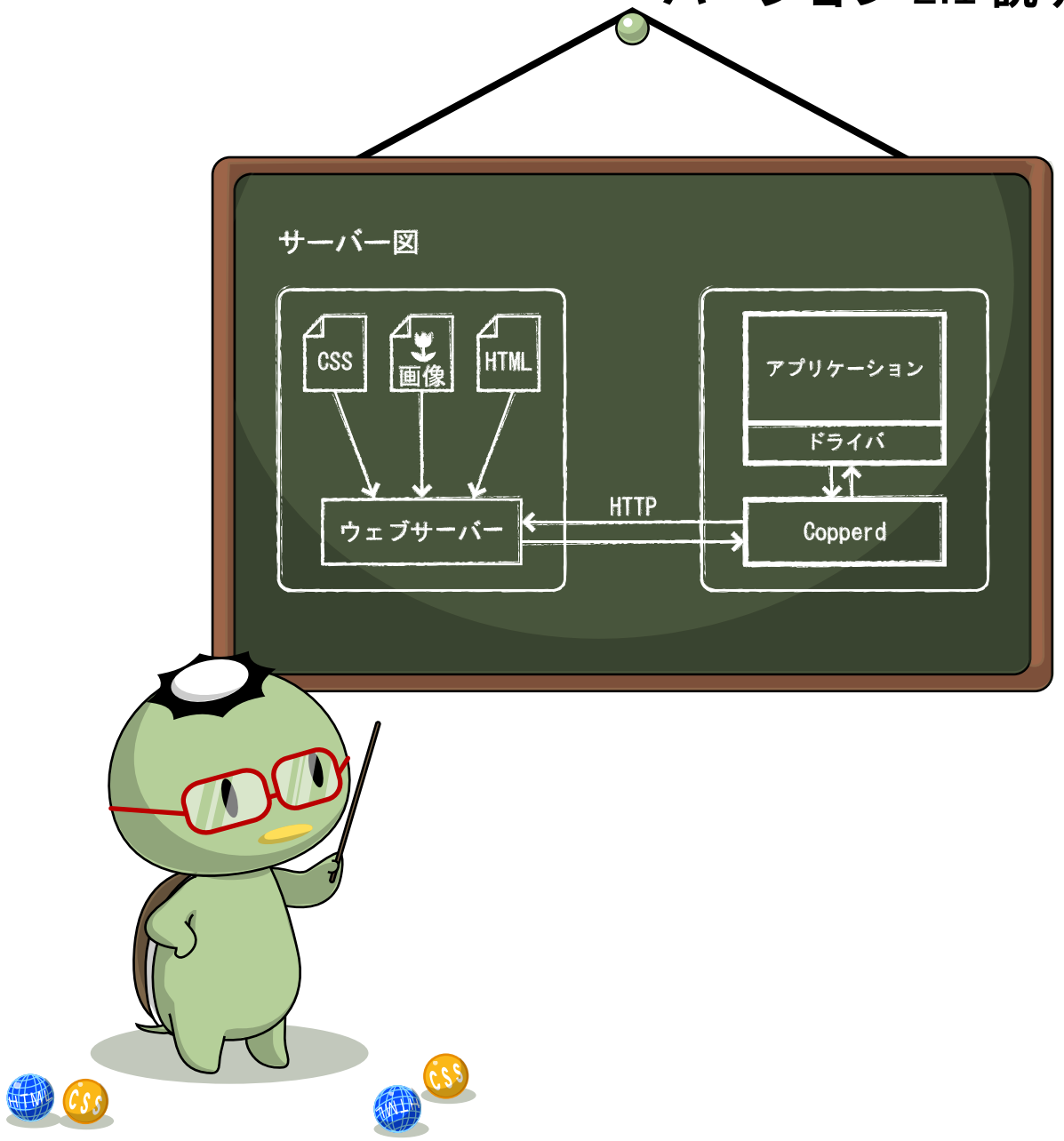
HTML→PDF変換サーバー

Copper PDF

バージョン 2.1 説明書



株式会社 GNN

2011-12-27

NOTICE

目次

1.はじめに

1.1 Copper PDFについて

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。一般的なブラウザでページ分割された文書を表示する場合に比べて、高速・高品質でレイアウト出来るように独自に機能強化されています。また文書を表示するブラウザ等のクライアントの環境に関係なく同一の表示が保障され、単なるHTML文書では実現出来ない、PDF特有の機能を利用することが出来ます。ECサイト等での帳票類の出力、統計資料やマニュアル等の文書生成、官公庁や銀行等での各種申請書類の出力等、幅広く利用が可能です。

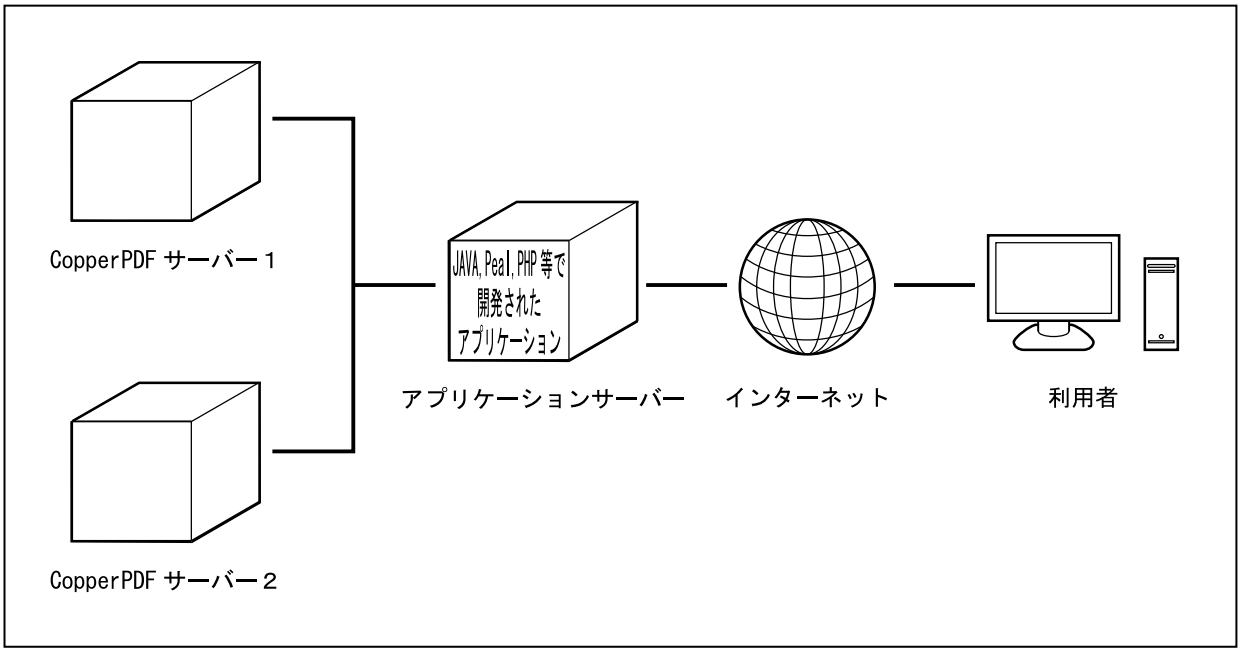
HTML, CSS以外にもXML, SVG, XSLTといったウェブ向けの標準規格に対応しており、JPEG, PNG等の画像を読み込んでPDFに変換することが出来ます。PDF以外の各種画像フォーマットへの変換も可能です。

Copper PDFは100% Pure Javaアプリケーションであり、Java実行環境を利用出来る、Linux, Windowsをはじめとする様々な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をはじめとするプログラミング言語向けに、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のもととなる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。

高速で強力な独自の通信プロトコル(CTIP)の他、HTTP/HTTPSによる接続をサポートしており*[2.1.0]*、Ruby, Python, 4D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



1.2 動作環境

1.2.1 サーバー

copperd(CopperPDF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)および copper Antタスクの実行には最低限以下の環境が必要です。

- Java JRE/JDK(国際化対応版) バージョン1.4.2, 5.0, 6.0
- 128MB以上の空きメモリ
- 50MB以上の空きディスクスペース

以下の環境を推奨します。

- Java JDK バージョン1.4.2, 5.0, 6.0
- 1GB以上の空きメモリ
- 2GB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 2000/XP/Vista
- Red Hat Enterprise Linux(RHEL) 4/5
- Debian 4.0 "etch"/5.0 "lenny"
- Solaris 10
- MacOS X
- FreeBSD 7.2

ウェブインターフェースは以下のブラウザが必要です。

- Firefox 2 以降
- Internet Explorer 7 以降
- Google Chrome 2 以降
- Safari 4 以降

1.2.2 プログラミングインターフェース

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出来ます。

旧インターフェース(CTIP 1.0)

Javaドライバ

Java バージョン1.4.2, 5.0, 6.0

Perlドライバ

Perl バージョン5.6.1以降

PHPドライバ

PHP バージョン4.3.0以降またはPHP バージョン5.0.0以降

新インターフェース(CTIP 2.0)

Java ドライバ / transcode Antタスク

Java バージョン 5.0, 6.0*[2.1.0]*

Java ドライバ 2.0.x までは JDK1.4.2 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 (126ページ) - HTML/XHTML文書をレイアウトします。
- XML (126ページ) - XML文書をレイアウトします。

スタイルシート

- CSS 2.1 (119ページ) - HTML/XHTML/XML文書をCSSでスタイル付けします。
- XSLT (130ページ) - XML文書をXSLTで変換します。
- メディアタイプの設定 (119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 (123ページ)
- ユーザーのXSLTスタイルシート (130ページ)

ベクター画像データ

- SVG (133ページ) - SVGを直接PDF等のデータ形式に変換します*[2.1.0]*。または、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

画像を直接PDF等のデータ形式に変換します*[2.1.0]*。または、文書中に画像を埋め込みます。

- GIF/PNG (132ページ)
- JPEG (132ページ) - JPEGをそのままPDF中に埋め込むことが出来ます。
- その他Java実行環境が対応可能な画像形式 (132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 (149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 (155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 (152ページ)*[2.1.8]*
- PDFの圧縮の設定(無圧縮/Flate圧縮/ASCIIテキスト) (151ページ)
- 画像の圧縮(可逆圧縮/JPEG/JPEG2000) (151ページ)
- 添付ファイル (150ページ) - PDFに別のファイルを添付します。
- ブックマーク (127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 (149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 (149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出来ます。
■ ハイパーリンク (129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 (129ページ) - PDFにリンクを反映する他、ページ番号を出力出来ます。
■ メタ情報(文書タイトル、Creator、CreationDate等)の設定 (126ページ)
■ CreationDate, ModDate, ファイルIDの設定 (152ページ)
■ ViewerPreferencesの設定 (155ページ)*[3.0.2/2.1.11]*
■ オープン時に実行されるJavaScriptの設定 (155ページ)*[3.0.2/2.1.11]*

1.3.3 画像出力

最後のページだけを画像として画像として出力します。

■ JPEG/PNG (118ページ)
■ その他Java実行環境が対応可能な画像形式 (132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 (118ページ) - 任意の解像度で画像を出力出来ます。

1.3.4 フォント関連機能

■ CID-Keyedフォント (36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出来るPDFを出力出来ます。
■ 外部(CID Identity)フォント (36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を出力出来ます。
■ 埋め込みフォント (30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 (31ページ)
■ フォント幅情報・MSフォントの使用 (37ページ)

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 (114ページ)
■ ヘッダ(Referer, User-Agent等)の変更 (112ページ) - 参照元やUser-Agentを偽装することが出来ます。
■ プロクシを介した接続 (112ページ)
■ BASIC認証/Digest認証 (111ページ)
■ クッキー (113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 (135ページ)
■ ページの拡大・縮小・変形 (135ページ)
■ トンボの出力 (135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 (137ページ)
■ ページのマージン (135ページ)
■ 片面印刷・両面印刷の切り替え (137ページ)

