目次

改訂情報

変更年月日	変更内容
2013-10-11	初版
2014-04-01	第2版 下記を追加・変更しました。 ▪ ドキュメント全般 Windows Server 2012 向けの記述を追加しました。
2016-08-01	第3版 下記を追加・変更しました。 ▪ 「トラブルシューティング」→「原因と対処一覧」に注意事項を追加しました。
2018-12-01	第4版 下記を追加・変更しました。 ▪ 表記のゆれを訂正しました。
2019-04-01	第5版 下記を追加・変更しました。 ▪ トラブルシューティングを本書から独立させました。
2020-04-01	第6版 下記を追加・変更しました。 ▪ Windows 7 / Windows Server 2008 の記述を削除しました。 ▪ 「 *はじめに* 」の トラブルシューティングに関する記載を削除しました。
2020-08-01	第7版 下記を追加・変更しました。 ▪ 「 *サポート* 」の内容を見直しました。
2021-08-01	第8版 下記を追加・変更しました。 ▪ 「 *エラーコード一覧* 」へ注意を追加しました。
2021-12-01	第9版 下記を追加・変更しました。 ▪ 「 *PDFファイルの事前チェック* 」のサンプルプログラムをLinux対応版にしました。
2022-12-01	第10版 下記を追加・変更しました。 ▪ 「 *エディット機能（Edit） の文字追記を Linux 環境で使用する際の注意点* 」を追加しました。 ▪ 「 *チュートリアル* 」 　▪ 「 サンプルプログラムの場所（すべての機能） 」から「 保存場所制限 」、および、「 閲覧期限制御 」のサンプルプログラムを削除しました。 　▪ 「 JSPプログラムの作成(セキュリティAPI) 」のサンプルプログラムを変更しました。
2023-04-01	第11版 下記を追加・変更しました。 ▪ 「 *APIリストについて* 」の スクリプト開発モデル の記述を変更 ▪ 「 *スクリプト開発モデル* 」の記述を変更 ▪ 「 *APIの種類と性質* 」の スクリプト開発モデル の記述を変更 ▪ 「 *チュートリアル* 」の構成、および、記述を変更 　▪ 「 *JSPプログラムの作成（JavaEE開発モデル ）* 」を追加 　▪ 「 *jsプログラムの作成（ スクリプト開発モデル ）* 」を追加 　▪ 「 前提条件 」を削除し、記述を「 *JSPプログラムの作成（JavaEE開発モデル ）* 」へ移動 　▪ 「 *用語解説* 」の記述を変更し、「 *はじめに* 」へ移動 　▪ 「 環境 」を削除 　▪ 「 サンプルプログラムの場所（すべての機能） 」を削除し、記述を「 *JSPプログラムの作成（JavaEE開発モデル ）* 」へ移動 　▪ 「 プログラム実行 」を削除し、記述を「 *JSPプログラムの作成（JavaEE開発モデル ）* 」へ移動

変更年月日	変更内容
2023-10-01	第12版 下記を追加・変更しました。 ■ 「 *APIリストについて* 」のAPIリストの所在を変更 ■ 「 *PDFファイルの事前チェック* 」のサンプルプログラムを変更 ■ 「 *JSPファイルの作成* 」のサンプルプログラムを変更 ■ 「 *jsファイルの作成* 」のサンプルプログラムを変更 ■ 「 *サポート* 」のサポート窓口先の記述を変更

はじめに

目次

本書の目的

本書では、 IM-PDFCoordinator for Accel Platform を利用した基本的なプログラム開発や注意点等について説明します。

対象読者

本書は、開発をスムーズに開始するための手引書となっています。

したがって、実際に IM-PDFCoordinator for Accel Platform を利用したアプリケーションを開発するプログラマの方が対象です。

- 以下のいずれかを理解していることが必須です。
  - JavaEE開発モデル （ Java ）
  - スクリプト開発モデル （ サーバサイドJavaScript ）

また、本書は、以下に列挙する技術に関する知識を有することを前提として構成されています。

これらの技術に関して不明な点がある場合、本ドキュメントの内容を正しく理解することが困難になることがありますので、予めご了承ください。

なお、前提知識となる技術に関しては、一般の専門書籍等を参照してください。

- Javaプログラミング言語
- Java Servlet および JSP
- オペレーティングシステム
- ネットワーク

本書の構成

- *API リスト*

  利用できるAPIについて説明します。

- *プログラミング*

  プログラム開発の際の注意点や、プログラムの方法などを説明します。

- *チュートリアル*

  本製品のAPI を利用して実際にプログラムを作成する過程を学びます。

- *エラーコード*

  エラー発生時に返されるエラーコードを説明します。

- *サポート*

  製品サポートおよび技術情報の公開について説明します。

用語解説

Resin をインストールしたディレクトリ	%RESIN_HOME% と略します。
PDFメイクアップ をインストールしたディレクトリ	%PDFMAKEUP% と略します。

APIリスト

目次

APIリストについて

IM-PDFCoordinator for Accel Platform は、 JavaEE開発モデル 用のAPI、および、 スクリプト開発モデル 用のAPIを用意しています。

No.	フォルダ	説明
1	pdfprotection	セキュリティ機能（Security） API
2	pdfmakeup	マージ機能（Merge） / ページ機能（Page） / エディット機能（Edit） API

IM-PDFCoordinator for Accel Platform のAPIリストは、次の通りです。

- IM-PDFCoordinator for Accel Platform - PDF Makeup API ドキュメント
- IM-PDFCoordinator for Accel Platform - PDF Protection API ドキュメント

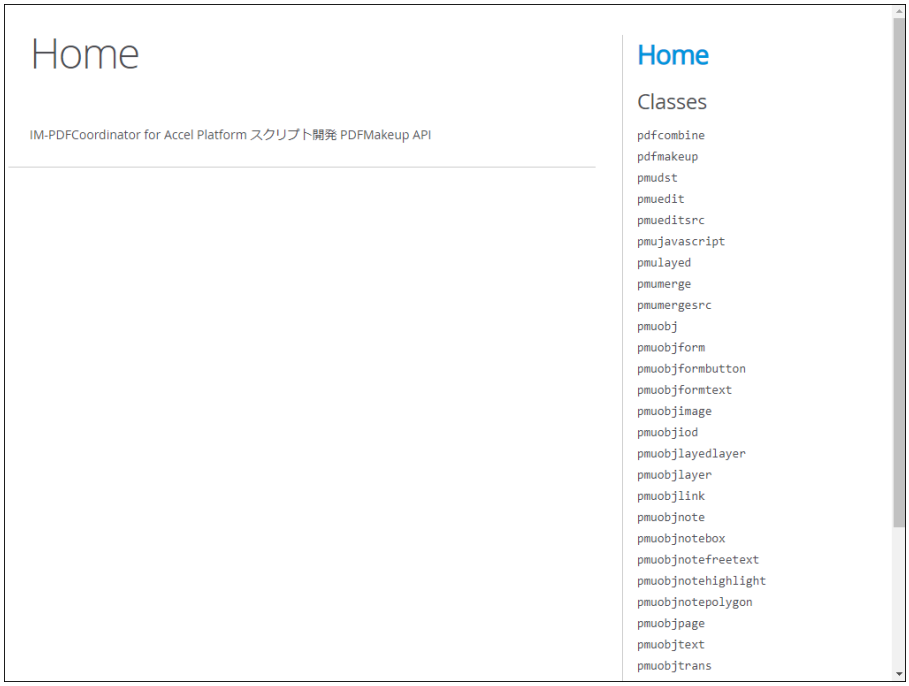
> コラム
> スクリプト開発モデル のAPIリストは、「pdfmakeup」のみ用意しています。

JavaEE開発モデル

IM-PDFCoordinator for Accel Platform は、JavaEE開発モデル で利用可能なJava-API（クラス）を用意しています。

スクリプト開発モデル

IM-PDFCoordinator for Accel Platform は、スクリプト開発モデル で利用可能なスクリプトAPI（クラス）を用意しています。



コラム
スクリプト開発モデル のAPIは、「pdfmakeup」のみ用意しています。

目次

動作概念

通常の JavaEE開発モデル ・ スクリプト開発モデル プログラムは、ApplicationRuntime で実行されます。

IM-PDFCoordinator for Accel Platform で提供されるAPI も、ApplicationRuntime で動作します。

詳しくは、APIリストをご覧ください。

APIの種類と性質

IM-PDFCoordinator for Accel Platform は、次のAPIを用意しています。

- JavaEE開発モデル で利用可能なJava-API（クラス）
- スクリプト開発モデル で利用可能なスクリプトAPI（クラス）

プログラム開発における注意点

PDFファイルへのアクセス

IM-PDFCoordinator for Accel Platform が提供するAPIで加工・編集前後のファイルのパスを指定する際には、AppRuntimeからアクセス可能なパスを指定してください。

加工・編集するPDFファイルのサイズによっては、ネットワーク、APIのレスポンス、PDFファイルの編集が完全に終了するタイミングが大きく異なる場合があります。

特にサイズの大きいPDFファイルを加工・編集する場合は、十分な時間が経過した後に加工・編集したPDF ファイルにアクセスするようにしてください。

PDFファイルの事前チェック

外部から不特定のPDFファイルが投入されるシステムでは、サーバの安定運用の点からPDFファイルの事前チェックを推奨します。

これは、PDFファイルに問題がないかチェックをすることで、サーバに害を与えるPDFファイルを事前にはじくことが目的です。

以下のサンプルでは、PDF結合処理を実行し正常終了するか確認をしています。

PDF事前チェックを exe で処理するのは、何か問題が発生した際に影響範囲をこのプロセス内に抑えるためです。

以下にサンプルを記載します。

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * PDFファイルの結合処理を実行し、元ファイルに問題があるか判定します。
 * サーバに悪影響を与えるファイルを事前にはじくことが目的です。
 * 本ソースは、利用方法の説明のためのサンプルですのでサポート対象です。
 * @version 1.0
 */
public class pdfcheck {

 private static final int STATUSCODE_SUCCESS = 0;
 private static final int STATUSCODE_MESSAGE_WITHOUT_ERRORCODE = -9990;
 private static final int STATUSCODE_EXCEPTION = -9991;
 private static final int STATUSCODE_INVALID_ARGS = -9992;

 private static class CheckResults {
  // 初期値として、ステータスコードに成功時の値／メッセージに空文字を設定
  int code = STATUSCODE_SUCCESS;
  String message = "";
 }

 /**
  * ypdfcomb.exeを実行して元ファイルに問題があるかどうかチェックします。
  * @param infile 元ファイル
  * @param pwd パスワード
  */
 public static CheckResults execute(String infile, String pwd) {
  CheckResults checkResults = new CheckResults();

  try {
   // 出力先となる一時ファイルの作成
   File outfile = File.createTempFile("pdfcheck_", ".pdf");
   // プログラム終了時に一時ファイルを自動削除するように設定
   outfile.deleteOnExit();

   List<String> command = getCommandList(infile, outfile.getPath(), pwd);

   ProcessBuilder processBuilder = new ProcessBuilder(command);
   // 標準エラーを標準出力にマージ
   processBuilder.redirectErrorStream(true);
   Process process = processBuilder.start();

   try (InputStreamReader inputStreamReader = new InputStreamReader(process.getInputStream());
   BufferedReader bufferedReader = new BufferedReader(inputStreamReader)) {
    String line;
    StringBuilder message = new StringBuilder();
    while ((line = bufferedReader.readLine()) != null) {
     message.append(line);
    }

