

PDF テンプレートエンジン

Field Reports for Windows

ユーザーズ・マニュアル

第 1.53 版

2015 年 8 月 1 日

合同会社 フィールドワークス

Field Works, LLC.

まえがき

本書では、PDF テンプレートエンジン Field Reports（以降、Field Reports と表記します）のインストール手順、Field Reports を利用したプログラムの作成手順および作成上の注意事項について説明します。また、レンダリングパラメータの書式および API について詳細に解説します。

販売と保守について

ライセンスのご購入

Field Reports のライセンスのご購入は、以下 Web サイトよりお願いします。

<http://www.field-works.co.jp/>

エラーや不具合

エラーや不具合を発見した場合、あるいは改善要望などがございましたら、下記までご連絡ください。

support@field-works.co.jp

ご注意

本書について

1. 本書の内容の一部または全部を無断で転載することはお断りします。
2. 本書の内容は、将来予告なしに変更することがあります。
3. 本書の作成にあたっては正確な記述に努めましたが、本書に基づく運用結果について、合同会社フィールドワークスは責任を負いかねますのでご了承ください。

版権について

すべての権利は、合同会社フィールドワークスに属しています。書面による同意なしに本書の内容を複製・改変および翻訳することを禁じます。

Copyright © 2011–2015 Field Works, LLC All rights reserved.

商標について

- Adobe, Acrobat, Adobe PDF, Adobe Reader は、Adobe 社の登録商標です。
- Mac, Mac OS は、Apple 社の登録商標です。
- Microsoft, Windows, Windows XP, Windows Vista, Windows 7, Windows 8, Microsoft .NET, Visual Studio, Microsoft Word, Excel は、Microsoft Corporation の登録商標です。
- Linux は、Linus Torvalds 氏の登録商標です。
- その他、本書に掲載されている会社名・製品名は、各社の商標または登録商標です。
- 本書では、® および™ を明記しておりません。

参考文献

1. アドビシステムズ著 ドキュメントシステム訳 (2001) 『PDF リファレンス 第2版 Adobe Portable Document Format Version 1.3』ピアソン・エデュケーション
2. Adobe Systems. (2004). *PDF Reference fifth edition: Adobe Portable Document Format Version 1.6.*

改版履歴

2011 年 2 月 25 日	第 1.0 版	新規リリース
2011 年 4 月 6 日	第 1.1 版	Perl Bridge, OCaml I/F を追加
2011 年 7 月 1 日	第 1.2 版	PHP Bridge を追加
2011 年 9 月 21 日	第 1.3 版	Windows 版追加に伴い OS 毎にマニュアルを分割 Java Bridge, .NET Framework Bridge を追加, C 言語 I/F を変更
2012 年 1 月 30 日	第 1.4 版	フォント埋込機能, 縦組テキスト機能他を追加
2013 年 8 月 23 日	第 1.5 版	第 3 章, 4 章の構成を変更 Windows 64bit 対応を追加 crop-box, bleed-box, trim-box, art-box エントリを追加
2015 年 8 月 1 日	第 1.53 版	年間ライセンスに関する記述を追加

目次

第 1 章	はじめに	1
1.1	Field Reports とは	1
1.1.1	主な特長	1
1.1.2	製品ラインナップ	2
1.1.3	動作環境	2
1.2	PDF 帳票作成手順	3
1.2.1	PDF テンプレートの作成	3
1.2.2	レンダリングパラメータの作成 (Python 編)	6
1.2.3	レンダリングパラメータの作成 (コマンドライン編)	7
1.2.4	PDF 帳票の生成	8
第 2 章	インストール	9
2.1	インストールの概要	9
2.1.1	Field Reports の構成	9
2.1.2	インストール媒体のファイル構成	9
2.2	Field Reports 本体のインストール	11
2.2.1	インストーラの実行	11
2.2.2	環境変数の設定	11
2.2.3	初期設定ファイルについて	12
2.2.4	アンインストール方法	12
2.3	言語 Bridge のインストール	13
2.3.1	Python Bridge	13
2.3.2	Ruby Bridge	13
2.3.3	Perl Bridge	14
2.3.4	PHP Bridge	14
2.3.5	.NET Framework Bridge	15
2.3.6	Java Bridge	16
2.4	ライセンス認証	18
2.4.1	ライセンスキーの種類	18
2.4.2	開発ライセンスキー	18
2.4.3	運用ライセンスキー	18
2.4.4	年間ライセンスキー	19

2.4.5 ライセンスキーの登録	19
第3章 帳票定義	21
3.1 帳票定義の概要	21
3.1.1 PDF テンプレート	22
3.1.2 レンダリングパラメータ	24
3.2 テンプレート要素	25
3.2.1 単票の場合のテンプレート指定	25
3.2.2 複合帳票の場合のテンプレート指定 (Standard 版のみ)	25
3.2.3 連続帳票の場合のテンプレート指定 (Standard 版のみ)	26
3.2.4 複合帳票+連続帳票でのテンプレート指定 (Standard 版のみ)	27
3.3 コンテキスト要素	28
3.3.1 単票でのフィールド値の指定	28
3.3.2 複合帳票でのフィールド値の指定	29
3.3.3 連続帳票でのフィールド値の指定	30
3.4 スタイル要素	31
3.4.1 スタイル指定の例	31
3.4.2 セレクタとは	31
3.5 リソース要素	32
3.5.1 フォントリソース	32
3.5.2 画像リソース	33
3.5.3 リソース URL 指定について	34
3.5.4 リソースのキャッシュについて	34
第4章 帳票生成	35
4.1 帳票生成処理の概要	35
4.1.1 テーブル分割処理 (Standard 版のみ)	35
4.1.2 レンダリング処理	35
4.2 テーブル分割処理の詳細 (Standard 版のみ)	38
4.3 レンダリング処理の詳細	40
4.3.1 テキスト・フィールドの外観生成	40
4.3.2 ボタン・フィールドの外観生成	41
4.3.3 境界線の外観生成	42
4.3.4 回転角度	43
4.3.5 座標変換	43
4.3.6 透明度	44
4.3.7 ブレンドモード	45
4.4 組版処理	47
4.4.1 行分割	47
4.4.2 ハイフネーション処理	48

