

- 3 開発者ガイド -



3.開発者ガイド

3.1 プログラムインターフェース

3.1.1 ソケット通信を用いたインターフェース

CSSJ自身はJavaで書かれているため、JavaプログラマはライブラリとしてリンクすることでCSSJを利用したアプリケーションを作ることができます。また、CSSJには直接ライブラリとして使用する以外に、ソケット通信を用いたインターフェースが用意されています。

ソケット通信を用いたインターフェースは、Java以外の言語からCSSJを利用するための唯一の実用的なインターフェースです。また、このインターフェースを用いることで、CSSJをインストールした複数のマシンをネットワークでつないで負荷分散することができるため、Javaプログラマにとっても有用なものです。

各言語のためにCSSJドライバが用意されています。CSSJドライバは、サーバーにデーモンとして常駐しているcssjdプロセスにアクセスするためのライブラリです。一般的なデータベース・ドライバなどと同様に、通信にTCPソケットを用いるので、言語に依存することはありません。

各言語で用意されているドライバのAPIドキュメントおよびソースコードは公開されています。CSSJサーバーと通信するための用途に限り、ドライバのソースを改変すること、他の言語に移植することを許可します。また、改変されたドライバや新しく開発されたドライバを配布する際はソースの出所および、誰がどのように変更を加えたかを明示してください。この通信プロトコルは特許申請中です。詳細についてはライセンス許諾書を参照してください。

3.2 APIによるアクセスの概要

この章では、CSSJサーバーにアクセスするための、各言語で共通の手続きを説明します。ライブラリのリンク方法など、具体的な手続きを知りたい方は、まず各言語別の説明をご覧ください。後でこの章を読み直しても構いません。

CSSJサーバーによる文書の出力は、以下の流れが基本となります。

1. サーバーへの接続・認証
2. エラーハンドラ・プログレスリスナの設定
3. 出力先の設定
4. プロパティの設定
5. リソースの送信・アクセス許可
6. 本体の送信
7. 通信の終了

上記のうち、2,3,4は順序が入れ替わっても構いませんが、それ以外は正しい順序で実行することが重要となります。以下では、各手続きについて順を追って説明します。

3.2.1 サーバーへの接続・認証

CSSJサーバーにアクセスするためには、サーバーのホスト名、ポート番号を知る必要があります。これらはCSSJサーバーの設定によって決まり、詳細は[管理者ガイド](#)に記述されています。典型的な例として、同一のマシン上にある、デフォルトのセットアップを行ったCSSJサーバーの場合、ホスト名はlocalhost(あるいは127.0.0.1)、ポート番号は8099になります。

また、CSSJサーバーには、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。ユーザーIDは現時点では"user"で固定です。パスワードはCSSJサーバーの管理者が設定します。加えて、クライアントのIPアドレスによるアクセス制御機能があります。サーバーへの接続を拒否される場合は、クライアントマシンのアクセスが許可されていない可能性があります。セキュリティ機能の詳細は[管理者ガイド](#)をご参照下さい。このマニュアルに記述されたサンプルでは、パスワードが"secret"に設定されているものとしていますが、実際は、管理者が設定したパスワードになるため、適宜読み替えてください。

3.2.2 エラーハンドラ・プログレスリスナの設定

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を受け取るための機構で、プログレスリスナは変換処理の進行状況や最終的なデータサイズをプログラムが知るための機構です。どちらも

必須ではありませんが、開発過程においてデバッグのため、あるいはHTTPのContent-Lengthヘッダの送信のために重要なことがあります。

エラーハンドラが受け取ることができるエラーメッセージは以下の3つに分類されます。

警告

CSSの文法ミスなど、些細な誤りです。ユーザーが意図的にそうしている可能性もあり、一般には処理や結果に大きく影響しない問題です。

エラー

画像などの関連するリソースが読み込めないなどの、出力結果に大きく影響する問題です。

致命的エラー

通信障害や、本体のXMLの文法ミスなど、処理の続行が不可能になる問題です。このエラーが発生した場合は、正常な出力を得ることができません。

プログレスハンドラには、サーバー側で処理済の入力データのバイト数が渡されます。また、プログレスハンドラが要求した場合は、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。前者はプログレスバーなどでユーザーに進行状況を伝えるために利用できます。後者はHTTPのContent-Lengthヘッダを送るために利用できます。