■ グレイスケール変換 (137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 (117ページ) - 目次やページ参照のために、文書を複数回処理出来ます。
■ 欄外のページ番号、見出し (138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 (128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 (117ページ) - 文書中の指定した部分のページ番号を出力出来ます。

1.3.8 プログラムインターフェース

Copper PDF 2.0以前

■ Java (55ページ)
■ PHP (62ページ)
■ Perl (59ページ)
■ Ant (65ページ)
■ 処理中のページ、内容の取得 (48ページ)
■ 処理状況の取得 (48ページ)

Copper PDF 2.1以降

■ Java (JRuby) (69ページ)
■ PHP (85ページ)
■ Perl (82ページ)
■ .NET (C# / VB.NET) (88ページ)
■ Ant (97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 (99ページ) (Apache経由での接続 (27ページ) / Ruby (102ページ) / Python (103ページ) / C# (104ページ) / ASP.NET (C# / VB.NET) (106ページ) / 4th Dimension (107ページ))

1.3.9 その他の機能

■ 改ページをせずに文書を出力する (137ページ)
■ 解像度(pxの大きさ)の設定 (123ページ)
■ IE互換モード (130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 (115ページ) - DoS攻撃対策等のため出力サイズを制限出来ます。
■ ページ数制限 (115ページ) - DoS攻撃対策等のため出力ページ数を制限出来ます。
■ 実行経過情報取得 (172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出来ます。

1.4 CSSJ 1.x系統からの変更点

1.4.1 Copper PDFへの移行

CSSJ 1.x向けのJava, PHP, Perlのドライバは、そのままCopper PDFでも利用可能です。以前のドライバを利用したアプリケーションは、接続先をCopper PDFにするだけで、PDFの出力サーバーをCSSJ 1.xからCopper PDFに移行することが出来ます。

ただし、文書のレイアウト機能、サーバー側の設定ファイルについては上位互換性を維持していません。Copper PDFへの移行は、新しい機能を使う場合だけ行ってください。なお、CSSJ 1.x系統のメンテナンス期限は**2013年3月31日まで**です。

1.4.2 主な変更

配布パッケージ

CSSJ 1.xではサーバーとドライバを一緒に配布していましたが、Copper PDFではドライバは別途配布しています。弊社サイト(http://copper-pdf.com/download/) から、利用環境に合うものをダウンロードしてください。

設定ファイル

CSSJ 1.xのresourcesディレクトリがなくなり、全ての設定はconfディレクトリにまとめられました。また、fonts.xml設定ファイルの仕様が大きく変更されたため、CSSJ 1.xのものをCopper PDFで使用することは出来ません。詳細はフォントの設定 (30ページ)を参照してください。

改ページ可能な場所

Copper PDFでは改ページ制御関連の機能が強化されました。詳細は「改ページ制御」(140ページ)の章を参照してください。

CSSJ 1.xでは常に禁止されていた、テーブルセル内部、浮動ボックス内部での改ページが出来るようになりました。また、CSSJ 1.xでは`{page-break-inside: avoid;}`で改ページが禁止された領域が ページ高さを超えてしまう場合は内容がページからはみ出していましたが、Copper PDFでは最悪の場合は改ページするように改善されています。

1.4.3 廃止・置き換えられた機能

ライブラリとしてリンクするJavaドライバの廃止

Copper PDFからは、ライブラリとしてリンクさせて動作するJavaドライバはサポートしません。

Copper PDFはサーブレットコンテナ等で広く使用されている各種ライブラリに依存するため、他のアプリケーションから利用する場合に、ライブラリの競合が問題となることがあることと、近年はマルチコアやハイパースレッディングに対応したCPUが普及したため、ソケット通信により処理を分散したほうが高速に動作するのが実情であり、ライブラリとしてリンクさせることのメリットがないためです。

-cssj-regeneratable[css]の非推奨

-cssj-regeneratable[css]、-cssj-regeneratable-clear[css]は下位互換性のため Copper PDF でも使用出来ますが、代わりに -cssj-page-content[css]、-cssj-page-content-clear[css]の使用を推奨します。詳細は「ページごとに生成されるコンテンツ」 (138ページ)を参照してください。

なお、{position: fixed;}が指定されたボックスの 再生成は完全に廃止されました。

<cssj:toc>要素の廃止

<cssj:toc>要素が廃止され、<cssj:make-toc>に置き換えられ、仕様が変更されました。詳細は「目次の生成」 (128ページ)を参照してください。

また、新しい目次の生成の仕様では常に２度目のパスから目次を生成するため、目次を生成するタイミングを指定する以下の入出力プロパティは廃止されました。

- processing.make-toc-before[io]

ウィンドウアプリケーションと直接印刷の廃止

CSSJ無料版として配布していたウィンドウアプリケーションをCopper PDFでは廃止し、PDFを出力するウェブアプリケーションに置き換えました。それに伴い、ウィンドウアプリケーションで表示する際の見栄えを設定する、以下の入出力プロパティを廃止しました。

- output.j2d.antialiasing[io]
- output.j2d.pdf-layout[io]

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、http://www.java.com/download/ 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実行環境はバージョン1.4.2以降の国際化対応Java実行環境です。JDK(開発環境およびサーバー用Java VMを含むもの)を推奨します。

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。(2.x.xの部分はCopper PDFのバージョンにより異なります)

copper-pdf-2_x_x.zip
ZIPアーカイブ

copper-pdf-2.x.x-x.noarch.rpm
RPMパッケージ

copper-pdf_2.x.x_all.deb
Debianパッケージ

copper-pdf-2.x.x.tar.gz
tar.gzアーカイブ

2.1.3 Windows 2000/XP/Vista

http://copper-pdf.com/download/ で配布されているZIPアーカイブ copper-pdf-2_x_x.zip をダウンロードしてください。

copper-pdf-2_x_x.zip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_x/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と、Cドライブ以外にある場合はサービスの起動が出来ません。Cドライブ直下、あるいは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。

あるいは、後で説明するJava Service Wrapperの使用を推奨します。

Windows XP以前

Windows版にはCopper PDFサーバーをサービスとしてインストールするためのバッチファイルが用意されています。サービスをインストールするには、Administrator権限を持つユーザーでInstallService.bat(x64版JavaVM の場合はInstallService-x64.bat)を実行するだけです。サービスを削除する場合はRemoveService.bat(x64 版 JavaVM の場合はRemoveService-x64.bat) を実行してください。

Windows Vista以降

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RemoveService-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InstallService-x64.bat)を実行してください。

例 2.1 Windows Vista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
```

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 によってサービスとしてインストールことが出来ます。付属のプログラムより制約が少なく、多機能です。Java Service Wrapperの全ての機能を利用し、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。

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。

- CopperPDFディレクトリ直下にbinディレクトリを作り、bin/wrapper.exeをその中にコピーする。
- 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。
- 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。
- 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・停止することが出来ます。

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出来ます。

例 2.2 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
```

2.1.4 Red Hat Enterprise Linux(RHEL) 4/5

RHEL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。

copper-pdf-2.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 2.3 RPMパッケージのインストール

```
# sudo rpm -ivh copper-pdf-2.x.x-0.noarch.rpm
```

例 2.4 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サーバーを管理出来ます。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 2.5 copper-pdfの起動

```
# sudo service copperd start
```

例 2.6 copper-pdfの動作状態確認

```
# sudo copperd -status
```

例 2.7 copper-pdfの停止

```
# sudo service copperd stop
```

例 2.8 copper-pdfの再起動

```
# sudo service copperd restart
```

例 2.9 copper-pdfの自動起動の有効化

```
# sudo chkconfig copperd on
```

例 2.10 copper-pdfの自動起動の無効化

```
# sudo chkconfig copperd off
```

2.1.5 Debian

Debian向けにはdebパッケージを配布しています。

Debianでは、Java実行環境としてsun-java5-jdkまたはsun-java6-jdkが必要です。kaffeまたは java-gcj-compatでは動作しません。sun-java5-jdkまたはsun-java6-jdkをインストールし、`update-alternatives --config java` コマンドで java コマンドを `/usr/lib/jvm/java-1.x.0-sun/jre/bin/java`に切り替えてください。

他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/j2sdk1.4.2_19
```

copper-pdf_2.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 2.11 debパッケージのインストール

```
# sudo dpkg -i copper-pdf_2.x.x_all.deb
```

例 2.12 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の手法に従います。

例 2.13 copper-pdfの起動

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

例 2.14 copper-pdfの動作状態確認

```
# sudo copperd -status
```

例 2.15 copper-pdfの停止

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

例 2.16 copper-pdfの再起動

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

例 2.17 copper-pdfの自動起動の有効化

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

例 2.18 copper-pdfの自動起動の無効化

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

2.1.6 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-2.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、展開して出来たCopperPDF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を適切な場所に配置してください。

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 2.19 JAVA_HOME の設定

```
export JAVA_HOME=/usr/local/j2sdk1.4.2_19
```

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 (17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。

2.1.7 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-2_x_xというディレクトリが出来ます。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 2.2 ディレクトリ構成

```
copper-pdf-2_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。

ディレクトリ	配置場所
jetty	/var/lib/copper-pdf/jetty
webapp	/var/lib/copper-pdf/webapp
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。

2.1.8 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 (171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http://copper-pdf.com/buy/で購入してください。

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

2.2 Copper PDFのツール

Copper PDFはプログラムとして実行可能な以下の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/jdk1.6.0_05"等)です。設定されていない場合は、環境変数PATH の設定により実行出来る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オプションは使用出来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 2.20 JAVA_OPTS の設定

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

付属の実行ファイルを使えない環境等で、Javaコマンドで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出来ます。

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出来る、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。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>]

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を直接使用します。

-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]*。

-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

コマンドラインで手軽に実行出来るドキュメント変換ツールです。

ツールの実行ごとに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copper 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変換出来ます。-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 (68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から取得出来るデータが使われ、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。

java.awt.headless
: Java 1.4.2でディスプレイのない環境で実行する場合は"true"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.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ことが出来ます。

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出来ます。なお、ブラウザが実行出来る環境では、プログラムの起動後に自動的にブラウザが開きます。

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。

java.awt.headless
Java 1.4.2でディスプレイのない環境で実行する場合は"true"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。

START
jettyディレクトリ内のstart.configファイル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>] [-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] [-user <ユーザー名>] [-userdel <ユーザー名>]

概要

Copper PDFサーバーの制御と、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 (10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]*。

-jkp, --jk-port ポート番号
HTTP/RESTインターフェースをAJP13で起動するポート番号です*[2.1.2]*。

-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]*。

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出来るため、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出来ます。

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 (26ページ)を参照してください。

Copper PDF 2.1.2以降ではAJP13ポート(-jkp)を設定可能です。詳細はmod_jkの設定 (27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、直接編集出来ないため、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出来ます。

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。

java.awt.headless
: Java 1.4.2でディスプレイのない環境で実行する場合は"true"を設定してください。

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 2.3 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 2.21 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。 ドライバの接続先のポートです。
jp.cssj.cssjd.control-command	制御コマンドです。
jp.cssj.cssjd.control-port	制御用ポートです。
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。
jp.cssj.cssjd.maxThreads	最大スレッド数です。 copperdが同時に処理出来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、処理に空きが出来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]*。

プロパティ	説明
jp.cssj.cssjd.jk.port	AJP13ポート番号です。 指定したポートで、AJP13でHTTP/RESTインターフェースを起動します*[2.1.2]*。mod_jkによる接続が可能になります。
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]*。

2.3.2 SSL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。

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 2.22 秘密鍵の生成

```
openssl genrsa -des3 -out ssl.key 1024
```

例 2.23 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 2.24 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 2.25 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 2.26 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通信に使用する場合は、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 mod_jkの設定