4.4.3	禁則処理	49
4.4.4	割付け処理	50
4.4.5	縦組みテキスト	51
4.5	拡張漢字の利用	53
4.5.1	追加面に格納された Unicode 文字の指定	53
4.5.2	異体字セレクタによる異体字の指定	54
4.5.3	グリフ直接指定	55
第 5 章	レンダリングパラメータ	58
5.1	基本データ	58
5.1.1	Python から利用する場合	58
5.1.2	Ruby から利用する場合	59
5.1.3	Perl から利用する場合	59
5.1.4	PHP から利用する場合	59
5.1.5	JSON で記述する場合	60
5.2	共通データ構造	61
5.2.1	日付／時刻文字列	61
5.2.2	色指定	61
5.2.3	URL	62
5.3	セレクタ文字列	63
5.4	レンダリング辞書	64
5.5	template 要素	65
5.5.1	名前空間	66
5.5.2	PDF 要素	66
5.6	resources 辞書	68
5.6.1	font 要素	68
5.6.2	image 要素	70
5.7	context 要素	71
5.8	style リスト	72
5.9	filed 要素	72
5.9.1	共通フィールド属性	73
5.9.2	テキストフィールド属性	76
5.9.3	ボタンフィールド属性	89
5.10	property 辞書	89
5.10.1	docinfo 辞書	90
5.10.2	encryption 辞書	91
5.10.3	viewer-preferences 辞書	92
5.11	環境変数	93
5.11.1	ユーザ定義環境変数	93
5.11.2	システム定義環境変数	94

5.12	settings 辞書	94
第 6 章	API リファレンス	95
6.1	Python Bridge	95
6.1.1	field.reports モジュール	95
6.2	Ruby Bridge	96
6.2.1	Field::Reports モジュール	96
6.3	Perl Bridge	96
6.3.1	Field::Reports モジュール	96
6.4	PHP Bridge	97
6.4.1	php_reports モジュール	97
6.5	Java Bridge	97
6.5.1	jp.co.field_works.Reports クラス	97
6.6	.NET Framework Bridge	98
6.6.1	Field.Reports クラス	98
6.6.2	COM コンポーネント	99
6.7	コマンドライン I/F	100
6.7.1	reports コマンド	100
6.8	C I/F	100
6.8.1	API 一覧	100
6.8.2	caml_value 型について	102
6.8.3	コールバック関数について	102
6.9	OCaml I/F	104
6.9.1	Field.Reports モジュール	104
付録 A	言語 Bridge のビルト手順	105
A.1	前提条件	105
A.2	Python	105
A.2.1	準備	105
A.2.2	環境変数の設定	105
A.2.3	ビルト手順	106
A.2.4	インストール手順	106
A.2.5	アンインストール手順	106
A.3	Ruby	106
A.3.1	準備	106
A.3.2	DevKit の初期設定	107
A.3.3	環境変数の設定	107
A.3.4	ビルトならびにインストールの手順	107
A.3.5	アンインストール手順	107
A.4	Perl	107

A.4.1	準備	107
A.4.2	環境変数の設定	108
A.4.3	ビルド手順	108
A.4.4	インストール手順	108
A.4.5	アンインストール手順	108
A.5	PHP	109
A.5.1	準備	109
A.5.2	ビルド手順	110
A.5.3	インストール手順	110
A.5.4	アンインストール手順	110
A.6	Java	111
A.6.1	準備	111
A.6.2	環境変数の設定	111
A.6.3	ビルド手順	111
A.6.4	インストール手順	111
A.6.5	アンインストール手順	112
付録 B	依存ライブラリ	113
B.1	OCaml ライブラリ	113
B.2	C ライブラリ	114

第 1 章

はじめに

1.1 Field Reports とは

Field Reports は、マルチプラットフォーム／マルチ言語対応の PDF テンプレートエンジンです。

業務システムでの帳票出力をはじめとして、チラシ・パンフレット・ダイレクトメール (DM) 等のバリアブル印刷など、さまざまな用途において柔軟にお使いいただくことができます。

固定の下絵に対し、個別の可変データを配置した PDF ドキュメント（以後「帳票」と総称する）を容易に生成することができます。

下絵（PDF テンプレート）は Excel, Word 等のオフィスソフトと Adobe Acrobat で作成し、テキスト・画像等の可変データは JSON ベースのレンダリングパラメータとして記述します。Field Reports は、PDF テンプレートとレンダリングパラメータを合成（レンダリング）して、新たな PDF 帳票を生成します。

1.1.1 主な特長

- マルチプラットフォーム対応

Linux, Windows, Mac OS X の各プラットフォーム上で動作します。

- マルチ言語対応

Python, Ruby, Perl, PHP などの LL 言語および JVM, .NET Framework (Windows 版のみ) プラットフォーム上で動作するプログラミング言語をサポートします。特に LL 言語においては、ネイティブのデータ構造を用いてシームレスにレンダリングパラメータを記述することができます。

- テンプレート方式

PDF テンプレートに配置された（フォーム）フィールドをプレースホルダとして、レンダリングパラメータから取得した値を流し込みます。単にフィールドに値をセットするだけではなく、コンテンツストリームと結合させ、外観を恒久化することができます。