3.2.3 出力先の設定

変換結果はストリーム、ファイル、あるいは変数などに出力されることが考えられますが、どのような出力先が指定できるかは、プログラミング言語によります。CSSJ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。

また、変換結果となるデータ形式はデフォルトではPDFとなりますが、他の形式を指定することもできます。

3.2.4 プロパティの設定

CSSJサーバーによる変換方法の詳細はプロパティによって細かく指定することができます。利用可能なプロパティの完全なリストは[資料集](#)に記述されています。例として、変換元のHTMLのキャラクタ・エンコーディング、出力されるPDFの用紙サイズといった設定はプロパティを使って行います。

3.2.5 リソースの送信・アクセス許可

CSSJで文書を変換する場合、CSSや画像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CSSJサーバーからアクセスできる場所(例:CSSJサーバーのマシンのディスク上)にある場合は、CSSJサーバーから積極的にリソースを取得できます。リソースがCSSJサーバーからアクセスできる場所ではなく、代わりにクライアント側(ここで言うクライアントとは、ブラウザなどの端末ではなく、CSSJドライバが存在するマシンのこと)からアクセスできる場所にある場合は、まず、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、CSSJサーバーはまず、そのURLで表されるリソースが既にクライアントから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、ディスク上、あるいはHTTPによってネットワーク越しにリソースを取得しようと試みます。

リソースをクライアントからサーバーに送る

CSSJドライバには、CSSJサーバーに必要なリソースを送る機能が含まれています。このとき送ることができる情報は、リソース本体のデータに加え、リソースの仮想的なURL、MIME型およびキャラクタ・エンコーディングです。最後の2つは必須ではなく、MIME型やキャラクタ・エンコーディングが不明な場合は、CSSJサーバーはそれらを(拡張子、ファイルのヘッダなどを手がかりに)自動検出します。キャラクタ・エンコーディングはXML宣言やHTMLのmeta要素、CSSの@charset指示子により検出します。このような手がかりが全くない場合は、文字化けが発生する可能性があります。そのような状況では、必ずキャラクタ・エンコーディングを明示することを推奨します。当然、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。CSSJサーバーはこれらのURLで表されるリソースが必要になった場合、サーバー側のディスク上、あるいはネットワーク越しにその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。CSSJドライバにはアプリケーション側で必要なリソースを自動的に判断する機能がないため、どのようなリソースが必要になるかの判断はアプリケーションの開発者に委ねられます。

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積極的にアクセスする場合

CSSJサーバーが必要とするリソースがクライアントから送られなかった場合、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。

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、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、ソケット通信によってCSSJサーバーにリソースを送る手間が省けるため、パフォーマンス上有利です。

欠点として、セキュリティの問題が挙げられます。変換対象となる文書を誰でも編集できる場合、そこにサーバー上のファイル名を指定することで、サーバー上のファイルが盗まれてしまう可能性があります。また、`http://`で始まる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になります。

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、アクセスでき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。**デフォルトの状態では、サーバーからいかなるリソースにもアクセスすることはできません。**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。

URLパターン

リソースへのアクセス許可・制限は、URLパターンによって行います。本文の変換を始める前に、クライアントは、利用できるURLと除外するURLのパターンを指定しておきます。具体的な手段については、各言語別の説明をご参照ください。ここでは、各言語共通のURLパターンの形式について説明します。

URLパターンにはワイルドカードを使うことができます。例えば、`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。"*"というワイルドカードは、0個以上の"/" (スラッシュ) も含めることを表します。例えば `http://www.company.com/image/**` は `http://www.company.com/image/` 以下の全てのリソースを表し、`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 `http://www.company.com/secret` へのアクセス禁止を指定し、次に `http://www.company.com/**` へのアクセス許可を指定した場合、`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 `http://www.company.com/**` へのアクセスを許可すると、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。

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、組み合わせることができます。前にも説明した通り、CSSJサーバーはまず、クライアントから送られたリソースを利用しようと試みるため、前者の方法が優先されます。従って、URL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無関係です。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` と共に送られたリソースへのアクセスすることは可能です。