Copper PDF 2.1.2からは、mod_jkによる接続が出来るように、AJP13をサポートしました。AJP13により、Apacheにより公開されているドメイン・ポート上にCopper PDFのHTTP/RESTインターフェースへの窓口を設けることができます。また、SSL通信にApacheを利用することが出来ます。

AJP13によるマウントポイントは、任意のパスに設定することが出来ます。(HTTP/RESTインターフェースでは、最後の'/'以降だけでアクションを判別するためです。) 以下の例は、Copper PDFがlocalhostの8095ポートでAJP13を起動している場合に、/cti/パスでHTTP/RESTインターフェースを使用可能にする、mod_jkがインストールされたApacheの設定例です。

例 2.27 mod_jkの設定

```
JkWorkerProperty worker.list=copper
JkWorkerProperty worker.copper.type=ajp13
JkWorkerProperty worker.copper.host=127.0.0.1
JkWorkerProperty worker.copper.port=8095
JkLogFile /var/log/httpd/jk.log
JkLogLevel info
JkMount /cti/* copper
```

2.3.4 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 2.28 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
java.util.logging.FileHandler.level = INFO
java.util.logging.FileHandler.formatter =
java.util.logging.SimpleFormatter
java.util.logging.FileHandler.encoding = UTF-8

# 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
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
prefix	copperd.	ログファイル名の先頭の文字列
suffix	.log	ログファイル名の末尾の文字列
level	ALL	ログレベル
filter	N/A	ログのフィルタ
formatter	java.util.logging.SimpleFormatter	ログのフォーマッタ
encoding	UTF-8	ログファイルのキャラクタエンコーディング

2.3.5 アクセス制御の設定(access.txt)

access.txtはIPアドレスでアクセス出来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 2.29 アクセス制御の設定例

```
allow   127.0.0.1
deny    *
```

2.3.6 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.copper/webapp.proprtiesにコピーされます。*[2.0.1]*

2.3.7 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。この設定の変更を反映するためにCopper PDFサーバーを再起動する必要はありません。

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変更出来ますが、default.propertiesに書いておくことにより、あらかじめ入出力プロパティを設定した状態にすることが出来ます。

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。

2.3.8 fontsディレクトリ

フォント関連の設定です。詳細は次の章で解説します。

2.4 フォントの設定

Copper PDFは外部のフォントを使用するPDF、あるいはフォント情報を埋め込んだPDFを出力することが出来ます。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。

図 2.4 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。ドキュメント中で利用出来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利用出来る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 フォントしか使用出来なかったためで、実際はOpenType CFF/Type2(.otf)フォントも配置することが出来ます。

warrays

実際のフォントファイルを用いずに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。

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。

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。

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とfonts.xml.dbのタイムスタンプを比較し、異なっていればfonts.xmlを自動的に読み込むため、fonts.xmlへの変更はすぐに反映されます。

ただし、システムへのフォントのインストールや、フォントファイルの追加・置き換え等を自動検出しません。この場合、変更を反映させるためには、fonts.xmlファイルを再保存するか、fonts.xml.dbファイルを削除してください。

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するフォントを特定する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出来ます)。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出来る最も近いフォントが選択され, 使用されます。

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, italicとobliqueの場合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、どうしても表示出来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] によって選択可能です。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出来ます。

これらの入出力プロパティまたは拡張CSSプロパティにcid-keyed, cid-identity, cid-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出来ます。なお、コアフォントは常に使用されます。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]*。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。コアフォントの優先度は常に最低となります*[2.0.9]*。

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]*。

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出来ます。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。

以下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
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フォントで使用出来る文字セット(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別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&#0x2021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	ƒ 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で使用出来る文字の一覧です。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で使用出来る文字の一覧です。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ディレクトリにフォントファイルを配置すると フォントが利用可能になるようにfonts.xmlが設定されています。

埋め込みフォント

PDFにフォントを字体のデータを埋め込む方式で、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。

Copper PDFはフォントの埋め込みをサポートしています。フォントを埋め込む場合、文書で使用す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、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インデックス)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、全く表示出来なくなります。

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、使用出来る文字はAdobe社が提供するCMapファイルで定義された文字に限られ、全ての環境でPDFを表示出来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。

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、さらのクラス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出来ます。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。

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 2.30 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 2.31 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

http://www.panose.com/ProductsServices/pan1.aspx
PANOSE-1コードの仕様です。

http://developer.apple.com/textfonts/TTRefMan/RM06/Chap6OS2.html
TrueTypeフォントのOS/2テーブルの仕様です。

http://www.microsoft.com/typography/tools/tools.aspx
Microsoft社のFontToolsのページです。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 2.32 幅情報の抽出

```
java -cp lib/cssj-sakae.jar:lib/sakae-opentype.jar:lib/commons-
logging.jar:lib/cti-server.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出来ます。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、使用出来る文字セットには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出来ます。

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出来ます。

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出来ます。

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出来ます。

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]で参照することが出来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出来ます*[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出来、excludeにより除外する文字範囲を明示することが出来ます。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 2.33 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出来ます。以下の記述は U+1F00-1FFFと書くのと等価です。

例 2.34 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 2.35 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出来ます。

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。
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。
vcmap		縦書きのCMap名です。
warray	✔	フォントの幅情報のファイルです。

フォントの幅情報のファイルはwarraysディレクトリに等幅フォント用の既定のものが用意されています。

また、手持ちのフォントから幅情報を抽出するツールが用意されています。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出来ます。

例 2.36 幅情報の抽出