- オフィスソフトと Adobe Acrobat を用いた帳票設計

Excel, Word 等のオフィスソフトを利用して、イメージ通りの帳票を作成できます。

- 高度な組版機能

「均等割付」「禁則処理」「ハイフネーション処理」「縦組み」など高度なテキスト整形機能を有します。

- PDF1.6 に準拠

PDF version 1.6 に準拠した PDF 帳票を出力します。

RC4, AES 暗号化、文書情報の設定、「開き方」の設定、データ圧縮、WEB 表示用に最適化など PDF の各種属性を設定できます。

1.1.2 製品ラインナップ

Field Reports の製品ラインナップを表 1.1 に示します。

表 1.1: 製品ラインナップ

エディション	説明
Standard	Field Reports の基本パッケージとなります。 ページ構成を動的に変更可能な複合帳票、ページ数可変の連続帳票の作成が可能です。
Lite	出力できる帳票の種類が単票に限定されます。 ページ構成が固定の帳票作成をご利用いただけます。

1.1.3 動作環境

Field Reports は、表 1.2 の環境で動作します。

表 1.2: 動作環境

項目	条件
対象 OS	Windows XP SP3 以上 アーキテクチャ : x86, x86_64
必要なソフトウェア	Python 2.6 以上 Ruby 1.8 以上 Perl 5.8 以上 (Strawberry Perl 推奨。Perl 6 には未対応) PHP 5.1 以上 Java SE 6 以上 Adobe Acrobat Professional ^{*1}
ハードウェアスペック	CPU : お使いの OS が推奨する環境以上 メモリ : 同上 HDD : 200MB 以上

その他の環境での動作報告については、弊社 Web サイト (<http://www.field-works.co.jp/>) でご確認ください。

依存ライブラリの詳細については、B を参照してください。

^{*1} PDF にフィールドを配置する必要があるため、Professional 版以上が必要になります。

1.2 PDF 帳票作成手順

写真付き料理レシピの帳票作成を題材として、PDF 帳票を生成するまでの手順を説明します。

1.2.1 PDF テンプレートの作成

最初に、Microsoft Word, Excel などのオフィスソフトを使って、PDF テンプレートの下絵となる文書を作成します。その文書を、Adobe Acrobat などのツールを用いて、PDF ファイルに変換します。

今回は、オフィスソフトとして OpenOffice.org の Calc を使用しましたので（図 1.1），PDF ファイルへの変換まで OpenOffice.org で行うことができます（図 1.2）。



図 1.1: 下絵の作成



図 1.2: PDF への変換

次に、作成した PDF ファイルを Adobe Acrobat などの PDF 編集が可能なアプリケーションで開き、フィールドを配置します。フィールド名は、後でフィールドを参照する際に使用しますので、わかりやすい名前をつけてください。フォント・フォントサイズ・表示色などの表示属性もこの時点では設定します（帳票生成時に動的に変更することも可能です）。

図 1.3 は、Adobe Acrobat を使ってフィールドの配置を行っている様子です。テーブル形式のフィールドを作成する際には、「複数のフィールドを配置...」を使用すると便利です。

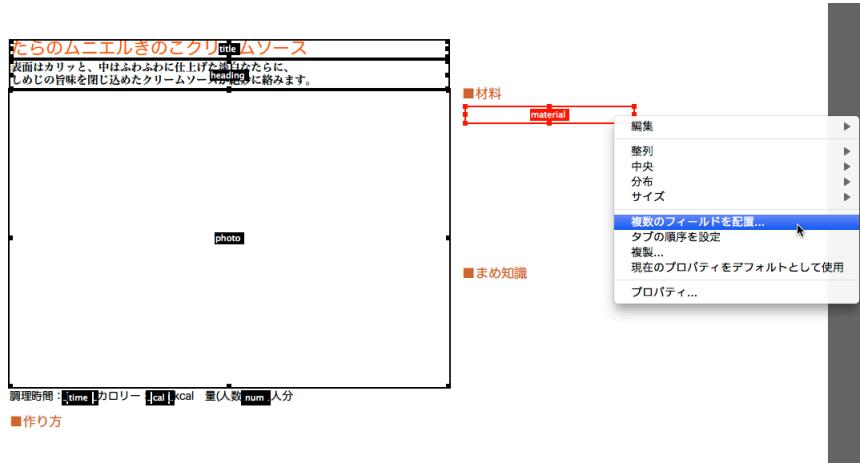


図 1.3: フィールドの配置

すべてのフィールドの配置が終わったら、保存します（図 3.2）。ここでは、ファイル名を“recipe.pdf”としました。

ここまでで、PDF テンプレートの作成は完了です。



図 1.4: フィールドの配置が完了した状態

1.2.2 レンダリングパラメータの作成（Python 編）

レンダリングパラメータを Python プログラムで記述します。レンダリングパラメータには、先ほど作成した PDF テンプレートのパス名と PDF テンプレートに配置したフィールドに設定する値を辞書オブジェクトとして記述します。

