直後

ただし、上記の場所であっても浮動ボックス内、絶対配置ボックス内、テーブルセル内では強制改ページを発生することはできません。

要素の直前の強制改ページの指定は{page-break-before: always;}です。要素の直後の強制改ページの指定は{page-break-after: always;}です。

以下は強制改ページを使って表紙を作る例です。

例 4.67 強制改ページ

```
<html>
  <head>
    <title>ドキュメント</title>
  </head>
  <body>
    <h1 style="page-break-after: always;">表紙</h1>
    <p>本文...</p>
  </body>
</html>
```

また、単純に改ページするためではなく、改ページした直後のページが右になるか、左になるかを指定することができます。この場合、調整のために空白のページが１つつくられる可能性があります。

強制改ページの後のページが右になるか、左になるかを指定するには、page-break-beforeおよびpage-break-afterプロパティの値として、alwaysの代わりに left(左ページにする場合)またはright(右ページにする場合)を指定します。

以下の例では、必ず右側になる中表紙を生成しています。

例 4.68 空ページが生じるケース

```
<html>
  <head>
    <title>ドキュメント</title>
  </head>
  <body>
    <h1 style="page-break-after: always;">表紙</h1>
    <p style="page-break-after: always;">本文1...</p>
    <p>本文2...</p>
    <h1 style="page-break-before: right;">中表紙</h1>
  </body>
</html>
```

ただし、強制改ページのleft, rightの指定はテーブル内部では適用されず、いずれもalwaysと解釈されます。

4.10.3 orphans[css]とwidows[css]

orphans[css]とwidows[css]プロパティは、段落(ここでは通常のフローのブロックを指し、
による空行等は段落の区切りとは認識されません) の途中で改ページが発生する場合、必ず前のページに残す行数と、後のページに表示される行数を指定するものです。

なお、Copper PDFは実際の行数ではなく、行から行までの長さを標準的な行の高さ (段落に適用されたline-height[css]による高さ) で割った値を整数に丸めた数値を基準に計算します。そのため、行内に大きな画像が存在したり、インラインに対するfont-size[css]の指定により、例えば通常の2倍の高さに拡張されている行が存在すれば、2行として計算します。これはCSS 2.1の仕様にはありませんが、より直感的な改ページとするための仕様です。

orphans[css]

ある段落がページの下端にかかっている場合、段落を途中で分割して改ページする必要があります。orphans[css]はそのような場合に、改ページされる前のページに最低限残さなければならない行数です。例えば、以下の例ではorphans[css]が3に対して 改ページ前のページに3行があるので、条件を満たしています。

例 4.69 orphansが3の場合

(段落1)1行目... 2行目... 3行目... 4行目... 5行目... 6行目... 7行目... 8行目... (段落2)1行目... 2行目... 3行目...	4行目... 5行目...
1ページ目	**2ページ目**

同じ文書でorphans[css]を4に指定すると、そのままではorphans[css]を満たすことができないため、段落をまるごと次ページに移動してしまいます。

例 4.70 orphansが4の場合

(段落1)1行目... 2行目... 3行目... 4行目... 5行目... 6行目... 7行目... 8行目...	(段落2)1行目... 2行目... 3行目... 4行目... 5行目...
1ページ目	**2ページ目**

`widows` [css]

文書の内容の高さがページの高さよりわずかに高い場合、次のページに文書の内容のうち何行かを先送りしなければなりません。`widows` [css]は改ページされた後のページに最低限表示されなければならない行数で、Copper PDFは`widows` [css]を満たすように先送りする行数を調整します。例えば、`widows` [css]が2の場合、以下の例では2ページ目に2行存在するので条件を満たしています。

例 4.71 widowsが2の場合

(段落1)1行目... 2行目... 3行目... 4行目... 5行目... 6行目... 7行目... 8行目... (段落2)1行目... 2行目... 3行目...	4行目... 5行目...
1ページ目	**2ページ目**

同じ文書で`widows` [css]を3に指定すると、以下のように前のページから次のページへ行を移動して、`widows` [css]を満たすようにします。

例 4.72 widowsが3の場合

(段落1)1行目... 2行目... 3行目... 4行目... 5行目... 6行目... 7行目... 8行目... (段落2)1行目... 2行目...	3行目... 4行目... 5行目...
1ページ目	2ページ目

orphans[css]とwidows[css]の競合

orphans[css]とwidows[css]の両方の条件を同時に満たすことができない場合も、orphans[css]を満たせなかった場合と同様に段落を丸ごと次ページに移動します。

例えば、以下の状況ではorphans[css]とwindows[css]の両方が満たされています。

例 4.73 orphansが3でwidowsが2の場合

(段落1)1行目... 2行目... 3行目... 4行目... 5行目... 6行目... 7行目... 8行目... (段落2)1行目... 2行目... 3行目...	4行目... 5行目...
1ページ目	2ページ目

この状態でwidows[css]を3に設定すると、orphans[css]は満たせるがwidows[css]は 満たせない状態になります。

例 4.74 orphansが3でwidowsが3の場合

(段落1)1行目... 2行目... 3行目... 4行目... 5行目... 6行目... 7行目... 8行目...	(段落2)1行目... 2行目... 3行目... 4行目... 5行目...
1ページ目	2ページ目

ただし、段落がページの先頭にある場合は、orphans [css]が無視され、少なくとも 1 行が前ページに残されます。

4.10.4 改ページの抑制

次の場所には、改ページの抑制を指定することができます。

- 通常のフローのブロックの内部
- 通常のフローのブロックの直前
- 通常のフローのブロックの直後
- 浮動ボックスの内部
- テーブルの内部
- テーブルの直前
- テーブルの直後
- テーブル行グループの直前
- テーブル行グループの直後
- テーブル行の内部
- テーブル行の直前
- テーブル行の直後
- テーブルセルの内部
- テーブルセルの直前
- テーブルセルの直後

内部の改ページ抑制

内部での改ページを抑制するには、{page-break-inside: avoid;}という指定をします。

改ページ抑制されたボックスがページをはみ出す場合は、ボックスが丸ごと次のページの先頭に先送りされます。ただし、ボックスが(先送りの結果か、元々そこにあるかに関わらず)ページの先頭にある場合、かつボックスの高さがページの高さを超えてしまう場合は、改ページの抑制を無視してページ分割されます。

前後の改ページ抑制

ボックスの前後での改ページを抑制するには、`{page-break-before: avoid;}`(ボックスの前)あるいは`{page-break-after: avoid;}`(ボックスの後)を指定します。Copper PDFは改ページが抑制された箇所での改ページを避け、前後の何行かを必ず1つのページに含めるようにします。改ページが抑制されたポイントの前に入れる行数は`orphans` [css]に依存します。

強制改ページと改ページの抑制が競合する場合は、強制改ページが優先されます。。例えば、`{page-break-after: always;}`と指定された段落の直後に、`{page-break-before: avoid;}`と指定された段落がある場合です。このような競合が起こった場合、常に強制改ページが優先されます。つまり、このケースでは`{page-break-before: avoid;}`は無視されて改ページが発生します。

なお、HTMLのh1〜h6要素にはデフォルトで`{page-break-before: avoid;}` が指定されています。

4.10.5 自動改ページ

Copper PDFは、文書の内容がページの下端にさしかかった部分で、自動的に改ページします。自動的な改ページが発生するのは次の場所です。

- 通常のフローのブロックの間
- 画像以外の通常のフローのブロックの内部
- 画像以外の浮動ボックスの内部
- 行の間
- テーブル行グループの間
- テーブル行グループ内の行の間
- テーブル行グループ内の行の内部

逆に、以下の場所ではどのような場合も改ページされることはありません。

- 行の内部
- 画像の内部
- 絶対配置ボックスの内部
- テーブルキャプションの内部
- テーブルキャプションとテーブルの間
- テーブルヘッダの内部
- テーブルフッタの内部
- テーブルヘッダと行グループの間
- テーブルフッタと行グループの間

通常のフローのブロック

通常のフローでは orphans [css], widows [css] を尊重して改ページが行われます。ブロックに境界線がある場合に境界線の直後、あるいは高さが指定されていて内容がないブロックの内部では なるべく改ページを避けますが、ブロックがページの先頭ある場合は改ページが発生します。

浮動ボックス

浮動ボックスがページの下端をはみ出した場合、浮動ボックスは分割され、次ページに送られた部分は浮動ボックスとして再配置されます。浮動ボックスの分割でも、orphans [css], windows [css] の指定は尊重されますが、条件を満たせない場合であっても浮動ボックスを丸ごと次ページに送られることはなく、その場合はorphans [css], widows [css]を無視して分割されます。

画像および{page-break-inside: avoid;}が指定された浮動ボックスは分割されることはなく、ページの下端をはみ出した場合は丸ごと次のページに持ち越されます。ただし、浮動ボックスの中に入れ子になった浮動ボックスでは{page-break-inside: avoid;}は無効です。

4.11 テーブル内での改ページ

Copper PDFはテーブルの行間、テーブルの行の途中(セルの途中)での改ページが可能です。また、テーブルのヘッダとテーブルのフッタは各ページで繰り返し表示されます。

4.11.1 改ページされない場所

テーブル内の次の場所では、どのような場合も改ページされることはありません。

- テーブルのキャプション内
- テーブルのキャプションとテーブルの間
- テーブルヘッダの内部
- テーブルフッタの内部
- テーブルのヘッダと行グループの間
- テーブルのフッタと行グループの間

従って、上記の部分に指定されたpage-break-after [css], page-break-before [css], page-break-inside [css] は無視されます。また、上記の規則が適用された結果、テーブルの行グループが存在する限り、どのように改ページが発生する場合も、テーブルの行グループの一部が常に表示され、ヘッダかフッタだけのテーブルが現れることはありません。

上記の部分がページの高さを超える場合は、テーブルがページの下端をはみ出します。従って、適切なレイアウトとなるためには、テーブルのキャプションとヘッダとフッタの高さが、ページの高さに対して十分に小さいことが望ましいです。

4.11.2 page-break-XXXの適用

改ページに関する特性は行の間には通常のフローのブロックと同様に適用されます。改ページに関する特性の指定は行グループの間にも適用されます。ただし、強制改ページでleft, rightの指定は有効ではなく、効果はalwaysと同じになります。

セルに対するpage-break-inside [css]は、行に適用されます。同じ行内でautoとavoidが指定されたセルが競合する場合、avoidが優先されます。すなわち、行に属するセルのうち１つでもavoidが指定された場合、行全体の中での改ページが禁止されます。また、セルがrowspanで連結されている場合、セルが属する全ての行の内部で改ページが抑制されるのに加えて、行の間で page-break-after [css] および page-break-before [css] にavoidが指定されたものと見なされます。

セルに対するpage-break-after [css]および page-break-before [css]は、行に適用されます。この場合の優先順位は次の順になります。

always > avoid > auto

テーブルがページの先頭にあり、かつ改ページ禁止指定のために、ページの下端までの間で改ページできない場合は、行間の改ページ禁止を無視します。この場合は、改ページが禁止された部分であっても改ページが発生します。

4.11.3 テーブル行内部(セル内部)での改ページ

テーブル行内に、{page-break-inside: avoid;}が指定されたセルがなく、テーブル行がページの下端に差し掛かっている場合、そのテーブル行の分割が試みられます。このとき、orphans [css]とwidows [css]が尊重され、行に属する全てのセルが分割不可能な場合は、行全体が次のページに先送りされます。1つでも分割可能なセルがあった場合は、行が分割されます。この場合、他のセルに対してはorphans [css]とwidows [css] を無視した分割が起こる可能性があります。



分割されたセルに対してはvertical-align [css]による垂直アラインメントの指定が無効となり、セルの内容は全てセルの上端につけられます。

4.11.4 デフォルトの改ページ禁止

HTMLのtd,th要素は、デフォルトで {page-break-inside: avoid;} が設定されています。テーブルセル内の改ページを有効にするには、HTMLのtd, th要素に対して明示的に {page-break-inside:: auto;} を指定する必要があります。(td, th以外の要素に対して {display: table-cell;} を指定したことによるテーブルセルは、この限りではありません。)

例 4.75 テーブルセル内での改ページを有効にするCSS

```
td, th {
  page-break-inside: auto;
}
```

上のスタイルシートをデフォルトのスタイルシートとして指定しておけば、あらゆるページでテーブルセル内での改ページがされるようになります。

4.12 WebFont

Copper PDF 3.0.0 からWebFontがサポートされました。WebFontは、CSSによりファイルシステム上やネットワーク上のフォントを指定し、文書をレイアウトする際に読み込むものです。Internet Explorer, Safari, Chrome, Firefox等の最新のブラウザはWebFontをサポートしており、これらのブラウザ向けに、表示環境に関わらず同じフォントが表示されるようにした文書はCopper PDFでも、同じフォントを表示することができます。

WebFontは非常に手軽に使える反面、文書のレイアウトの度にフォントファイルを読み込むため、処理速度が遅くなります。WebFontの利用は開発時や、どうしても使用する必要がある場合にとどめ、可能な限りシステムのフォント設定 (44ページ)で対応することを推奨します。

4.12.1 @font-face ルール

文書からフォントファイルを読み込むにはCSSの@font-faceルールを使います。以下は、欧文フォントを読み込む例です。

例 4.76 ネットワーク上のフォントの読み込み(ソース)

```
<html>
<head>
  <style type="text/css">
    @font-face {
      font-family: "VeraSerif";
      src: url("http://dl.cssj.jp/docs/copper/misc/bitstream-
vera/Vera.ttf");
    }
    @font-face {
      font-family: "VeraSerif";
      font-weight: bold;
      src: url("http://dl.cssj.jp/docs/copper/misc/bitstream-
vera/VeraBd.ttf");
    }
    @font-face {
      font-family: "VeraSerif";
      font-style: italic;
      src: url("http://dl.cssj.jp/docs/copper/misc/bitstream-
vera/VeraIt.ttf");
    }
    body {
      font-family: "VeraSerif"
    }
    .bold {
      font-weight: bold;
    }
    .italic {
```

```
      font-style: italic;
    }
  </style>
</head>
<body>
  <p>This is Bitstream Vera Serif.</p>
  <p class="bold">This is Bitstream Vera Serif Bold.</p>
  <p class="italic">This is Bitstream Vera Serif Italic.</p>
</body>
</html>
```

図 4.21 ネットワーク上のフォントの読み込み(表示結果)



@font-face 内では、`font-family`[css]、`font-style`[css]、`font-weight`[css]、`unicode-range`[css]、`src`[css] の各プロパティを設定することができます。このうち、`font-family`[css]、`src`[css]は必須です。