```
java -cp lib/cssj-sakae.jar:lib/sakae-opentype.jar:lib/commons-
logging.jar:lib/cti-server.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
```

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出来ます。
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出来ます。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出来ます。100から900まで100刻みの値で設定してください。

属性	必須	説明
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出来ますが、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出来ます。
src	✔	フォント名です。
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 **cid-keyed** CID-Keyedフォント

属性	必須	説明
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出来ます。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出来ます。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。

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
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出来ます。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出来ます。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]で指定出来る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出来ます。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IPAフォント (http://ossipedia.ipa.go.jp/ipafont/) 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 2.37 例のタイトル

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

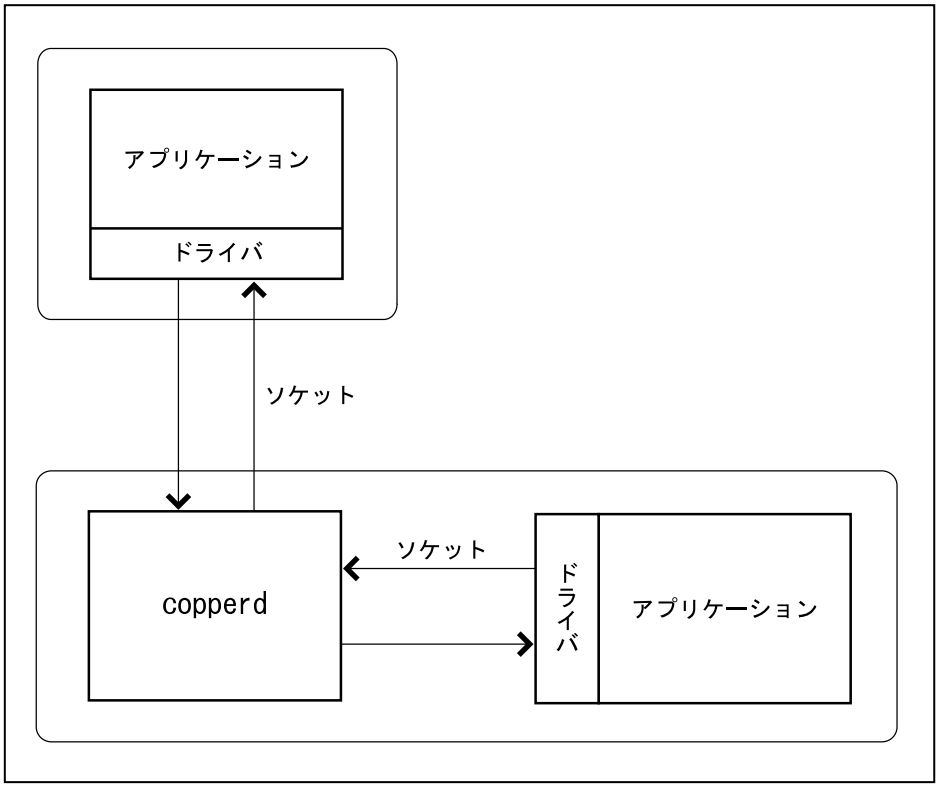
3.開発者ガイド

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出来ます。いずれにしても、ネットワーク越しの接続が可能です。

図 3.5 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３種類があります。

CTIP 1.0 (54ページ)
Copper PDFの前の製品(CSSJ)から使われている、高速で信頼性の高い通信方式です。

CTIP 2.0 (68ページ)*[2.1.0]*
CTIP 1.0の高速・高信頼性をそのままに、機能を強化した通信方式です。

HTTP/REST (99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出来ます。CTIP 2.0と同等の機能を持ちます。

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出来ます。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 1.0, CTIP 2.0で共通)、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出来ます (20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 (28ページ)起動中に変更することが出来ます。

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 (CTIP 1.0ではエラーハンドラ」という名前になっています)。

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出来ます。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出来ます。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出来ます。

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出来ます。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を参照してください。

3.1.9 リソースの送信・アクセス許可

Copper PDFで文書を変換する場合、CSSファイルや画像ファイルといったリソースがどこにあるかが問題となります。リソースがサーバー側からアクセス出来る場所(例:copperdが動作しているマシンのディスク上)にある場合は、copperdが直接リソースを取得出来ます。リソースがクライアント側(ドライバを使用するアプリケーションが動いているマシン)からアクセス出来、サーバー側からアクセス出来ない場所にある場合は、ドライバがリソースをサーバーに送る必要があります。

文書内から画像などがURLによって参照されている場合、copperdはまず、そのURLで表されるリソースが既にドライバにより送られているかどうかを調べます。送られている場合は、そのリソースを使います。そうでない場合は、後で説明するソースリゾルバ*[2.1.0]*によって、クライアントからリソースを送るように要求します。ソースリゾルバによってんもリソースを取得できない場合は、サーバー側でディスクやネットワーク等からリソースを取得しようと試みます。

図 3.6 copperdがリソースにアクセスする場合(1)

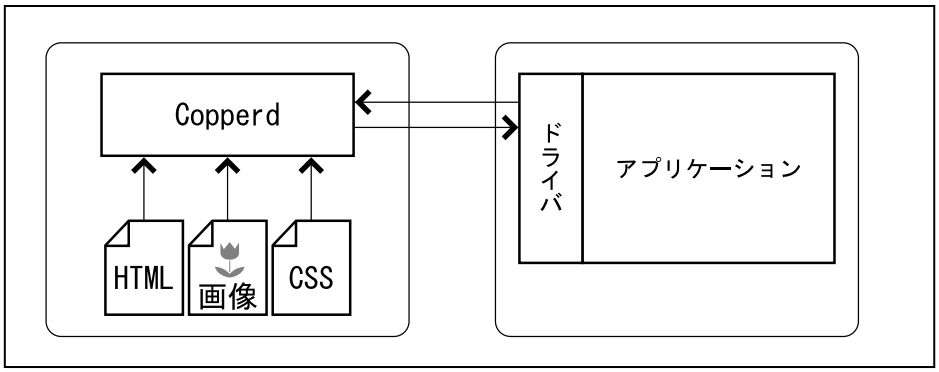


図 3.7 copperdがリソースにアクセスする場合(2)

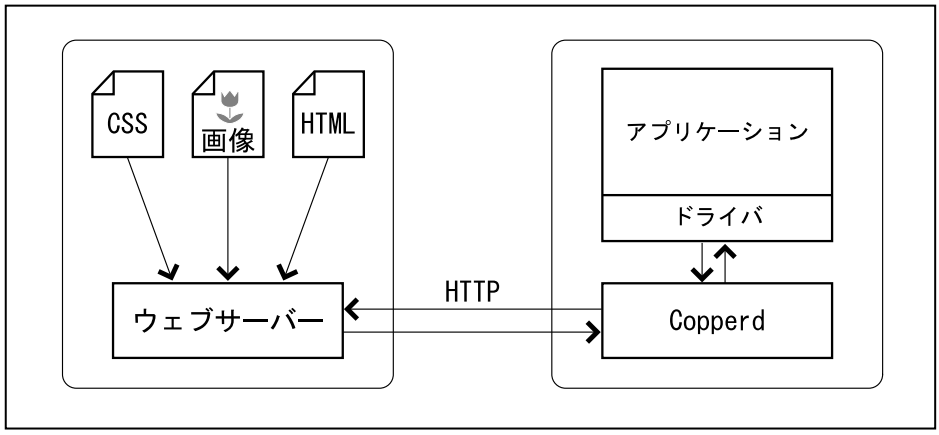
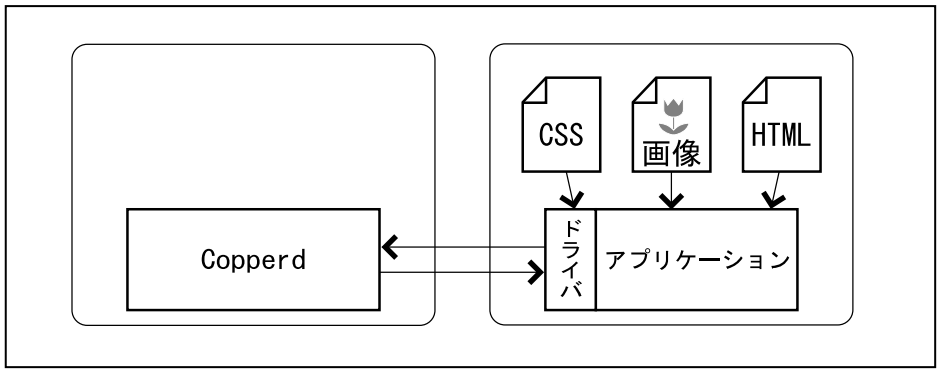


図 3.8 クライアント側からcopperdにリソースを送る場合



リソースをサーバーに送る

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URL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出来ます。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URLは、*file:///var/data/image.gif*や、*http://host/style.css*といった文字列です。copperdはこれらのURLで表されるリソースが必要になった場合、そのURL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。

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。ドライバには必要なリソースを自動的に判断する機能がないため、事前にどのリソースを送るかはアプリケーション側に委ねられます。

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。

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URL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URL)、SSL(https://で始まるURL)によりアクセス可能なデータを取得することが出来ます。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URL)へアクセスすることが出来ます。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がアクセス出来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出来る場合、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URL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、利用出来る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出来ます。"*"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]*

両者の組み合わせ

クライアントからリソースを事前に送る方法と、サーバー側から積極的にリソースを取得する方法は、組み合わせて使うことが出来ます。このとき、URIパターンによるアクセス制御はクライアントから送られたリソースには適用されません。例えば、*file:///var/data/*.gif* へのアクセスが禁止されていたとしても、クライアントから仮想URL *file:///var/data/image.gif* として送られたリソースは有効です。

3.1.10 ソースリゾルバの設定

ソースリゾルバは、指定したURIでアクセスできるデータを取得する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、例えば実際にHTMLファイルの内容を解析しなければ、必要なリソースが分からないということももあります。CTIP 2.0では、サーバー側がドキュメントを解析しつつ、必要になったリソースをその都度クライアントが探すように要求し、ドライバはソースリゾルバで探し出したリソースをサーバーに送ります。

3.1.11 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出来ます。リセットにより、全ての状態は接続直後に戻ります。

3.1.12 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

リソースの場合と同様に、変換対象のドキュメント本体をサーバー側に送ることが出来ます。

このとき、ドキュメントの仮想的なURLを送る必要があります。このURLは、ドキュメント中で使われる相対パスを解決するために使われます。例えば、ドキュメントの仮想的なURL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。

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・エンコーディングを明示することが出来ます。

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URL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出来ます。ドキュメント中の相対パスは、ドキュメントのURLを基点に解決します。ドキュメントのURL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URL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URLには適用されません。

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出来ます。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出来ますが、必ず要求どおりに処理が終わる保障はありません。

3.1.14 通信の終了

通信の終了をサーバーに通知すると、サーバーの方から接続を切断します。CTIP 1.0では接続を切断して接続しなおさない限り、次のドキュメント変換処理は実行できませんが、CTIP 2.0ではそのまま処理を繰り返すことができます。

3.2 CTIP 1.0 インターフェースの特徴

Copper PDF 2.1.0からは、より強力なインターフェース(CTIP 2.0)と、より手軽なHTTP/RESTインターフェースを使用可能です。従来のインターフェースもサポートされていますが、新しいインターフェースの使用を検討してください。

- CTIP 2.0インターフェースの説明 (68ページ)
- HTTP/RESTインターフェースの説明 (99ページ)

3.2.1 結果サイズの取得

HTTPのContent-Lengthヘッダを送るためには、結果全体のデータのバイト数を事前に知る必要がありますが、CTIP 1.0ではプログレスリスナをその目的に使用することが出来ます。プログレスリスナが要求した場合は、変換結果の先頭が得られる前に、結果全体のデータのバイト数が渡されます。

3.2.2 メッセージハンドラ（エラーハンドラ）の設定

エラーハンドラは、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 (処理情報を受け取る機能は後で追加されたため「エラーハンドラ」という名前になっています)。エラーハンドラが受け取ることが出来るメッセージは以下の4つに分類されます。

種類	コード	説明
警告	1	処理の続行が可能なエラーです。入出力設定や、変換対象文書に問題がありますが、処理結果自体は得ることが出来ます。
エラー	2	処理の続行が不可能になるか、出力結果を得られなくなるエラーです。このエラーが発生した場合、正常な処理結果が得られることは期待出来ません(PDF等のデータが壊れている可能性があります)。
致命的エラー	3	通信障害など、システムの問題に起因する深刻なエラーです。このエラーが発生した場合、正常な処理結果が得られることは期待出来ません(PDF等のデータが壊れている可能性があります)。
処理情報	4	これはエラーメッセージではありません。出力済みのページ数、処理中の内容などの情報です。

以上のうち、コード4の処理情報は "カテゴリ:値" という形式で渡されます (Java版のドライバではカテゴリと値を別々に受け取ることも出来ます)。カテゴリと値は、資料集のメッセージハンドラから取得出来る情報 (172ページ)を参照してください。

3.2.3 CTIP 1.0 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。
http://sourceforge.jp/projects/copper/docs/ctip-v1

3.3 Javaドライバ

3.3.1 使用方法

Java用ドライバはCopper PDF本体とは別に配布されています。http://sourceforge.jp/projects/copper/releases/?package_id=8742 からcssj-driver-java 1.x.xをダウンロードしてください。アーカイブを展開した後にできるlibディレクトリ内のcssj-driver-1.x.x.jarがドライバのライブラリです。このファイルをクラスパスに追加(あるいはアプリケーションのライブラリディレクトにコピー)してください。

ドライバの窓口となるクラスはjp.cssj.cti.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 3.1 copperdへの接続

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

...

CTIDriver driver = CTIDriverManager.createDriverFor("localhost",
"8099");
CTISession session = driver.createSession("user", "kappa");

//各種操作
...
```

3.3.2 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。各関数の詳細はドライバのapidocディレクトリ内のJavadocか、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 createDriverFor(String host, int port)
- public CTISession createSession(String user, String password) throws IOException, SecurityException

エラーハンドラ・プログレスリスナの設定

- public void setErrorHandler(ErrorHandler eh)
- public void setProgressListener(ProgressListener l)

出力先の設定

- public void setOutput(OutputStream out, String mimeType) throws IOException

プロパティの設定

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

リソースの送信・アクセス許可

- public void includeResource(String uriPattern) throws IOException
- public void excludeResource(String uriPattern) throws IOException
- public OutputStream sendResource(String uri, String mimeType, String encoding) throws IOException

本体の送信

- public void formatMain(String uri) throws IOException
- public OutputStream sendMain(String uri, String mimeType, String encoding) throws IOException

通信の終了

- public void close() throws IOException

3.3.3 サンプル

以下は、サーバー側から取り出したデータを変換するサンプルです。

例 3.2 jp.cssj.cti.examples.ServerResource

```
package jp.cssj.cti.examples;

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;

import jp.cssj.cti.CTIDriver;
import jp.cssj.cti.CTIDriverManager;
import jp.cssj.cti.CTISession;
import jp.cssj.cti.helpers.StdioErrorHandler;

/**
 * サーバーでデータを取得して変換します。
 */
public class ServerResource {
    /** 接続先のホスト名。 */
    private static final String HOST = "localhost";

    /** 接続先のポート番号。 */
    private static final int PORT = 8099;

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

    public static void main(String[] args) throws Exception {
```

```
        //ドライバを取得
        CTIDriver driver = CTIDriverManager.createDriverFor(HOST,
PORT);
        //接続する(ユーザー名は"user"で固定)
        CTISession session = driver.createSession("user",
PASSWORD);
        try {
            //test.pdfに結果を出力する
            OutputStream out = new BufferedOutputStream(new
FileOutputStream(
                    "test.pdf"));
            try {
                session.setOutput(out, "application/pdf");
                //エラーメッセージを標準出力に表示する
                session.setErrorHandler
(StdioErrorHandler.getInstance());

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

                // http://www.cssj.jp/以下にあるリソースへアクセスする
                session.includeResource("http://www.cssj.jp/**");
                // index.htmlを変換
                session.formatMain
("http://www.cssj.jp/index.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();
        }
    }
}
```

クライアント側のデータを変換するサンプルを含め、これらのファイルはドライバのsrc/examplesに収められています。

3.3.4 サーブレットの作成

webappにウェブアプリケーションのサンプルが収められています。このサンプルのソースコードはwebapp/WEB-INF/srcにあります。

このサンプルは、index.pdfにアクセスすると、index.jspの出力結果をPDFに変換されたものが 表示されるというものです。webappディレクトリをサーブレット・コンテナに配備することで実際に動かすことが出来ます。web.xml内のcontext-paramの部分を接続先のcopperdに合わせて設定してください。

サンプルのjp.cssj.cti.servlet.AbstractCSSJServletを継承することで、PDFを出力するサーブレットを簡単に作ることが出来ます。ドライバを利用してユーザーが全く独自にサーブレットを作ることも可能ですが、変換結果を直接クライアントを送る場合は、以下のようにContent-Lengthヘッダを送ることを 忘れないで下さい。Content-Lengthヘッダを送らないと、Acrobat Readerプラグインで表示されない場合があります。

例 3.3 Content-Lengthヘッダの送出

```
...
import jp.cssj.cti.helpers.ProgressAdapter;
...セッションの作成...
session.setProgressListener(new ProgressAdapter(true) {
    public void contentLength(long contentLength) {
        response.setContentLength((int) contentLength);
    }
});
...変換処理...
```

3.3.5 フィルターを使ったServlet/JSPの変換

ウェブアプリケーションのサンプルに含まれるjp.cssj.cti.servlet.CSSJFilterは、Servlet 2.3の「フィルタ」を利用してServletまたはJSPの出力結果をPDFに変換するものです。

サンプルでは、filterディレクトリ内に置かれたファイルをPDFに変換します。ファイルは静的なファイルのほか、JSPなど動的なファイルでも構いません。

CSSなどのリソースはresourcesファイルに置いています。これらのファイルに直接アクセスされるのを防ぐために、jp.cssj.cti.servlet.AccessFilterを使うことが出来ます。このフィルタは、CSSJFilter以外からのアクセスに対して、404エラーを返します。