ここでは、作成したプログラムを “recipe.py” というファイル名で保存するものとします。エンコーディングは、UTF-8 とします。画像ファイル “meuniere_photo.jpg” もレンダリングパラメータと同じディレクトリに用意しておきます。

recipe.py

```
#!/usr/bin/env python
# coding: utf-8

import field.reports

param = {
    "template": "./recipe.pdf",

    "context": {
        "title": {"value": u"たらのムニエルきのこクリームソース", "color": [204, 102, 51]},
        "heading": u"""表面はカリッと、中はふわふわに仕上げた淡白なたらに、
        しめじの旨味を閉じ込めたクリームソースが絶妙に絡みます."""
    },
    "photo": {"icon": "./recipe_photo.jpg"},
    "time": u"15 分",
    "cal": 310,
    "num": "2",
    "material": [
        [u"たら（切身）", u"2 枚"],
        [u"しめじ", u"1 パック"],
        [u"小麦粉（強力粉）", u"適量"],
        [u"生クリーム", u"50cc"],
        [u"卵黄", u"1 個"],
        [u"バター", "20g"],
        [u"塩", u"適量"],
        [u"白ワイン", u"大さじ 2"],
        [u"チャーピル", u"適量"]
    ],
    "procedure": [
        u"(1) たらの表面に塩をふり、小麦粉（強力粉）をまぶし、バター 50g を入れフライパンで、\
        皮の方から焼く。",
        u"(2) 焼き上がったら皿に写し、フライパンの余分な油をとる。",
        u"(3) 残りのバター・白ワイン・しめじを炒め、生クリーム・卵黄を加え軽く火を通す。",
        u"(4)(2) に (3) をかけ、チャーピルを飾る。"
    ],
    "tips": u"昔、風車を回して小麦粉を作っている人をフランス語で「ムニエ」と呼んでいました。\
    小麦粉を使った料理「ムニエル」は、この「ムニエ」が由来しているそうです。"
}

field.reports.render(param, "meuniere.pdf")
```

1.2.3 レンダリングパラメータの作成（コマンドライン編）

レンダリングパラメータを JSON 形式のファイルとして作成します。

作成したレンダリングパラメータは，“recipe.json” というファイル名で保存します。エンコーディングは、UTF-8 とします。画像ファイル “meuniere_photo.jpg” もレンダリングパラメータと同一のディレクトリに配置しておきます。

```
----- recipe.json -----  
  
{  
    "template": "./recipe.pdf",  
  
    "context": {  
        "title": {"value": "たらのムニエルきのこクリームソース", "color": [204, 102, 51]},  
        "heading": "表面はカリッと、中はふわふわに仕上げた淡白なたらに、\nしめ  
じの旨味を閉じ込めたクリームソースが絶妙に絡みます。",  
        "photo": {"icon": "./recipe_photo.jpg"},  
        "time": "15 分",  
        "cal": 310,  
        "num": "2",  
        "material": [  
            ["たら (切身)", "2 枚"],  
            ["しめじ", "1 パック"],  
            ["小麦粉 (強力粉)", "適量"],  
            ["生クリーム", "50cc"],  
            ["卵黄", "1 個"],  
            ["バター", "20g"],  
            ["塩", "適量"],  
            ["白ワイン", "大さじ 2"],  
            ["チャーピル", "適量"]  
        ],  
        "procedure": [  
            "(1) たらの表面に塩をふり、小麦粉 (強力粉) をまぶし、バター 50g を入れフライパンで、皮の方から焼く。",  
            "(2) 焼き上がったら皿に写し、フライパンの余分な油をとる。",  
            "(3) 残りのバター・白ワイン・しめじを炒め、生クリーム・卵黄を加え軽く火を通す。",  
            "(4)(2) に (3) をかけ、チャーピルを飾る。"  
        ],  
        "tips": "昔、風車を回して小麦粉を作っている人をフランス語で「ムニエ」と  
呼んでいました。小麦粉を使った料理「ムニエル」は、この「ムニエ」が  
由来しているそうです。"  
    }  
}
```

1.2.4 PDF 帳票の生成

Python プログラムの実行

“recipe.pdf” と “recipe.py” が存在するディレクトリで、以下のコマンドを実行します。生成された PDF 帳票は、“meuniere.pdf” に保存されます。

```
$ python recipe.py
```

reports コマンドの実行

“recipe.pdf” と “recipe.json” が存在するディレクトリで、以下のコマンドを実行します。生成された PDF 帳票は、“meuniere.pdf” に保存されます。

```
$ reports recipe.json meuniere.pdf
```

完成イメージ

完成した PDF 帳票のイメージを図 1.5 に示します。

たらのムニエルきのこクリームソース
表面はカリッと、中はふわふわに仕上げた淡白なたらに、しめじの旨味を閉じ込めたクリームソースが絶妙に絡みます。



■材料

たら (切身)	2枚
しめじ	1パック
小麦粉（強力粉）	適量
生クリーム	50cc
卵黄	1個
バター	20g
塩	適量
白ワイン	大さじ2
チャーピル	適量

■まめ知識
昔、風車を回して小麦粉を作っている人をフランス語で「ムニエ」と呼んでいました。小麦粉を使った料理「ムニエル」は、この「ムニエ」が由来しているそうです。

調理時間：15分 カロリー：310kcal 量(人数)： 2人分

■作り方

- (1)たらの表面に塩をふり、小麦粉(強力粉)をまぶし、バター50gを入れフライパンで、皮の方から焼く。
- (2)焼き上がったら皿に写し、フライパンの余分な油をとる。
- (3)残りのバター・白ワイン・しめじを炒め、生クリーム・卵黄を加え軽く火を通す。
- (4)(2)に(3)をかけ、チャーピルを飾る。

図 1.5: 生成された PDF 帳票

第 2 章

インストール

2.1 インストールの概要

2.1.1 Field Reports の構成

Field Reports は、表 2.1 の要素により構成されています。

表 2.1: ソフトウェアの構成

分類	内訳
Field Reports 本体	コマンドラインプログラム ダイナミックリンクライブラリ (DLL) .NET Framework ライブラリアセンブル ヘッダファイル
言語 Bridge	Python Bridge (ソース+バイナリ提供 : 2.6, 2.7, 3.3 / 32, 64 ビット) Ruby Bridge (ソース提供) Perl Bridge (ソース提供) PHP Bridge (ソース+バイナリ提供 : 5.3, 5.4 / 32 ビット) Java Bridge (ソース+バイナリ提供 : 1.6 / 32, 64 ビット) .NET Framework Bridge (バイナリ提供 : .Net Framework 4.0)