    // エラー発生時は標準出力からコードとメッセージを取得する
    if (process.waitFor() != 0) {
     checkResults.code = getErrorCode(message.toString());
     checkResults.message = getErrorMessage(message.toString());
    }
   }
  }
  catch (IOException | InterruptedException | NumberFormatException | SecurityException e) {
   // 例外についてはエラーコードを取得することができないため、固定の数値を返す
   e.printStackTrace();
   checkResults.code = STATUSCODE_EXCEPTION;
   checkResults.message = e.getMessage();
  }

  return checkResults;
```

```
75    }
76
77    /**
78     * 実行用のコマンドを返す。
79     * @param infile 元ファイル
80     * @param outfile 出力ファイル
81     * @param pwd パスワード
82     * @return コマンド
83     */
84    private static List<String> getCommandList(String infile, String outfile, String pwd) {
85     List<String> command = new ArrayList<String>();
86     command.add("ypdfcomb");
87     // エラー発生時、メッセージが表示されるように設定
88     command.add("-se");
89     command.add("y");
90     // チェック後、出力されるPDFファイルを設定
91     command.add("-o");
92     command.add(outfile);
93     // PDF連結時の作業フォルダとして、システムのデフォルト一時フォルダを設定
94     command.add("-temp");
95     command.add(System.getProperty("java.io.tmpdir"));
96
97     // パスワードの有無で引数を分岐
98     if (Objects.isNull(pwd) || pwd.isEmpty()) {
99      // チェックを行うPDFを設定
100     command.add("-i");
101     command.add(infile);
102     } else {
103     // チェックを行うPDF、およびパスワードを設定
104     command.add("-ip");
105     command.add(infile);
106     command.add(pwd);
107     }
108
109     return command;
110   }
111
112   /**
113    * 標準出力に出力されるエラーメッセージからエラーコード部分を取得します。
114    * メッセージの書式は以下の通りです。
115    * [エラーコード]:エラーメッセージ[備考]
116    * @param message 標準出力
117    * @return エラーコード
118    */
119   private static int getErrorCode(String message) throws NumberFormatException {
120    // エラーメッセージからエラーコードが取得できなかった場合は、固定の数値を返す
121    int sts = STATUSCODE_MESSAGE_WITHOUT_ERRORCODE;
122
123    // エラーメッセージ全体を取得する正規表現のうち、
124    // エラーコード部分を正規表現グループ1とする
125    Pattern pattern = Pattern.compile("\\[(\\d*)\\]:.*");
126    Matcher matcher = pattern.matcher(message);
127
128    if (matcher.find() && matcher.groupCount() >= 1) {
129     // 正規表現グループ1(エラーコード部分)を取得
130     sts = Integer.parseInt(matcher.group(1));
131    }
132
133    return sts;
134   }
135
136   /**
137    * 標準出力に出力されるエラーメッセージから、
138    * エラーコード部分を除いたエラーメッセージを取得します。
139    * メッセージの書式は以下の通りです。
140    * [エラーコード]:エラーメッセージ[備考]
141    * @param message 標準出力
142    * @return エラーメッセージ
143    */
144   private static String getErrorMessage(String message) throws NumberFormatException {
145    // エラーコードが取得できなかった場合は、空文字を返す
146    String errorMessage = "";
147
148    // エラーメッセージ全体を取得する正規表現のうち、
149    // エラーコード部分を除いたメッセージを正規表現グループ1とする
```

```
    Pattern pattern = Pattern.compile("\\[\\d*\\]:(.*)");
    Matcher matcher = pattern.matcher(message);

    if (matcher.find() && matcher.groupCount() >= 1) {
     // 正規表現グループ1(エラーメッセージ部分)を取得
     errorMessage = matcher.group(1);
    }

    return errorMessage;
   }

   /**
    * PDF事前チェックを実行します。
    * @param args 実行引数
    * 第一引数にはチェックを行うPDFファイルパス（必須）、
    * 第二引数にはパスワード（保護されていない場合は不要）を指定します。
    * 第三引数以降は無視されます。
    * @throws Exception
    */
   public static void main (String[] args) throws Exception{

    CheckResults checkResults = new CheckResults();

    // 引数チェック
    if (args.length >= 1) {
     // チェックを行うPDF、およびそのパスワード(パスワードで保護されていない場合は不要)を指定する
     checkResults = pdfcheck.execute(args[0], (args.length >= 2) ? args[1] : null);
    } else {
     // 引数が不正な(チェックを行うPDFが指定されていない)場合は固定の数値／メッセージとする
     checkResults.code = STATUSCODE_INVALID_ARGS;
     checkResults.message = "引数が不正です";
    }

    if (checkResults.code == STATUSCODE_SUCCESS) {
     System.out.println("PDFチェックで異常は確認されませんでした");
    } else {
     System.out.println("PDFチェックでエラー[" + checkResults.code + "]が発生しました\r\n" + checkResults.message);
    }

    return;
   }
  }
```

エディット機能（Edit） の文字追記を Linux 環境で使用する際の注意点

Linux 環境で エディット機能（Edit） の文字追記を使用する場合は、MS932（ Shift_JIS ）の文字コードを設定する必要があります。

```
// 出力先インスタンス作成
pmudst dst = new pmudst();

// テキストオブジェクト生成
pmuobjtext textobj = dst.createobjtext();

//-----------------------------------
// Linux環境では文字コード指定が必須
//-----------------------------------
textobj.m_encode = "MS932";

// 文字列を追記
sts = text.setstring("これはサンプルです。");
```

注意

Linux 環境での文字追記には制限事項があります。次を確認してください。

- 文字コードの指定はテキストオブジェクト（createobjtextメソッドで生成）に適用してください。
- m_encode には必ず “MS932” を指定してください。
- 追記する文字（setstringメソッドで指定）は文字コード MS932（ Shift_JIS ）で扱えるもののみとしてください。

体験版ライセンスにおける注意点

試用版ライセンスでご利用のお客様は、30〜60 日間の試用期間が終了するとAPIが自動的に利用できない状態となります。

この状態でAPIを利用したプログラムを実行した場合に、実行時エラーとなります。

その場合は、正規の製品ライセンスを購入いただき、アンインストール後に再インストールしてください。

アンインストール・再インストールの方法は、インストールマニュアルをご確認ください。

チュートリアル

本項では、 IM-PDFCoordinator for Accel Platform での開発の導入として、APIを利用したPDFファイルの編集/加工処理を作成することによって、 IM-PDFCoordinator for Accel Platform での開発の流れを体験します。

本項のチュートリアルを開始するにあたっての前提条件は次の通りです。

- intra-mart Accel Platform 、および、 IM-PDFCoordinator for Accel Platform が正しくセットアップされていること。

ここでは、 JavaEE開発モデル 、 スクリプト開発モデル それぞれについて開発の流れを説明します。

- JavaEE開発モデル

JSPプログラムの作成（ JavaEE開発モデル ）

JavaEE開発モデル として、JSPのプログラムを作成します。

目次

準備

本チュートリアルでは、 PDFファイルを処理対象とし、セキュリティ機能（Security）（利用するAPIは jp.co.iothe.pdfprotection パッケージ）のプログラムを作成します。

「 sample.pdf 」のファイル名でPDFファイルを用意し、 intra-mart Accel Platform サーバの< C:/temp >ディレクトリに配置してください。

JSPファイルの作成

テキストエディタを使用してJSPファイルを作成します。

Resin の場合、< %RESIN_HOME%/webapps/warファイルと同名のディレクトリ/ >の配下に「 protection.jsp 」の名前でファイルを作成し、次のソースを実装します。

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<%@ page import="java.util.Date" %>
<%@ page import="java.text.ParseException" %>
<%@ page import="java.text.SimpleDateFormat" %>

<%@ page import="jp.co.iothe.pdfprotection.PdfProtection" %>
<%@ page import="jp.co.iothe.pdfprotection.PdfProtectionException" %>
<%@ page import="jp.co.iothe.pdfprotection.PdfProtectionFactory" %>

<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<%
 String src  = "C:/temp/sample.pdf";
 String pdf  = "C:/temp/out.pdf";
 String outpdf  = "";
 String message  = "Success !!";

 PdfProtection  protection ;
 int   sts  ;
 String   docinfname ;

 // セキュリティ強化APIのインスタンスを生成します
 // 設定ファイルに従ってリモートまたは直接使用のインスタンスを生成します
 protection = PdfProtectionFactory.createPdfProtection();

 // 文書情報を設定します
 // このメソッドを実行しなかった場合、文書情報はすべて空になります
 protection.setDocInfo(
  "タイトル",
  "サブタイトル",
  "作成者",
  "アプリケーション",
  "キーワード"
  );

 // 標準セキュリティを設定します
 // RC4-128ビットとAES128ビットのセキュリティはどちらか片方しか付与されません（最後に実行した方が有効）
 // このメソッドを実行しなかった場合、標準セキュリティは付与されません
 if( true ) {
  System.out.println("RC4-128ビットのセキュリティ");
  // RC4-128ビットのセキュリティを設定します
  protection.setSecurity128("open", "security",
   PdfProtection.SEC128PRINT_DISABLE,
   PdfProtection.SEC128ACC_DISABLE,
   PdfProtection.SEC128COPY_DISABLE,
   PdfProtection.SEC128DOCCHANGE_DISABLE);
 }
 else {
  System.out.println(" AES128ビットのセキュリティ");
  // AES128ビットのセキュリティを設定します
  protection.setSecurityAES128("open", "security",
   PdfProtection.SEC128PRINT_DISABLE,
   PdfProtection.SEC128ACC_DISABLE,
   PdfProtection.SEC128COPY_DISABLE,
   PdfProtection.SEC128DOCCHANGE_DISABLE);
 }

 // Webに最適化するかどうかを設定します
 protection.setFastWebView(true);

 // 上記で設定した情報を元にセキュリティを強化して新しいPDFを出力します
 int result;
 if (true) {
  // 編集元PDFにセキュリティパスワードが設定されていない場合
  result = protection.outputPdf(src, pdf);
 }
 else {
  // 編集元PDFにセキュリティパスワードが設定されている場合
  result = protection.outputPdf(src, "password", pdf);
 }

 // outputPdfで発生した例外を取得します
 if (result < 0) {
  PdfProtectionException exception = protection.getException();
```

```
  message = exception.getMessage();
 }
%>
<imui:head>
 <title>IM-PDFCoordinator-チュートリアル-JavaEE開発モデル-protection</title>
</imui:head>

<div class="imui-title">
 <h1>IM-PDFCoordinator チュートリアル JavaEE開発モデル protection</h1>
</div>

<div class="imui-form-container">
 <div class="imui-chapter-title"><h2>実行結果</h2></div>
  <table class="imui-table">
   <tbody>
    <tr>
     <th class="wd-20">出力PDFファイル</th>
     <td><%= pdf %></td>
    </tr>
    <tr>
     <th>戻り値</th>
     <td><%= result %></td>
    </tr>
    <tr>
     <th>メッセージ</th>
     <td><%= message %></td>
    </tr>
   </tbody>
  </table>
</div>
```

注意
文字コードを UTF-8 にして保存してください。

コラム

標準セキュリティのRC4-128ビット、および、AES128ビットのセキュリティは、どちらか片方のみ付与されます。

セキュリティ設定処理を複数実行した場合、最後に実行したセキュリティ設定が有効になります。

コラム
URLのセキュリティはワイルドカード「*」が使用できます。このセキュリティを使用する場合、標準セキュリティで転載と文書変更を許可しないよう設定してください。

コラム
有効期限のセキュリティはfromとtoのどちらか片方だけ設定することもできます。このセキュリティを使用する場合、標準セキュリティで転載と文書変更を許可しないよう設定してください。setSecurityDateでは、年月日までしか指定できません。