3.3.6 ソースコード

ドライバのソースはドライバの配布ファイルのsrcディレクトリに収められています。このソースは、JDK1.4.2以降でコンパイルすることが出来ます。

3.4 Perlドライバ

3.4.1 使用方法

Perl用ドライバはCopper PDF本体とは別に配布されています。http://sourceforge.jp/projects/copper/releases/?package_id=8741 からダウンロードしてください。アプリケーションは、codeディレクトリをライブラリパスに含め、`use CSSJ::Driver;`でモジュールをインポートしてください。

3.4.2 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apidoc内のAPIドキュメントか、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

- create_driver_for HOST PORT [ENCODING]
- CSSJ::Driver->create_session USER PASSWORD

エラーハンドラ・プログレスリスナの設定

- CSSJ::Session->set_error_func FUNCTION
- CSSJ::Session->set_progress_func FUNCTION
- CSSJ::Session->set_content_length_func FUNCTION

出力先の設定

- CSSJ::Session->set_output OUTPUTHANDLE [MIME_TYPE]

プロパティの設定

- CSSJ::Session->set_property NAME VALUE

リソースの送信・アクセス許可

- CSSJ::Session->include_resource URI_PATTERN
- CSSJ::Session->exclude_resource URI_PATTERN
- CSSJ::Session->start_resource FILEHANDLE URI [MIME_TYPE ENCODING]
- CSSJ::Session->end_resource FILEHANDLE

本体の送信

- CSSJ::Session->format_main URI
- CSSJ::Session->start_main FILEHANDLE URI [MIME_TYPE ENCODING]
- CSSJ::Session->end_main FILEHANDLE

通信の終了

- CSSJ::Session->close

3.4.3 サンプル

Perl用インターフェースは、ファイルハンドルに対する出力をキャプチャしてサーバーに送ります。以下の例では$session->start_mainと$session->flush_mainの間で出力されたHTMLが変換されます。

例 3.4 content.pl

```
#!/usr/bin/perl
=head1 NAME

コンテンツ変換サンプル

=head2 概要

start_mainとend_mainの間の出力結果をPDFに変換します。

=cut
use lib '../code';
# ドライバのインポート
use CSSJ::Driver(create_driver_for);

# ドライバの作成
$driver = create_driver_for('localhost', 8099, 'EUC-JP');
# 接続
$session = $driver->create_session('user', 'kappa');

# Content-Lengthヘッダの送信
$session->set_content_length_func (sub {
    my $length = shift;
    print "Content-Length: $length\n\n";
    binmode(STDOUT);
});

# Content-Typeヘッダの送信
print "Content-Type: application/pdf\n";

# リソースの送信
$session->start_resource(STDOUT, 'file:/skin.css');
print << 'EOF';
  p {
    background-color: Gray;
  }
EOF
$session->end_resource(STDOUT);

# 本体の送信
$session->start_main(STDOUT, 'file:/test.html', 'text/html', 'EUC-
JP');
print << 'EOF';
<html>
```

```
  <head>
    <title>テストドキュメント</title>
    <link rel="StyleSheet" type="text/css" href="skin.css">
  </head>
  <body>
    <p>こんにちは</p>
  </body>
</html>
EOF
$session->end_main(STDOUT);

# セッションを閉じる
$session->close();
```

start_resource,start_mainにファイルハンドルを渡すと、それぞれend_resource,end_mainが呼び出されるまで、ファイルハンドルに出力されたデータをcopperdに送ります。その間、ファイルハンドルの本来の機能(STDOUTでは標準出力にデータを送るなど)は使えなくなります。

requireで他のプログラムを呼び出すことで、他のプログラムの出力結果を変換することも出来ます。ただし、CGIとして作成されたPerlプログラムは、HTTPヘッダを出力するため、そのままではヘッダもPDF内に表示されてしまいます。start_resource, start_mainの最後の引数に1を設定すると、最初の空行までの間をヘッダと認識して除去します。

例 3.5 content.pl

```
$session->start_main(STDOUT, 'file:/test.html', 'text/html', 'EUC-
JP', 1);
require 'program.cgi';
$session->end_main(STDOUT);
```

3.5 PHPドライバ

3.5.1 使用方法

PHP用ドライバはCopper PDF本体とは別に配布されています。http://sourceforge.jp/projects/copper/releases/?package_id=8743 からダウンロードしてください。アプリケーションは、codeディレクトリ内のcssj_driver.phpをインクルード(require_once)してください。PHP用ドライバではcssj_driver.php(driverパッケージ)の関数を用いてください。他の関数は低レベルな処理をするもので、通常は必要ありません。

また、ソース中の日本語のコメントのために、ファイルはEUC-JPエンコーディングとなっています。PHPの内部エンコーディングにEUC-JP以外を使う場合は、asciiディレクトリの中にASCIIコードに変換したものがありますので、そちらを利用してください。

3.5.2 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apidocディレクトリ内のAPIドキュメント(phpDocumentor)またはオンラインの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

- mixed &cssj_create_driver_for ($host $host, $port $port, [$encoding $encoding = 'ISO-8859-1'])
- mixed &cssj_create_session ($driver &$driver, $user $user, $password $password)

エラーハンドラ・プログレスリスナの設定

- void cssj_set_error_func ($session &$session, $errorFunc &$errorFunc)
- void cssj_set_progress_func ($session &$session, $progressFunc &$progressFunc)

出力先の設定

- boolean cssj_set_output ($session &$session, $out &$out, [$mimeType $mimeType = 'application/pdf'])

プロパティの設定

- boolean cssj_set_property ($session &$session, $name $name, $value $value)

リソースの送信・アクセス許可

- boolean cssj_include_resource ($session &$session, $uriPattern $uriPattern)
- boolean cssj_exclude_resource ($session &$session, $uriPattern $uriPattern)
- boolean cssj_ob_start_resource ($session &$session, $uri $uri, [$mimeType $mimeType = 'text/css'], [$encoding $encoding = ''])
- boolean cssj_ob_end_flush_resource ()

本体の送信

- boolean cssj_format_main ($session &$session, $uri $uri)
- boolean cssj_ob_start_main ($session &$session, $uri $uri, [$mimeType $mimeType = 'text/html'], [$encoding $encoding = ''])
- boolean cssj_ob_end_flush_main ()

通信の終了

- boolean cssj_close ($session &$session)

3.5.3 サンプル

PHP用インターフェースは、出力バッファを介してドキュメントを変換するのが特徴です。以下の例ではcssj_ob_start_mainとcssj_ob_end_flush_mainの間に記述されたHTMLが変換されます。

例 3.6 ob.php

```
<?php
require_once ('../code/cssj_driver.php');
header("Content-Type: application/pdf");

//ドライバの作成
$driver = cssj_create_driver_for('localhost', 8099);

//セッションの開始
$session = cssj_create_session($driver, 'user', 'kappa') or die('
サーバーに接続出来ません');

//リソースの送信
cssj_ob_start_resource($session, 'file:test.css');
readfile('test.css');
cssj_ob_end_flush_resource();

//出力結果の変換の開始
cssj_ob_start_main($session, 'file:ob.html');
?>

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=EUC-JP">
    <link rel="StyleSheet" type="text/css" href="test.css">
    <title>Hello CSSJ</title>
  </head>
  <body>
    <h2>ただいまの時刻</h2>
    <p><?php echo date("l dS of F Y h:i:s A") ?></p>
  </body>
```

```
</html>

<?php
//出力結果の変換の終了
cssj_ob_end_flush_main();

//セッションの終了
cssj_close($session);

?>
```

単純に変数から変数に変換する場合は以下のような関数を定義しておくと便利です。実際の使用例はvar.phpを参照してください。

例 3.7 変数の変換

```
/**
 * 与えられたデータをPDFに変換して返します。
 *
 * @param $session セッション
 * @param $input 元のデータ
 * @return 変換結果PDF
 */
function &toPDF(&$session, $input) {
  //出力先
  $output = '';//nullの場合標準出力となるので、必ず文字列を代入しておく必要がある
  cssj_set_output($session, $output);

  //変換
  cssj_ob_start_main($session, 'file:test.html');
  echo $input;
  cssj_ob_end_flush_main();

  //セッションの終了
  cssj_close($session);

  return $output;
}
```

3.5.4 Content-Lengthヘッダの送信

変換結果の出力先はデフォルトではクライアントにそのまま返されます。このとき、自動的にContent-Lengthヘッダが送信されます。

それ以外の出力先を設定した場合はContent-Lengthヘッダは出力されません。最終的にクライアントにPDFを送信する場合は、Content-Lengthヘッダを送らないとAdobe Readerプラグインで表示されない場合がありますのでご注意ください。

3.6 copper Antタスク

3.6.1 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についての詳細は書籍などをご参照下さい。

Antタスクは、Copper PDF本体がインストールされたサーバー上で実行する必要があります。別途ライブラリをダウンロードする必要はありません。

3.6.2 copper タスクの使用方法

copper タスクはドキュメントの一括変換を行うためのタスクです。なお、CSSJ 1.x系のcssjというタスク名も使用出来ます。copper 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 3.8 copper タスクの宣言

```
<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/anttask/tasks.properties"/>
```

以降、copperという要素名でタスクを使えるようになります。

copperタスクに指定することが出来る属性は次の通りです。

表 3.1 copper タスクの属性

属性名	説明
configFile	Copper PDFの設定ファイル(profiles/default.properties)です。
configDir	Copper PDFの(license-keyファイルが置かれた)設定ディレクトリです。
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。
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。

copper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出来ます。

表 3.2 property要素の属性

属性名	説明
name	プロパティの名前。

属性名	説明
value	プロパティの値。

以下の例では、docs/manual.htmlからdocs/manual.pdfを出力します。

例 3.9 copper タスクの使用例

```
<copper includes="docs/manual.html" suffix=".pdf"
    configFile="conf/profiles/default.properties">
    <includeresource pattern="**" />
    <property name="output.pdf.bookmarks" value="true" />
    <property name="output.pdf.hyperlinks" value="true" />
    <property name="output.pdf.font.policy" value="cid-keyed" />
</copper>
```

上記のサンプルは実際にドキュメントディレクトリ(docsあるいは/usr/share/doc/copper-pdf)に収められています。一旦manual.pdfを削除して、ドキュメントディレクトリ内でantを実行することで試すことが出来ます。

3.6.3 うまく動かない場合

Can't connect to X11... というエラーが表示される

JDK 1.4.2では以下のようなエラーが発生することがあります。

例 3.10 エラーメッセージの例

```
   [copper] java.lang.InternalError: Can't connect to X11 window
server using ':0.0' as the value of the DISPLAY variable.
   [copper]     at sun.awt.X11GraphicsEnvironment.initDisplay
(Native Method)
   [copper]     at sun.awt.X11GraphicsEnvironment.<clinit>
(X11GraphicsEnvironment.java:134)
   [copper]     at java.lang.Class.forName0(Native Method)
...省略...
   [copper]     at org.apache.tools.ant.launch.Launcher.run
(Launcher.java:246)
   [copper]     at org.apache.tools.ant.launch.Launcher.main
(Launcher.java:67)
   [copper] Transcoded 0 file(s).

BUILD FAILED
/home/miyabe/workspaces/cssj/cssj/build/release/docs/build.xml:11:
java.lang.InternalError: Can't connect to X11 window server using
':0.0' as the value of the DISPLAY variable.
```

これは、Copper PDFが一部でグラフィック環境を必要とする処理を行っているため、ディスプレイが接続されていない環境で発生するものです。以下のように、ANT_OPTS環境変数でjava.awt.headless システムプロパティをtrueにするように設定することで、問題が解消されます。