最初に Field Reports 本体のインストールを行い、次に必要なプログラミング言語用の Bridge のインストールを行います。

2.1.2 インストール媒体のファイル構成

インストール媒体は ZIP 形式で圧縮されています。適切なソフトウェアを用いて展開してください。

展開したインストール媒体は、以下のファイル構成となっています。“1.x” または “1.x.x” は、Field Reports のバージョン番号を示します（以下同様）。

```
reports-1.x.x-windows
├── c/
│   └── README.txt
├── dotNET/
│   ├── README.txt
│   └── reports-1.x.x.zip
├── java/
│   ├── 1.6/
│   │   └── README.txt
│   └── jni_reports-1.x.x.tar.gz
├── ocaml/
│   ├── 3.12.1/
│   └── README.txt
├── perl/
│   ├── Field-Reports-1.x.x.tar.gz
│   └── README
├── php/
│   ├── x86/
│   │   └── README.txt
│   └── php_reports-1.x.x.zip
├── python/
│   ├── README.txt
│   ├── field.reports-1.x.win-amd64-py2.6.exe
│   ├── field.reports-1.x.win-amd64-py2.7.exe
│   ├── field.reports-1.x.win-amd64-py3.3.exe
│   ├── field.reports-1.x.win32-py2.6.exe
│   ├── field.reports-1.x.win32-py2.7.exe
│   ├── field.reports-1.x.win32-py3.3.exe
│   └── field.reports-1.x.zip
├── ruby/
│   ├── README.rdoc
│   └── reports-1.x.x.gem
├── samples/
├── LICENSE.txt
└── README.txt
```

reports-1.x.x-x64.msi
reports-1.x.x-x86.msi
users-man.pdf

2.2 Field Reports 本体のインストール

2.2.1 インストーラの実行

インストールファイル “reports-1.x.x-x64.msi” または “reports-1.x.x-x86.msi” を開いて、インストーラを実行させてください。

標準では “Program Files” (64 ビット OS に 32 ビット版をインストールした場合は “Program Files (x86)”) 以下のディレクトリに、コマンドライン・プログラムと DLL ファイルならびにヘッダファイルをインストールします。

```
C:\Program FieIs
└─ Field Works
    └─ Field Reports 1.x
        ├─ bin
        |   ├─ libcurl.dll
        |   ├─ libreports.dll
        |   ├─ Reports.dll
        |   ├─ reports.exe
        |   └─ zlib1.dll
        ├─ include
        |   └─ reports.h
        ├─ lib
        |   └─ libreports.lib
        ├─ LICENSE.txt
        ├─ README.txt
        └─ users-man.pdf
```

2.2.2 環境変数の設定

実行ファイルの検索パスに Field Reports の格納ディレクトリを追加してください¹。

```
> set PATH=%PROGRAMFILES%\Field Works\Field Reports 1.x\bin;%PATH%
```

コマンドライン・プログラムが起動することを確認してください。

¹ 64 ビット OS に 32 ビット版をインストールした場合は、“%PROGRAMFILES(x86)%” を設定してください。

```
> reports
Field Reports 1.x.x [x86/windows] (Trial)
usage: reports <subcommand> [options] [args]
Type 'reports <subcommand> ----help' for help on a specific subcommand.
```

Available subcommands:

reports create	create PDF report file.
reports info	inspect PDF information.
reports font	probe font file.
reports activate	generate check code for activation.
reports check	check initial config file.

Copyright 2011-2013 Field Works, LLC
<http://www.field-works.co.jp/>

2.2.3 初期設定ファイルについて

初期設定ファイルを所定の場所に置いて、コンピュータ内で共通のパラメータを記述しておくことができます。初期設定ファイルに記述された値は、個々の PDF 帳票を生成する際に指定するレンダリングパラメータの初期値として利用されます。

初期設定ファイルには主にライセンスキーを登録するための settings 辞書を記述しますが、style リスト・environ 辞書・property 辞書を記述することも可能です（context 要素が記述されていても無視します）。

初期設定ファイルのデフォルトの格納場所は OS により異なり、Windows XP の場合は、

%ALLUSERSPROFILE%\Application Data\Field Works\Field Reports 1.x\reports.conf

Vista 以降の場合は、

%ALLUSERSPROFILE%\Field Works\Field Reports 1.x\reports.conf

となります。

初期設定ファイルのパス名は、環境変数 REPORTS_CONFIG の設定により変更することができます。

2.2.4 アンインストール方法

コントロールパネルの「プログラムの追加と削除」を使用してください。

2.3 言語 Bridge のインストール

ここでは、主に 32 ビット OS をご使用の場合の手順を説明しますが、64 ビット OS をご使用の場合は、win32 または x86 を win64 または x64(x86_64) に適宜読み替えてください。

2.3.1 Python Bridge

Python 处理系について

ここでは、Python 公式サイト (<http://www.python.org/>) より入手した Windows 版バイナリ配布物を用いて Python がインストールしてあるものとして、拡張モジュールのインストール方法を説明します。

Python 拡張モジュールのインストール

Python 拡張モジュールのインストール媒体は実行形式になっています。使用する Python のバージョンに応じて、適切な exe ファイルを選択してしてください（以下は Python2.6 を使用する場合の実行例）。

```
> field.reports-1.x.win32-py2.6.exe
```

注意事項

- 使用する Python 处理系に合ったインストールファイルが用意されていない場合は、ソースからビルドしてください。拡張モジュールのビルドならびにインストールの手順については、付録 A.2 を参照してください。
- アンインストール手順については、付録 A.2 を参照してください。