`font-family`[css]、`font-style`[css]、`font-weight`[css] は、読み込まれたフォントの属性となります。文書中で適切なフォントが選択される際の手がかりとなります。Copper PDF がフォントを選択する方法については、ドキュメント中でのフォントの利用 (45ページ) をご参照ください。

font-family

フォントのファミリ名です。

font-style

フォントのスタイルです。normal, italic, obliqueのいずれかです。デフォルトはnormalです。

font-weight

フォントの太さです。normal, bold または100から900までの100刻みの値です。デフォルトはnormalです。

unicode-range

フォントが利用可能な文字コードの範囲です。ここで指定されたコード範囲にあり、かつフォントファイルに定義されている文字が利用可能な文字となります。デフォルトはU+0-10FFFFです。

記述例は次のとおりです。

unicode-range: U+A5;
円記号（¥）の文字だけに適用します。

unicode-range: U+0-7F;
ASCII文字（文字コード0から127）だけに適用します。

unicode-range: U+30??;
ひらがな、カタカナ（文字コード16進数で3000番台）だけに適用します。

unicode-range: U+A5, U+0-7F, U+30??;
前記の3つのコード範囲を合わせたものです。

src

フォントファイルの位置です。記述例は次のとおりです。

src: url(fonts/IPAMincho.otf);
fonts/IPAMincho.otfというパスにあるフォントファイルを読み込みます。

src: local(ＭＳ明朝)
(Copper PDFが動作している)OSにインストールされたＭＳ明朝という名前のフォントを読み込みます。

src: local(ＭＳ明朝), url(fonts/IPAMincho.otf)
OSにインストールされたＭＳ明朝が利用可能であればそれを使い、なければfonts/IPAMincho.otfを読み込みます。

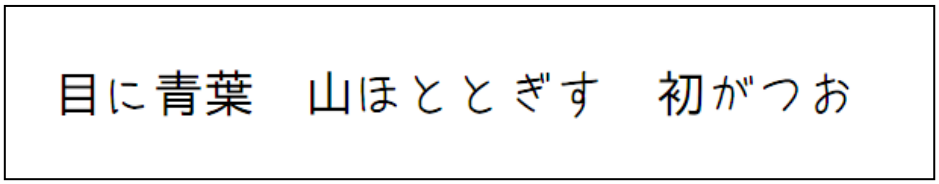
対応しているフォントフォーマットは、TrueType, OTF, WOFF*[3.1.0]*です。SVGフォント等はサポートしていません。

次の例は、漢字にはIPA Pゴシック、ひらがなと英数字にはきろ字 を使用します。

例 4.77 複数のフォントの併用(ソース)

```
<html>
<head>
  <style type="text/css">
    @font-face {
      font-family: "MyFont";
      src: url("http://dl.cssj.jp/docs/copper/misc/ipagp.otf");
      unicode-range: U+4E00-9FFF;
    }
    @font-face {
      font-family: "MyFont";
      src: url("http://dl.cssj.jp/docs/copper/misc/kiloji.ttf");
      unicode-range: U+A5, U+0-7F, U+30??;
    }
    body {
      font-family: "MyFont";
    }
  </style>
</head>
<body>
  <p>目に青葉／山ほととぎす／初がつお</p>
</body>
</html>
```

図 4.22 複数のフォントの併用(表示結果)



4.13 縦書き

Copper PDF は縦書きを独自にサポートしています*[3.0.0]*。これはドラフト段階のCSS3 Writing Modes(2010年12月現在)仕様の一部を先行して実装したものです。Copper PDFの現在の実装は将来リリースされるW3C規格との互換性を保証するものではありません。またCopper PDFの将来のバージョンでは仕様に変更が生ずる可能性があります。

4.13.1 -cssj-writing-mode

`-cssj-writing-mode` [css]は、CSS3 Writing Modesの`writing-mode` [css]の先行実装です。仕様は次のとおりです。

値

horizontal-tb | vertical-rl | lr | lr-tb | rl | tb | tb-rl

初期値

horizontal-tb

適用対象

テーブル行グループ、テーブルカラムグループ、テーブル行、テーブルカラム以外の要素

値の継承

する

要素に対して horizontal-tbは横書き、vertical-rl は縦書きを適用します。SVG, Internet Explorerとの互換性のために用意されている、lr, lr-tb, rlはhorizaontal-tbと同じ意味であり、同様にtb, tb-rlはvertical-rlと同じ意味です。

テーブルセル(td, th)に`-cssj-writing-mode` [css]を指定することができますが、テーブルセルの書字方向を変えることは推奨しません。書字方向が変えられたテーブルセルの途中では常に改ページできなくなります。代わりに、テーブルセル内に`-cssj-writing-mode` [css]を設定したdivタグを入れ子にするなどしてください。

4.13.2 文書の書字方向

文書全体の書字方向は、文書のドキュメント要素（XMLではルート要素、HTMLでは、BODY要素）に対する、`-cssj-writing-mode` [css]の指定によります。

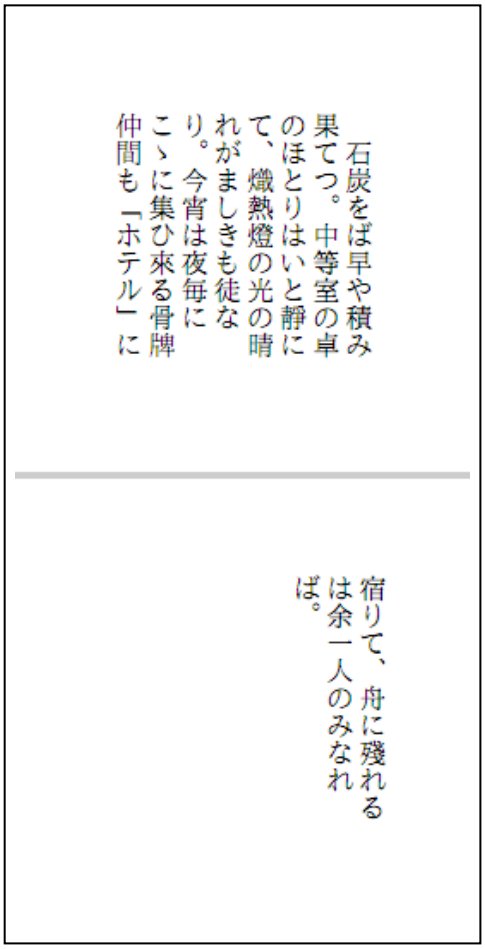
文書全体の書字方向は、文書の綴じ方向に影響します。すなわち、横書きでは左綴じ、縦書きでは右綴じとなります。`page-break-before` [css], `page-break-after` [css] に対するleft, rightによる強制改ページでは、綴じ方向が考慮されます。

横書きの場合は、内容が下にはみ出したところから改ページされますが、縦書きでは内容が左にはみ出したところから改ページされます。

例 4.78 文書全体を縦書きにする(ソース)

```
<html>
<head>
  <style type="text/css">
    body {
      writing-mode: vertical-rl;
    }
  </style>
</head>
<body>
<p>
石炭をば早や積み果てつ。
中等室の卓のほとりはいと靜にて、熾熱燈の光の晴れがましきも徒なり。
今宵は夜毎にこゝに集ひ來る骨牌仲間も「ホテル」に宿りて、舟に殘れるは余一人のみなれば。
</p>
</body>
</html>
```

図 4.23 文書全体を縦書きにする(表示結果)



4.13.3 書字方向の混在

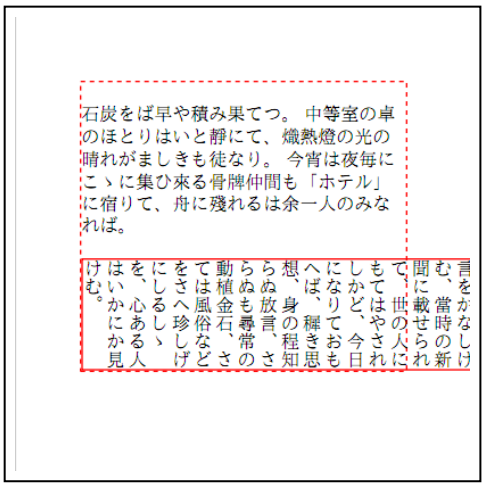
横書き中に縦書き指定された要素がある、あるいは縦書き中に横書き指定された要素がある場合、書字方向が混在するものとして処理します。文書の書字方向と異なる要素の中では改ページすることはできません。

ブロックの書字方向が異なる場合、そのブロックのページ進行方向の幅（親要素が横書きでは高さ、縦書きでは幅）がautoであれば、ページの高さとなります。しかし、書字方向の異なるブロックを、ページ進行方向の幅をautoで配置することは推奨しません。行方向の幅（親要素横書きでは幅、縦書きでは高さ）の計算方法は通常の場合と同じです。そのため、内容が長ければ、親要素の行方向にはみ出すことになります。あるいは、多段組 (ページ)を活用してください。

例 4.79 横書きの文書の一部を縦書きにする(ソース)

```
<html>
<head>
  <style type="text/css">
    body {
      border: 1pt dashed Red;
    }
    #a {
      writing-mode: vertical-rl;
      height: 6em;
      border: 1pt solid Red;
    }
  </style>
</head>
<body>
<p>
石炭をば早や積み果てつ。
中等室の卓のほとりはいと靜にて、熾熱燈の光の晴れがましきも徒なり。
今宵は夜毎にこゝに集ひ來る骨牌仲間も「ホテル」に宿りて、舟に殘れるは余一人のみなれば。
</p>
<div id="a">
五年前の事なりしが、平生の望足りて、洋行の官命を蒙り、このセイゴンの港まで來し頃は、目に見るもの、耳に聞くもの、一つとして新ならぬはなく、筆に任せて書き記しつる紀行文日ごとに幾千言をかなしけむ、當時の新聞に載せられて、世の人にもてはやされしかど、今日になりておもへば、稚き思想、身の程知らぬ放言、さらぬも尋常の動植金石、さては風俗などをさへ珍しげにしるしゝを、心ある人はいかにか見けむ。
</div>
</body>
</html>
```

図 4.24 横書きの文書の一部を縦書きにする(表示結果)

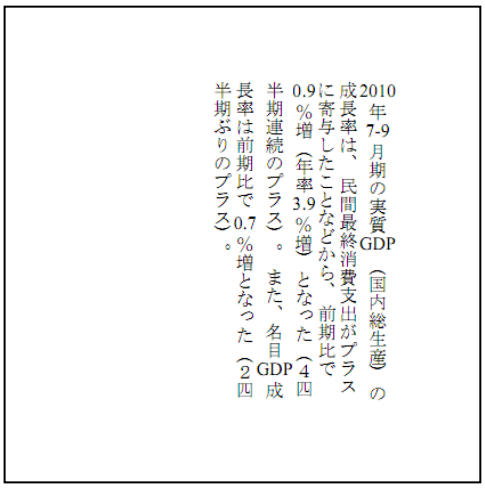


インラインの書字方向が異なる場合、インラインブロックとして配置します。これは、縦中横のために使うことができます。

例 4.80 縦中横(ソース)

```
<html>
<head>
  <style type="text/css">
    body {
      writing-mode: vertical-rl;
    }
    .tcy {
      writing-mode: horizontal-tb;
    }
  </style>
</head>
<body>
<p>
<span class="tcy">2010</span>年<span class="tcy">7-9</span>月期の実質
<span class="tcy">GDP</span>（国内総生産）の成長率は、
民間最終消費支出がプラスに寄与したことなどから、
前期比で<span class="tcy">0.9</span>%増（年率<span class="tcy">3.9
</span>%増）となった（４四半期連続のプラス）。
また、名目<span class="tcy">GDP</span>成長率は前期比で <span class="tcy"
>0.7</span>%増となった（２四半期ぶりのプラス）。
</p>
</body>
</html>
```

図 4.25 縦中横(表示結果)



4.13.4 -cssj-direction-mode

`margin-top` [css], `border-left` [css], `padding-bottom` [css] といったマージン、境界、パディング等の方向に依存するプロパティは、書字方向に関わらず、物理的な方向 (*-topなら常に上、*-leftなら常に左のように) に適用されます。しかし、横書きの文書を縦書きに切り替えるスタイルシートを作成する場合、横書きで上だったものを右、右だったものを下、下だったものを左、左だったものを上に、それぞれ回転させることで、新しいスタイルシートの記述が最小限で済む場合があります。特に、style属性によるスタイルでマージン等が指定された既存の文書の書字方向を変えるには、方向に依存するスタイル指定を回転させるしかありません。

`-cssj-direction-mode` [css]は、方向に依存するプロパティを回転させます。仕様は次のとおりです。

値

physical | logical | horizontal-tb*[3.0.12]* | vertical-rl*[3.0.12]*

初期値

physical

適用対象

全ての要素

値の継承

する

`-cssj-direction-mode` [css]がlogicalまたはhorizontal-tbの場合、方向に依存するプロパティを、横書きを基準として論理的に適用します。topはページ進行方向の前、bottomはページ進行方向の後、leftは行頭、rightは行末という意味になります。値の継承は変わりません。例えば`margin-top` [css]は、常に`margin-top` [css]に継承します。継承などにより得られた計算値を論理的な方向に読み替えて、スタイルを構成する要素に適用します。

page-break-before [css], page-break-after [css] に対するleft, right指定は、それぞれ偶数ページ(verso)、奇数ページ(recto)として処理します。すなわち、全体が縦書き（右綴じ）の文書では左右の指定が逆になります。

background-position [css] は、yがパーセント値の場合は (100%-y) の値が適用され、そしてx値とy値が逆転されます。

background-repeat [css] repeat-xとrepeat-y が逆転されます。

縦書きでは次の通りにプロパティの計算値を適用します。

元の計算値	計算値を適用するプロパティ
width	height
height	width
max-width	max-height
max-height	max-width
min-width	min-height
min-height	min-width
padding-top	padding-right
padding-right	padding-bottom
padding-bottom	padding-left
padding-left	padding-top
border-top-color	border-right-color
border-right-color	border-bottom-color
border-bottom-color	border-left-color
border-left-color	border-top-color
border-top-style	border-right-style
border-right-style	border-bottom-style
border-bottom-style	border-left-style
border-left-style	border-top-style
border-top-width	border-right-width
border-right-width	border-bottom-width
border-bottom-width	border-left-width

元の計算値	計算値を適用するプロパティ
border-left-width	border-top-width
margin-top	margin-right
margin-right	margin-bottom
margin-bottom	margin-left
margin-left	margin-top
top	right
right	bottom
bottom	left
left	top

また、width [css]は行進行方向の幅、height [css]はいずれの場合もページ進行方向の幅として処理します。最大幅、最小幅を指定するプロパティ（max-width [css]など）も同様です。しかし、一般的に画像は初期方向が切り替わっても回転させることはないため、縦横の幅を維持する必要があります。そのため、画像の幅と高さに対しては、logicalまたはhorizontal-tb, horizontal-tbの設定は影響しません。

float [css]、clear [css]、text-align [css]、caption-side [css] は、もともと-cssj-direction-mode [css] の影響を受けません。常に設定値のleftは行頭、rightは行末、topはページ進行方向の前、bottomはページ進行方向の後として処理されます。