3.2.6 本体の送信

最後に変換対象となる文書の本体の送信を行います。リソースの場合と同様に、文書の仮想的なURLを送る必要があります。このURLは、文書中で使われる相対パスを絶対パスに変換するために使われます。また、同様に必須ではありませんが、文書のMIME型とキャラクタ・エンコーディングを明示することができます。CSSJサーバー側での文書の変換処理は、本体の送信と並行して行われます。

あるいは、リソースと同様にサーバー側で積極的に変換対象の文書を取得することもできます。このとき、CSSJドライバからは、対象となる文書のURLだけを送ります。

3.2.7 通信の終了

各言語のCSSJドライバに含まれている、通信を完了する操作を最後に必ず実行する必要があります。このとき、CSSJサーバーは全ての変換結果をクライアントに送り返し、ドライバとの接続を終了します。

3.3 各言語のAPI

CSSJドライバはCSSJのリリースのdriversディレクトリに格納されています。ライブラリ、ソースおよびAPIドキュメントが各言語別のディレクトリに含まれます。

ドライバは各言語でなるべく近いインターフェースを持ち、シンプルに設計されています。各言語の事情に合わせた応用例は、サンプルコードを参照してください。サンプルコードはCSSJを利用したアプリケーションを開発するために、自由に改変して利用していただいて構いません。

3.3.1 Java

ソケット通信による接続

drivers/java/libディレクトリ内のcssj-driver.jarがドライバのライブラリです。ソケット通信で接続する場合、アプリケーションはこのファイルをCLASSPATH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リにコピー)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CSSJサーバーに、ユーザーID"user"、パスワード"secret"で接続するには、以下のようになります。

```
//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", "secret");

//各種操作
...
```

例 3.1 ソケット通信インターフェース

ライブラリとしてのリンク

CSSJをライブラリとして直接利用する場合、cssj-driver.jarは必要ありません。lib/server,lib/share,lib/plugins内の全てのjarをCLASSPATH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リにコピー)してください。このとき、他のアプリケーションや、サブレット・コンテナとのライブラリの競合に注意してください。

また、ライブラリとしてリンクする際も、ライセンス情報やフォント情報の入ったresourcesディレクトリが必要です。resourcesディレクトリの位置はシステム・プロパティ"jp.cssj.resources.dir"に絶対パスで指定する必要があります。システム・プロパティは、javaコマンドの-Dオプションを使って、-Djp.cssj.resources.dir=/usr/local/cssj-server/resourcesのように指定するか、java.lang.System#setPropertyで設定してください。また、サブレット・アプリケーションなどではセキュリティのために後者の方法が使えない場合があります。

CSSJはjava.awtパッケージを利用するため、X Window Systemが動作していないUNIX環境で起動すると、以下のエラーメッセージが出力されることがあります。

```
Can't connect to X11 window server using ':0.0' as the value of the
DISPLAY variable.
```

例 3.2 ディスプレイがない場合のエラーメッセージ

これを回避するためには、Java VMをヘッドレスモードで起動する必要があります。ヘッドレスモードで起動するにはjavaコマンドの-Dオプションで、-Djava.awt-headless=trueと指定します。

Apache Jakartaプロジェクトの[Tomcat](#)サブレット・コンテナを使用する場合は、起動時のオプションは環境変数JAVA_OPTSで指定することができます。Tomcatを起動する前に以下のコマンドを実行しておいてください。

```
UNIXまたはLinuxの場合
```

```
# export JAVA_OPTS="-Djp.cssj.resources.dir=resourcesディレクトリへのフルパス -Djava.
awt-headless=true"

# Windowsの場合
set JAVA_OPTS="-Djp.cssj.resources.dir=resourcesディレクトリへのフルパス -Djava.
awt-headless=true"
```

例 3.3 Tomcatの起動オプションの設定

他のサーブレット・コンテナを使用する場合のオプションの設定方法は、サーブレット・コンテナのマニュアルをご参照ください。

インターフェースはソケット通信で接続する場合と同じですが、ホスト名、ポート、ユーザーID、パスワードを設定する必要はありません。これらにはどのような値を渡しても構いませんが、通常は以下の例のようにnullまたは0を渡してください。

```
//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;

...

System.setProperty("jp.cssj.resources.dir", "/usr/local/cssj-server/resources");
CTIDriver driver = CTIDriverManager.createDriverFor(null, 0);
CTISession session = driver.createSession(null, null);

//各種操作
...
```