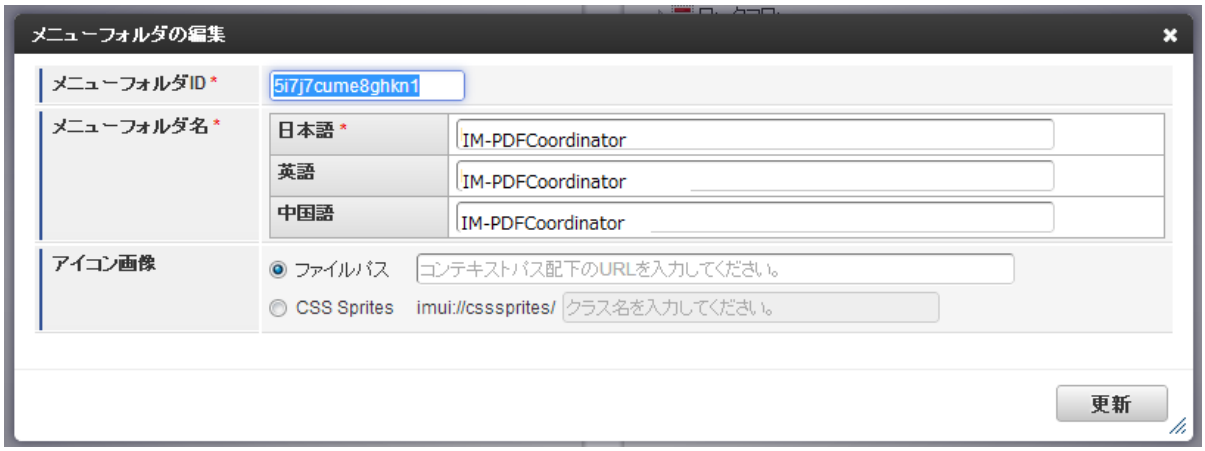
プログラムの登録

作成したJSPファイルを環境に適用するため、 Web Application Server を再起動してください。

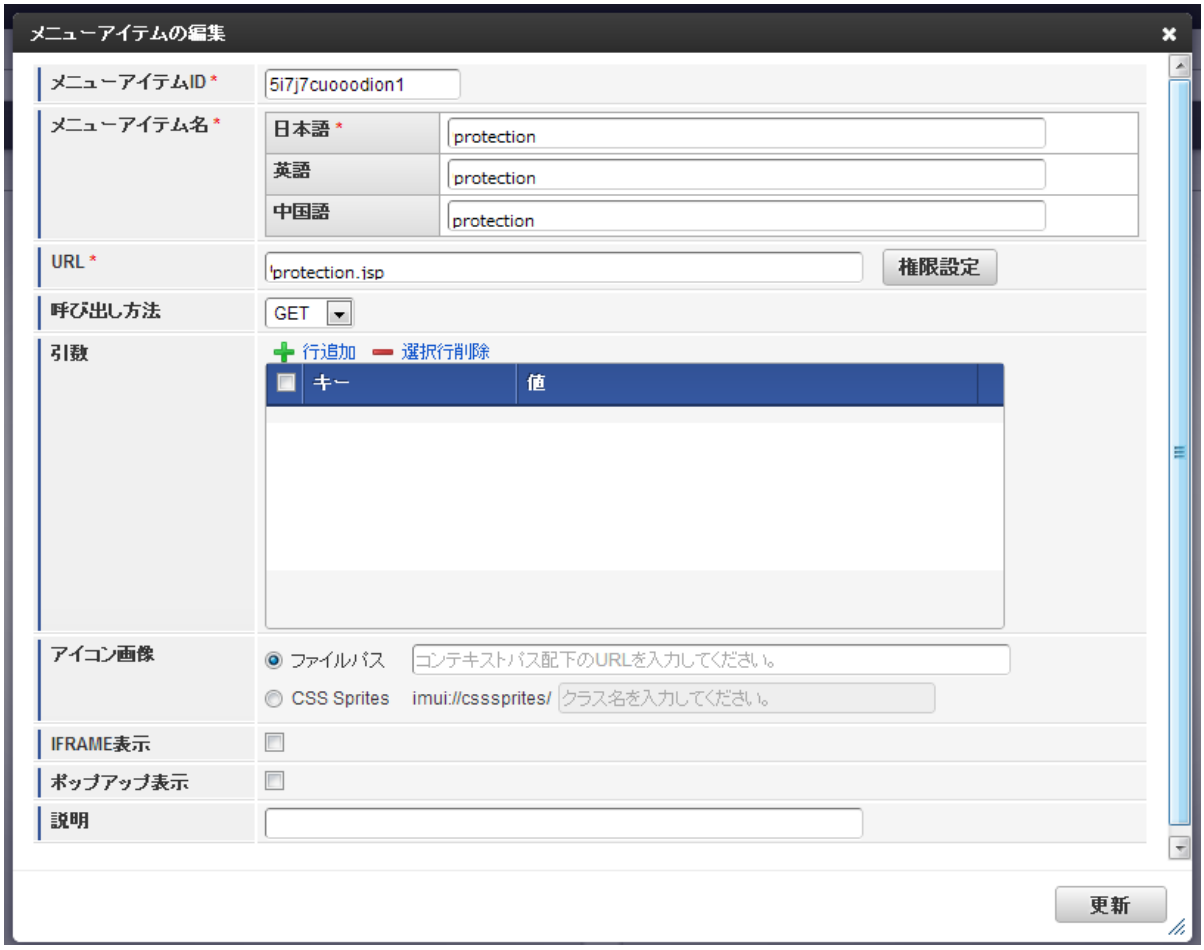
再起動後、プログラムをメニューに設定します。

メニュー設定

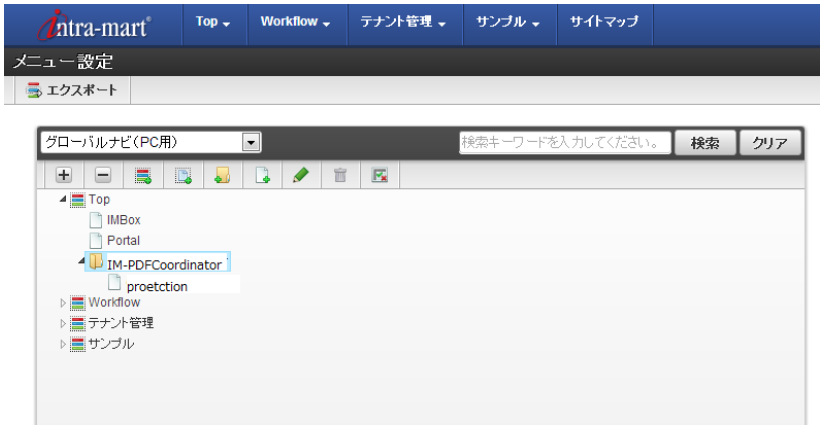
1. テナント管理者でログインし、次のメニューを設定します。
2. [テナント管理]-[メニュー]画面を開きます。
3. フォルダを作成します。



4. 作成したフォルダの下にメニューアイテムを新規作成し、URLに< protection.jsp >を設定します。



5. メニュー設定は完了です。



プログラムの実行と確認

メニューで「 protection 」を選択することにより、作成したプログラムが実行されます。

実行後は intra-mart Accel Platform サーバの< C:/temp >ディレクトリに、処理されたPDFファイル「 out.pdf 」が出力されます。

PDFビューア（ Adobe Acrobat Reader など）でファイルが正しく表示されることを確認し、このチュートリアルは完了です。

サンプルプログラムの場所

機能毎のサンプルプログラムを< %PDFMAKEUP%/sample/java >に用意しています。

使用用途に対応するフォルダの例は、次の通りです。

- < %PDFMAKEUP%/sample/java >直下のフォルダ群

No.	用途	サンプルフォルダ
1	パスワード付与	/samplesetproperty
2	パスワード解除	/samplesetproperty
3	PDFファイルの 重ね合わせ	/samplemerge
4	PDFファイルに 透かしの挿入	/sampletrans
5	用紙サイズの変更	/sampleedit
6	PDFファイルの 結合	/samplecomb
7	PDFファイルの 抽出・分割	/sampleextractpage、/sampleiv
8	PDFファイルの 回転	/sampleiv
9	PDFファイルへの 印影付与	/sampleiod、/sampletrans、/sample1
10	PDFファイルへの 文字・画像追記	/sampletrans、/sample1
11	PDFファイルへの しおり・リンク付与	/sampleol、/sample1
12	PDFファイルへの フォーム・注釈追加	/sampleform、/samplenote
13	PDFファイルへの JavaScriptの挿入	/sample1

> **コラム**
>
> 機能に合わせて< %PDFMAKEUP%/sample/data >にサンプルデータを用意しています。
>
> サンプルプログラムを実行する際に使用してください。

- スクリプト開発モデル

jsプログラムの作成（ スクリプト開発モデル ）

スクリプト開発モデル として、HTML/JavaScriptのプログラムを作成します。

目次

準備

本チュートリアルでは、 後述で作成する画面から処理対象ファイルをアップロードすることで処理を実行します。

処理対象のPDFファイルを用意してください。

jsファイルの作成

テキストエディタを使用してhtmlファイルとjsファイルを作成します。

Resin の場合、< %RESIN_HOME%/webapps/warファイルと同名のディレクトリ/WEB-INF/jssp/src/pdfc >の配下に、それぞれ次の名前でファイルを作成し、ソースを実装します。

機能	htmlファイル名	jsファイル名
ページ機能（Page）	combine.html	combine.js
マージ機能（Merge）	merge.html	merge.js
エディット機能（Edit）	edit.html	edit.js

注意
文字コードを UTF-8 にして保存してください。

コラム

RC4-40ビット、RC4-128ビット、および、AES128ビットのセキュリティは、いずれか一つのみ付与されます。

セキュリティ設定処理を複数実行した場合、最後に実行したセキュリティ設定が有効になります。

ページ機能（Page）

combine.html

```
<imart type="head">
 <title>IM-PDFCoordinator-チュートリアル-スクリプト開発モデル-combine</title>
<script type="text/javascript">
 $(function (){
  $("#combine_submit").click(function() {
   if($("#comb_file_1_path").val().length == 0 || $("#comb_file_2_path").val().length == 0) {
    imuiAlert("ファイルを2つ選択してください。", "警告");
    return;
   }
   $("#combine_form").submit();
  });
 });
</script>
</imart>

<div class="imui-title">
 <h1>IM-PDFCoordinator チュートリアル スクリプト開発モデル combine</h1>
</div>

<div class="imui-form-container">
 <div class="imui-chapter-title"><h2>combine プログラム実行</h2></div>
  <div class="imui-box-supplementation">
   <div class="supplementation-left-m">
    <span class="im-ui-icon-common-24-information"></span>
   </div>
   <p class="imui-pgh-section supplementation-left-m">
    アップロードした2つのPDFファイルを結合し、結合後PDFをダウンロードします。<br>
    PDFファイルを2つ指定し、「PDF結合」ボタンを押下してください。
   </p>
  </div>
  <imart type="form" action="combinePDF" method="POST" id="combine_form" enctype="multipart/form-data">
   <table class="imui-table">
    <tbody>
     <tr>
      <th class="wd-225px">結合対象PDFファイル1</th>
      <td><input type="file" id="comb_file_1_path" name="comb_file_1_path"></td>
     </tr>
     <tr>
      <th class="wd-225px">結合対象PDFファイル2</th>
      <td><input type="file" id="comb_file_2_path" name="comb_file_2_path"></td>
     </tr>
    </tbody>
   </table>
   <div class="imui-operation-parts">
    <imart type="imuiButton" value="PDF結合" class="imui-medium-button" id="combine_submit"></imart>
   </div>
  </imart>
 </div>
</div>
```

combine.js

```
/**
 * アップロードした2つのファイルを結合します。
 * @param {Object} request リクエスト
 */
function combinePDF(request) {

 // リクエストからアップロードしたPDFファイル1の情報を取得します。
 let uploadFile = request.getParameter("comb_file_1_path");
 let combFile1Stream = uploadFile.getValueAsStream();
 let combFile1Name = uploadFile.getFileName();

 // リクエストからアップロードしたPDFファイル2の情報を取得します。
 uploadFile = request.getParameter("comb_file_2_path");
 let combFile2Stream = uploadFile.getValueAsStream();
 let combFile2Name = uploadFile.getFileName();

 // アップロードしたファイルを一時ファイルに保管します。
 let sessionid  = Client.identifier();
 let combFile1 = File.createTempFile("comb1_" + sessionid, ".pdf", "", false);
 combFile1.save(combFile1Stream);

 let combFile2 = File.createTempFile("comb_" + sessionid, ".pdf", "", false);
 combFile2.save(combFile2Stream);

 // 出力するPDFファイルパスを作成します。
 let outFile = File.createTempFile("out_" + sessionid, ".pdf", "", false);
 let pdfFileStream = null;
 let errorMessage;

 try {
  // IM-PDFCoordinatorを実行し、PDFファイルを結合します。
  execPdfcoordinatorCombine(combFile1.path(), combFile2.path(), outFile.path());

  // PDFファイルを取得します。
  if(outFile.exist()) {
   pdfFileStream = outFile.load();
  }
 }
 catch(e) {
  errorMessage = e.message + (isUndefined(e.stack) ? '' : '\r\n' + e.stack);
 }

 // 保存したファイルを削除します。
 combFile1.remove();
 combFile2.remove();
 outFile.remove();

 // 生成したPDFファイルをダウンロードします。
 if(pdfFileStream != null) {
  let pdfFileName = combFile1Name.substr(0, combFile1Name.lastIndexOf("."))
                  + "_" + combFile2Name.substr(0, combFile2Name.lastIndexOf(".")) + ".pdf";
  Module.download.send(pdfFileStream, pdfFileName);
 }
 else {
  let logger = Logger.getLogger();
  logger.error(errorMessage);
  let response = Web.getHTTPResponse();
  response.sendError(500, "Failed to combine PDF file.");
 }
}

/**
 * IM-PDFCoordinatorを実行し、PDFファイルを結合します。
 * @param {String} combFile1Path 被結合対象のファイルパス
 * @param {String} combFile2Path 結合対象のファイルパス
 * @param {String} outFilePath 結合後の出力先PDFファイルパス
 */
function execPdfcoordinatorCombine(combFile1Path, combFile2Path, outFilePath)
{
 // PDFをファイル単位で結合するクラスのインスタンスを生成します。
 // @return {Object} PDFをファイル単位で結合するクラスのインスタンス
 let comb = new pdfcombine();
       checkerror(0, comb);