次のような横書きの文書があるとします。

例 4.81 横書きの文書(ソース)

```
<html>
<head>
  <style type="text/css">
    p {
      text-indent: 1em;
      text-align: justify;
    }
  </style>
</head>
<body>
<h1 style="border-bottom: 2pt dashed">かっぱ</h1>
```

```
<img src="kappa.png" width="148" height="199" alt="かっぱ"
align="left"/>
<p>
河童（かっぱ）は、日本の妖怪・伝説上の動物、または未確認動物。標準和名の「かっぱ」は、「かわ（川）」に「わらは（童）」の変化形「わっぱ」が複合した「かわわっぱ」が変化したもの。河太郎（かわたろう）とも言う。ほぼ日本全国で伝承され、その呼び名や形状も各地方によって異なる。
</p>
</body>
</html>
```

図 4.26 横書きの文書(表示結果)



この文書のbodyに対して `{writing-mode: vertical-rl;}` を適用すると文書が縦書きになりますが、見出しの境界線や画像のマージンの方向はそのままになります。

図 4.27 縦書きに変換(表示結果)



さらにbodyに対して `{-cssj-direction-mode: logical;}` を適用すると、見出しの境界線や画像のマージンが回転され、より適切な表示になります。

図 4.28 論理方向モードで縦書きに変換(表示結果)



`-cssj-direction-mode` [css]にvertical-rlを設定した場合は、方向に依存するプロパティを、縦書を基準として論理的に適用します。

`page-break-before` [css], `page-break-after` [css] に対するleft, right指定は、それぞれ奇数ページ(recto)、偶数ページ(verso)として処理します。すなわち、全体が縦書き（右綴じ）の文書では左右の指定が逆になります。

`background-position` [css] は、xがパーセント値の場合は (100%-x) の値が適用され、そしてx値とy値が逆転されます。

`background-repeat` [css] repeat-xとrepeat-y が逆転されます。

横書きでは次の通りにプロパティの計算値を適用します。

元の計算値	計算値を適用するプロパティ
width	height
height	width
max-width	max-height
max-height	max-width
min-width	min-height
min-height	min-width
padding-top	padding-left
padding-right	padding-top
padding-bottom	padding-right
padding-left	padding-bottom

元の計算値	計算値を適用するプロパティ
border-top-color	border-left-color
border-right-color	border-top-color
border-bottom-color	border-right-color
border-left-color	border-bottom-color
border-top-style	border-left-style
border-right-style	border-top-style
border-bottom-style	border-right-style
border-left-style	border-bottom-style
border-top-width	border-left-width
border-right-width	border-top-width
border-bottom-width	border-right-width
border-left-width	border-bottom-width
margin-top	margin-left
margin-right	margin-top
margin-bottom	margin-right
margin-left	margin-bottom
top	left
right	top
bottom	right
left	bottom

その他のプロパティの扱いに関しては `{-cssj-direction-mode: logical;}` の場合と同様です。

4.14 多段組

Copper PDF は多段組を独自にサポートしています*[3.0.0]*。これは勧告候補段階のCSS3 Multi-column Layout(2010年12月現在)仕様の一部を先行して実装したものです。Copper PDFの現在の実装は将来リリースされるW3C規格との互換性を保証するものではありません。またCopper PDFの将来のバージョンでは仕様に変更が生ずる可能性があります。

印刷面の大きさに比べて文字が小さい場合、1行の幅が大きくなってしまい非常に読みにくいことがあります。そんな時は、多段組をすると読みやすくなります。また、空白が少なくなるため、より紙面を節約できます。

多段組では、段数に応じて行方向の幅が圧縮されます。%指定で配置された画像やボックスの大きさは、段の幅に対する比率となります。

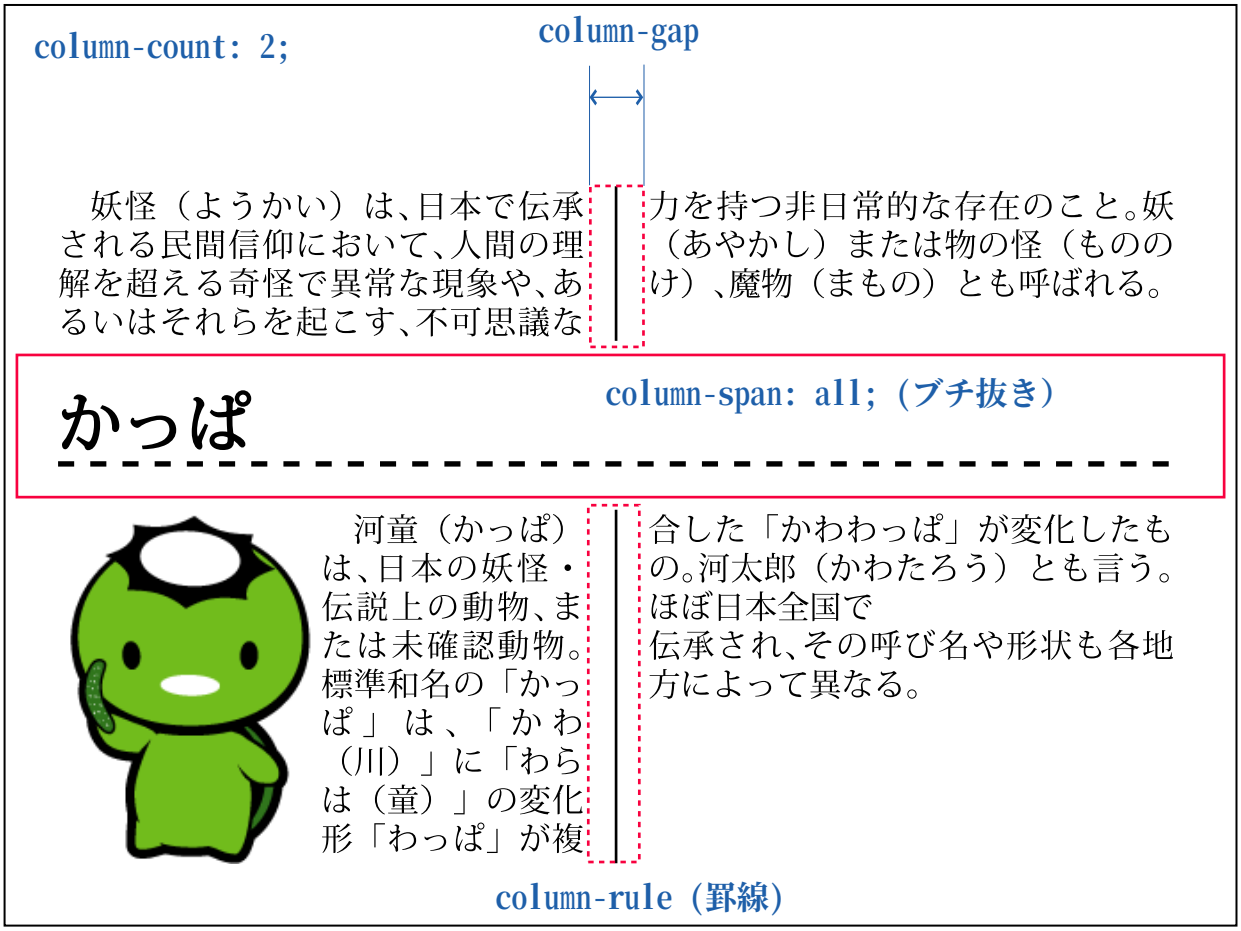
見出しなどは、ブチ抜きで配置することができます。

例 4.82 ２段組(ソース)

```
<html>
<head>
  <style type="text/css">
    div {
      column-count: 2;
      column-gap: 2em;
      column-rule: 1pt solid;
    }
    h1 {
      border-bottom: 2pt dashed;
      column-span: all;
    }
    img {
      float: left;
      width: 50%;
    }
    p {
      text-indent: 1em;
      text-align: justify;
      margin: 0;
    }
  </style>
</head>
<body>
<div>
<p>
妖怪（ようかい）は、日本で伝承される民間信仰において、人間の理解を超える奇怪で異常な現象や、あるいはそれらを起こす、不可思議な力を持つ非日常的な存在のこと。妖（あやかし）または物の怪（もののけ）、魔物（まもの）とも呼ばれる。
```

```
</p>
<h1>かっぱ</h1>
<img src="kappa.png" alt="かっぱ" />
<p>
河童（かっぱ）は、日本の妖怪・伝説上の動物、または未確認動物。標準和名の「かっぱ」は、「かわ（川）」に「わらは（童）」の変化形「わっぱ」が複合した「かわわっぱ」が変化したもの。河太郎（かわたろう）とも言う。ほぼ日本全国で伝承され、その呼び名や形状も各地方によって異なる。
</p>
</div>
</body>
</html>
```

図 4.29 ２段組(表示結果)



段組に関する各プロパティの説明は次のとおりです。

4.14.1 -cssj-column-count

値

auto | 1以上の整数

初期値

auto

適用対象

置換不可能なブロックレベル要素（テーブルを除く）、テーブルセル、インラインブロック

値の継承

しない

段組の段数です。ブロックの幅に余裕がある限り、このプロパティの計算値が、実際の段組の段数になります。`-cssj-column-width` [css], `-cssj-column-gap` [css] の兼ね合いで、指定した段数が確保できない場合、あるいはautoを指定した場合はできる限りの段数が確保されます。詳しい仕様は CSS3 Multi-column layout 3.4 Pseudo-algorithm のとおりです。

4.14.2 -cssj-column-width

値

auto | 長さ

初期値

auto

適用対象

置換不可能なブロックレベル要素（テーブルを除く）、テーブルセル、インラインブロック

値の継承

しない

段の幅を設定します。ブロックの幅が固定されている場合は、ブロックの幅を満たすように段の幅が調整されるため、実際の段の幅は計算値より広くなることがあります。詳しい仕様は CSS3 Multi-column layout 3.4 Pseudo-algorithm のとおりです。

4.14.3 -cssj-columns

`-cssj-column-count` [css]と `-cssj-column-width` [css] を同時に設定することができるプロパティです。例えば、`{-cssj-columns: 2 10em;}` は `{-cssj-column-count: 2; -cssj-column-width: 10em;}` と同じ意味になります。

4.14.4 -cssj-column-gap

値

normal | 長さ

初期値

normal

適用対象

段組された要素

値の継承

しない

段の間の幅を設定します。normal は 1em と同じです。

4.14.5 -cssj-column-rule-color

値

色

初期値

`color` [css]と同じ値

適用対象

段組された要素

値の継承

しない

段の間の境界線の色です。

4.14.6 -cssj-column-rule-style

値

境界のスタイル（`border-top-style` [css]等と同様）

初期値

none

適用対象

段組された要素

値の継承

しない

段の間の境界線のスタイルです。none,dotted, dashed, solid, double, groove, ridge, inset, outsetのいずれかです。noneは境界線を表示せず、境界線の太さもゼロになります。inset はridge, outsetはgrooveとそれぞれ同じ表示になります。

4.14.7 -cssj-column-rule-width

値

境界の太さ（`border-top-width` [css]等と同様）

初期値

medium

適用対象

段組された要素

値の継承

しない

段の間の境界線の太さです。

4.14.8 -cssj-column-rule

`-cssj-column-rule-color`[css], `-cssj-column-rule-style`[css], `-cssj-column-rule-width`[css] をまとめて設定するプロパティです。`border-top`[css]等と同様の記述方法です。例えば、`{-cssj-column-rule: 2pt dashed Red;}` は、`{-cssj-column-rule-color: Red; -cssj-column-rule-style: dashed; -cssj-column-rule-width: 2pt;}` と同じ意味になります。

4.14.9 -cssj-column-fill

値

balance | auto

初期値

balance

適用対象

段組された要素

値の継承

しない

段組の末尾の揃え方です。balanceは両方の段のページ進行方向の幅がなるべく同じになるように揃えます。autoは、揃えることをしません。横書きではbalance、縦書きではautoにするのが一般的です。また、autoの方が処理速度は速くなります。

4.14.10 -cssj-column-span

値

1 | all

初期値

1

適用対象

静的な、浮動体以外の要素

値の継承

しない

段組の、いわゆる「ぶち抜き」を指定します。指定可能なのは、1段または全段抜きのいずれかです。例えば、見出しだけを全段抜きにすることができます。

全段抜きにした場合、そこで段組が区切られます。上位の要素に段組があれば、全段抜きの前で一旦閉じられ、後で再開するのと同じことになります。ただし、全段抜きの直前では `-cssj-column-fill` [css] の指定に関係なく、段組の末尾が揃えられます。

4.14.11 改段と改ページ

自動的な改段と改ページ

段組みされた内容がページ末端をはみ出す場合、改段できるのであれば改段し、そうでなければ改ページします。例えば3段組の要素内で、1段目と2段目の内容がはみ出せば改段し、3段目の内容がはみ出せば改ページします。改段による内容の分割のされ方は改ページと同じです。

Copper PDF は浮動ボックス、テーブルセル内でも改ページをしますが、段組みされている浮動ボックス、テーブルセル内では改ページしません。また、高さが指定されているボックス内でも改ページしません。高さが指定されているボックスでは、最後の段がいっぱいになっても、改ページの代わりに改段をするため、実際の段数が設定した段数より多くなることがあります。

強制的な改段と改ページ

`page-break-before` [css], `page-break-after` [css] のleft, right, alwaysは段組中でも改ページとして機能します。改段のためには、columnというキーワードを使うことができます。pageというキーワードも使用可能ですが、alwaysと同じ意味です。

例えば、`{page-break-after: column;}` という指定がされた場合、段組の途中であれば改段します。最後の段で、かつ改ページが可能な場合、あるいは段組されていない場合は改ページします。

4.15 バーコード・QRコード

Copper PDFはバージョン2.1.3以降でプラグインとしてバーコードの機能を提供してきましたが、バージョン3.1.0からはビルトインの機能となりました。

バーコードの表示は、Barcode4J(http://barcode4j.sourceforge.net/)、QRコードの表示はQRコードクラスライブラリ for Java(http://www.swetake.com/qr/java/qr_java.html) により、それぞれ実現されています。

4.15.1 バーコードの表示

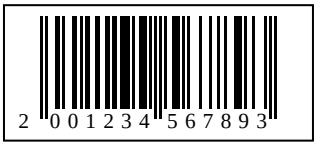
バーコードタグの詳細な仕様は、オンラインドキュメントのBarcode4J ドキュメント抄訳 (http://copper-pdf.com/?p=905) または英語の原文 (http://barcode4j.sourceforge.net/2.0/barcode-xml.html) を参照してください。ここでは概要を解説します。

HTML/XHTML中に記述された、http://barcode4j.krysalis.org/ns名前空間に属する要素がバーコードに置き換えられます。例えば、以下の記述はEAN-13(JAN-13)形式のバーコード画像に置き換えられます。