動作確認

インストールした拡張モジュールが Python から使用できることを確認してください。

```
> python
>>> import field.reports
>>> field.reports.renders({})
"%PDF-1.6\n%\x80\x81\x82\x83\n ..."
```

2.3.2 Ruby Bridge

RubyGems のインストール

Ruby Bridge は、ソースのみのご提供となります。RubyGems のビルドならびにインストールの手順については、付録 A.3 を参照してください。

動作確認

インストールした拡張モジュールが ruby から使用できることを確認してください。

```
> irb
> require 'rubygems'
> require 'field/reports'
> Field::Reports.renders({})
=> "%PDF-1.6\n%\x80\x81\x82\x83\n ..."
```

2.3.3 Perl Bridge

Perl 拡張モジュールのインストール

Perl Bridge は、ソースのみのご提供となります。拡張モジュールのビルドならびにインストールの手順については、付録 A.4 を参照してください。

動作確認

インストールした拡張モジュールが perl から使用できることを確認してください。

```
> perl -e "use Field::Reports; print Field::Reports::renders({});"
%PDF-1.6\n%\x80\x81\x82\x83\n ..."
```

2.3.4 PHP Bridge

PHP 処理系について

ここでは、PHP 公式サイト (<http://windows.php.net/>) より入手した Windows 版バイナリ配布物 (x86) を用いて PHP がインストールされているものとして、拡張モジュールのインストール方法を説明します。

拡張モジュールのインストール

動作環境でアーキテクチャ、PHP のバージョンに対応したディレクトリにある “php_reports_ts.dll” または “php_reports.dll” ファイルをコピーします。PHP 処理系がスレッドセーフの場合は “php_reports_ts.dll” を、非スレッドセーフの場合は “php_reports.dll” を使用します（以下は、スレッドセーフの場合の実行例）。

```
> copy x86\<PHP バージョン>\php_reports_ts.dll <拡張モジュール格納場所>
```

既定の拡張モジュール格納場所が不明な場合は、コマンドプロンプトから以下のコマンドを実行して確認してください。

```
> php -i | find "extension_dir"
```

PHP 設定ファイル (php.ini) の編集

php.ini を編集して、次の行を追加してください。

php.ini

```
extension = php_reports_ts.dll
```

注意事項

- 使用する PHP 実行環境に合った dll ファイルが用意されていない場合は、ソースからビルドしてください。拡張モジュールのビルドならびにインストールの手順については、付録 A.5 を参照してください。
- 現在、x64 版 PHP での使用はサポートしておりません。
- アンインストール手順については、付録 A.5 を参照してください。

動作確認

インストールした拡張モジュールが PHP から使用できることを確認してください。

まず以下の内容のテキストファイルを作成し、test.php という名称で保存します。

test.php

```
<?php  
echo fr_renders("{}");  
?>
```

次に、以下のコマンドを実行します。

```
> php test.php  
%PDF-1.6\x80\x81\x82\x83\n ...
```

2.3.5 .NET Framework Bridge

インストール

Field Reports 本体と一緒にインストールされますので、追加でインストールする必要はありません。

“Program Files” 配下にインストールされた “Reports.dll” を.NET Framework ライブラリアセンブルとしてご利用いただけます。また、インストール時にレジストリに COM 登録されますので、COM 呼び出しも可能です。

注意事項

- .NET Framework Bridge は、.Net Framework 4.0 でビルドされています。
- 依存する.NET Framework のバージョンを変更したい場合は、添付ソースファイルを再ビルドして使用してください。

動作確認

WSH の JScript から COM 呼び出しができることを確認します。

以下の内容のテキストファイルを作成し，“test.js”というファイル名で保存します。

test.conf

```
var reports = new ActiveXObject("Field.Reports");
reports.Render("{}", "out.pdf");
```

保存したファイルを cscript コマンドの引数として与え、プログラムを実行します。

“out.pdf”というファイルが作成されれば成功です (out.pdf 自体は正常な PDF ではありませんので、Adobe Reader 等で開くことはできません)。

```
> cscript test.js
> type out.pdf
%PDF-1.6 ...
```

2.3.6 Java Bridge

Java 実行環境について

ここでは、<http://java.com/> より取得した JDK がインストールされているものとして、拡張ライブラリのインストール手順を説明します。

jar ファイルと JNI ライブラリのインストール

jar ファイルと JNI ライブラリを拡張ライブラリの格納場所にコピーしてください^{*2}。環境変数 JAVA_HOME には、JDK のインストール先ディレクトリが設定されているものとします。

```
> copy <JAVA バージョン>\reports.jar "%JAVA_HOME%\jre\lib\ext"
> copy <JAVA バージョン>\<アーキテクチャ>\Reports.dll "%JAVA_HOME%\jre\bin"
```

注意事項

- 使用する Java 実行環境に合った拡張ライブラリが用意されていない場合は、ソースからビルドしてください。拡張モジュールのビルドならびにインストールの手順については、付録 A.6 を参照してください。
- アンインストール手順については、付録 A.6 を参照してください。

動作確認

インストールした拡張モジュールが Java から使用できることを確認してください。

^{*2} <JAVA バージョン>は、JDK のバージョン番号に対応します。

```
> set PATH=%JAVA_HOME%\bin;%PATH%
> java jp.co.field_works.Reports
0000: 25 50 44 46 2D 31 2E 36  0A 25 80 81 82 83 0A 31  %PDF-1.6.%.....1
0010: 20 30 20 6F 62 6A 0A 3C  3C 20 2F 54 79 70 65 20  0 obj.<< /Type
0020: 2F 50 61 67 65 73 20 2F  4B 69 64 73 20 5B 20 5D  /Pages /Kids [ ]
...
...
```