```

```
        // 内部メンバの初期化等を行います。
        // @returns {Number} 正常時は0、エラー時は-1を返します。
        let sts = comb.init();
        checkerror(sts, comb);

 // 出力PDFの文書情報を設定します。
 // setdocinfo(title, subTitle, creator, app, keyword);
 // @param {String} title タイトルに設定する文字列を指定します。
 // @param {String} subtitle サブタイトルに設定する文字列を指定します。
 // @param {String} creator 作成者に設定する文字列を指定します。
 // @param {String} app 作成アプリケーションに設定する文字列を指定します。
 // @param {String} keyword キーワードに設定する文字列を指定します。
 sts = comb.setdocinfo("文書タイトル", "文書サブタイトル", "作成者", "作成アプリケーション名", "キーワード");
        checkerror(sts, comb);

 // 出力PDFのRC4 40ビットセキュリティを設定します。
 // setsecurity(fromtop, showpasswd, securitypasswd, noprint, noedit, nocopy, noaddnote);
 // @param {boolean} fromtop 連結元の先頭のPDFを引継ぎます。
 // @param {String} showpasswd 参照用のパスワードを指定します。
 // @param {String} securitypasswd セキュリティ設定用のパスワードを指定します。
 // @param {boolean} noprint 印刷(true:不可,false:可能)
 // @param {boolean} noedit 編集(true:不可,false:可能)
 // @param {boolean} nocopy 転載(true:不可,false:可能)
 // @param {boolean} noaddnote 注釈追加(true:不可,false:可能)
// sts = comb.setsecurity(false, "open", "security", false, true, false, true);
// checkerror(sts, comb);

 // 出力PDFのRC4 128ビットセキュリティを設定します。
        // setsecurity128(showpasswd, securitypasswd, print, access, copy, change);
 // @param {String} showpasswd 参照用のパスワードを指定します。
 // @param {String} securitypasswd セキュリティ設定用のパスワードを指定します。
 // @param {String} print 128bit security(印刷)を指定します。
 // "PRINT_DISABLE":許可しない
 // "PRINT_DEGRADED":低解像度で許可する
 // "PRINT_ENABLE":許可する
 // @param {String} access 128bit security(アクセス)を指定します。
 // "ACC_DISABLE":許可しない
 // "ACC_ENABLE":許可する
 // @param {String} copy 128bit security(転載)を指定します。
 // "COPY_DISABLE":許可しない
 // "COPY_ENABLE":許可する
 // @param {String} change 128bit security(文書変更)を指定します。
 // "DOCCHANGE_DISABLE":許可しない
 // "DOCCHANGE_ASSEMBLE":アセンブリを許可する
 // "DOCCHANGE_FORMFILL":フォーム入力を許可する
 // "DOCCHANGE_ADDNOTE":フォーム入力と注釈追加を許可する
 // "DOCCHANGE_ENABLE":許可する
// sts = comb.setsecurity128("open", "security", "PRINT_DEGRADED", "ACC_ENABLE", "COPY_DISABLE",
"DOCCHANGE_ADDNOTE");
// checkerror(sts, comb);

 // 出力PDFのAES 128ビットセキュリティを設定します。
 // setsecurityaes128(showpasswd, securitypasswd, print, access, copy, change);
 // @param {String} showpasswd 参照用のパスワードを指定します。
 // @param {String} securitypasswd セキュリティ設定用のパスワードを指定します。
 // @param {String} print 128bit security(印刷)を指定します。
 // "PRINT_DISABLE":許可しない
 // "PRINT_DEGRADED":低解像度で許可する
 // "PRINT_ENABLE":許可する
 // @param {String} access 128bit security(アクセス)を指定します。
 // "ACC_DISABLE":許可しない
 // "ACC_ENABLE":許可する
 // @param {String} copy 128bit security(転載)を指定します。
 // "COPY_DISABLE":許可しない
 // "COPY_ENABLE":許可する
 // @param {String} change 128bit security(文書変更)を指定します。
 // "DOCCHANGE_DISABLE":許可しない
 // "DOCCHANGE_ASSEMBLE":アセンブリを許可する
 // "DOCCHANGE_FORMFILL":フォーム入力を許可する
 // "DOCCHANGE_ADDNOTE":フォーム入力と注釈追加を許可する
 // "DOCCHANGE_ENABLE":許可する
// sts = comb.setsecurityaes128("open", "security", "PRINT_DEGRADED", "ACC_ENABLE", "COPY_DISABLE",
"DOCCHANGE_ADDNOTE");
// checkerror(sts, comb);

```

```
150         // PDF出力後のWebに最適化の処理の有無を設定します。
151         // 特にこのメソッドを呼び出さない場合はデフォルトで最適化されます。
152         // setfastwebview(bfastwebview);
153         // @param {boolean} bfastwebview true:最適化する,false:最適化しない
154         sts = comb.setfastwebview(true);
155         checkerror(sts, comb);
156 
157         // 出力PDFをオープンし、連結の準備をします。
158         // open(outpdf);
159         // @param {String} outpdf 出力先PDFのファイル名を指定します。
160         sts = comb.open(outFilePath);
161         checkerror(sts, comb);
162 
163         // 指定ファイルのPDF連結の準備をします。
164         // combine(pdf);
165         // @param {String} pdf 連結するPDFのファイル名を指定します。
166         sts = comb.combine(combFile1Path);
167         checkerror(sts, comb);
168         sts = comb.combine(combFile2Path);
169         checkerror(sts, comb);
170 
171         // 出力PDFを連結及びクローズし、連結を終了します。
172         sts = comb.close();
173         checkerror(sts, comb);
174 
175     // 内部のハンドルを開放します。
176         comb.release();
177 }
178 
179 /**
180  * メソッドの戻り値からエラーを判定し、エラーであれば例外を投げます。
181  * @param {Number} sts メソッドの戻り値
182  * @param {Object} obj メソッドを実行したインスタンス
183  */
184 function checkerror(sts, obj) {
185  if (obj == null) {
186             throw new Error("did not create the object");
187  }
188         if (sts < 0) {
189             throw new Error("error code:" + obj.geterrorno() + ", error message:" + obj.geterror());
        }
}
```

マージ機能（Merge）

merge.html

```
<imart type="head">
 <title>IM-PDFCoordinator-チュートリアル-スクリプト開発モデル-merge</title>
<script type="text/javascript">
 $(function (){
  $("#merge_submit").click(function() {
   if($("#src_file_path").val().length == 0) {
    imuiAlert("PDFファイルを選択してください。", "警告");
    return;
   }
   $("#merge_form").submit();
  });
 });
</script>
</imart>

<div class="imui-title">
 <h1>IM-PDFCoordinator チュートリアル スクリプト開発モデル merge</h1>
</div>

<div class="imui-form-container">
 <div class="imui-chapter-title"><h2>merge プログラム実行</h2></div>
  <div class="imui-box-supplementation">
   <div class="supplementation-left-m">
    <span class="im-ui-icon-common-24-information"></span>
   </div>
   <p class="imui-pgh-section supplementation-left-m">
    アップロードしたPDFファイルに、印影が記載されたPDFファイルを重ね合わせ、処理後PDFファイルをダウンロードします。<br>
    印影が記載されたPDFファイルのサイズ関係上、A4サイズ以上のPDFファイルを指定し、「PDF重ね合わせ」ボタンを押下してください。
   </p>
  </div>
  <imart type="form" action="mergePDF" method="POST" id="merge_form" enctype="multipart/form-data">
   <table class="imui-table">
    <tbody>
     <tr>
      <th class="wd-225px">重ね合わせ対象PDFファイル</th>
      <td><input type="file" id="src_file_path" name="src_file_path"></td>
     </tr>
    </tbody>
   </table>
   <div class="imui-operation-parts">
    <imart type="imuiButton" value="PDF重ね合わせ" class="imui-medium-button" id="merge_submit"></imart>
   </div>
  </imart>
 </div>
</div>
```