例 3.4 直接インターフェース

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drivers/java/apidoc内のAPIドキュメントをご参照ください。

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](#)

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。

```
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 = "password";

    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();
        }
    }
}
```

```
}  
}
```

例 3.5 jp.cssj.cti.examples.ServerResource

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drivers/java/src/examplesに収められています。

サープレットの作成

drivers/java/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drivers/java/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プレット・コンテナに配備することで実際に動かすことができます。web.xml内のcontext-paramの部分で接続先のCSSJサーバー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を出力するサープレットを簡単に作ることができます。CSSJドライバを利用してユーザーが全く独自にサープレットを作っても可能ですが、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。

```
...  
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...
```

例 3.6 Content-Lengthヘッダの送出

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を使うことができます。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。

ドキュメントおよびソースコード

詳細なAPIドキュメントがdrivers/java/apidocに収められています。

ドライバのソースはdrivers/java/src内にあります。このソースは、JDK1.4.1以降でコンパイルすることができます。

3.3.2 Perl

ソースコードとドキュメント

ドライバはdrivers/perl/codeディレクトリ内にあります。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。

ドライバのファイルは暗号化されていないため、そのままソースを見ることができます。Perl5.8.0で動作が確認されています。

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drivers/perl/apidoc内のAPIドキュメントをご参照ください。

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](#)

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。

```
#!/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', 'secret');

# Content-Lengthヘッダの送信
$session->set_content_length_func (sub {
```

```

        my $length = shift;
        print "Content-Length: $length \n \n";
    });

# 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();

```

例 3.7 content.pl

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CSSJサーバーに送ります。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。

3.3.3 PHP

ソースコードとドキュメント

ドライバはdrivers/php/codeディレクトリ内にあります。アプリケーションは、この中の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。

ドライバのファイルは暗号化されていないため、そのままソースを見ることができます。PHP4.3.9で動作が確認されています。

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、drivers/php/asciiディレクトリの中にASCIIコードに変換したものが 있으므로、そちらを利用してください。

APIドキュメント(phiDocumentor)はdrivers/php/apidoc内にあります。

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drivers/php/apidoc内のAPIドキュメントをご参照ください。

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 \(\\$session &\\$session\)](#)

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。

```
<?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', 'secret') or die('サーバーに接続できません');

//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);

?>

```

例 3.8 ob.php

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う便利です。実際の使用例はvar.phpを参照してください。

```

/**
 * 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;
}

```

例 3.9 変数の変換

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。

3.4 Antを使ったバッチ処理

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。

3.4.1 CSSJタスク

CSSJ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。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。

```
<taskdef resource="jp/cssj/anttask/tasks.properties"/>
```

例 3.10 CSSJタスクの宣言

以降、cssjという要素名でCSSJタスクを使えるようになります。

表 3.1 CSSJタスクの属性

属性名	省略	説明
srcDir	可	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	可	srcDir中の変換対象となるファイルのパターンです。
excludes	可	srcDir中の変換対象外となるファイルのパターンです。
destDir	可	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	可	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。
resources	可	ライセンスキー、フォント情報などが入ったディレクトリです。省略すると、カレントディレクトリ中のresourcesディレクトリが使われます。

cssj要素内で、property要素を使って[入出力プロパティ](#)を設定することができます。プロパティ名はname属性、値はvalue属性となります。

以下の例では、docs/examplesのfont-embed.html,font-external.htmlを除くHTMLファイルをPDFに変換し、buildディレクトリに出力します。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<project default="make" basedir="..">
  <taskdef resource="jp/cssj/anttask/tasks.properties"/>

  <target name="make">
    <mkdir dir="build"/>
    <cssj srcDir="docs/examples"
      includes="*.html"
      excludes="font-embed.html,font-external.html"
      destDir="build"
      suffix=".pdf"
      resources="resources">
      <property name="output.pdf.bookmarks" value="true"/>
      <property name="output.pdf.hyperlinks" value="true"/>
      <property name="output.pdf.font.policy" value="generic"/>
    </cssj>
  </target>
</project>
```

例 3.11 CSSJタスクの使用例

3.4.2 サンプル

Antプロジェクトのサンプルはantディレクトリ内にあります。antコマンド(Windowsではant.bat)を実行することにより、サンプルのHTMLファイルがPDFに変換されます。