例 4.83 バーコード(ソース)

```
<bc:barcode xmlns:bc="http://barcode4j.krysalis.org/ns"
 message="200123456789">
 <bc:ean-13>
 <bc:height>15mm</bc:height>
 <bc:module-width>0.33mm</bc:module-width>
 <bc:quiet-zone enabled="true">10mw</bc:quiet-zone>
 <bc:checksum>add</bc:checksum>
 <bc:human-readable>
   <bc:placement>bottom</bc:placement>
   <bc:font-size>8pt</bc:font-size>
 </bc:human-readable>
 </bc:ean-13>
</bc:barcode>
```

図 4.30 バーコード(表示結果)



Barcode4Jがサポートする、全ての形式のバーコードを同様の方法で表示させることができます。加えて、EAN-13(JAN-13)の変形として、日本国内で流通する図書のISBNを表記するためのバーコードをサポートしています。

4.15.2 ISBNバーコード

ISBNバーコードは、EAN-13(JAN-13)の変形であり、記述方法も同じです。ただし、デフォルト値は異なり、以下のとおりです。

例 4.84 ISBNバーコード

```
<bc:barcode>
  <bc:isbn>
    <bc:height>{length:14mm}</bc:height>
    <bc:module-width>{length:0.33mm}</bc:module-width>
    <bc:quiet-zone enabled="{boolean:true}">{length:5mm}</bc:quiet-
zone>
    <bc:checksum>{checksum-mode:auto=add|check}</bc:checksum>
    <bc:human-readable>
      <bc:placement>{human-readable-placement:bottom}</bc:placement
>
      <bc:font-name>{font-name:OCRB}</bc:font-name>
      <bc:font-size>{length:3.7mm}</bc:font-size>
    </bc:human-readable>
  </bc:isbn>
</bc:barcode>
```

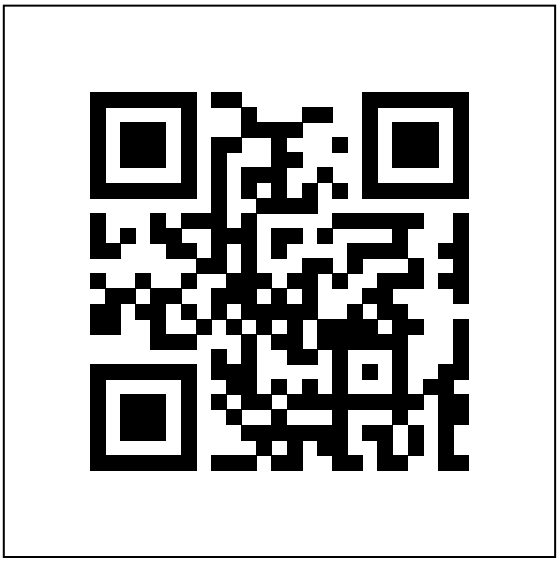
4.15.3 QRコード

QRコードについては、株式会社デンソーウェーブのサイト(http://www.qrcode.com/)も参考にしてください。QRコードもバーコードと同様のXMLで、以下のように文書中に記述します。

例 4.85 QRコード(ソース)

```
<bc:barcode xmlns:bc="http://barcode4j.krysalis.org/ns"
 message="エンコードする文字列">
 <bc:qrcode>
   <bc:version>0</bc:version>
   <bc:ecc>M</bc:ecc>
   <bc:encmode>B</bc:encmode>
   <bc:module-width>0.25mm</bc:module-width>
   <bc:quiet-zone>2mw</bc:quiet-zone>
 </bc:qrcode>
</bc:barcode>
```

図 4.31 QRコード(表示結果)



qrcode要素内の要素の意味は、次に説明するとおりです。

version

QRコードのバージョンです。0から40の整数で指定します。0の場合は自動設定です。デフォルトは0です。

ecc

エラー訂正レベルです。L, M, Q, Hのいずれかを指定します。デフォルトはMです。

encmode

エンコードモードです。N(数字), M(英数字), B(8ビットバイト)のいずれかを指定します。デフォルトはBです。

module-width

セルの大きさです。単位はcm, mm, pt, inのいずれかを使用可能です。デフォルトは0.25mmです。

quiet-zone

上下左右のマージンです。単位はcm, mm, pt, in, mw(セルの大きさ)のいずれかを使用可能です。デフォルトは1mwです。

4.15.4 郵便カスタマーバーコード

郵便カスタマーバーコードについては、郵便局のバーコードマニュアル(http://www.post.japanpost.jp/zipcode/zipmanual/index.html)も参考にしてください。カスタマーバーコードも他のバーコードと同様のXMLで、以下のように文書中に記述します。

例 4.86 カスタマーバーコード(ソース)

```
<bc:barcode xmlns:bc="http://barcode4j.krysalis.org/ns"
 message="1008798 1-3-2">
 <bc:japanpost>
    <bc:module-width>0.6mm</bc:module-width>
 </bc:japanpost>
</bc:barcode>
```

図 4.32 カスタマーバーコード(表示結果)



message内に含まれる数字、アルファベット、ハイフン以外の文字は無視されます。

japanpost要素内の要素の意味は、次に説明するとおりです。

module-width

バーの太さです。単位はcm, mm, pt, inのいずれかを使用可能です。デフォルトは0.6mmで、0.48mmから0.69mmの間で設定することが推奨されます。

4.16 MathML

Copper PDF 3.1.0からMathMLをサポートしています。

MathMLの描画はJEuclid(http://jeuclid.sourceforge.net/)を使用しています。MathML 2.0の機能のほとんどを利用することができます。

MathMLは、以下のとおり http://www.w3.org/1998/Math/MathML 名前空間の要素をXHTML内に記述します。

例 4.87 MathMLによる二次方程式の解(ソース)