merge.js

```
/**
 * アップロードしたファイルに印影を重ね合わせます。
 * @param {Object} request リクエスト
 */
function mergePDF(request) {

 // リクエストからアップロードした重ね合わせ対象のPDFファイル情報を取得します。
 let uploadFile = request.getParameter("src_file_path");
 let srcFileStream = uploadFile.getValueAsStream();
 let srcFileName = uploadFile.getFileName();

 // アップロードしたファイルを一時ファイルに保管します。
 let sessionid  = Client.identifier();
 let srcFile = File.createTempFile("src_" + sessionid, ".pdf", "", false);
 srcFile.save(srcFileStream);

 // 印影PDFファイルを取得し、一時ファイルとして保管します。
 let stampFile = File.createTempFile("stamp_" + sessionid, ".pdf", "", false);
 let stampFileReader = new PublicStorage("pdfc/tutorial/stamp.pdf").openAsBinary();
 let fileWriter = stampFile.createAsBinary();
 stampFileReader.transferTo(fileWriter);
 fileWriter.close();
 stampFileReader.close();

 // 出力するPDFファイルパスを作成します。
 let outFile = File.createTempFile("out_" + sessionid, ".pdf", "", false);
 let pdfFileStream = null;
 let errorMessage;

 try {
  // IM-PDFCoordinatorを実行し、PDFファイルに印影を重ね合わせます。
  execPdfcoordinatorMerge(srcFile.path(), stampFile.path(), outFile.path());

  // PDFファイルを取得します。
  if(outFile.exist()) {
   pdfFileStream = outFile.load();
  }
 }
 catch(e) {
  errorMessage = e.message + (isUndefined(e.stack) ? '' : '\r\n' + e.stack);
 }

 // 保存したファイルを削除します。
 srcFile.remove();
 stampFile.remove();
 outFile.remove();

 // 生成したPDFファイルをダウンロードします。
 if(pdfFileStream != null) {
  let pdfFileName = srcFileName.substr(0, srcFileName.lastIndexOf(".")) + "_merged.pdf";
  Module.download.send(pdfFileStream, pdfFileName);
 }
 else {
  let logger = Logger.getLogger();
  logger.error(errorMessage);
  let response = Web.getHTTPResponse();
  response.sendError(500, "Failed to merge PDF file.");
 }
}

/**
 * IM-PDFCoordinatorを実行し、PDFファイルに印影を重ね合わせます。
 * @param {String} srcFilePath 重ね合わせ対象のファイルパス
 * @param {String} stampFilePath 印影PDFのファイルパス
 * @param {String} outFilePath 追記後の出力先PDFファイルパス
 */
function execPdfcoordinatorMerge(srcFilePath, stampFilePath, outFilePath)
{
 // PDFマージクラスのインスタンスを生成します。
 // @return {Object} PDFマージクラスのインスタンス
 let merge = new pmumerge();
 checkerror(0, merge);

 // 内部メンバの初期化等を行います。環境ファイルパスを指定できます。
```

```
// @param {String} [etcpath] 環境ファイルパス
// @returns {Number} 正常時は0、エラー時は-1を返します。
let sts = merge.init();
checkerror(sts, merge);

// 出力PDFの文書情報を設定します。
// setdocinfo(title, subTitle, creator, app, keyword);
// @param {String} title タイトルに設定する文字列を指定します。
// @param {String} subtitle サブタイトルに設定する文字列を指定します。
// @param {String} creator 作成者に設定する文字列を指定します。
// @param {String} app 作成アプリケーションに設定する文字列を指定します。
// @param {String} keyword キーワードに設定する文字列を指定します。
sts = merge.setdocinfo("文書タイトル", "文書サブタイトル", "作成者", "作成アプリケーション名", "キーワード");
      checkerror(sts, merge);

// PDF出力時に設定するRC4 40ビットセキュリティ情報を指定します。
// setsecurity(openpassword, securitypassword, noprint, noedit, nocopy, noaddnote);
// @param {String} openpassword 参照用のパスワード
// @param {String} securitypassword セキュリティ設定用のパスワード
// @param {boolean} noprint 印刷を許可しない。
// @param {boolean} noedit 編集を許可しない。
// @param {boolean} nocopy 転載を許可しない。
// @param {boolean} noaddnote 注釈追加を許可しない。
// sts = merge.setsecurity(false, "open", "security", false, true, false, true);
// checkerror(sts, merge);

// PDF出力時に設定するRC4 128ビットセキュリティ情報を指定します。
// setsecurity128(openpassword, securitypassword, print, acc, copy, change);
// @param {String} openpassword 参照用のパスワード
// @param {String} securitypassword セキュリティ設定用のパスワード
// @param {String} print 印刷
// "PRINT_DISABLE":許可しない
// "PRINT_DEGRADED":低解像度で許可する
// "PRINT_ENABLE":許可する
// @param {String} acc アクセス
// "ACC_DISABLE":許可しない
// "ACC_ENABLE":許可する
// @param {String} copy 転載
// "COPY_DISABLE":許可しない
// "COPY_ENABLE":許可する
// @param {String} change 文書変更
// "DOCCHANGE_DISABLE":許可しない
// "DOCCHANGE_ASSEMBLE":アセンブリを許可する
// "DOCCHANGE_FORMFILL":フォーム入力を許可する
// "DOCCHANGE_ADDNOTE":フォーム入力と注釈追加を許可する
// "DOCCHANGE_ENABLE":許可する
// sts = merge.setsecurity128("open", "security", "PRINT_DEGRADED", "ACC_ENABLE", "COPY_DISABLE",
"DOCCHANGE_ADDNOTE");
// checkerror(sts, merge);

// PDF出力時に設定するAES 128ビットセキュリティ情報を指定します。
// setsecurityaes128(openpassword, securitypassword, print, acc, copy, change);
// @param {String} openpassword 参照用のパスワード
// @param {String} securitypassword セキュリティ設定用のパスワード
// @param {String} print 印刷
// "PRINT_DISABLE":許可しない
// "PRINT_DEGRADED":低解像度で許可する
// "PRINT_ENABLE":許可する
// @param {String} acc アクセス
// "ACC_DISABLE":許可しない
// "ACC_ENABLE":許可する
// @param {String} copy 転載
// "COPY_DISABLE":許可しない
// "COPY_ENABLE":許可する
// @param {String} change 文書変更
// "DOCCHANGE_DISABLE":許可しない
// "DOCCHANGE_ASSEMBLE":アセンブリを許可する
// "DOCCHANGE_FORMFILL":フォーム入力を許可する
// "DOCCHANGE_ADDNOTE":フォーム入力と注釈追加を許可する
// "DOCCHANGE_ENABLE":許可する
// sts = merge.setsecurityaes128("open", "security", "PRINT_DEGRADED", "ACC_ENABLE", "COPY_DISABLE",
"DOCCHANGE_ADDNOTE");
// checkerror(sts, merge);

// PDF出力後のWebに最適化の処理の有無を設定します。
```

```
// 特にこのメソッドを呼び出さない場合はデフォルトで最適化されます。
// setfastwebview(bfastwebview);
// @param {boolean} bfastwebview true:最適化する,false:最適化しない
sts = merge.setfastwebview(true);
checkerror(sts, merge);

// マージの基本になるPDFをオープンします。
// openbase(filename, passwd);
// @param {String} filename ファイル名
// @param {String} passwd パスワード
// @returns {pmumergesrc} マージ処理の基本PDFのpmumergesrcクラス。
// エラーの場合はnull。
let srcBase = merge.openbase(srcFilePath, null);
checkerror(0, srcBase);

// pmumerge.outpage()によるマージ時の上下関係を設定します。
// setorder(order);
// @param {Number} order 任意の数値を指定します。
// 0が一番下になります。
// 0以下は0と見なされます。
sts = srcBase.setorder(0);
checkerror(sts, srcBase);

// マージするPDFをオープンします。
// openmerge(filename, passwd);
// @param {String} filename ファイル名
// @param {String} passwd パスワード
// @returns {pmumergesrc} マージ処理のマージするPDFのpmumergesrcクラス。
// エラーの場合はnull。
let srcMerge = merge.openmerge(stampFilePath, null);
checkerror(0, srcMerge);

// pmumerge.outpage()によるマージ時の上下関係を設定します。
// setorder(order);
// @param {Number} order 任意の数値を指定します。
// 0が一番下になります。
// 0以下は0と見なされます。
sts = srcMerge.setorder(99);
checkerror(sts, srcMerge);

// pmumerge.outpage()によるマージ時の原点位置を設定します。
// setorigin(origin);
// @param {String} origin 以下の追記オブジェクトの基本位置を指定します。
// "LT":左上
// "LM":左中段
// "LB":左下
// "CT":中央上
// "CM":中央中段
// "CB":中央下
// "RT":右上
// "RM":右中段
// "RB":右下
sts = srcMerge.setorigin("CM");
checkerror(sts, srcMerge);

// どのレイヤに含めるかを設定します。
// setlayer(layer);
// @param {pmuobjlayer} layer pmumerge.createlayer()で作成したインスタンスを指定します。
// レイヤ指定を無効にするにはnullを指定します。
sts = srcMerge.setlayer(null);
checkerror(sts, srcMerge);

// 出力PDFをオープンします。
// openoutput(filename);
// @param {String} filename ファイル名
sts = merge.openoutput(outFilePath);
checkerror(sts, merge);

// オープンした切出し元のPDFのページ数を返します。
// @returns {Number} オープンした切出し元のPDFのページ数
let basePageCount = srcBase.getpagecount();
checkerror(basePageCount, srcBase);

for(let i = 0; i < basePageCount; i++) {
 // pmumerge.outpage()によるマージ対象のページを設定します。
```

```
 // setpage(page);
 // @param {Number} page pmumerge.outpage()が出力する、対象になるページを指定します(1以上の値)。
 // 存在しないページ番号を指定した場合は、マージ対象になりません。
 // また、基本PDFは、ページの順序、スキップ、削除はできません。
 // @returns {Number} 0:マージするPDFに、無効なページ番号を指定した。
 // 1:マージするPDFに、有効なページ番号を指定した。
 // 2:基本PDFに正しいページ番号を指定した。
 // 3:基本PDFに正しくない(現在ページ以外)のページ番号を指定した。
 sts = srcMerge.setpage(1);
 checkerror(sts, srcMerge);

 // マージしてページを出力します。
 sts = merge.outpage();
 checkerror(sts, merge);
 }

 // マージ用にオープンされている出力ファイルをクローズします。
 // また、pmumerge.outpage()呼び出しが、基本PDFの途中のページで打ち切られた場合は、残りのページも出力してからクローズします。
 sts = merge.closeoutput();
 checkerror(sts, merge);

 // 内部のハンドルを開放します。
 merge.release();
}

/**
 * メソッドの戻り値からエラーを判定し、エラーであれば例外を投げます。
 * @param {Number} sts メソッドの戻り値
 * @param {Object} obj メソッドを実行したインスタンス
 */
function checkerror(sts, obj) {
 if (obj == null) {
  throw new Error("did not create the object");
 }
 if (sts < 0) {
  throw new Error("error code:" + obj.geterrorno() + ", error message:" + obj.geterror());
 }
}
```

エディット機能（Edit）

edit.html

```
<imart type="head">
 <title>IM-PDFCoordinator-チュートリアル-スクリプト開発モデル-edit</title>
<script type="text/javascript">
 $(function (){
  $("#edit_submit").click(function() {
   if($("#src_file_path").val().length == 0) {
    imuiAlert("PDFファイルを選択してください。", "警告");
    return;
   }
   $("#edit_form").submit();
  });
 });
</script>
</imart>

<div class="imui-title">
 <h1>IM-PDFCoordinator チュートリアル スクリプト開発モデル edit</h1>
</div>

<div class="imui-form-container">
 <div class="imui-chapter-title"><h2>edit プログラム実行</h2></div>
  <div class="imui-box-supplementation">
   <div class="supplementation-left-m">
    <span class="im-ui-icon-common-24-information"></span>
   </div>
   <p class="imui-pgh-section supplementation-left-m">
    アップロードしたPDFファイルへ文字・画像を追記し、追記後PDFをダウンロードします。<br>
    追記対象ファイルを指定し、「PDF追記」ボタンを押下してください。
   </p>
  </div>
  <imart type="form" action="editPDF" method="POST" id="edit_form" enctype="multipart/form-data">
   <table class="imui-table">
    <tbody>
     <tr>
      <th class="wd-225px">追記対象PDFファイル</th>
      <td><input type="file" id="src_file_path" name="src_file_path"></td>
     </tr>
    </tbody>
   </table>
   <div class="imui-operation-parts">
    <imart type="imuiButton" value="PDF追記" class="imui-medium-button" id="edit_submit"></imart>
   </div>
  </imart>
 </div>
</div>
```