例 3.11 java.awt.headlessの設定

```
export ANT_OPTS=-Djava.awt.headless=true
```

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出来ます。

例 3.12 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 3.13 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.font.policy" value="cid-keyed" />
</copper>
```

3.7 CTIP 2.0 インターフェースの特徴

3.7.1 接続情報

以下の形式のURIで接続情報を設定します。

`ctip://ホスト名:ポート/`

Java版のCTIP 2.0ドライバはこの形式のURIを解釈する他、さらに次の形式のURIを解釈してHTTP/REST方式で接続します。

`http://ホスト名:ポート/`

前記の形式のURIは、Java版のCTIP 2.0ドライバを使用するCopper PDF 2.1.0以降のコマンドラインインターフェース、Antタスクでも共通して使用できます。

3.7.2 メッセージコード

メッセージハンドラは2バイトのメッセージコードと、メッセージに付随する値と、人間が読める形式の文字列が渡されます。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。

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。ドキュメント変換サーバー自体の問題によるもので、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 (172ページ)を参照してください。

3.7.3 URIパターンによるアクセス制御

アクセス制御のための、特別なメソッドは用意されていないため、プロパティの設定により行います。input.includeと、input.excludeは特別なプロパティで、値としてURIパターンを設定することが出来ます。同じプロパティを何度も設定することができ、先に設定されたパターンから順に検証されます。

3.7.4 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。
http://sourceforge.jp/projects/copper/docs/ctip-v2

3.8 Javaドライバ2

3.8.1 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。

3.8.2 ドライバの準備

Java用ドライバはCopper PDF本体とは別に配布されています。http://sourceforge.jp/projects/copper/releases/?package_id=8742 からcti-java 2.x.xをダウンロードしてください。アーカイブを展開した後にできるlibディレクトリ内のcti-driver-2.x.x.jarがドライバのライブラリです。このファイルをクラスパスに追加(あるいはアプリケーションのライブラリディレクトにコピー)してください。

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 3.14 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 (ページ)。

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。各関数の詳細はドライバのapidocディレクトリ内のJavadocか、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) 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

3.8.4 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 3.15 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 3.16 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
        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);

            // 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");

            // http://www.cssj.jp/以下にあるリソースへのアクセスを許可する
            session.property("input.include",
"http://print.cssj.jp/**");

            // ウェブページを変換
            session.transcode(URI.create("http://print.cssj.jp/"));
        } finally {
            // セッションを閉じる(忘れやすいので注意!)
            session.close();
        }
    }
}
```

他のサンプルはドライバのexamples/srcに収められています。

3.8.5 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 3.17 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 3.18 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による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 3.19 アクセス制限をしてSourceResolverを使う

```
...
RestrictedSourceResolver resolver = new RestrictedSourceResolver(
        CompositeSourceResolver.createGenericCompositeSo
urceResolver());
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 3.20 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。

3.8.6 サーブレット/JSPでの利用

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 3.21 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 3.22 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 3.23 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 3.24 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 3.25 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。

3.8.7 ソースコード

ドライバのソースコードはSourceForge.JPに公開しています。ドライバのソースコードが必要な方は、以下のガイドを参考にSVNから取得してください。
http://sourceforge.jp/projects/copper/cvs/

CTI Javaのソースコードのターゲットパスは以下の通りです。
http://svn.sourceforge.jp/svnroot/copper/drivers/java/trunk/

3.8.8 JRubyを使う場合

現在のところ純粋なRubyによるドライバは用意されていません。しかし、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/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 3.26 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をそのまま利用します。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。

3.9 Perlドライバ

3.9.1 使用方法

Perl用ドライバはCopper PDF本体とは別に配布されています。http://sourceforge.jp/projects/copper/releases/?package_id=8741 からcti-perl-2.x.xダウンロードしてください。アプリケーションは、codeディレクトリをライブラリパスに含め、`use CTI::DriverManager;`でモジュールをインポートしてください。

例 3.27 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
...
```

3.9.2 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apidoc内のAPIドキュメントか、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->set_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

処理の中断・リセット・通信の終了

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

3.9.3 サンプル

以下は、サーバー側からウェブページへアクセスして変換するサンプルです。

例 3.28 server_resource.pl

```
#!/usr/bin/perl
=head1 NAME

サーバー側リソース変換サンプル

=head2 概要

http://copper-pdf.com/を変換します。

=cut
use strict;
use lib '../code';
use CTI::DriverManager;

my $uri = 'ctip://localhost:8099/';
my $session = CTI::DriverManager::get_session($uri,
```

```
    user => 'user', password => 'kappa');

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

$session->property('output.pdf.compression', 'ascii');

$session->property('input.include', 'http://copper-pdf.com/**');
$session->transcode('http://copper-pdf.com/');

$session->close();
```

クライアント側のデータを変換するサンプルを含め、これらのファイルはドライバのsrc/examplesに収められています。

3.9.4 ソースコード

ドライバのソースコードはSourceForge.JPに公開しています。ドライバのソースコードが必要な方は、以下のガイドを参考にSVNから取得してください。
http://sourceforge.jp/projects/copper/cvs/

CTI Perlのソースコードのターゲットパスは以下の通りです。
http://svn.sourceforge.jp/svnroot/copper/drivers/perl/trunk/

3.10 PHPドライバ

3.10.1 使用方法

PHP用ドライバはCopper PDF本体とは別に配布されています。http://sourceforge.jp/projects/copper/releases/?package_id=8743 からcti-php-2.x.xダウンロードしてください。アプリケーションは、codeディレクトリをライブラリパスに含め、`require_once ('CTI/DriverManager.php');`でドライバを読み込んでください。

例 3.29 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
...
```

3.10.2 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apidoc内のAPIドキュメントか、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, $opts)
- Driver->get_session($uri, $opts)

サーバー情報の取得

- Session->set_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()

処理の中断・リセット・通信の終了

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

3.10.3 サンプル

以下は、サーバー側からウェブページへアクセスして変換するサンプルです。

例 3.30 server-resource.php

```
<?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('out/server-resource.pdf');

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

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

//セッションの終了
$session->close();
?>
```

クライアント側のデータを変換するサンプルを含め、これらのファイルはドライバのsrc/testに収められています。

3.10.4 ソースコード

ドライバのソースコードはSourceForge.JPに公開しています。ドライバのソースコードが必要な方は、以下のガイドを参考にSVNから取得してください。
http://sourceforge.jp/projects/copper/cvs/

CTI PHPのソースコードのターゲットパスは以下の通りです。
http://svn.sourceforge.jp/svnroot/copper/drivers/php/trunk/

3.11 .NETドライバ

3.11.1 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。

3.11.2 ドライバの準備

.NET用ドライバはCopper PDF本体とは別に配布されています。http://sourceforge.jp/projects/copper/releases/?package_id=12608 からcti-dotnet_2.x.x.zipをダウンロードしてください。このアーカイブに含まれるCTI.dllをアプリケーションのディレクトリまたはシステムディレクトリ(C:\WINDOWS\system32)に配置してください。

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 3.31 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
...
}
...
```

3.11.3 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。各関数の詳細はドライバのapidoc/htmlディレクトリ内のAPIドキュメントか、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()

3.11.4 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 3.32 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");

                // http://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", "http://copper-
pdf.com/**");

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

例 3.33 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")

            ' http://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", "http://copper-
pdf.com/**")

            ' ウェブページを変換
            session.Transcode("http://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 3.34 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 3.35 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 3.36 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 3.37 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。

3.11.5 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 3.38 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 3.39 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 3.40 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 3.41 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 3.42 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 3.43 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 3.44 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("http://print.cssj.jp/");
session.Join();
...
```

例 3.45 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://print.cssj.jp/")
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。

3.12 transcode Antタスク

3.12.1 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えます。CTIP 2.0ドライバを使用する場合は、後者の方法だけです。

3.12.2 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 3.46 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 3.47 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出来る属性は次の通りです。

表 3.3 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。
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 3.4 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出来ます。

表 3.5 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 3.48 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.font.policy" value="cid-keyed" />
</transcode>
```

3.12.3 その他問題が発生した場合

直接ローカルマシンのライブラリを使用する方法で問題が発生した場合は、旧copperタスクのトラブルシューティングの方法 (66ページ) を試みてください。

3.13 HTTP/RESTインターフェース

3.13.1 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。
http://sourceforge.jp/projects/copper/docs/cti-rest-v1

3.13.2 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)。どの方法を選ぶかはクライアントに任されています

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。
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。

3.13.3 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 3.49 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 3.50 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>
```

3.13.4 Ruby (httpclient)

※実行環境としてJRubyを使うと、より高速なJavaインターフェースを利用することができます。

Rubyではhttpclientモジュール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 3.51 Rubyでドキュメントを変換

```
require 'httpclient'

data = <<DATA
<html>
<body>
Rubyから Copper PDFを使う。
</body>
</html>
DATA
```

```
client = HTTPClient.new
postdata = {
  "rest.user" => "user",
  "rest.password" => "kappa",
  "rest.main" => data,
}
puts client.post_content("http://localhost:8097/transcode",
postdata)
```

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 3.52 Rubyでウェブサイトを変換

```
require 'httpclient'

client = HTTPClient.new
postdata = {
  "rest.user" => "user",
  "rest.password" => "kappa",
  "input.include" => "http://*.cssj.jp/**",
  "rest.mainURI" => "http://print.cssj.jp/",
}
puts client.post_content("http://localhost:8097/transcode",
postdata)
```

3.13.5 Python (urllib)

Pythonではurllib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 3.53 Pythonでドキュメントを変換

```
# -*- coding: utf_8 -*-
import urllib

url = 'http://localhost:8097/transcode'
data = """
<html>
<body>
Pythonからの Copper PDFを使う。
</body>
</html>
"""
params = urllib.urlencode({'rest.user':'user',
                           'rest.password':'kappa',
                           'rest.main':data,
                           })
f = urllib.urlopen(url, params)
print f.read()
```

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 3.54 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':'http://*.cssj.jp/**',
                           'rest.mainURI':'http://print.cssj.jp/',
                           })
f = urllib.urlopen(url, params)
print f.read()
```

3.13.6 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 3.55 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 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。

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 3.56 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=http://*.cssj.jp/**" + // 画像等へのアクセス許可
                "&rest.mainURI=http://print.cssj.jp/"; // 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();
        }
    }
}
```

3.13.7 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 http://print.cssj.jp/ 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 3.57 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=http://*.cssj.jp/**" + // 画像等へのアクセス許可
        "&rest.mainURI=http://print.cssj.jp/"; // 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 3.58 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=http://*.cssj.jp/**"
    ' 変換対象アドレス
    uri &= "&rest.mainURI=http://print.cssj.jp/"

    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()
%>
```

3.13.8 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 3.59 4Dでドキュメントを変換

```
C_TEXT($host;$user;$password;$boundary;$sourceType;$so
urce;$beforeContent;$afterContent;$resultFile;$text)
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 3.60 4Dでウェブサイトを変換

```


C_TEXT($host;$user;$password;$include;$uri;$query;$res
ultFile;$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:="http://*.cssj.jp/**"
$uri:="http://print.cssj.jp/"

  `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="+$pa
ssword+"&input.include="+$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)
```

3.13.9 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(http://copper-pdf.com/)で提供しています。

3.14 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。

3.14.1 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出来ません。*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)に対応することが出来ます。

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出来るようになります。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出来ます。
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 3.61 認証の設定

```
session.setProperty("input.http.authentication.preemptive",
"true");
session.setProperty("input.http.0.authentication.host",
"www.foo.com");
session.setProperty("input.http.0.authentication.user", "foouser");
session.setProperty("input.http.0.authentication.password",
"foopass");
session.setProperty("input.http.1.authentication.host",
"www.bar.com");
session.setProperty("input.http.1.authentication.user", "baruser");
session.setProperty("input.http.1.authentication.password",
"barpass");
```

3.14.2 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出来ます。

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出来ます。

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 3.62 プロクシサーバーの設定