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
 <mrow>
  <mi>x</mi>
  <mo>=</mo>
  <mfrac>
   <mrow>
    <mrow>
     <mo>-</mo>
     <mi>b</mi>
    </mrow>
    <mo>&PlusMinus;</mo>
    <msqrt>
     <mrow>
      <msup>
       <mi>b</mi>
       <mn>2</mn>
      </msup>
      <mo>-</mo>
      <mrow>
       <mn>4</mn>
       <mo>&InvisibleTimes;</mo>
       <mi>a</mi>
       <mo>&InvisibleTimes;</mo>
       <mi>c</mi>
      </mrow>
     </mrow>
    </msqrt>
   </mrow>
   <mrow>
    <mn>2</mn>
    <mo>&InvisibleTimes;</mo>
    <mi>a</mi>
   </mrow>
  </mfrac>
 </mrow>
</math>
```

上記の記述は以下のとおりに表示されます。

図 4.33 MathMLによる二次方程式の解(表示結果)

$$x = \frac{-b\sqrt{b^2-4ac}}{2a}$$

MathMLを手書きするのは大変ですが、インターネットで検索すると、MathMLを作成するための様々なツールがあります(LaTeXから変換するものなど)。

5.資料集

5.1 入出力プロパティー覧

以下は、各種プログラミング言語からCopper PDFにアクセスする際に設定できるプロパティの一覧です。プロパティの設定方法の詳細は開発マニュアルをご参照ください。

表 5.1 入力関連プロパティ

名前	デフォルト	バージョン	説明
input.default-encoding	JISAutoDetect 3.0.1 では JISUniAutoDetect がデフォルト	1.0.0	HTMLのMETA 要素でキャラクタ・エンコーディングを判断できない場合に使用するキャラクタ・エンコーディング名です。独自のエンコーディング名JISUniAutoDetectを設定すると、ISO-2022-JP, UTF-8, Windows-31J, EUC_JP_Solaris を自動判別するため、JISAutoDetectと違ってUTF-8を認識し、機種依存文字が文字化けしにくくなります。*[3.0.1]*
input.default-stylesheet	-	1.0.0	デフォルトのCSSスタイルシートのURIです。このプロパティが指定されている場合、最初にデフォルトのスタイルシートが読み込まれます。
input.filters	xslt default-to-xhtml loose-html	1.0.0	入力HTML(XML)のフィルタリングです。適用される順にフィルタ名をスペース区切りで並べます。以下のフィルタ名が用意されています。 **xslt** <xml-stylesheet...処理命令で指定されているXSL変換スタイルシートを適用します。 **default-to-xhtml** XMLで名前空間が指定されていない要素をXHTML名前空間に変換します。 **loose-html** 一般的なHTMLを解釈できるようにするためのフィルタです。
input.normalize-text	false	3.2.15	"true"を設定すると、全てのテキストをNFC（正規化形式C）に正規化します。
input.property-pi	false	2.0.0	trueを設定するとドキュメント中でjp.cssj.property-pi処理命令を使うことができるようになります。
input.stylesheet.titles	-	1.0.0	適用するCSSスタイルシートのタイトルをスペース区切りで並べます。link要素またはxml-stylesheet処理命令で関連付けられたスタイルシートについて、デフォルトでは代替スタイル以外が全て適用されますが、このプロパティを用いて適用するスタイルシートを指定することができます。
input.xslt.default-stylesheet	-	1.2.0	デフォルトのXSLTスタイルシートのURIです。このプロパティが指定されている場合、最初にデフォルトのスタイルシートが読み込まれます。input.filtersにxsltフィルタが存在するとき場合のみ有効です。

表 5.2 HTTPアクセス関連プロパティ

名前	デフォルト	バージョン	説明
input.html.change-default-namespace	false	3.2.12	falseの場合は文書のデフォルトの名前空間はXHTMLに強制されます。つまりxmlns="〜"という指定は無視されます。trueであればxmlns="〜"という指定が有効となります。
input.http.referer	true	1.0.1	HTTP通信でサーバー側のデータを取得するときにRefererヘッダを送るかどうかの指定です。trueまたはfalseで指定します。 falseを指定すると、Refererを用いて画像などのリソースへの直接アクセスを規制しているサイトでリソースにアクセスできなくなります。
input.http.proxy.host	-	1.2.6	プロクシのホスト名です。 この設定するとHTTP通信でプロクシを用います。
input.http.proxy.port	8080	1.2.6	プロクシを使う際のポート番号です。 この設定はinput.http.proxy.hostが設定されている場合のみ有効です。
input.http.proxy.authentication.user input.http.proxy.authentication.password	-	1.2.6	認証が必要なプロクシサーバーでの認証情報(user,password)です。この設定は`input.http.proxy.host`[io]が設定されている場合のみ有効です。
input.http.header.*n*.name input.http.header.*n*.value	-	2.0.0	HTTP接続で送信するヘッダです。 nは0から始まる通し番号で、nが同じ2つのプロパティで一組です。nameはヘッダ名でvalueはヘッダの値です。 通し番号は0から開始してカウントしていき、必要な情報(name)が欠けていた時点で以降のパラメータは無効となります。
input.http.authentication.preemptive	false	1.2.6	HTTP通信で認証を行う場合に、最初から認証情報を送るかどうかの設定です。trueまたはfalseで指定します。 trueを指定すると、Authorizationヘッダ等の認証情報を最初の接続で送ります。 falseを指定すると、最初にサーバーから401レスポンスを受け取ってレルムや認証スキーマ等の情報を取得します。 trueを指定した場合、Digest認証や複数のレルムが存在するサーバーで認証が行われなくなります。
input.http.authentication.*n*.host input.http.authentication.*n*.user input.http.authentication.*n*.password input.http.authentication.*n*.port input.http.authentication.*n*.realm input.http.authentication.*n*.schema	-	1.2.6	認証が必要なサイトでの認証情報です。 nは通し番号で、nが同じ4つのプロパティで一組です。hostは対象となるホストで、user,passwordが認証に用いられるユーザー名とパスワードです。 portはポート番号、realmは認証領域のレルム、schemaは認証スキーマ(basicまたはdigest)です。port,realm,schemaは省略可能で、省略した場合はそれぞれ全てのポート、レルム、スキーマでの認証が行われます。 通し番号は0から開始してカウントしていき、必要な情報(hostおよびuser)が欠けていた時点で以降のパラメータは無効となります。

名前	デフォルト	バージョン	説明
input.http.cookie.*n*.domain input.http.cookie.*n*.name input.http.cookie.*n*.value input.http.cookie.*n*.path	-	1.2.6	送信するクッキーです。nは通し番号で、nが同じ4つのプロパティで一組です。domain,name,valueはそれぞれクッキーのドメインと名前、値です。pathはクッキーのパスで、省略した場合はルート(/)となります。 通し番号は0から開始してカウントしていき、必要な情報(domainおよびname)が欠けていた時点で以降のパラメータは無効となります。
input.http.connection.timeout	0	2.0.7	HTTP接続の接続タイムアウト(ミリ秒)です。設定した時間内に接続が確立されない場合は、接続エラーとします。0の場合はタイムアウトなしです。
input.http.socket.timeout	0	2.0.7	HTTP接続のソケット通信タイムアウト(ミリ秒)です。設定した時間以上待ち時間が発生した場合は、通信エラーとします。0の場合はタイムアウトなしです。
input.viewport	false	3.1.0	trueにすると、<meta name="viewport"〜タグによりページサイズが設定されるようになります。

表 5.3 出力関連プロパティ

名前	デフォルト	バージョン	説明
output.auto-height	false	1.0.0	自動高さの指定です。falseまたはtrueで指定します。trueにすると、自動改ページをせず、ページの高さを文書の内容の高さに合わせます。このとき、`output.page-height`[io]（縦書きでは`output.page-width`[io]）プロパティは無効になります。 出力可能なページのサイズには制限があります。
output.auto-rotate	none	2.1.9	`output.page-width`[io], `output.page-height`[io], `output.paper-width`[io], `output.paper-height`[io] の設定により、用紙と内容の縦横比近くなるように、用紙または内容を回転します。 値はnone, content, paperで指定します。noneでは自動回転しません（デフォルト）。contentでは内容を回転します。paperでは用紙を回転します。
output.broken-image	none	1.2.2 noneは2.0.0 annotationは2.1.2	壊れた画像の表示方法。none,hidden,crossで指定します。noneでは代替テキスト(alt属性の値)だけが挿入されます。hiddenではwidth, height属性による矩形の範囲が空白となります。crossでは、width, height属性による矩形の範囲に×印が表示されます。annotationでは、crossと同じイメージがPDFのアノテーションとして表示されます。印刷時や、画像として出力する場合はhiddenと同じです。
output.clip	true	2.0.3	trueに設定した場合、印刷面の外側(トンボのドブの外側、あるいはページの外側)を描画しません。falseに設定した場合、印刷面の外側を描画します。

名前	デフォルト	バージョン	説明
output.color	rgb	1.2.1 cmykは3.1.0	出力結果のカラー・タイプです。rgb,cmyk,grayで指定します。rgbでは、指定通りのカラーで出力されます。cmykでは全てCMYKカラーに変換されます。grayでは、全てグレイスケールに変換されます。
output.compatible_mode	copper	2.0.0 3.1.0以降では廃止しました。	レイアウトの互換モードです。copper,msieで指定します。msieを指定すると、Internet Explorer 7に近いレイアウトを再現します。
output.default-font-family	serif	2.0.0	デフォルトのフォントファミリです。ドキュメント中でフォントが指定されていない場合、あるいは該当するフォントが見つからない場合、このフォントを使用します。 CSSの`font-family`[css]と同じ形式で複数のフォントを指定することができます。空白を含むフォント名はクウォート('または")で囲うことに注意してください。
output.expand-with-content	false	3.2.1	ページ高さを内容によって拡張します。`output.auto-height`[io]とは異なり、改ページは行われ、`output.page-height`[io]（縦書きでは`output.page-width`[io]）プロパティは無効になりません。画像など、分割不可能の要素が含まれ、その高さが既定のページの高さを超える場合に、ページの高さが拡張されます。`output.no-page-break`[io]の場合は、内容によってページの高さが拡張されます。ただし、`output.page-height`[io]（縦書きでは`output.page-width`[io]）により設定された高さより小さくなることはありません。
output.fit-to-paper	false	2.0.0 preserve-aspect-ratioは2.1.9	trueに設定した場合、内容が用紙に合わせて拡大されます。falseに設定した場合、中央寄せされます。preserve-aspect-ratioを設定すると、アスペクト比を固定して拡大します。
output.marks	none	1.0.0 hiddenは1.2.1	トンボおよび裁ち口の表示です。none,crop,cross,both,hiddenのいずれかを指定します。それぞれ、トンボ・裁ち口なし、コーナートンボを表示、センタートンボを表示、両方のトンボを表示、裁ち口だけを表示する、という意味になります。
output.media_types	all print paged visual bitmap static	2.0.0	適用するスタイルシートのメディアタイプです。
output.meta.*n*.name output.meta.*n*.value	-	2.0.3	文書情報をあらかじめ設定します。*n*は0から始まる通し番号で、*n*が同じ2つのプロパティで一組です。 文書情報はドキュメント内の`<meta name="名前" content="値">`要素によって上書きされます。詳細は文書情報 (167ページ)の節を参照してください。
output.no-page-break	false	2.0.3	trueに設定すると改ページを全くしなくなります。`output.auto-height`[io]をtrueに設定するのと異なり、ページの高さを内容に合わせて拡大しません。

名前	デフォルト	バージョン	説明
output.page-height	297mm	1.0.0	ページの高さです。デフォルトはA4の高さです。 CSSの長さの単位(mm,cm,in,pt,pc,px)を使ってください。 出力可能なページのサイズには制限があります。
output.page-limit	-	1.2.0	最大ページ数です。ページ数が限界に達すると、処理が中断されます。デフォルトでは無制限です。詳細はページ数の制限 (162ページ)の節を参照してください。
output.page-limi.abortt	force	3.0.11	forceを設定すると、ページ数の限界に達した場合に結果を破棄します。normalを設定すると、出来る限り途中までのファイルを出力します。詳細はページ数の制限 (162ページ)の節を参照してください。
output.page-margins	12.7mm	2.0.0	ページの余白です。CSSのmargin [css]プロパティと同じ形式で記述します。長さの単位は(mm,cm,in,pt,pc,px)が使用可能です。この設定は文書中の@pageルール内で上書きできます。
output.page-width	210mm	1.0.0	ページの幅です。デフォルトはA4の横幅です。 CSSの長さの単位(mm,cm,in,pt,pc,px)を使ってください。 出力可能なページのサイズには制限があります。
output.paper-height	output.page-heightの値	2.0.0	用紙の高さです。デフォルトはページの高さです。用紙とページの大きさが異なる場合の動作はoutput.fit-to-paper [io]の設定によります。 CSSの長さの単位(mm,cm,in,pt,pc,px)を使ってください。 出力可能なページのサイズには制限があります。
output.paper-width	output.paper-widthの値	2.0.0	用紙の幅です。デフォルトはページの横幅です。 用紙とページの大きさが異なる場合の動作はoutput.fit-to-paper [io]の設定によります。 CSSの長さの単位(mm,cm,in,pt,pc,px)を使ってください。 出力可能なページのサイズには制限があります。
output.print-mode	double-side	2.0.0 left-side, right-sideは3.0.0	印刷モードです。single-side, double-side, left-side, right-sideのいずれかを指定します。 single-sideでは片面印刷となり、@pageルールの:left, :right擬似クラスは適用されなくなります。 left-side, right-sideでは、文書の横書き、縦書きに関わらず綴じ方向がどちらかに固定されます。
output.resolution	96	2.0.0	px単位の基準となる解像度です。 ppi(1インチあたりのピクセル数)を指定します。 一般的なブラウザでは96という値が使われます。72を指定すると1pt(PDFの基本単位)と1pxの長さが同じになります。
output.size-limit	-	1.2.0	出力データの最大サイズ(バイト)です。サイズが限界に達すると、処理が中断されます。デフォルトでは無制限です。
output.htrim	1cm	2.0.0	左右の裁ち口の幅です。 CSSの長さの単位(mm,cm,in,pt,pc,px)を使ってください。

名前	デフォルト	バージョン	説明
output.vtrim	1cm	2.0.0	上下の裁ち口の幅です。 CSSの長さの単位(mm,cm,in,pt,pc,px)を使ってください。
output.text-size	1.0	2.1.9	文字のサイズの拡大率（実数）です。 例えば 0.5 を設定すると、文字サイズが通常の半分になり、2.0 を設定すると、2倍になります。
output.type	application/pdf	1.0.0	出力(MIME)形式です。"application/pdf"(PDFファイル)は必ず利用することができます。 Copper PDF 2.0.3から画像の出力に対応しました。画像の出力はJava Image I/Oに依存しており、Java実行環境がサポートする画像形式("image/png"など)を利用することができます。また、JAI-ImageI/O等のプラグインをJava実行環境にインストールすることで、利用可能な画像形式を追加することができます。 画像の出力では最後のページだけが出力されます。また、コア14フォントとフォント設定ファイルのcid-keyed-font要素によるCID-Keyedフォントは正確に描画できません。
output.trims	1cm	3.1.6	裁ち口の幅です。 CSSの`margin` [css]プロパティと同じ形式で記述します。長さの単位は(mm,cm,in,pt,pc,px)が使用可能です。

表 5.4 画像出力関連プロパティ

名前	デフォルト	バージョン	説明
output.image.resolution	96	2.0.4	`output.type`[io]の設定によりラスター画像を出力する際の解像度(dpi)です。 なお、2.0.8以前ではデフォルト値が72となっており、解像度が正しく反映されないバグがありました。2.0.9 以降では以前の設定 × `output.resolution`[io] / 72) で換算した値を設定してください。
output.image.antialias	true	3.0.1	ラスター画像出力の際のアンチエイリアスの設定です。trueを設定するとアンチエイリアスを有効にします、falseを設定するとアンチエイリアスを無効にします。
output.use-meta-info	true	3.1.8	HTMLのMETA，TITLE要素により文書情報を設定します。"false"にすると、この機能は無効になります。

表 5.5 PDF出力関連プロパティ

名前	デフォルト	バージョン	説明
output.pdf.attachments.*n*.name output.pdf.attachments.*n*.description output.pdf.attachments.*n*.mime-type output.pdf.attachments.*n*.uri	なし	1.2.0 (PDF 1.4)	添付ファイルです。 *n*は通し番号で、*n*が同じ4つのプロパティで一組です。nameはファイル名、descriptionはファイルについての説明で、省略可能です。mime-typeはファイルのMIME型で、省略可能です。uriはファイルの内容が置かれたURIで、省略できません。 通し番号は0から開始してカウントしていき、必要な情報(uri)が欠けていた時点でファイルの添付を終わります。nameにはASCII文字だけを使うことを推奨します(マルチバイト文字を含むことはできますが、表示環境によっては文字化けします)。マルチバイト文字が含まれる場合は、URLエンコードなどでASCIIに変換したものをnameに使い、実際のファイル名をdescriptionにセットしてください。
output.pdf.bookmarks	false	1.0.0 (PDF 1.2)	ブックマーク機能です。falseまたはtrueで指定します。 trueにすると、H1～H6要素をもとにブックマーク(アウトライン)を生成します。
output.pdf.compression	binary	1.0.0 (PDF 1.2)	圧縮方法です。none,ascii,binaryで指定します。 後者ほど圧縮効率がよくなります。noneでは画像以外は圧縮せず、asciiでは画像以外の内容も圧縮されますが、生成されるPDFはテキストファイルとなります。binaryの場合、生成されるPDFは圧縮され、かつバイナリ形式となります。 ただし、暗号化を行う場合は、結果的に全てバイナリとなることに注意してください。