edit.js

```
/**
 * アップロードしたファイルへ文字・画像を追記します。
 * @param {Object} request リクエスト
 */
function editPDF(request) {

 // リクエストからアップロードした追記されるPDFファイル情報を取得します。
 let uploadFile = request.getParameter("src_file_path");
 let srcFileStream = uploadFile.getValueAsStream();
 let srcFileName = uploadFile.getFileName();

 // アップロードしたファイルを一時ファイルに保管します。
 let sessionid  = Client.identifier();
 let srcFile = File.createTempFile("src_" + sessionid, ".pdf", "", false);
 srcFile.save(srcFileStream);

 // 画像ファイルを取得し、一時ファイルとして保管します。
 let picFile = File.createTempFile("pic_" + sessionid, ".jpg", "", false);
 let picFileReader = new PublicStorage("pdfc/tutorial/bitmap.jpg").openAsBinary();
 let fileWriter = picFile.createAsBinary();
 picFileReader.transferTo(fileWriter);
 fileWriter.close();
 picFileReader.close();

 // 出力するPDFファイルパスを作成します。
 let outFile = File.createTempFile("out_" + sessionid, ".pdf", "", false);
 let pdfFileStream = null;
 let errorMessage;

 try {
  // IM-PDFCoordinatorを実行し、PDFファイルへ追記します。
  execPdfcoordinatorEdit(srcFile.path(), picFile.path(), outFile.path());

  // PDFファイルを取得します。
  if(outFile.exist()) {
   pdfFileStream = outFile.load();
  }
 }
 catch(e) {
  errorMessage = e.message + (isUndefined(e.stack) ? '' : '\r\n' + e.stack);
 }

 // 保存したファイルを削除します。
 srcFile.remove();
 picFile.remove();
 outFile.remove();

 // 生成したPDFファイルをダウンロードします。
 if(pdfFileStream != null) {
  let pdfFileName = srcFileName.substr(0, srcFileName.lastIndexOf(".")) + "_edited.pdf";
  Module.download.send(pdfFileStream, pdfFileName);
 }
 else {
  let logger = Logger.getLogger();
  logger.error(errorMessage);
  let response = Web.getHTTPResponse();
  response.sendError(500, "Failed to edit PDF file.");
 }
}

/**
 * IM-PDFCoordinatorを実行し、PDFファイルへ文字・画像を追記します。
 * @param {String} srcFilePath 被追記対象のファイルパス
 * @param {String} picFilePath 追記対象の画像ファイルパス
 * @param {String} outFilePath 追記後の出力先PDFファイルパス
 */
function execPdfcoordinatorEdit(srcFilePath, picFilePath, outFilePath)
{
 // PDF出力制御クラスのインスタンスを生成します。
 // @return {Object} PDF出力制御クラスのインスタンス
 let dst = new pmudst();
 checkerror(0, dst);

 // 内部メンバの初期化等を行います。
```

```
// @returns {Number} 正常時は0、エラー時は-1を返します。
let sts = dst.init();
checkerror(sts, dst);

// 出力PDFの文書情報を設定します。
// setdocinfo(title, subTitle, creator, app, keyword);
// @param {String} title タイトルに設定する文字列を指定します。
// @param {String} subtitle サブタイトルに設定する文字列を指定します。
// @param {String} creator 作成者に設定する文字列を指定します。
// @param {String} app 作成アプリケーションに設定する文字列を指定します。
// @param {String} keyword キーワードに設定する文字列を指定します。
sts = dst.setdocinfo("文書タイトル", "文書サブタイトル", "作成者", "作成アプリケーション名", "キーワード");
      checkerror(sts, dst);

// PDF出力時に設定するRC4 40ビットセキュリティ情報を指定します。
// setsecurity(openpassword, securitypassword, noprint, noedit, nocopy, noaddnote);
// @param {String} openpassword 参照用のパスワード
// @param {String} securitypassword セキュリティ設定用のパスワード
// @param {boolean} noprint 印刷を許可しない。
// @param {boolean} noedit 編集を許可しない。
// @param {boolean} nocopy 転載を許可しない。
// @param {boolean} noaddnote 注釈追加を許可しない。
// sts = dst.setsecurity(false, "open", "security", false, true, false, true);
// checkerror(sts, dst);

// PDF出力時に設定するRC4 128ビットセキュリティ情報を指定します。
// setsecurity128(openpassword, securitypassword, print, acc, copy, change);
// @param {String} openpassword 参照用のパスワード
// @param {String} securitypassword セキュリティ設定用のパスワード
// @param {String} print 印刷
// "PRINT_DISABLE":許可しない
// "PRINT_DEGRADED":低解像度で許可する
// "PRINT_ENABLE":許可する
// @param {String} acc アクセス
// "ACC_DISABLE":許可しない
// "ACC_ENABLE":許可する
// @param {String} copy 転載
// "COPY_DISABLE":許可しない
// "COPY_ENABLE":許可する
// @param {String} change 文書変更
// "DOCCHANGE_DISABLE":許可しない
// "DOCCHANGE_ASSEMBLE":アセンブリを許可する
// "DOCCHANGE_FORMFILL":フォーム入力を許可する
// "DOCCHANGE_ADDNOTE":フォーム入力と注釈追加を許可する
// "DOCCHANGE_ENABLE":許可する
// sts = dst.setsecurity128("open", "security", "PRINT_DEGRADED", "ACC_ENABLE", "COPY_DISABLE",
"DOCCHANGE_ADDNOTE");
// checkerror(sts, dst);

// PDF出力時に設定するAES 128ビットセキュリティ情報を指定します。
// setsecurityaes128(openpassword, securitypassword, print, acc, copy, change);
// @param {String} openpassword 参照用のパスワード
// @param {String} securitypassword セキュリティ設定用のパスワード
// @param {String} print 印刷
// "PRINT_DISABLE":許可しない
// "PRINT_DEGRADED":低解像度で許可する
// "PRINT_ENABLE":許可する
// @param {String} acc アクセス
// "ACC_DISABLE":許可しない
// "ACC_ENABLE":許可する
// @param {String} copy 転載
// "COPY_DISABLE":許可しない
// "COPY_ENABLE":許可する
// @param {String} change 文書変更
// "DOCCHANGE_DISABLE":許可しない
// "DOCCHANGE_ASSEMBLE":アセンブリを許可する
// "DOCCHANGE_FORMFILL":フォーム入力を許可する
// "DOCCHANGE_ADDNOTE":フォーム入力と注釈追加を許可する
// "DOCCHANGE_ENABLE":許可する
// sts = dst.setsecurityaes128("open", "security", "PRINT_DEGRADED", "ACC_ENABLE", "COPY_DISABLE",
"DOCCHANGE_ADDNOTE");
// checkerror(sts, dst);

// PDF出力後のWebに最適化の処理の有無を設定します。
// 特にこのメソッドを呼び出さない場合はデフォルトで最適化されます。
```

```
150   // setfastwebview(bfastwebview);
151   // @param {boolean} bfastwebview true:最適化する,false:最適化しない
152   sts = dst.setfastwebview(true);
153   checkerror(sts, dst);
154
155   // 指定された切出し元のPDFの全てのページを出力時の対象とします。
156   // addsrcfile(filename, passwd);
157   // @param {String} filename 切出し元のPDFのファイル名
158   // @param {String} passwd 切出し元のPDFのパスワード
159   // @returns {Number} 切出し元のPDFファイルのページ数を返します。
160   sts = dst.addsrcfile(srcFilePath, null);
161   checkerror(sts, dst);
162
163   addtext(dst);
164
165   addimage(dst, picFilePath);
166
167   // 指定された切出し元のPDF、及び、追記オブジェクトを指定されたファイルにPDF出力します。
168   // outputpdf(filename);
169   // @param {String} filename 出力先のPDFファイルのファイル名
170   sts = dst.outputpdf(outFilePath);
171   checkerror(sts, dst);
172
173   // 内部のハンドルを開放します。
174   dst.release();
175  }
176
177  /**
178   * 文字列を追記します。
179   * @param {Object} dst PDF出力制御クラスのインスタンス
180   */
181  function addtext(dst) {
182   let sts = 0;
183
184   // PDF出力時に追記するテキスト枠オブジェクトクラスを作成します。
185   // @returns {pmuobjtext} テキスト枠オブジェクトクラス
186   let text = dst.createobjtext();
187   checkerror(sts, text);
188   text.m_encode = "MS932";
189
190   // オブジェクトの基本位置を指定します。
191   // setbasepos(postype);
192   // @param {String} postype 以下の追記オブジェクトの基本位置を指定します。
193   // "XY":XYを使用
194   // "LT":左上
195   // "LM":左中段
196   // "LB":左下
197   // "CT":中央上
198   // "CM":中央中段
199   // "CB":中央下
200   // "RT":右上
201   // "RM":右中段
202   // "RB":右下
203   sts = text.setbasepos("LT");
204   checkerror(sts, text);
205
206   // 追記オブジェクトをオリジナルPDFの上または下のどちらに追記するかを設定します。
207   // setlayer(layertype);
208   // @param {String} layertype 以下の追記位置を設定します。
209   // "FRONT":追記オブジェクトをオリジナルの上(前面)に配置
210   // "BACK":追記オブジェクトをオリジナルの下(背面)に配置
211   sts = text.setlayer("FRONT");
212   checkerror(sts, text);
213
214   // オブジェクトをどのページにするかを設定します。
215   // settargetpage(pagetype, pageno1, pageno2)
216   // ページ番号を指定する際は、1ページ目を「1」として指定してください。
217   // @param {String} pagetype 以下のページ指定の種類を指定します。
218   // "ALL":全てのページ
219   // "FROM":指定ページ以降
220   // "FROMTO":範囲指定
221   // "PAGE":特定のページ
222   // "TO":指定ページまで
223   // @param {Number} pageno1 ページ番号1
224   // @param {Number} pageno2 ページ番号2(FROMTOの場合のみ使用)
```

```
 sts = text.settargetpage("PAGE", 1, 0);
 checkerror(sts, text);

 // 追記オブジェクトが使用するフォントのサイズを設定します。
 // setfontsize(fontsize);
 // @param {Number} fontsize フォントのサイズ
 sts = text.setfontsize("32.0");
 checkerror(sts, text);

 // 追記オブジェクトが使用するフォントの色をRGBで設定します。
 // setfontcolor(r, g, b);
 // @param {Number} r 赤値
 // @param {Number} g 緑値
 // @param {Number} b 青値
 sts = text.setfontcolor(255, 0, 0);
 checkerror(sts, text);

 // テキストオブジェクトの文字列を設定します。
 // setstring(str);
 // @param {String} str 文字列
 sts = text.setstring("文字列追記テスト1 ");
 checkerror(sts, text);

 // テキストオブジェクトの枠線の種類を設定します。
 // setbordertype(bordertype);
 // @param {String} bordertype 以下の枠線の種類を指定します。
 // "AUTONEWLINE":自動改行
 // "BORDERFITSTRING":枠をテキストに合せる
 // "NONAUTONEWLINE":調節なし
 // "STRINGFITBORDER":テキストを枠に合せる
 sts = text.setbordertype("BORDERFITSTRING");
 checkerror(sts, text);
}

/**
 * 画像を追記します。
 * @param {Object} dst PDF出力制御クラスのインスタンス
 * @param {string} picFilePath 追記対象の画像ファイルパス
 */
function addimage(dst, picFilePath) {
 let sts = 0;

 // PDF出力時に追記するイメージオブジェクトクラスを作成します。
 // @returns {pmuobjimage} イメージオブジェクトクラス
 let objimg = dst.createobjimage();
 checkerror(sts, objimg);

 // オブジェクトの基本位置を指定します。
 // setbasepos(postype);
 // @param {String} postype 以下の追記オブジェクトの基本位置を指定します。
 // "XY":XYを使用
 // "LT":左上
 // "LM":左中段
 // "LB":左下
 // "CT":中央上
 // "CM":中央中段
 // "CB":中央下
 // "RT":右上
 // "RM":右中段
 // "RB":右下
 sts = objimg.setbasepos("CM");
 checkerror(sts, objimg);

 // 追記オブジェクトをオリジナルPDFの上または下のどちらに追記するかを設定します。
 // setlayer(layertype);
 // @param {String} layertype 以下の追記位置を設定します。
 // "FRONT":追記オブジェクトをオリジナルの上(前面)に配置
 // "BACK":追記オブジェクトをオリジナルの下(背面)に配置
 sts = objimg.setlayer("BACK");
 checkerror(sts, objimg);

 // オブジェクトをどのページにするかを設定します。
 // settargetpage(pagetype, pageno1, pageno2)
 // ページ番号を指定する際は、1ページ目を「1」として指定してください。
 // @param {String} pagetype 以下のページ指定の種類を指定します。
```

```
// "ALL":全てのページ
// "FROM":指定ページ以降
// "FROMTO":範囲指定
// "PAGE":特定のページ
// "TO":指定ページまで
// @param {Number} pageno1 ページ番号1
// @param {Number} pageno2 ページ番号2(FROMTOの場合のみ使用)
sts = objimg.settargetpage("ALL", 0, 0);
checkerror(sts, objimg);

// イメージオブジェクトの表示の大きさを指定します。
// setsize(option, width, height);
// @param {String} option 以下のサイズ指定方法のオプションを指定します。
// "SIZE":イメージのサイズのまま
// "WH":指定されたボックスの幅高さ
// "LT":原寸の比率固定(左上段)
// "LM":原寸の比率固定(左中段)
// "LB":原寸の比率固定(左下段)
// "CT":原寸の比率固定(中央上段)
// "CM":原寸の比率固定(中央中段)
// "CB":原寸の比率固定(中央下段)
// "RT":原寸の比率固定(右上段)
// "RM":原寸の比率固定(右中段)
// "RB":原寸の比率固定(右下段)
// 比率固定を指定した場合、以下のように表示されます。
// ・原寸に対して、ボックスが縦長の場合:
//      widthの値を基準とし、画像の高さを調整。
//      調整後、ボックスに対して上段、中段、下段揃えで表示。
// ・原寸に対して、ボックスが横長の場合:
//      heightの値を基準とし、画像の幅を調整。
//      調整後、ボックスに対して左、中央、右揃えで表示。
// @param {Number} width 幅
// @param {Number} height 高さ
sts = objimg.setsize("SIZE", 255, 407);
checkerror(sts, objimg);

// イメージオブジェクトのファイル名を設定します。
// setfilename(imgtype, filename);
// @param {String} imgtype 以下のイメージファイルの種類を指定します。
// "BMP":Windows BMP
// "JPG":JPG
// "PNG":PNG
// "PNGALPHA":アルファチャンネルを持つPNG
// "TIFFG4":TIFF G4
// @param {String} filename イメージファイル名
sts = objimg.setfilename("JPG", picFilePath);
checkerror(sts, objimg);
}

/**
 * メソッドの戻り値からエラーを判定し、エラーであれば例外を投げます。
 * @param {Number} sts メソッドの戻り値
 * @param {Object} obj メソッドを実行したインスタンス
 */
function checkerror(sts, obj) {
 if (obj == null) {
  throw new Error("did not create the object");
 }
 if (sts < 0) {
  throw new Error("error code:" + obj.geterrorno() + ", error message:" + obj.geterror());
 }
}
```

ルーティング設定ファイルの作成

ルーティング用のxmlファイルを作成します。次のようにファイルのマッピング情報を記述します。

- 認可の設定に当たる authz-default の mapper 属性には “welcome-all” を設定します。
- URLの設定に当たる file-mapping には、 path 属性に画面のURLを、 page 属性に画面のファイルパスをそれぞれ設定します。

Resin の場合、作成した設定ファイルは< %RESIN_HOME%/webapps/{アプリケーション名}/WEB-INF/conf/routing-jssp-config >の配下に設置してください。

ファイル名については、次の通りです。

機能	xmlファイル名
ページ機能（Page）	sample-pdfc-combine.xml
マージ機能（Merge）	sample-pdfc-merge.xml
エディット機能（Edit）	sample-pdfc-edit.xml

注意

文字コードを UTF-8 にして保存してください。

ページ機能（Page）

sample-pdfc-combine.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config
 xmlns="http://www.intra-mart.jp/router/routing-jssp-config" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config ../../schema/routing-jssp-config.xsd ">

 <authz-default mapper="welcome-all" />
 <file-mapping path="pdfc/combine" page="pdfc/combine">
  <authz uri="service://pdfc/combine" action="execute" />
 </file-mapping>

</routing-jssp-config>
```

マージ機能（Merge）

sample-pdfc-merge.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config
 xmlns="http://www.intra-mart.jp/router/routing-jssp-config" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config ../../schema/routing-jssp-config.xsd ">

 <authz-default mapper="welcome-all" />
 <file-mapping path="pdfc/merge" page="pdfc/merge">
  <authz uri="service://pdfc/merge" action="execute" />
 </file-mapping>

</routing-jssp-config>
```

エディット機能（Edit）

sample-pdfc-edit.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config
 xmlns="http://www.intra-mart.jp/router/routing-jssp-config" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config ../../schema/routing-jssp-config.xsd ">

 <authz-default mapper="welcome-all" />
 <file-mapping path="pdfc/edit" page="pdfc/edit">
  <authz uri="service://pdfc/edit" action="execute" />
 </file-mapping>