```
session.setProperty("input.http.proxy.host", "proxy.foo.com");
session.setProperty("input.http.proxy.authentication.user", "mei");
session.setProperty("input.http.proxy.authentication.password",
"pass");
```

3.14.3 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出来ます。*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出来ます。

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 3.63 HTTPヘッダの設定

```
session.setProperty("input.http.header.0.name", "Accept-Language");
session.setProperty("input.http.header.0.value", "ko");
session.setProperty("input.http.header.1.name", "User-Agent");
session.setProperty("input.http.header.1.value", "Mozilla/4.0
(compatible; MSIE 7.0; Windows NT 5.1)");
```

3.14.4 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出来ます。

3.14.5 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出来ます。

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 3.64 クッキーの設定

```
String sessionId = request.getSession().getId();
session.setProperty("input.http.cookie.0.domain", "www.foo.com");
session.setProperty("input.http.cookie.0.name", "JSESSIONID");
session.setProperty("input.http.cookie.0.value", sessionId);
session.setProperty("input.http.cookie.0.path", "/");
```

3.14.6 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出来ます。*[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 3.65 タイムアウトの設定

```
session.setProperty("input.http.connection.timeout", "30000");
session.setProperty("input.http.socket.timeout", "10000");
```

3.15 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。

3.15.1 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。

3.15.2 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の作成も同様に出来るのが特徴です。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に変換することが出来ます。

4.1.2 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 (162ページ)を参照してください。

入出力プロパティはシステム管理者やプログラマにより設定されますが、管理者やプログラマにより許可されている場合は (`input.property-pi`[io]がtrueに設定されている)、HTMLやXML文書中でjp.cssj.property処理命令により設定することも出来ます。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"?>
<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出来ないプロパティがあります。詳細は資料集を参照してください。

4.1.3 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。

> CSSJ 1.xのcssj:toc要素, processing.make-toc-before[io]は廃止されました。

4.1.4 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。

4.2 出力結果の形式

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出来ます。デフォルトではPDF("application/pdf")です。

4.2.1 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 (149ページ)を参照してください。

4.2.2 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出来ます。*[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出来ます。Java実行環境に拡張ライブラリを導入することにより、出力出来る画像を増やすことが出来ます。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

現在のところ、画像として出力出来るのは1ページのみです。改ページを行わないモード (137ページ)を利用することを推奨します。

また、コアフォント (32ページ) とCID-Keyedフォント (36ページ)を描画出来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 (35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 (31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出来ます*[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。

4.3 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出来ます。

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。

4.3.1 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。

4.3.2 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出来ます。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。

4.3.3 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 4.2 style属性によるスタイルの指定

```
<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 4.3 style属性によるスタイルの指定(表示)

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 4.4 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出来ます。

例 4.5 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出来ます。

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 4.6 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>
```

4.3.4 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出来ます。

例 4.7 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出来ます。title属性は代替スタイルシートの名前です。

例 4.8 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出来ます。

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 4.9 @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出来ます。

例 4.10 @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 4.11 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 4.12 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出来ます。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。

4.3.5 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。

4.3.6 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

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 (118ページ)を参照してください。

4.4 CSSの拡張機能

4.4.1 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出来ます。

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 4.13 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 4.14 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

4.4.2 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 4.15 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] を使うことが出来ます。例えば、pageという名前を1に設定しなおす場合は {counter-reset: page 1;} と宣言します。

4.5 HTML/XMLの処理

4.5.1 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。

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が認識出来るキャラクタ・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、次の順に判別します。

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. 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、次の順に判別します。

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. `<meta http-equiv="Content-Type" content="text/html; charset=エンコーディング名">`
3. `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。

4.5.2 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出来ます。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。

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出来ます *[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出来ます。*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出来ます。この設定は、前記のmeta要素で上書きされます。

4.5.3 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出来ます。

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出来ます。

例 4.16 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出来ます。これは、ノンブルに見出しを表示する場合に便利です。

4.5.4 目次の生成

見出しは、目次の生成に利用することが出来ます。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 4.17 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 4.18 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 4.19 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;
}
```

4.5.5 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、変更することが出来ます。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]内で使用出来る-cssj-page-ref関数を用意しています。

例 4.20 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関数を使うことが出来ます。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出来ます。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。

4.5.6 Internet Explorerとの互換モード

完全ではありませんが、Copper PDFはWindows Internet Explorerに近いレイアウトを再現します。このモードを有効にするには、output.compatible_mode [io] にtrueを設定してください。ただし、次のいずれかのpublic idを持つDOCTYPE宣言がある場合は、通常のモードでレイアウトします。

- -//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

4.5.7 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出来ます。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。

4.5.8 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出来ます。

4.5.9 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出来ます。

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 画像

4.6.1 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要素によってドキュメント中に含めることが出来ます。また、object, embed要素の使用もサポートされています。

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。

4.6.2 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サンマイクロシステムズ社により配布されている 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出来ます。

JAI-ImageI/Oは画像形式での出力 (118ページ)にも利用することが出来ます*[2.0.3]*。

4.6.3 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に含めることが出来ます。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に含むことも出来ますが、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は全く利用出来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。

4.6.4 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 4.21 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出来ます。img要素を利用することが出来、background-image[css]プロパティにより背景に設定することも出来ます。

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出来ます。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 4.22 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>
```

4.6.5 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)ことが出来ます。

4.7 ページ処理機能

4.7.1 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。

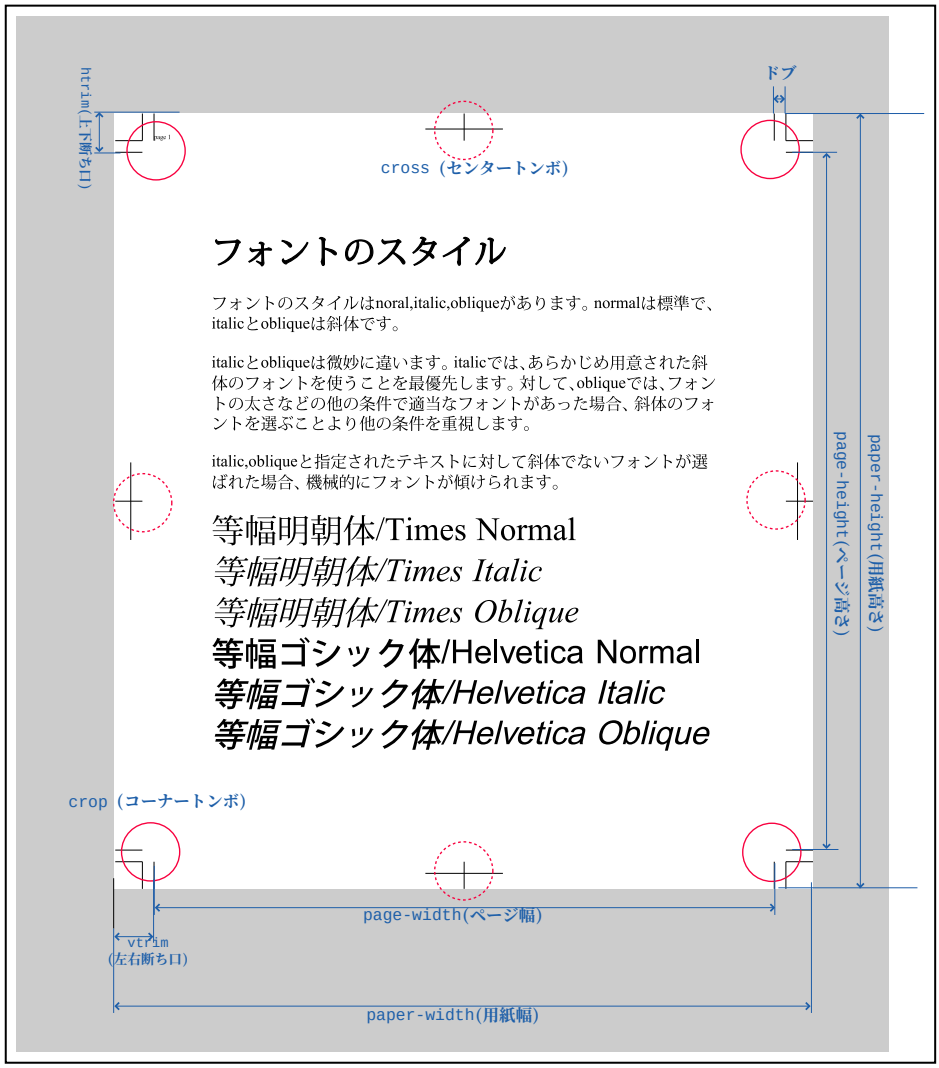
マージン
通常は内容がみだすことのない、ページの余白部分です。

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 4.9 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出来ます。

4.7.2 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]*。

4.7.3 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

4.7.4 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出来ます。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。

4.7.5 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出来ます。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]*

以下は再生成ボックスを利用して、ページごとに「ノンブル」を振る例です。各ページの下部中央にページ番号が表示されます。

例 4.23 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);
  }
</style>
<div id="page-number"></div>
... 本文 ...
```

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出来ます。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出来ます。

例 4.24 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:  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。

4.8 改ページ制御

4.8.1 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。

4.8.2 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出来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出来ません。

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 4.25 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出来ます。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 4.26 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。

4.8.3 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 4.27 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]を満たすことが出来ないため、段落をまるごと次ページに移動してしまいます。

例 4.28 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 4.29 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 4.30 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出来ない場合も、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 4.31 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 4.32 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。

4.8.4 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出来ます。

- 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。

4.8.5 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。

4.9 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。

4.9.1 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。

4.9.2 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出来ない場合は、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。

4.9.3 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。

4.9.4 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 4.33 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。

4.10 PDFの機能

4.10.1 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] で設定することが出来ます。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出来ます。

PDF 1.4以前では、使用出来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出来ます (155ページ)。

4.10.2 暗号化

Copper PDFは暗号化したPDFを出力することが出来ます。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を出力することが出来ます。

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] で設定出来ます。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] で設定することが出来ます。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出来る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出来ます。

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出来ます。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出来ます。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。

4.10.3 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出来ます。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出来ます。

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出来ますが、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。

4.10.4 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を生成することが出来ます。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&ペースト出来るようなファイルが出来上がります。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出来ます。

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] で指定することが出来ます*[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出来ます

[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。

4.10.5 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。

4.10.6 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出来ます*[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。

4.10.7 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出来ます*[2.1.8]*。すかしを利用出来るのは、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出来ますが、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。