output.pdf.encryption	none	1.2.0 (PDF 1.2) (v2はPDF 1.3)	暗号化方式です。none,v1,v2で指定します。 noneは暗号化なし、v1は40ビットArcfour暗号、v2は40-128ビットArcfour暗号です。
output.pdf.encryption.length	128	1.2.0 (PDF 1.3)	暗号化キーの長さ(ビット)です。 output.pdf.encryption=v1では40で固定です。v2では40から128の間で、8ビット刻みで指定可能です。
output.pdf.encryption.user-password	空	1.2.0 (PDF 1.2)	文書を開くためのパスワードです。このパスワードを使って文書を閲覧する場合は、文書に設定されたパーミッションによる制限がかかります。
output.pdf.encryption.owner-password	ユーザーのパスワード	1.2.0 (PDF 1.2)	文書の権限を変更するためのパスワード(マスタパスワード)です。文書に対するあらゆる操作を可能にします。
output.pdf.encryption.permissions.print	true	1.2.0 (PDF 1.2)	文書を印刷する権限です。 true=許可,false=禁止 です。
output.pdf.encryption.permissions.modify	true	1.2.0 (PDF 1.2)	文書中の内容を変更をする権限です。 true=許可,false=禁止 です。
output.pdf.encryption.permissions.copy	true	1.2.0 (PDF 1.2)	文書中のテキストや画像をコピーする権限です。 true=許可,false=禁止 です。

名前	デフォルト	バージョン	説明
output.pdf.encryption.permissions.add	true	1.2.0 (PDF 1.2)	注釈を追加・変更する、あるいはフォームに入力する権限です。output.pdf.encryption.permissions.modify=true であればフォームの追加・変更も許可されます。 true=許可,false=禁止 です。
output.pdf.encryption.permissions.fill	true	1.2.0 (PDF 1.3)	フォームに入力する権限です。 output.pdf.encryptionがv2のときだけ有効です true=許可,false=禁止 です。
output.pdf.encryption.permissions.extract	true	1.2.0 (PDF 1.3)	障害のあるユーザーのために文書中のテキストや画像を抽出する権限です。 output.pdf.encryptionがv2のときだけ有効です true=許可,false=禁止 です。
output.pdf.encryption.permissions.assemble	true	1.2.0 (PDF 1.3)	文書中に新しいページ、ブックマーク、サムネイル画像を追加する権限です。 output.pdf.encryptionがv2のときだけ有効です true=許可,false=禁止 です。
output.pdf.encryption.permissions.print-high	true	1.2.0 (PDF 1.3)	文書を高画質で印刷する権限です。 output.pdf.encryptionがv2のときだけ有効です true=許可,false=禁止 です。
output.pdf.file-id	ランダムに生成	2.0.9	PDFのファイルIDを設定します。32桁固定の16進数を使用してください。 例: "000067A36902BF8D2A0617B9CD02BCFA"

名前	デフォルト	バージョン	説明
output.pdf.fonts.policy	cid-keyed	1.1.0 outlinesは3.1.1 (PDF 1.2)	デフォルトのフォントの埋め込みポリシーです。cid-keyed, cid-identity, embedded, outlines, -coreで指定します。指定する値と、使用するフォントの対応は以下の通りです。 **cid-keyed** コア14フォント、CID-Keyed外部フォント **cid-identity** コア14フォント、CID Identity外部フォント **embedded** コア14フォント、埋め込みフォント **outlines** 埋め込みフォントを使用し、なおかつフォントをアウトライン化 **-core** コア14フォントを排除 なお、CSSJ 1.x系ではそれぞれgeneric, external, embedというキーワードが使われていました。互換性のため、このキーワードはCopper PDFでも利用可能です。 Copper PDF 2.0.1からは、スペース区切りで複数の指定が可能になりました。例えば"embedded cid-keyed"という指定をすると、埋め込みフォントが見つからない場合はCID Keyedフォントを使用します。 PDF/A-1を出力する場合、この設定は無視され、常に埋め込みフォントだけが使われます。
output.pdf.hyperlinks	false	1.0.0 (PDF 1.2)	ハイパーリンク機能です。falseまたはtrueで指定します。 trueにすると、PDFからWWWなどへのハイパーリンクが有効になります。
output.pdf.hyperlinks.href	relative	1.1.0 (PDF 1.2)	ハイパーリンクのアドレスの記述方法です。relativeまたはabsoluteで指定します。 relativeでは相対アドレス指定となり、HTMLのa要素のhref属性がそのまま使われます。absoluteでは絶対URIに変換されてPDFに反映されます。
output.pdf.hyperlinks.base	ドキュメントのURI	2.0.0 (PDF 1.2)	output.pdf.hyperlinks.hrefにrelativeを指定した場合の、基準となるURIです。output.pdf.hyperlinks.hrefがabsoluteの場合は、このプロパティは無効です。
output.pdf.hyperlinks.fragment	true	2.0.0 (PDF 1.2)	trueを設定するとHTMLの<a name〜あるいはid属性によりドキュメントフラグメントが配置され、URLのフラグメント識別子によって ドキュメント内の特定の場所へリンクすることができるようになります。 falseを設定した場合はドキュメントフラグメントを配置しません。

名前	デフォルト	バージョン	説明
output.pdf.image.compression	flate	2.0.3	画像をPDFに埋め込むときの圧縮形式です。以下の値を指定可能です。 **flate** FlateDecode形式です。ただし output.pdf.compression[ib] がnoneのときは圧縮されずHEX形式になります。 可逆圧縮のため高画質ですが、出力されるファイルのサイズは大きくなります。 **jpeg** JPEG形式です。 **jpeg2000** JPEG2000形式です。PDF 1.5以降の出力で、Java実行環境にJAI-ImageI/Oがインストールされている必要があります。
output.pdf.image.compression.lossless	200	2.0.3	output.pdf.image.compression[io] により非可逆圧縮(JPEG形式等)を使用する場合、非可逆圧縮を適用する画像サイズの閾値です。指定されたサイズ(縦のピクセル数と横のピクセル数を足したもの)より小さければ可逆圧縮(FlateDecode)を使用します。
output.pdf.image.max-width	無制限	3.0.0	PDFで使用される画像の横方向の最大ピクセル数（整数）です。アスペクト比を維持して、画像の幅がこのピクセル数に収まるように自動的に縮小します。これは解像度を制限するためのもので、表示上の物理的な大きさは変わりません。
output.pdf.image.min-width	無制限	3.0.0	PDFで使用される画像の縦方向の最大ピクセル数（整数）です。アスペクト比を維持して、画像の高さがこのピクセル数に収まるように自動的に縮小します。これは解像度を制限するためのもので、表示上の物理的な大きさは変わりません。
output.pdf.jpeg-image	raw	1.1.0 (PDF 1.2)	JPEG画像の埋め込み方法です。raw,to-flate,recompress(Copper PDF 2.0.3以降)で指定します。 rawでは元のデータをそのまま使います。to-flateまたはrecompressを設定すると、データを再圧縮します(Copper PDF 2.0.2以前ではto-flateしか使用できず、文字通りFlateDecode形式に再圧縮していました。Copper PDF 2.0.3以降では output.pdf.image.compression[io] の設定により他の圧縮形式も使用できますが、to-flateとrecompressは全く同じ意味です)。 Copper PDF 2.0.3以降ではシステムにJAI-ImageI/Oがインストールされ、かつPDF 1.5以降の出力でJPEG 2000に対しても有効になります。

名前	デフォルト	バージョン	説明
output.pdf.meta.creation-date	サーバーの現在時刻	2.0.9	PDFのメタ情報のCreationDateを設定します。 設定例: "2009-05-22 21:10:14" "2009-06-04 15:53:02 +09:00" (タイムゾーンを明示する場合)
output.pdf.meta.mod-date	output.pdf.meta.creation-dateの値	2.0.9	PDFのメタ情報のModDateを設定します。 時刻の形式は output.pdf.meta.creation-date[io] と同じです。
output.pdf.open-action.java-script	-	3.0.2/2.1.11 (PDF 1.2)	文書を開いた時に実行するJavaScriptを設定します。
output.pdf.platform-encoding	MS932	1.2.0 (PDF 1.2)	PDFを表示する環境のプラットフォームのキャラクタ・エンコーディングです。 PDF1.2以前ではフォント名が影響を受けます。 PDF1.3以降ではユニコードが使われるため無関係です。 PDF1.6以前では添付ファイル名が影響を受けます。ファイル名にマルチバイト文字が使われている場合、このエンコーディングが表示するプラットフォームのものと一致しないと文字化けします。日本語の文書であればMS932(Windows版Shift_JIS)、韓国語であればEUC-KR、繁体字中国語ではBig5といった指定をしてください。 PDF1.7以降ではユニコードが使われるため無関係です(Copper PDF 2.0.3)。

名前	デフォルト	バージョン	説明
output.pdf.version	1.5	1.1.0	出力されるPDFファイルのバージョンです。 1.2,1.3,1.4,1.5,1.6,1.7が指定可能です(1.5,1.6,1.7はCSSJ 1.xでは対応していません)。PDFのバージョンによって利用できる機能が変わります。指定されたバージョンで未対応の機能を使おうとすると警告が出力され、PDFに反映されません。 Copper PDF 2.1.0からは1.4A-1を指定することができます。 PDFの一定のバージョンで有効になる機能は以下の通りです。 **1.3以降** 40から128ビットの暗号化とfill,extract,assemble,print-high 権限。 **1.4以降** ファイルの添付、PNG半透明化、SVG透明度。 **1.5以降** JPEG 2000画像の使用(要JAI-ImageI/O)。 **1.7以降** 添付ファイル名にユニコードを使用(Copper PDF 2.0.3)。 **1.4A-1** PDF/A-1bに準拠したファイルの出力(Copper PDF 2.1.0)。
output.pdf.viewer-preferences.hide-toolber	false	3.0.2/2.1.11	ビューワアプリケーションのツールバーの非表示、表示を設定します。 trueを設定すると非表示となります。
output.pdf.viewer-preferences.hide-menubar	false	3.0.2/2.1.11	ビューワアプリケーションのメニューバーの非表示、表示を設定します。 trueを設定すると非表示となります。
output.pdf.viewer-preferences.hide-windowUI	false	3.0.2/2.1.11	ビューワアプリケーションのウィンドウ内UI(サムネール、添付など)の非表示、表示を設定します。 trueを設定すると非表示となります。
output.pdf.viewer-preferences.fit-window	false	3.0.2/2.1.11	内容に合わせてビューワアプリケーションのウィンドウサイズをフィットさせるかどうかを設定します。 trueを設定すると非表示となります。
output.pdf.viewer-preferences.center-window	false	3.0.2/2.1.11	内容に合わせてビューワアプリケーションのウィンドウサイズをスクリーンに対して中央表示させるかどうかを設定します。 trueを設定すると中央表示となります。
output.pdf.viewer-preferences.display-doc-title	false	3.0.2/2.1.11 PDF 1.4	ビューワアプリケーションのタイトルバーに文書のタイトルを表示させるかどうかを設定します。 trueを設定すると表示します。

名前	デフォルト	バージョン	説明
output.pdf.viewer-preferences.non-full-screen-page-mode	use-none	3.0.2/2.1.11	ビューワアプリケーションのサイドパネルの表示内容を設定します。 **use-none** しおりかサムネイルパネルを表示します。 **use-outlines** しおりパネルを表示します。 **use-thumbs** サムネイルパネルを表示します。 **use-oc** レイヤーパネルを表示します。
output.pdf.viewer-preferences.print-scaling	app-default	3.0.2/2.1.11 PDF 1.6	ビューワアプリケーションの印刷設定の拡大縮小を設定します。 **scaling-none** 拡大縮小をしません。 **app-default** 拡大縮小をビューワに任せます。
output.pdf.viewer-preferences.duplex	none	3.0.2/2.1.11 PDF 1.7	ビューワアプリケーションの印刷設定の片面・両面印刷の方法を設定します。 **none** ビューワのデフォルト設定のままです。 **simplex** 片面印刷をします。 **flip-short-edge** 短辺綴じで両面印刷をします。 **flip-long-edge** 長辺綴じで両面印刷をします。
output.pdf.viewer-preferences.pick-tray-by-pdf-size	false	3.0.2/2.1.11 PDF 1.7	ビューワアプリケーションの印刷設定の「PDFのページサイズに合わせて用紙を選択」のチェック状態を設定します。 trueを設定するとチェックした状態になります。
output.pdf.viewer-preferences.print-page-range	-	3.0.2/2.1.11 PDF 1.7	初期の印刷対象ページを設定します。 ページはカンマ区切りで "1,2,3,5"のように設定します。範囲をしていするためにハイフンを使って"1-3,5"のように設定することもできます。
output.pdf.viewer-preferences.num-copies	0	3.0.2/2.1.11 PDF 1.7	初期の印刷枚数を設定します。0ではビューワのデフォルトで、その他は2から5が有効な値です。6以上の枚数を設定することができません。

名前	デフォルト	バージョン	説明
output.pdf.watermark.uri	-	2.1.8 PDF 1.4	すかし画像のアドレスを、絶対パスで設定してください。 すかしはPDFの前面または背面に繰り返しパターンとして描画されます。
output.pdf.watermark.mode	back	2.1.8 PDF 1.4	すかし画像の配置方法です。 **front** 前面に配置 **back** 背面に配置
output.pdf.watermark.opacity	1	2.1.8 PDF 1.4	すかし画像の不透明度です。 0〜1までの小数で指定します。
output.pdf.watermark.view	true	2.1.8 PDF 1.4(説明参照)	すかし画像が、画面表示の場合に見えるようにします。 すかしを背面に配置する場合、false(非表示)を設定できるのはPDF 1.5以降です。
output.pdf.watermark.print	true	2.1.8 PDF 1.4(説明参照)	すかし画像が、印刷時に見えるようにします。 すかしを背面に配置する場合、false(非表示)を設定できるのはPDF 1.5以降です。

表 5.6 その他のプロパティ

名前	デフォルト	バージョン	説明
processing.exclude-message	-	3.0.0	除外するメッセージのパターンを設定します。パターンはメッセージコードの16進数表記で、'?'は全ての文字にマッチします。このプロパティは何度も設定でき、除外ルールは設定順に適用されます。デフォルトではどのメッセージも除外されません。 例えば"3???"を設定すると、3000から3FFFまでのメッセージが送られなくなります。
processing.include-message	-	3.0.0	ドライバに送るメッセージのパターンを設定します。設定方法はprocessing.exclude-messageと同じです。 例えばprocessing.include-messageに"3012"を設定した後、processing.exclude-messageに"3???"を設定すると、3000から3FFFまでのメッセージのうち3012だけが送られるようになります。
processing.middle-pass	false	3.0.4	trueを設定すると、実際は結果を生成しない中間の処理を実行します。後でfalseを設定してドキュメントを処理すると、結果が生成されます。 詳細は2パス以上の変換処理を参照してください。
processing.page-references	false	2.0.0	trueを設定すると、目次、ページ参照のための情報を収集します。falseを設定すると、目次、ページ参照のための情報を収集しないため一部の機能が利用できなくなります。 詳細はページの参照を参照してください。
processing.pass-count	1	1.2.0	1回のフォーマット処理のために、文書を処理する回数です。 詳細は2パス以上の変換処理を参照してください。

5.1.1 文書中で設定できないプロパティ

以下のリストにあるプロパティは`input.property-pi`[io]の設定とは関係なく、jp.cssj.property処理命令による設定ができません。

- input.http.で始まるプロパティ
- input.default-encoding
- input.property-pi
- output.type
- processing.pass-count

5.1.2 機能限定版

機能限定版の場合、原則として入出力プロパティはデフォルトのままで固定されます。ただし、以下のプロパティは例外的に変更することができます。

- input.default-encoding
- output.pdf.bookmarks
- output.pdf.hyperlinks
- processing.page-references
- processing.pass-count
- processing.include-message
- processing.exclude-message

また、利用できるフォントがCID-Keyedフォントのみに固定されているため、`-cssj-font-policy`[css]プロパティを利用できません。

5.2 メッセージハンドラから取得できる情報

5.2.1 Copper PDF 2.0以前(CTIP 1.0)

メッセージハンドラに渡される、コード4の処理情報 (66ページ)から得ることができる情報の一覧です。

カテゴリ	値の形式	説明
page-number	数値	これから生成されるページの番号です。最終的なページ番号が総ページ数となります。
heading-title	文字列	見出し(h1〜h6)として認識された文字列です。
broken-image-uri	文字列	表示できない画像のURIです。
pass-count	数値	残りパス数です。
annot	文字列	cssj:annot属性で任意の要素に指定された注釈です。

5.2.2 Copper PDF 2.1以降(CTIP 2.0)

新しいプログラム・インターフェースではメッセージコード (73ページ)により、さらに詳細な情報を得ることができます。

表 5.7 情報

コード	値	説明
1001		abort等により、正常に処理が中断された。
1801	ページ番号(int)	現在処理を開始したページ。
1802	見出し(string)	現在出力した見出し。
1803	パス番号(int)	現在処理を開始した処理のパス。
1804	注釈(string)	現在出力した注釈。
1805	タイトル(string)	ドキュメントのタイトル。
1806*[2.1.2]*	ページの高さ(double)	pt単位のページの高さです。`output.auto-height`[io]がtrueのときだけ通知します。

表 5.8 警告

コード	値	説明
2001	リソースのURI(string)	ドキュメントから参照されたリソースURIの形式の不正。
2002	ベースURI(string)	文書のベースURIの形式の不正。
2801	CSSファイルのURI(string) エラーメッセージ(string)	形式が不正なCSSがあった。
2802	CSSプロパティ名(string)	サポートされないCSSプロパティがあった。
2803	CSSファイルのURI(string)	CSSファイルが存在しない。