</routing-jssp-config>
```

プログラムの登録

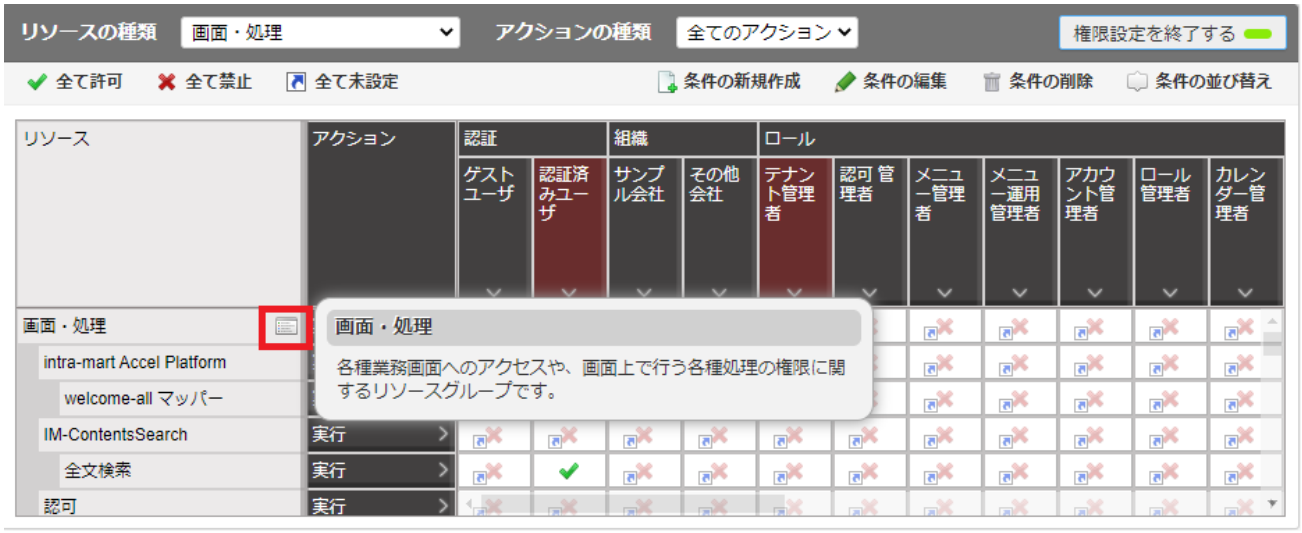
作成したhtmlファイルとjsファイルを環境に適用するため、 Web Application Server を再起動してください。

再起動後、プログラムを認可とメニューに設定します。

認可設定

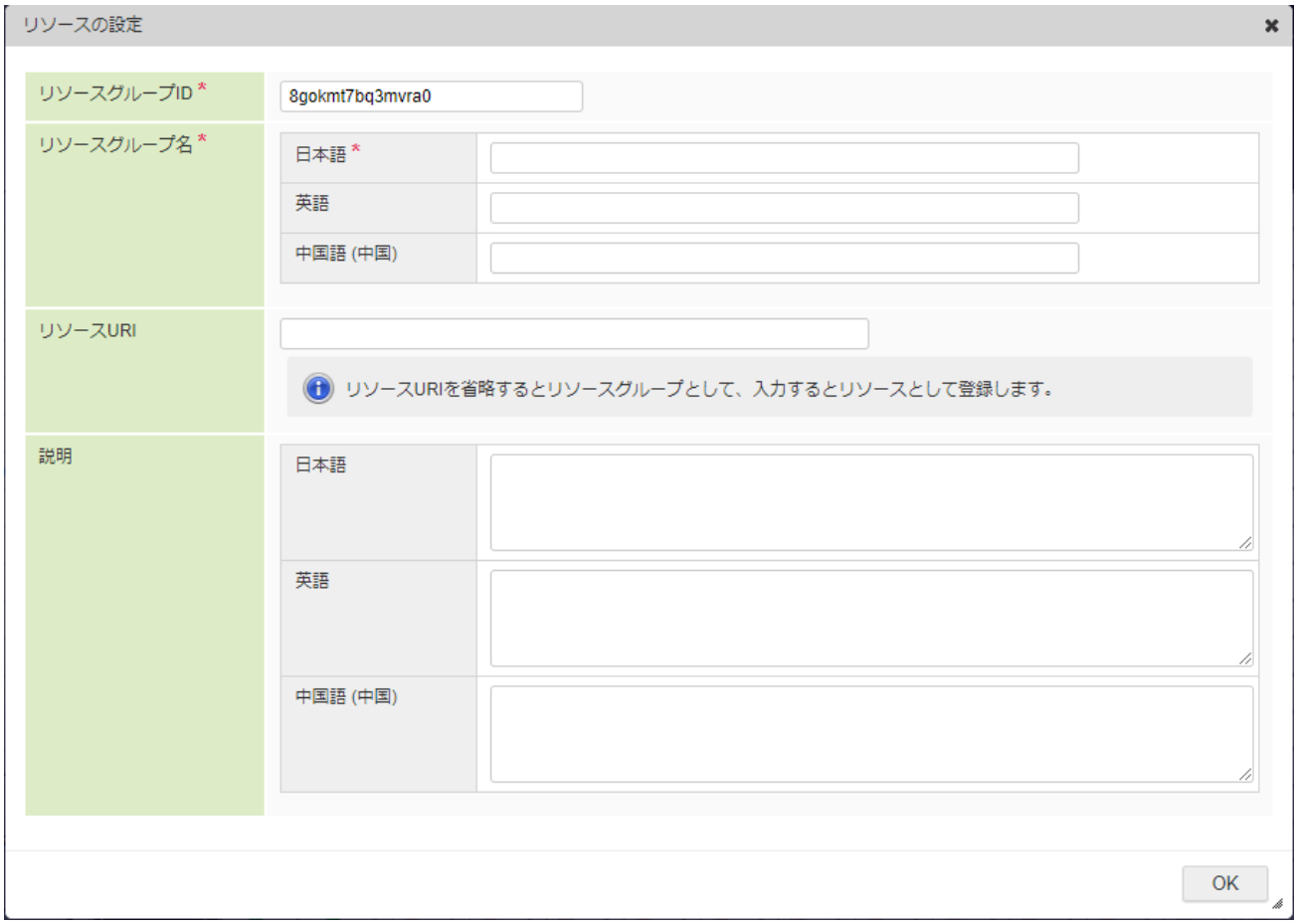
1. テナント管理者でログインし、次のメニューを設定します。
2. [テナント管理]-[認可]画面を開きます。
3. [権限設定を開始する]ボタンを押下します。

4. [リソース]を選択し、[リソースの詳細を開く]押下します。



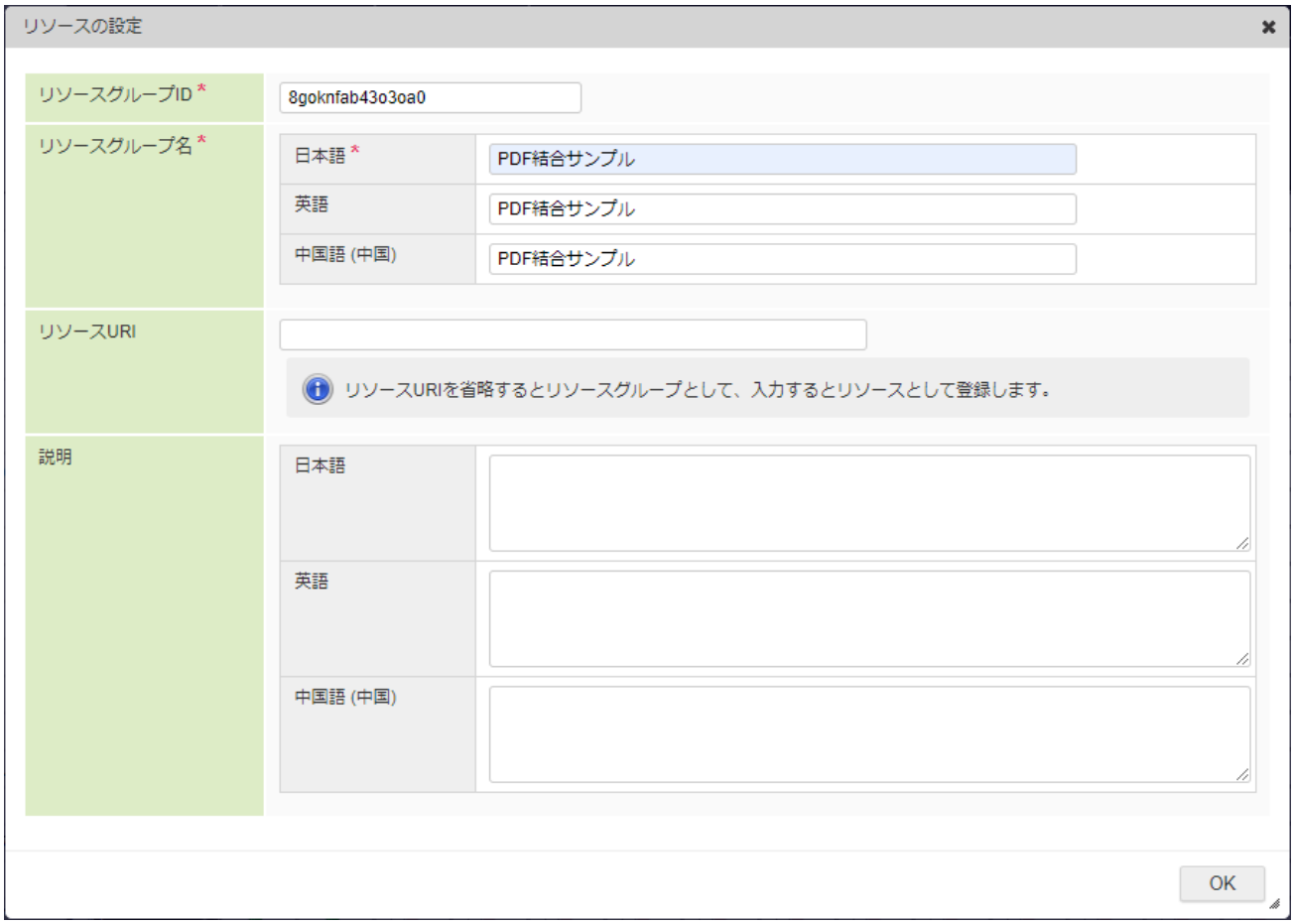
5. [配下にリソースを新規作成]を押下します。

6. リソースグループを作成します。



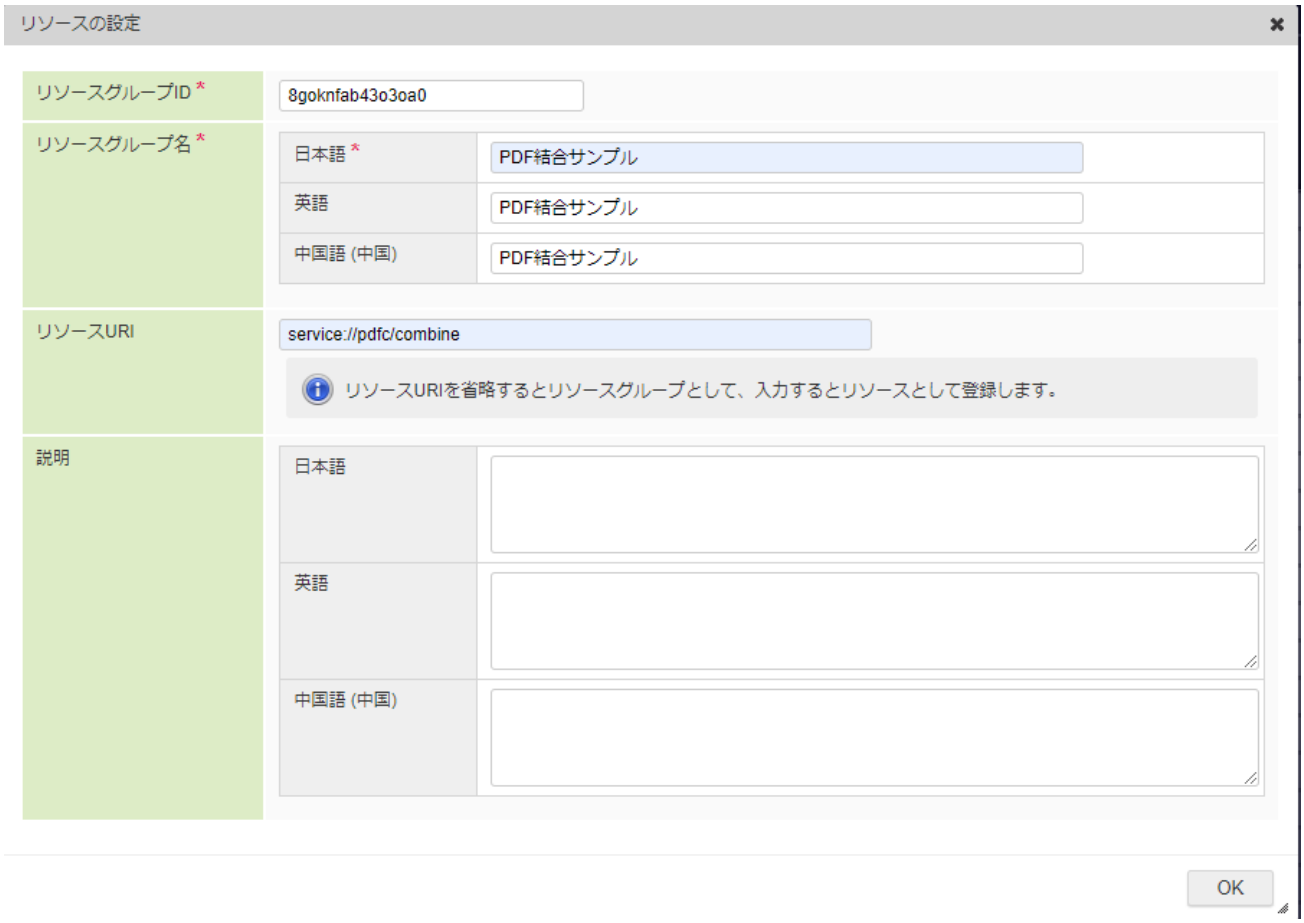
7. リソースグループ名に、次の値を設定します。

機能	リソースグループ名
ページ機能（Page）	PDF結合サンプル
マージ機能（Merge）	PDFマージサンプル
エディット機能（Edit）	PDF追記サンプル



8. リソースURIに、次の値を設定します。

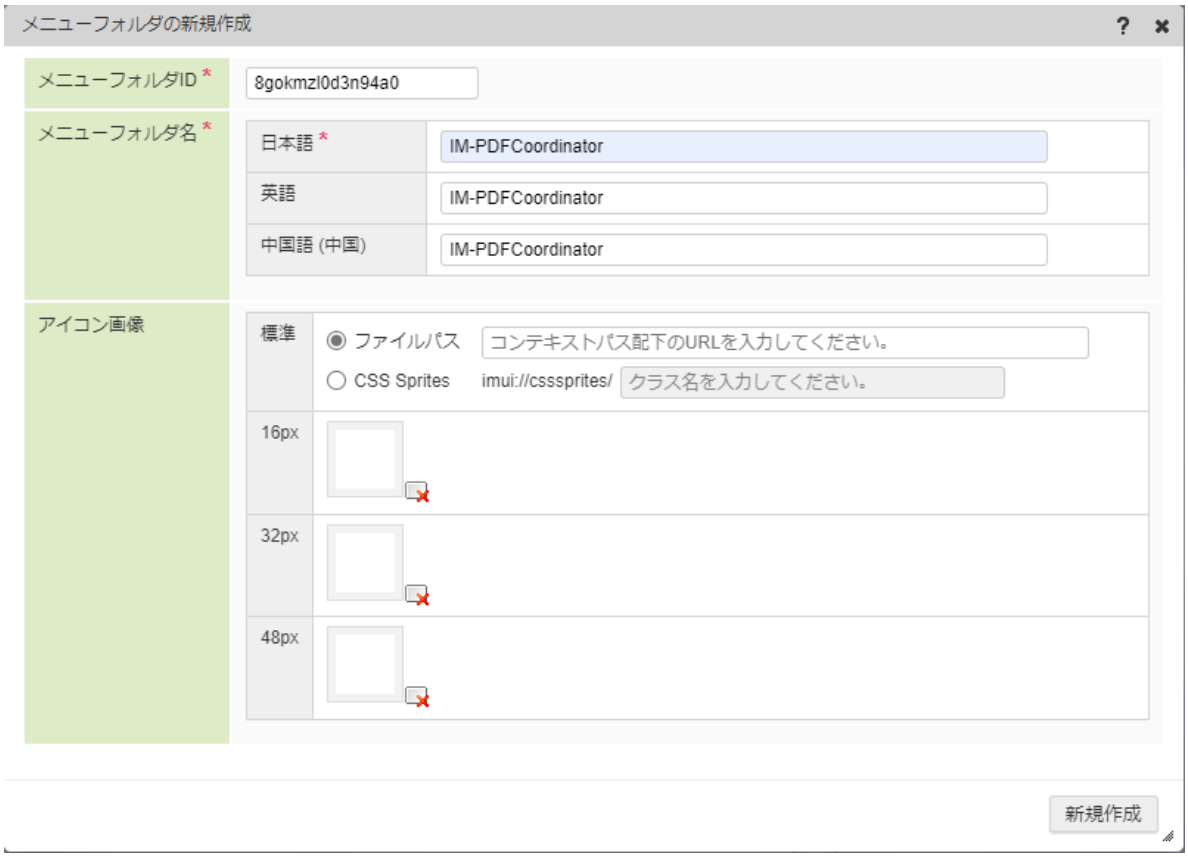
機能	リソースURI
ページ機能（Page）	service://pdfc/combine
マージ機能（Merge）	service://pdfc/merge
エディット機能（Edit）	service://pdfc/edit



9. 作成したリソースグループで「認証済みユーザ」に「全て許可」を付与します。

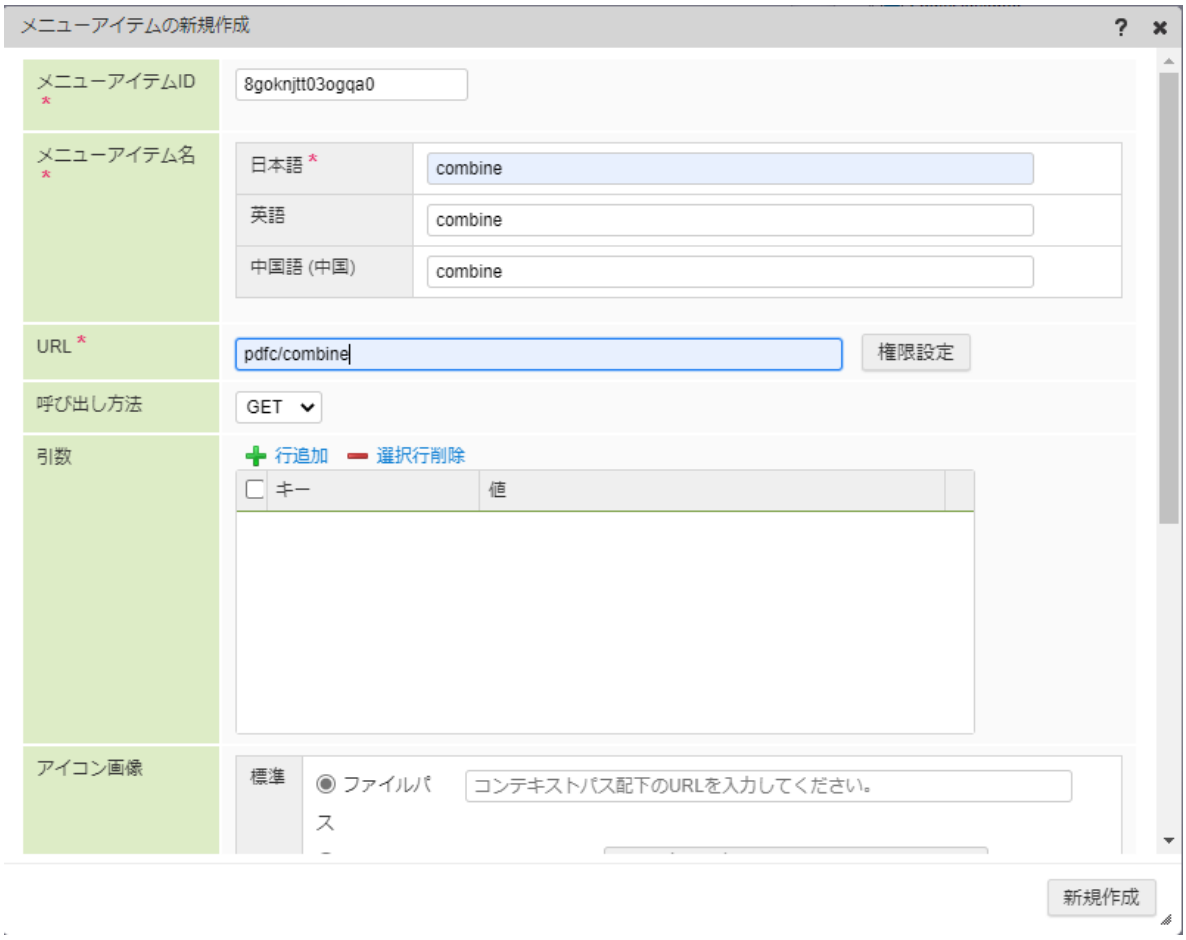
メニュー設定

1. テナント管理者でログインし、次のメニューを設定します。
2. [テナント管理]-[メニュー]画面を開きます。
3. フォルダを作成します。

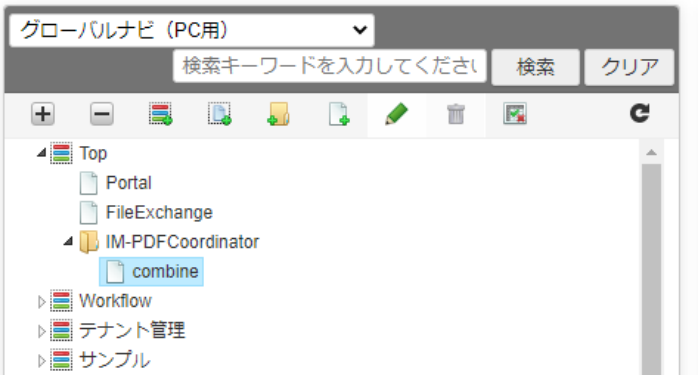


4. 作成したフォルダの下にメニューアイテムを新規作成し、メニューアイテム名、および、URLに次の値を設定します。

機能	メニューアイテム名	URL
ページ機能（Page）	combine	pdfc/combine
マージ機能（Merge）	merge	pdfc/merge