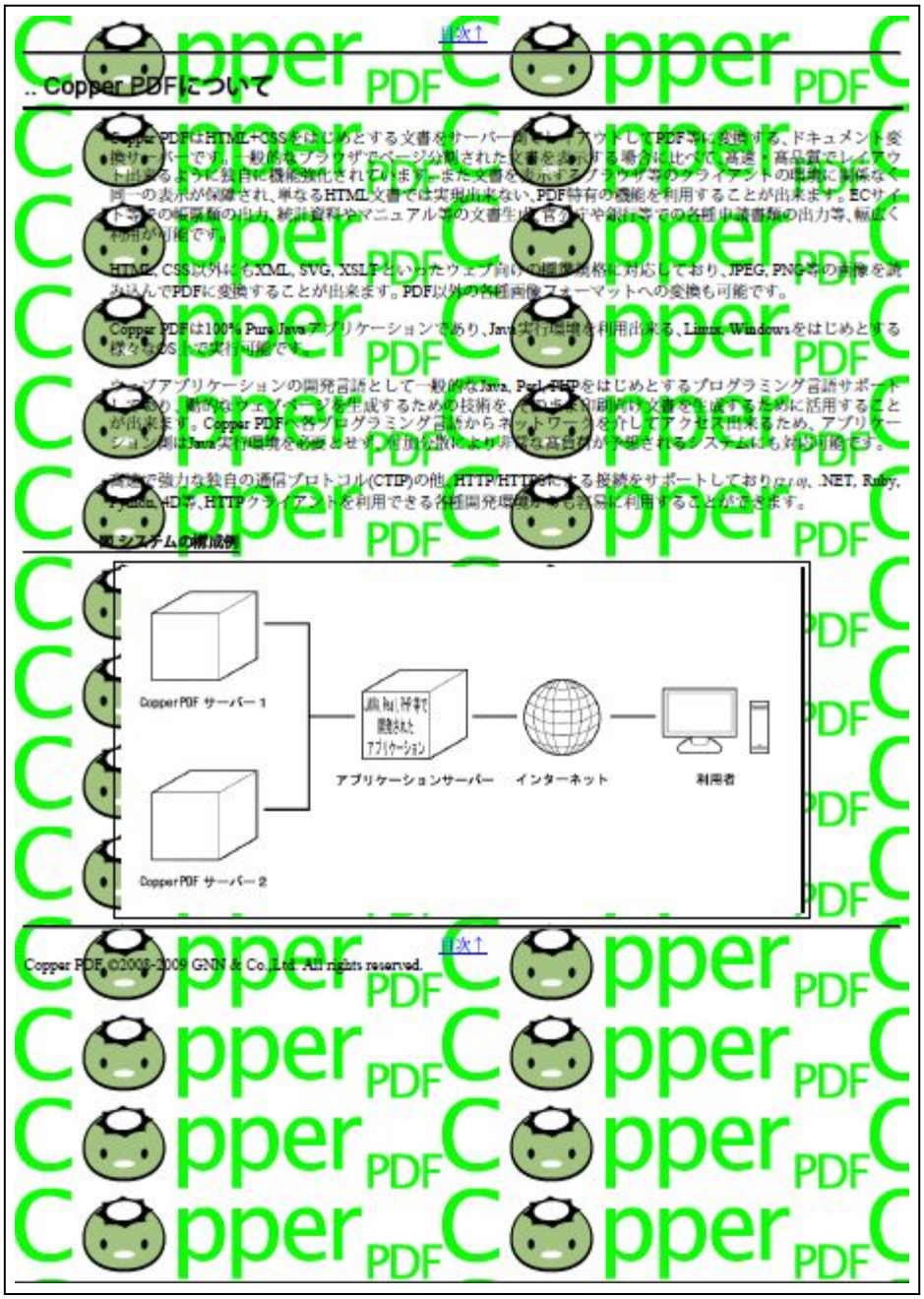
`output.pdf.watermark.uri`[io]に、すかしに使う画像を、絶対アドレスで指定すると、繰り返しパターンとして、画像がPDFの全ページに表示されるようになります。

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出来ます。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。

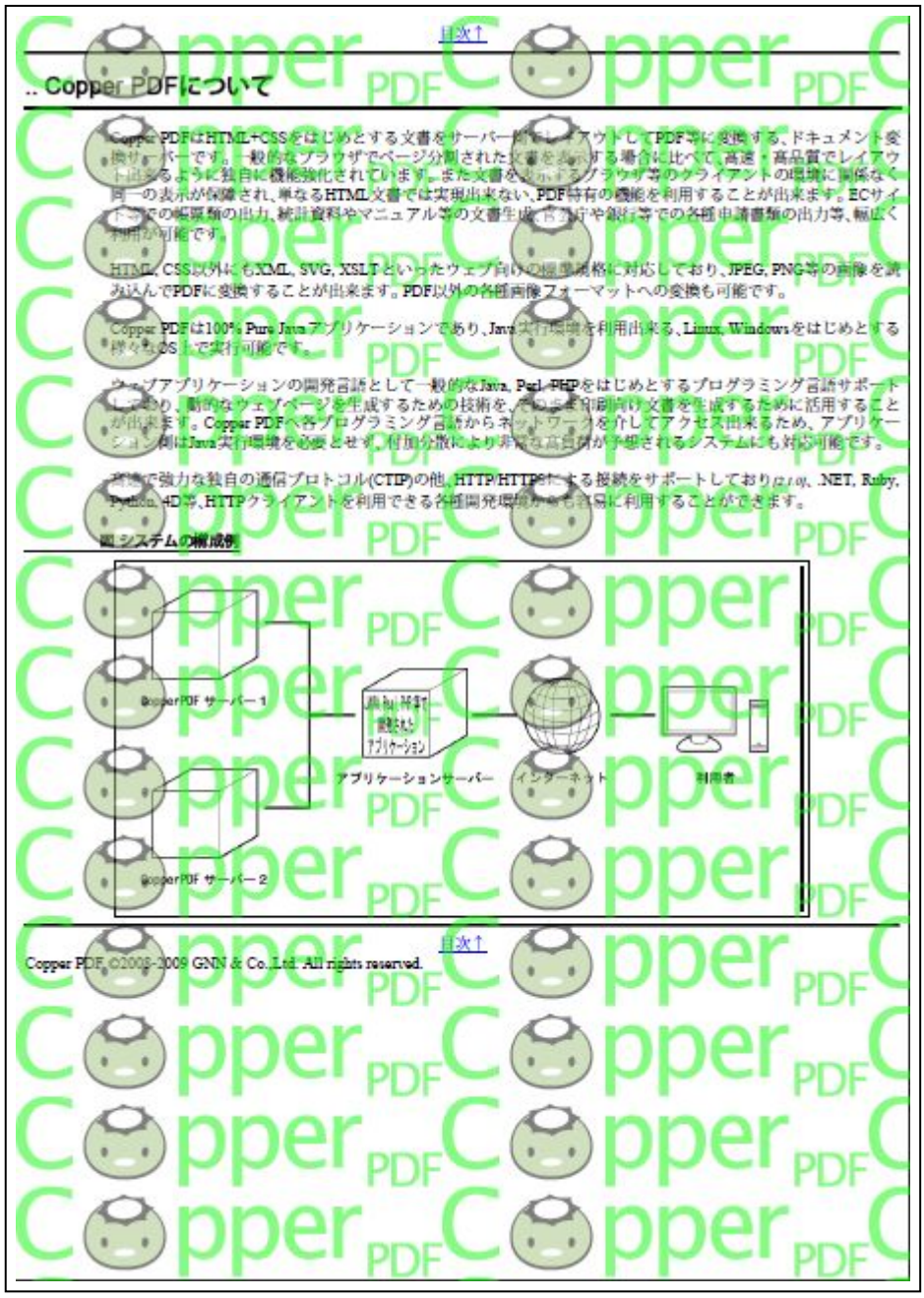
図 4.10 背面にすかしを配置



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 4.11 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。

4.10.8 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/jp/support/products/pdfs/acrojs_j.pdf)を参照してください。

4.11 一般的なブラウザとの互換性

4.11.1 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。このモードは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。

4.11.2 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 4.34 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'='が使用出来る

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'='も使用することが出来ます。

例 4.35 ':'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、#を省略出来ます。

例 4.36 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。

4.11.3 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]

4.11.4 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。

5.資料集

5.1 入出力プロパティ

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出来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	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出来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。
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を解釈出来るようにするためのフィルタです。
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出来るようになります。
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出来ます。
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.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出来なくなります。
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。

名前	デフォルト	バージョン	説明
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。
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。

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。

名前	デフォルト	バージョン	説明
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。
output.color	rgb	1.2.1	出力結果のカラー・タイプです。rgb,grayで指定します。rgbではフルカラーで出力されます。grayでは、グレイスケールに変換されます。
output.compatible_mode	copper	2.0.0	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出来ます。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.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 (126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。この設定は文書中の@pageルール内で上書き出来ます。
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。
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ルール内で上書き出来ます。
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。この設定は文書中の@pageルール内で上書き出来ます。
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	印刷モードです。single-side, double-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。
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。
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出来ます。 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)を利用することが出来ます。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出来ます。 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出来ません。

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。

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 (PDF1.4)	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、省略出来ません。 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出来ますが、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 (PDF1.2)	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 (PDF1.2)	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 (PDF1.3)	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 (PDF1.2)	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 (PDF1.2)	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 (PDF1.2)	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 (PDF1.2)	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 (PDF1.2)	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 (PDF1.2)	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 (PDF1.3)	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 (PDF1.3)	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 (PDF1.3)	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 (PDF1.3)	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"
output.pdf.fonts.policy	cid-keyed	1.1.0 (PDF1.2)	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 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 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。 CopperPDF 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 (PDF1.2)	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 (PDF1.2)	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。

名前	デフォルト	バージョン	説明
output.pdf.hyperlinks.base	ドキュメントのURI	2.0.0 (PDF1.2)	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 (PDF1.2)	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出来るようになります。 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。
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[io]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.jpeg-image	raw	1.1.0 (PDF1.2)	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しか使用出来ず、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出来ますが、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。
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。

名前	デフォルト	バージョン	説明
output.pdf.open-action.java-script	-	3.0.2/2.1.11 (PDF1.2)	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 (PDF1.2)	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)。
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出来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を指定することが出来ます。 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。

名前	デフォルト	バージョン	説明
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 PDF1.4	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。
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 PDF1.6	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 PDF1.7	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 PDF1.7	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 PDF1.7	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 PDF1.7	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。
output.pdf.watermark.uri	-	2.1.8 PDF1.4	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。

名前	デフォルト	バージョン	説明
output.pdf.watermark.mode	back	2.1.8 PDF1.4	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 PDF1.4	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 PDF1.4(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 PDF1.4(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.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出来なくなります。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 文書中で設定出来ないプロパティ

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出来ません。

- 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出来ます。

- input.default-encoding
- output.pdf.bookmarks
- output.pdf.hyperlinks
- processing.page-references
- processing.pass-count

また、利用出来るフォントがCID-Keyedフォントのみに固定されているため、`-cssj-font-policy`[css]プロパティを利用出来ません。

5.2 エラーハンドラから取得出来る情報

5.2.1 Copper PDF 2.0以前(CTIP 1.0)

エラーハンドラに渡される、コード4の処理情報 (48ページ)から得ることが出来る情報の一覧です。

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	表示出来ない画像の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 (68ページ)により、さらに詳細な情報を得ることが出来ます。

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	ページの高さ(double)*[2.1.2]*	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。
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。

コード	値	説明
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出来ない。
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。

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。
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。

コード	値	説明
380C		ライセンスファイルを読み込むことが出来ない。

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。

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。

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	
border-bottom-width	する	
border-left-width	する	
border-width	する	
border	する	
bottom	する	
caption-side	する	

特性	サポート	備考
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出来ません。
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	
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	

特性	サポート	備考
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	
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	

特性	サポート	備考
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		しない	
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。
comment		しない	
dd		する	
del		する	
dfn		する	

要素・機能	属性	サポート	備考
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	画像の表示に使用出来ます。
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	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。
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	

要素・機能	属性	サポート	備考
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出来ません。
map		しない	
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出来ます (126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。
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)	しない	画像の表示に使用出来ます。
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	

要素・機能	属性	サポート	備考
q		する	
ruby rb rt rp		しない	
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。
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		しない	
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	使用するフォントの種類を指定します。指定出来る値はcid-keyed, cid-identity, embedded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。 CopperPDF 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。 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。
-cssj-page-content	2.0.0		none	すべての要素	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出来ます。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	`-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出来ます。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。

名前	バージョン	継承	デフォルト値	適用対象	説明
-cssj-background-size background-size	2.0.8		auto	すべての要素	CSS level 3 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。
-cssj-text-align-last text-align-last	2.0.8		start	すべての要素	CSS level 3 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。

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。
-cssj-cmyk	1.0.0	3	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の順です。
-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。

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	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。

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出来ます(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
svg	http://www.w3.org/2000/svg

XML要素

表 5.18 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 5.19 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 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: http://copper-pdf.com/