コード	値	説明
2804	プロパティ名(string) プロパティ値(string)	プロパティの値の形式が不正。
2805	処理命令名(string) 処理命令の値(string)	不正な形式の処理命令があった。
2806	CSSファイルのURI(string) 深さの限界値(string)	CSSの@importが深すぎる。
2807	参照元CSSのURI(string) 参照先CSSのURI(string)	CSSの@importがループしている。
2808	HTML要素名(string) 属性名(string) 属性地(string)	HTMLの属性名の形式に不正がある。
280A	cssj:header属性の値(string)	cssj:header属性の値の形式に不正がある。
280B	リソースのURI(string)	リソースのURIの形式に不正がある。
280C	リンクのURI(string)	リンクのURIの形式に不正がある。
280D	SVGファイルのURI(string) エラーメッセージ(string)	SVGの形式に不正がある。
280E	XSLTファイルのURI(string)	XSLTファイルが存在しない。
280F		PIによる入出力プロパティの上書きが禁止されている。
2810	添付ファイルのURI(string)	PDFに添付しようとしたファイルが存在しない。
2811	画像ファイルのURI(string)	画像ファイルが存在しない。
2812	PDFバージョン(string) 設定名(string) 設定値(string)	現在のPDFバージョンで利用できない機能を使おうとした。
2813	エラーメッセージ(string)	インラインオブジェクトの形式に不正がある。
2814	リソースのURI(string)	リソースへのアクセスが許可されていない。
2815	CSSプロパティ名(string)	使用を許可されていないCSSプロパティを使おうとした。
2816	CSSプロパティ名(string) 値(string) エラーメッセージ(string)	CSSプロパティの値の形式に不正がある。
2817	インラインスタイル(string) エラーメッセージ(string)	インラインCSSの形式に不正がある。
2818	プロパティ名(string)	サポートされない入出力プロパティがある。
281B	プロパティ名(string)	使用が許可されない入出力プロパティがある。
281C	プロパティ設定ファイルのURI(string)	プロパティ設定ファイルを読み込むことができない。
281D	文字エンコーディング名(string)	サポートされない文字エンコーディング名を使おうとした。
281E	フォントファイルのURI(string)	フォントファイルを読み込むことができない。
281F	対象のテキスト(string)	使用可能なフォントがない。
2820 *[3.2.16]*	対象のテキスト(string)	CID-Keyedフォントまたは絵文字は`{background-clip: text;}`で使えない。

表 5.9 エラー

コード	値	説明
3001	ドキュメントのURI(string)	メインドキュメントのURIの形式に不正がある。

コード	値	説明
3002	エラーメッセージ(string)	入出力エラー。
3801	XSLTファイルのURI(string)	XSLTファイルの形式に不正がある。
3802	制限値(string) 設定値(double)	ページサイズの設定が制限を超えている。
3803	エラーメッセージ(string)	XMLの形式に不正がある。
3804	バイト数(long)	出力ファイルの大きさが制限値を超えている。
3805	制限ページ数(int)	出力ページ数が制限を超えている。
3806	ドキュメントのURI(string)	サーバー側のメインドキュメントが存在しない。
3807		ライセンス認証ファイルが不正。
3808	XSLTファイルのURI(string) メッセージ(string)	XSLTプロセッサの警告メッセージ。
3809	XSLTファイルのURI(string) メッセージ(string)	XSLTプロセッサのエラーメッセージ。
380B		ライセンスファイルの期限切れ。
380C		ライセンスファイルを読み込むことができない。
380D*[3.0.0]*	ドキュメントの内容が空なのでページを生成できない。	

表 5.10 深刻なエラー

コード	値	説明
4001	エラーメッセージ(string)	予期しないエラー。
4801	XSLTファイルのURI(string) メッセージ(string)	XSLTプロセッサの致命的エラーメッセージ。

メッセージコードのフィルタリング *[3.0.0]*

何もしなければドライバには全てのメッセージが送られますが、必要なメッセージだけ送るようにフィルタリングすることができます。

`processing.include-message`[io], `processing.exclude-message`[io]に、それぞれクライアントに送る、クライアントに送らないメッセージコードのパターンを指定します。パターンはメッセージコードの16進数表記で、'?'は全ての文字にマッチします(例: "3???"は3000-3FFFのコードにマッチする)。ルールは設定順に適用され、デフォルトでは全てのメッセージが送られます。

5.3 CSSプロパティのサポート状況

以下の表はW3C CSS2.1仕様の各プロパティのサポート状況です。

□ …対応
□ …一部対応
□ …未対応

表 5.11 HTML/XML要素に対するCSSプロパティ

特性	サポート	備考
azimuth	しない	音声スタイルのため、印刷には無関係です。
background-attachment	する	
background-color	する	
background-image	する	
background-position	する	
background-repeat	する	
background	する	
border-collapse	する	
border-color	する	
border-spacing	する	
border-style	する	
border-top	する	
border-right	する	
border-bottom	する	
border-left	する	
border-top-color	する	
border-right-color	する	
border-bottom-color	する	
border-left-color	する	
border-top-style	する	
border-right-style	する	
border-bottom-style	する	
border-left-style	する	
border-top-width	する	
border-right-width	する	

特性	サポート	備考
border-bottom-width	する	
border-left-width	する	
border-width	する	
border	する	
bottom	する	
caption-side	する	
clear	する	
clip	する	
color	する	
content	する	
counter-increment	する	
counter-reset	する	
cue-after	しない	音声スタイルのため、印刷には無関係です。
cue-before	しない	音声スタイルのため、印刷には無関係です。
cue	しない	音声スタイルのため、印刷には無関係です。
cursor	しない	インタラクティブスタイルのため、印刷には無関係です。
direction	しない	左から右へ書く言語(アラビア語・ヘブライ語など)はサポートしていません。
display	する	
elevation	しない	音声スタイルのため、印刷には無関係です。
empty-cells	する	
float	する	
font-family	する	
font-size	する	
font-style	する	
font-variant	しない	スモール・キャップフォントは利用できません。
font-weight	する	
font	する	
height	する	
left	する	
letter-spacing	する	
line-height	する	
list-style-image	する	
list-style-position	する	

特性	サポート	備考
list-style-type	する	hebrew, armenian, georgianはサポートしません。
list-style	する	
margin-right	する	
margin-left	する	
margin-top	する	
margin-bottom	する	
margin	する	
max-height	する	
max-width	する	
min-height	する	
min-width	する	
orphans	する	
outline-color	しない	インタラクティブスタイルのため、印刷には無関係です。
outline-style	しない	インタラクティブスタイルのため、印刷には無関係です。
outline-width	しない	インタラクティブスタイルのため、印刷には無関係です。
outline	しない	インタラクティブスタイルのため、印刷には無関係です。
overflow	する	
padding-top	する	
padding-right	する	
padding-bottom	する	
padding-left	する	
padding	する	
page-break-after	する	
page-break-before	する	
page-break-inside	する	
pause-after	しない	音声スタイルのため、印刷には無関係です。
pause-before	しない	音声スタイルのため、印刷には無関係です。
pause	しない	音声スタイルのため、印刷には無関係です。
pitch-range	しない	音声スタイルのため、印刷には無関係です。
pitch	しない	音声スタイルのため、印刷には無関係です。
play-during	しない	音声スタイルのため、印刷には無関係です。
position	する	
quotes	する	

特性	サポート	備考
richness	しない	音声スタイルのため、印刷には無関係です。
right	する	
speak-header	しない	音声スタイルのため、印刷には無関係です。
speak-numeral	しない	音声スタイルのため、印刷には無関係です。
speak-punctuation	しない	音声スタイルのため、印刷には無関係です。
speak	しない	音声スタイルのため、印刷には無関係です。
speech-rate	しない	音声スタイルのため、印刷には無関係です。
stress	しない	音声スタイルのため、印刷には無関係です。
table-layout	する	
text-align	する	
text-decoration	する	
text-indent	する	
text-transform	する	
top	する	
unicode-bidi	しない	
vertical-align	する	
visibility	する	
voice-family	しない	音声スタイルのため、印刷には無関係です。
volume	しない	音声スタイルのため、印刷には無関係です。
white-space	する	
widows	する	
width	する	
word-spacing	する	
z-index	する	

表 5.12 ページに対するCSSプロパティ

特性	サポート	備考
margin-top	する	
margin-right	する	
margin-bottom	する	
margin-left	する	
margin	する	

5.4 HTMLの各要素・属性のサポート状況

以下の表はHTMLの各要素のサポート状況です。

表 5.13 HTMLサポート状況一覧

要素・機能	属性	サポート	備考
a	href name	する	name属性を用いた文書内リンクとハイパーリンクをサポートします。XML文書中で使用可能です。
abbr		する	
acronym		する	
address		する	
applet	width height hspace vspace alt align	しない	枠だけが表示されます。
area	href shape coords	する *[3.2.6]*	shapeが円形(circle)または、角が４つ以外の多角形(polygon)のリンクはそれを囲む四角形になります
b		する	
base	href	する	以降のハイパーリンクなどは、hrefからの相対パスになります。XML文書中で使用可能です。
basefont	size color face	しない	font要素と同じ働きをします。
bgsound		しない	
bdo		しない	
big		する	
blink		しない	
blockquote		する	
body	marginheight marginwidth topmargin leftmargin rightmargin bottommargin bgcolor background bgproperties text link	する	alink,vlink属性はサポートしません。
br	clear	する	
button	disabled	しない	
caption	align valign	する	
center		する	
cite		する	
code		する	
colgroup		する	align, bgcolor, charoff, span, valign, width属性はサポートされません。
col		する	charoff属性はサポートされません。

要素・機能	属性	サポート	備考
comment		しない	
dd		する	
del		する	
dfn		する	
dir	type	する	compact属性はサポートしません。
div	align	する	
dl		する	compact属性はサポートしません。
dt		する	
em		する	
embed	border width height hspace vspace alt hidden frameborder units	しない	画像の表示に使用できます。
fieldset	align	する	
font	size color face font-weight point-size	する	
form		しない	枠だけ表示されます。
frame		しない	
framset		しない	
h1 h2 h3 h4 h5 h6	align	する	
head		する	
hr	align color noshade size width	する	
html		する	
i		する	
iframe		しない	
ilayer	background bgcolor clip height src visibility width left pagex pagey top z-index	する	above,below属性はサポートしません。
img	src alt border width height hspace vspace align usemap*[3.2.6]*	する	XML文書中で使用可能です。ただし、XML文書中ではsrc,width,height属性のみ有効です。
input	disabled	しない	枠だけ表示されます。
input [type=checkbox]	size	しない	枠だけ表示されます。
input[type=text]	size	しない	枠だけ表示されます。
input [type=password]	size	しない	枠だけ表示されます。
input[type=file]	size	しない	枠だけ表示されます。

要素・機能	属性	サポート	備考
input [type=radio]		しない	枠だけ表示されます。
input[type=reset]		しない	枠だけ表示されます。
input [type=button]		しない	枠だけ表示されます。
input [type=submit]		しない	枠だけ表示されます。
input [type=image]	src border width height align	しない	img要素と同様に表示されます。
ins		する	
isindex		しない	
kbd		しない	
keygen		しない	
label		しない	
layer	background bgcolor clip height src visibility width left pagex pagey top z-index	する	above,below属性はサポートしません。
legend		しない	ブロックとして表示します。
li	type	する	value属性はCopper PDF 2.1.9 からサポートします。
listing		する	
link	rel type media href	する	rel="StyleSheet" type="text/css"の場合にhrefのCSSスタイルシートをリンクします。これはCSSスタイルシートに対するxml-stylesheet処理命令と同様に動作します。link要素でXSLTスタイルシートをリンクすることはできません。
map	name	する *[3.2.6]*	対応するusemapがある要素の、前か直後にある必要があります。確実にリンクを反映するには2パス以上の処理が必要です。
marquee	bgcolor width height hspace vspace	しない	スクロールはしません。
menu	type	する	compact属性はサポートしません。
meta	http-equiv name content	する	XML文書中で使用可能です。 HTML文書では、`<meta http-equiv="Content-Type" content="text/html; charset=エンコーディング名">` という指定を行うことでエンコーディングを指定可能です。 また、PDF出力の際の文書情報を設定することができます (167 ページ)。
multicol		しない	ブロックとして表示されます。
nextid		しない	
nobr		する	
noembed		しない	表示しません。
noframes		する	表示します。

要素・機能	属性	サポート	備考
nolayer		しない	表示しません。
noscript		する	表示します。
object	border width height hspace vspace alt align (absbottom,absmiddle,texttop を除く) usemap*[3.2.6]*	しない	画像の表示に使用できます。
ol	type	する	compact属性はサポートしません。 start属性はCopper PDF 2.1.9 からサポートします。
optgroup		しない	
option		しない	
p	align	する	
param		しない	
plaintext		する	
pre	cols width wrap	する	
q		する	
ruby rb rt rp		する *[3.0.0]*	Copper PDF 3.0.4 からはHTML5(rtがない)形式のrubyに対応しました。
s		する	
samp		する	
script		しない	表示しません。
select		しない	枠だけ表示されます。
server		しない	
small		する	
spacer		しない	
span		する	
strike		する	
strong		する	
style	disabled type media	する	XML文書中で使用可能です。
sub		する	
sup		する	
table	width height bgcolor bgackground align hspace vspace border frame rules cellspacing cellpadding	する	bordercolordark,bordercolorlight,cols,summary属性はサポートされません。
tbody	align bgcolor valign	する	charoff属性はサポートされません。
thead	align bgcolor valign	する	charoff属性はサポートされません。

要素・機能	属性	サポート	備考
tfoot	align bgcolor valign	する	charoff属性はサポートされません。
td	bordercolor background bgcolor align valign height width nowrap colspan rowspan	する	charoff,bordercolordark,bordercolorlight属性はサポートされません。colspan,rowspan属性は"display: table-cell"スタイルが指定されている要素に付けることでXML文書でも利用可能です。
th	bordercolor background bgcolor align valign height width nowrap colspan rowspan	する	charoff,bordercolordark,bordercolorlight属性はサポートされません。colspan,rowspan属性は"display: table-cell"スタイルが指定されている要素に付けることでXML文書でも利用可能です。
tr	bordercolor background bgcolor align valign height	する	charoff,bordercolordark,bordercolorlight属性はサポートされません。
textarea	cols rows disabled	しない	枠だけ表示されます。wrap属性は無視されます。
title		する	タイトルバーまたは、PDF文書情報のタイトルとして使われます。
u		する	
ul	type	する	compact属性はサポートされません。
var		する	
wbr		する	
xmp		する	
dir一般属性		しない	
style一般属性		する	XML文書中で使用可能です。

5.5 拡張機能

Copper PDFには独自の処理命令、CSSプロパティ、CSS関数、XML要素、XML属性があります。また、CSS 2.1ではサポートされず、CSS 3で追加されるプロパティを先行して実装したものがあります。

5.5.1 処理命令の拡張

表 5.14 処理命令一覧

名前	バージョン	説明
jp.cssj.default-encoding	1.1.0	これはHTMLの<meta http-equiv="Content-Type" content="text/html; charset=...">要素の代替機能を提供するものです。エンコーディング名を値に使用します。
jp.cssj.default-style-type	1.1.0	これはHTMLの<meta name="content-style-type" content="...">要素の代替機能を提供するものです。MIME型を値に使用します。