2.4 ライセンス認証

Field Reports から試用版の制限を解除するためには、シリアル番号とライセンスキーを登録する必要があります。

2.4.1 ライセンスキーの種類

Field Reports のライセンス契約には、通常ライセンスと年間ライセンスのふたつの形態があり、発行されるライセンスキーそれぞれが異なります（表 2.2）。

表 2.2: ライセンスキーの種類

ライセンスの種類	キーの種類	使用期限	用途
通常ライセンス	開発ライセンスキー	1 年間	開発
	運用ライセンスキー	無期限	運用
年間ライセンス	年間ライセンスキー	1 年間	開発と運用

2.4.2 開発ライセンスキー

通常ライセンス契約でご購入頂いた場合にシリアル番号と共に発行されるライセンスキーです。

Field Reports を利用したプログラムを開発する際に、契約者所有の開発用コンピュータに登録してお使いいただくことができます。使用期間は限定されますが、ご使用頂くコンピュータの台数は問いません。

次年度以降の再発行について

バージョンアップ・保守などの理由により、次年度以降にも開発ライセンスキーが必要になった場合には、再発行が可能です。再発行をおこなうためには、保守サポート契約が有効である必要があります。

- 保守サポート契約が終了している場合には、再加入した上で申請してください。
- 再発行時の開発ライセンスキーの有効期限は、3ヶ月となります。
- 開発期間が3ヶ月を超える場合は、再度期間延長の申請を行ってください。

ライセンスキー申請窓口^{*1} で、再発行の手続きを行ってください。

2.4.3 運用ライセンスキー

本番稼動を行うコンピュータに対して発行されるライセンスキーです。

Field Reports を利用して開発したプログラムを本番運用する際に、契約した台数の運用コンピュータに登録してお使いいただすることができます。運用ライセンスキーは、1つのシリアル番号につき原則1回発行されます。1つの運用ライセンスキーは、1台のコンピュータ（搭載CPU数およびコア数は問いません）でお使い頂くことができます。

^{*1} <http://www.field-works.co.jp/サポート/ライセンスキー申請/>

新規発行手続き

運用ライセンスキーを発行する際には、チェックコード^{*2}を通知していただく必要があります。
実際に運用を行うコンピュータ上で、シリアル番号を引数に与えて以下のコマンドを実行してください。

```
> reports activate xx-xxxx-xxxx-xxxx
```

標準出力に出力されるテキストの内容をライセンスキー申請窓口^{*1}までご連絡ください。折り返し運用ライセンスキーをお送りいたします。

再発行について

運用ライセンスキーの発行は原則1回限りですが、以下のような理由でライセンスキーが無効になった場合に再発行いたします。ただし、保守サポートサービスの契約期間中である必要があります。

- 故障により、ハードウェアを変更した。
- ハードウェア構成を変更した。

ライセンスキー申請窓口^{*1}で、再発行の手続きを行ってください。

2.4.4 年間ライセンスキー

年間ライセンスを新規にご契約頂いた際にシリアル番号と共に年間ライセンスキーが発行されます。次年度以降の年度更新時には、新しい年間ライセンスキーが発行されます。

プログラム開発時には、契約者所有の開発用コンピュータに登録してお使いいただけます。運用時には、契約いただいた台数の本番稼働用コンピュータに登録してお使いいただけます。

2.4.5 ライセンスキーの登録

初期設定ファイル(2.2.3)に下記の書式でライセンス情報を追加してください（初期設定ファイルが存在しない場合は、テキストエディタ等で作成してください）。

reports.conf

```
{  
    "settings": {  
        "serial-number": "<シリアル番号>",  
        "auth-code": "<ライセンスキー>"  
    }  
}
```

確認のため、コマンドライン・プログラムを実行してください。

^{*2} チックコードは、シリアル番号とコンピュータのハードウェア情報を元に生成されるハッシュ値です。

```
> reports
usage: reports <subcommand> [options] [args]
Field Reports Standard x.x.x -- Field Reports command-line tool
...
```

バージョン番号から「(Trial)」の文字が消えていれば、ライセンスキーの登録は成功です。
「Standard」部分の表示は、エディションにより異なります。開発・年間ライセンスキーを登録した場合は、
使用期限が表示されます。

第3章

帳票定義

3.1 帳票定義の概要

Field Reports では、PDF テンプレートとレンダリングパラメータを用いて生成する PDF 帳票を定義します。PDF テンプレートで固定のデザインを規定し、レンダリングパラメータで可変データを指定します。