エディット機能（Edit）	edit	pdfc/edit



5. メニュー設定は完了です。



プログラムの実行と確認

メニューで次のアイテムを選択することにより、作成した画面が表示されます。

機能	メニューアイテム名
ページ機能（Page）	combine
マージ機能（Merge）	merge
エディット機能（Edit）	edit

画面上で処理対象ファイルをアップロードすることで、編集/加工処理のプログラムが実行され、処理されたPDFファイルがダウンロードされます。

PDFビューア（ Adobe Acrobat Reader など）で処理後のファイルが正しく表示されることを確認し、このチュートリアルは完了です。

エラーコード

目次

エラーコード一覧

ステータスコード	エラー内容
-1	編集元PDFのパスが指定されていません。
-2	編集元PDFが見つかりません。
-3	編集元PDFに問題があるため処理できません。
-4	出力先PDFのパスが指定されていません。
-5	一時フォルダの作成に失敗しました。
-6	一時ファイルの作成に失敗しました。
-7	ypdfcombコマンドの実行に失敗しました。
-8	致命的エラーが発生した為、除外リストに追加しました。PDFメイクアップでエラーが発生しました。
-9	Javascriptファイルの編集に失敗しました。
-10	ファイルの作成に失敗しました。
-11	リソースのコピーに失敗しました。
-12	利用可能なサーバが見つかりません。
-13	PDFセキュリティ強化サービスでエラーが発生しました。
-100	システムエラーが発生しました。

コラム

PDFエンジン部分のエラーについては、スタート→YSS PDF Makeup→ドキュメント→エラー一覧 から確認ください。

注意

「Password error」と表示された場合でも、パスワードに起因するエラーではないケースがあります。

上記のエラーが発生する主なPDFファイルの形式やケースは、次の通りです。

- パスワードが付与され、暗号化されているPDFファイル
- 電子署名やタイムスタンプ等が付与されているPDFファイル
- Adobe Acrobat の拡張機能等が使用されているPDFファイル
- 内部構造が一部破損しているPDFファイル
- PDFの規格に準拠していないPDFファイル

サポート

Webにて当製品に対するサポート、および、技術情報を公開しています。

当製品に関して不明な点などがある場合、情報検索、または、「 intra-mart サポートサイト 」に問い合わせしてください。