jp.cssj.document-info	1.1.0	これはHTMLの<meta name="..." content="...">要素の代替機能を提供するものです。name,content属性に対して、name,value擬似属性が用意されています。
jp.cssj.property	2.0.0	`input.property-pi`[io]がtrueのときだけ利用可能です。 入出力プロパティをドキュメント中で再設定します。name,value擬似属性が用意されています。valueを省略すると、デフォルト値に設定されます。
jp.cssj.stylesheet	1.1.0	これはHTMLのstyle要素の代替機能を提供するものです。type,media属性に対して、同名の擬似属性が用意されています。スタイルシートは'[]'で囲って記述します。

5.5.2 CSSプロパティの拡張

CSSプロパティ

表 5.15 CSSプロパティー覧

名前	バージョン	継承	デフォルト値	適用対象	説明
-cssj-font-policy	2.0.0	✔	cid-keyed	すべての要素	独自プロパティです 使用するフォントの種類を指定します。指定できる値はcid-keyed, cid-identity, embedded, outlines*[3.1.1]*のいずれかです。詳細はフォントの設定の章を参照してください。 Copper PDF 2.0.1以降では、複数の値を指定可能になりました。例えば"embedded cid-keyed"という指定をすると、埋め込みフォントが見つからない場合はCID Keyedフォントを使用します。 コアフォントは常に使われます。ただし、-coreという指定をすると除外されます*[3.0.0]*。 PDF/A-1を出力する場合、この設定は無視され、常に埋め込みフォントだけが使われます。

名前	バージョン	継承	デフォルト値	適用対象	説明
-cssj-page-content	2.0.0		none	すべての要素	独自プロパティです 要素をページごとに生成します。1つめの値は、ページごとに生成されるコンテンツの名前です。2つめ以降の値は、コンテンツを生成するページ(first, left, right, singleのいずれか)で、省略するか複数列挙することができます。詳細はページごとに生成される内容の章を参照してください。
-cssj-page-content-clear	2.0.0		none	すべての要素	独自プロパティです `-cssj-page-content` [css] によりページごとに生成されるコンテンツの再生成を停止します。指定する値は、ページごとに生成されるコンテンツの名前で、複数列挙することができます。詳細はページごとに生成される内容の章を参照してください。
-cssj-background-size background-size	2.0.8		auto	すべての要素	CSS3 Backgrounds and Borders の先行実装です。 背景画像のサイズを指定します。1つめの値は画像の幅で、2つめの値は高さです。%指定は、要素の幅または高さに対する割合です。backgroundでも指定可能です*[3.2.16]*。
-cssj-text-align-last text-align-last -epub-text-align-last	2.0.8	✔	start	すべての要素	CSS3 Text の先行実装です。 段落末のテキストの合わせ方です。値はstart, end, left, right, center, justify のいずれかです。 プロパティ名 `-epub-text-align-last` [css] は Copper PDF 3.0.4 から対応しています。
-cssj-writing-mode writing-mode -epub-writing-mode	3.0.0	✔	horizontal-tb	テーブル行グループ、テーブルカラムグループ、テーブル行、テーブルカラム以外の要素	CSS3 Writing Modes の先行実装です。 縦書き、横書きを設定します。値はhorizontal-tb(横書き), vertical-rl(縦書き)のいずれかです。Internet Explorer/SVG 1.1との互換性のため、lr, lr-tb, rl, tb, tb-rlも値として設定可能です。 プロパティ名 `-epub-writing-mode` [css] は Copper PDF 3.0.4 から対応しています。
-cssj-direction-mode	3.0.0	✔	physical	すべての要素	独自プロパティです マージン、境界などを縦書きで回転させるかどうかを設定します。値はphysical, logical, horizontal-tb*[3.0.12]*, vertical-rl*[3.0.12]*のいずれかです。詳細は論理方向モード (250ページ)を参照してください。

名前	バージョン	継承	デフォルト値	適用対象	説明
-cssj-column-width column-width	3.0.0		auto	置換不可能なブロックレベル要素（テーブルを除く）、テーブルセル、インラインブロック	CSS3 Multi-column Layout の先行実装です。 段組の幅を設定します。値はautoまたは長さです。
-cssj-columns columns	3.0.0			置換不可能なブロックレベル要素（テーブルを除く）、テーブルセル、インラインブロック	CSS3 Multi-column Layout の先行実装です。 `column-width` [css], `column-count` [css] の一方の値か、あるいは両方の値をまとめて設定することができます。
-cssj-column-count column-count oeb-column-number	3.0.0		auto	置換不可能なブロックレベル要素（テーブルを除く）、テーブルセル、インラインブロック	CSS3 Multi-column Layout の先行実装です。 段組の数を設定します。値はautoまたはカラムの数です。 プロパティ名 `oeb-column-number` [css] は Copper PDF 3.0.4 から対応しています。
-cssj-column-gap column-gap	3.0.0		normal	段組された要素	CSS3 Multi-column Layout の先行実装です。 段組の間の幅を設定します。値はnormalまたは長さです。
-cssj-column-rule-color column-rule-color	3.0.0		色	段組された要素	CSS3 Multi-column Layout の先行実装です。 段組の間に入る罫線の色を設定します。
-cssj-column-rule-style column-rule-style	3.0.0		none	段組された要素	CSS3 Multi-column Layout の先行実装です。 段組の間に入る罫線のスタイルを設定します。値は `border-*-style` [css]の値と同じです。
-cssj-column-rule-width column-rule-width	3.0.0		medium	段組された要素	CSS3 Multi-column Layout の先行実装です。 段組の間に入る罫線の幅を設定します。値は`border-*-width` [css]の値と同じです。
-cssj-column-rule column-rule	3.0.0			段組された要素	CSS3 Multi-column Layout の先行実装です。 段組の間に入る罫線の色、スタイル、幅をまとめて設定します。値は `border-*` [css]の値と同じです。
-cssj-column-full column-full	3.0.0		balance	段組された要素	CSS3 Multi-column Layout の先行実装です。 段組の最後を均等に揃えるかどうかの設定です。値はauto, balanceのいずれかです。

名前	バージョン	継承	デフォルト値	適用対象	説明
-cssj-column-span column-span	3.0.0		1	静的な、浮動体以外の要素	CSS3 Multi-column Layout の先行実装です。 段落のブチヌキを設定します。値は1またはallです。
-cssj-text-combine -epub-text-combine	3.0.4	✔	none	すべての要素	CSS3 Text の先行実装です。 縦中横を実現するためのものです。現在のところ、horizontalだけを指定でき、`{writing-mode: horizontal-tb;}`を指定するのと変わりません。**このプロパティは将来仕様が変更されるか、サポートされなくなる可能性があります。**
-cssj-text-emphasis -epub-text-emphasis	3.0.4	✔	none	すべての要素	CSS3 Text の先行実装です。 `-cssj-text-emphasis-style` [css], `-cssj-text-emphasis-color` [css] をまとめて指定します。詳細は圏点 (207ページ)を参照してください。
-cssj-text-style -epub-text-style	3.0.4	✔	none	すべての要素	CSS3 Text の先行実装です。 圏点のスタイルを指定します。詳細は圏点 (207ページ)を参照してください。
-cssj-text-color -epub-text-color	3.0.4	✔		すべての要素	CSS3 Text の先行実装です。 圏点の色を指定します。詳細は圏点 (207ページ)を参照してください。
src	3.0.0			@font-faceルール	CSS3 Font の先行実装です。 フォントの位置を示します。詳細はWebFont (242ページ)を参照してください。
unicode-range	3.0.0		U+0-10FFFF	@font-faceルール	CSS3 Font の先行実装です。 フォントのコード範囲です。詳細はWebFont (242ページ)を参照してください。
word-wrap	3.0.0	✔	normal	すべての要素	CSS3 Text の先行実装です。 英単語の中での折り返しを許可するかどうかの設定です。値はnormal, break-wordのいずれかです。
word-break	3.2.2	✔	normal	すべての要素	CSS3 Text の先行実装です。 禁則処理の設定です。値はnormal, break-all, keep-allのいずれかです。
opacity	3.0.6		1	すべての要素	CSS Color Module Level 3 の先行実装です。 要素の透明度を指定します。詳細は透明化 (215ページ)を参照してください。
border-top-left-radius	3.0.6		0	すべての要素	CSS Backgrounds and Borders Module Level 3 の先行実装です。 境界線の左上の半径を指定します。詳細は角丸境界 (217ページ)を参照してください。

名前	バージョン	継承	デフォルト値	適用対象	説明
border-top-right-radius	3.0.6		0	すべての要素	CSS Backgrounds and Borders Module Level 3 の先行実装です。 境界線の右上の半径を指定します。詳細は角丸境界 (217ページ)を参照してください。
border-bottom-left-radius	3.0.6		0	すべての要素	CSS Backgrounds and Borders Module Level 3 の先行実装です。 境界線の左下の半径を指定します。詳細は角丸境界 (217ページ)を参照してください。
border-bottom-right-radius	3.0.6		0	すべての要素	CSS Backgrounds and Borders Module Level 3 の先行実装です。 境界線の右下の半径を指定します。詳細は角丸境界 (217ページ)を参照してください。
border-radius	3.0.6			すべての要素	CSS Backgrounds and Borders Module Level 3 の先行実装です。 境界線の半径をまとめて指定します。詳細は角丸境界 (217ページ)を参照してください。
-cssj-transform -webkit-transform -moz-transform transform*[3.2.16]*	3.0.8		none	ブロックレベル要素	CSS Transforms の先行実装です。 ２次元のアフィン変換を指定します。３次元変換には対応していません。詳細は回転・縮小・変形 (220ページ)を参照してください。
-cssj-transform-origin -webkit-transform-origin -moz-transform-origin transform-origin*[3.2.16]*	3.0.8		50% 50%	ブロックレベル要素	CSS Transforms の先行実装です。 transformプロパティによる変換の基点を指定します。詳細は回転・縮小・変形 (220ページ)を参照してください。
-cssj-text-fill-color -webkit-text-fill-color	3.0.8			すべての要素	Chrome/Safariとの互換性のための独自プロパティです テキストの塗りつぶし色を指定します。詳細は袋文字 (212ページ)を参照してください。
-cssj-text-stroke-color -webkit-text-stroke-color	3.0.8			すべての要素	Chrome/Safariとの互換性のための独自プロパティです テキストの枠線の色を指定します。詳細は袋文字 (212ページ)を参照してください。
-cssj-text-stroke-width -webkit-text-stroke-width	3.0.8		0	すべての要素	Chrome/Safariとの互換性のための独自プロパティです テキストの枠線の太さを指定します。詳細は袋文字 (212ページ)を参照してください。
-cssj-text-stroke -webkit-text-stroke	3.0.8			すべての要素	Chrome/Safariとの互換性のための独自プロパティです テキストの枠線の太さと色を指定します。詳細は袋文字 (212ページ)を参照してください。

名前	バージョン	継承	デフォルト値	適用対象	説明
text-shadow	3.0.8		none	すべての要素	CSS Text Level 3 の先行実装です。テキストの影を指定します。詳細は文字の影 (211ページ)を参照してください。
-cssj-no-break-characters	3.0.6	✔	none	すべての要素	独自プロパティです 禁則文字を追加します。詳細は禁則処理 (203ページ)を参照してください。
-cssj-break-characters	3.0.6	✔	none	すべての要素	独自プロパティです 禁則文字を解除します。詳細は禁則処理 (203ページ)を参照してください。
background-clip	3.2.16		border-box	すべての要素。	CSS Backgrounds and Borders Module Level 4 の先行実装です。値がtextの場合、テーブルと複数カラムには未対応であり、CID-Keyedフォントまたは絵文字以外のフォントに対してのみ有効です。
box-sizing	3.1.10		content-box	`width` [css], `height` [css]を指定可能な要素	CSS Basic User Interface Module Level 3 の先行実装です。

CSS関数

表 5.16 CSS関数一覧

名前	バージョン	引数の数	引数の型	適用プロパティ	説明
-cssj-heading	1.2.0	1	整数	content	いちばん最後に表示された見出し(HTML の h1 〜 h6 または cssj:header属性がついた要素)の内容を出力します。引数の値は、見出しのレベルです。
-cssj-page-ref	2.0.0	2,3,4	文字列[, 文字列, 文字列]	content	指定したドキュメントフラグメントでのカウンタの値を出力します。詳細はリンクとフラグメントの節を参照してください。

名前	バージョン	引数の数	引数の型	適用プロパティ	説明
-cssj-cmyk	1.0.0 オーバープリントの指定は3.1.0	3,4	整数 小数 パーセント値	color 他、色を指定するプロパティ	CSSのrgbカラーの代わりに、CMYKで色を指定します。引数の値はそれぞれCyan, Magenta, Yellow, Black, オーバプリントモードの順です。 オーバープリントモードはstandard, illustratorのいずれかです。standardの場合はインクを重ねあわせしませんが、illustratorはインクを重ねます。デフォルトはstandardです。 オーバープリントモードの指定はPDF出力のみ有効です。
-cssj-gray	2.0.0	1	整数 小数 パーセント値	color 他、色を指定するプロパティ	CSSのrgbカラーの代わりに、グレイスケールで色を指定します。引数の値は黒味の強さです。
linear-gradient	3.2.16	-	角度 色 パーセント値	background	グラデーション塗りを適用します。ただし、使用できる色が2色までで、3色目以降は無視されます。
rgba	3.0.8	4	整数 小数 パーセント値	color 他、色を指定するプロパティ	CSS Color Module Level 3の先行実装です。 rgbカラーに加えて不透明度(Alpha)を指定します。引数の値はそれぞれRed, Green, Blue, Alphaの順です。詳細は透明色 (216ページ)を参照してください。

CSS識別子

表 5.17 CSS識別子一覧

名前	バージョン	適用対象	説明
~~-cssj-decimal-full-width~~ -cssj-full-width-decimal*[3.0.0]*	2.1.2	list-style-typeプロパティ counter関数	decimal同様に番号を出力しますが、全角文字を用います。
-cssj-cjk-decimal	3.0.0	list-style-typeプロパティ counter関数	位取り漢数字を出力します。
-cssj-title	3.1.4	contentプロパティ	文書のタイトル(<title>タグの内容)を出力します。

名前	バージョン	適用対象	説明
pages	3.1.4	counter関数	CSS Paged Media Module Level 3 の先行実装です。 文書の総ページ数を記録するカウンタです。2パス以降の処理で有効です。
page	3.0.0	page-break-beforeプロパティ page-break-afterプロパティ	CSS3 Multi-column Layout の先行実装です。 alwaysと同じ意味です。
column	3.0.0	page-break-beforeプロパティ page-break-afterプロパティ	CSS3 Multi-column Layout の先行実装です。 強制的に改段します。
transparent	3.2.16	color等の色指定全般	rgba(0, 0, 0, 0)と同値です。

CSSルール

表 5.18 CSSルール一覧

名前	バージョン	説明
@-cssj-page-content	3.1.4	@pageルール内で、再生成ボックスの作成を指定します。詳細はページごとに生成するコンテンツの節を参照してください。

CSS擬似クラス

表 5.19 CSS擬似クラス一覧

名前	バージョン	説明
root	3.2.16	文書のbody要素に対応します。

CSS単位

表 5.20 CSS単位一覧

単位	バージョン	説明
rem	3.1.9	ルート要素(HTMLまたはBODY)のフォントサイズです。
ch	3.1.9	CSS3の仕様上は数字の0の幅ですが、現状はexと同じです。

5.5.3 XMLの拡張

Copper PDFはXML中で特別の意味をもつ要素、属性を処理します。

以降の表では、便宜上以下のように接頭辞と名前空間が対応しているものとして記述します。当然、実際のドキュメント中では別の接頭辞を使うことができます(xmlで始まる接頭辞を除く)。なお、接頭辞のないものは、任意の名前空間に属することを意味します。

接頭辞	名前空間
cssj	http://www.cssj.jp/ns/cssjml
html	http://www.w3.org/1999/xhtml

接頭辞	名前空間
svg	http://www.w3.org/2000/svg

XML要素

表 5.21 XML要素一覧

名前	バージョン	属性	説明
cssj:make-toc	2.0.0	counter, type	目次を生成します。counterはページ番号付けに使用するページカウンタの名前で、typeはページ番号のスタイルです。詳細は目次の章を参照してください。
html:img	1.0.0	alt, src, width, height	HTMLのimg要素と同等の働きをします。
html:a	1.0.0	href,name	HTMLのa要素と同等の働きをします。
html:br	2.0.0		HTMLのbr要素と同等の働きをします。
html:h1〜html:h6	1.0.0		HTMLのh1〜h6要素と同等の働きをします。
svg:svg	1.2.0		SVG画像として処理します。詳細はインラインSVGの節を参照してください。

XML属性

表 5.22 XML属性一覧

名前	バージョン	説明
cssj:annot	1.2.0	処理中に注釈メッセージを出力します。設定された値がannotメッセージとしてドライバに送り返されます。
cssj:header	1.2.0	一般的な要素にHTMLのh1〜h6と同じ意味を持たせ、見出しとして目次やブックマークの生成に使います。値は見出しのレベルです。
html:style	1.0.0	HTMLのstyle属性と同等の働きをします。
html:class	1.0.0	HTMLのclass属性と同等の働きをします。
html:colspan	1.0.0	HTMLのcolspan属性と同等の働きをします。
html:rowspan	1.0.0	HTMLのrowspan属性と同等の働きをします。
xml:lang	2.0.0	HTMLのlang属性と同等の働きをします。
id	1.0.0	HTMLのid属性と同等の働きをします。

GNN 発売元 株式会社 GNN
〒103-0022 東京都中央区日本橋室町1-10-10 LXS室町803号
e-mail: info@cssj.jp
ホームページ: http://www.gnn.co.jp/
Copper PDFサイト: https://copper-pdf.com/