図 3.1 に PDF テンプレートとレンダリングパラメータを元に帳票を生成するまでの処理の流れを示します。

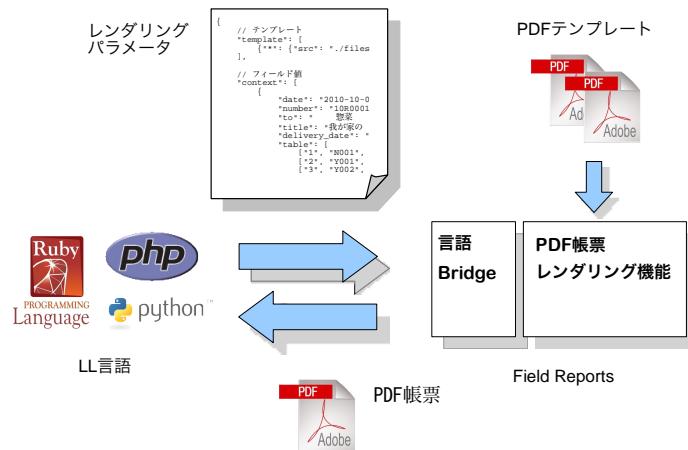


図 3.1: Field Reports 概念図

3.1.1 PDF テンプレート

テキスト・画像を表示したい位置に（フォーム）フィールドを配置した PDF ファイルを PDF テンプレートと呼びます。

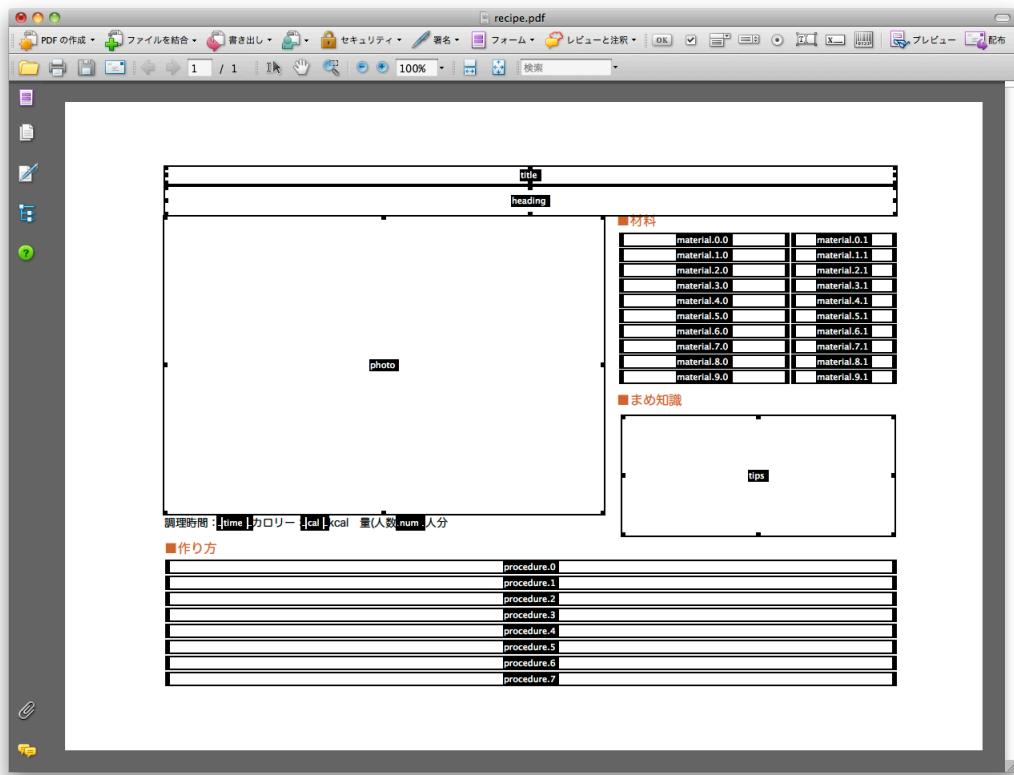


図 3.2: PDF テンプレートの例

フィールドとは

フィールドとは、PDF のページ上に配置された入力エリアです。フォームもしくはフォーム・フィールドとも呼ばれます。本書では、フィールドという呼称で統一しています。

フィールドの本来の用途は、ユーザにインタラクティブに値を入力させるためのものですが、Field Reports ではテキストや画像を流し込む先のプレースホルダとして利用しています。

フィールド名とは

PDF に配置されたフィールドは「フィールド名」により一意に識別することができます。

フィールド名をピリオドで区切ることで、フィールドの親子関係（階層構造）が表現できます。ルートから末端のフィールド名までをピリオドで区切って並べた形式のフィールド名を「完全修飾フィールド名」と呼び

ます。一方、完全修飾フィールド名の一部を構成するフィールド名を「部分フィールド名」と呼びます。

完全修飾フィールド名 → 部分フィールド名₁ . 部分フィールド名₂ 部分フィールド名_n

テーブル形式でのフィールド名

テーブル形式のデータを配置する際には、テーブルを構成するフィールド要素の一つ一つに、以下の形式でフィールド名が付けられることを想定しています。

1次元テーブル： テーブル名 . 行番号

2次元テーブル： テーブル名 . 行番号 . 列番号

ここで、行番号・列番号は0始まりの整数です。

基本フィールド属性とは

フィールド属性のうち、PDFの仕様に対応するものを基本フィールド属性と呼んでいます。Field Reportsで対応している基本フィールド属性の一覧を表3.1に示します。

表3.1: 基本フィールド属性

フィールド属性		テキスト	ボタン
値	テキスト	<input checked="" type="radio"/>	-
アイコン	画像	-	<input checked="" type="radio"/>
境界線と色	境界線の色・幅・スタイル、塗りつぶしの色	<input checked="" type="radio"/>	<input checked="" type="radio"/>
テキスト	フォント・サイズ	<input checked="" type="radio"/>	-
オプション	整列・複数行	<input checked="" type="radio"/>	-
座標	位置・幅・高さ・向き	<input checked="" type="radio"/>	<input checked="" type="radio"/>

これら以外の属性（例えば、Adobe Acrobatのフィールドのプロパティダイアログにおける「アクション」「フォーマット」「検証」「計算」に相当する属性など）は変更することはできません。また、テキスト・ボタン以外の種類のフィールドの属性を変更することもできません。

拡張フィールド属性とは

回転角度・透明度・ブレンドモードなど、Field Reportsが独自に拡張したフィールド属性を設定することができます（表3.2）。

表3.2: 拡張フィールド属性

フィールド属性		テキスト	ボタン
座標変換	回転角度、変換行列	<input checked="" type="radio"/>	<input checked="" type="radio"/>
重ねあわせ	透明度、ブレンドモード	<input checked="" type="radio"/>	<input checked="" type="radio"/>
余白調整	パディング	<input checked="" type="radio"/>	<input checked="" type="radio"/>
レイアウト調整	垂直方向整列、行の高さ、	<input checked="" type="radio"/>	-
組版処理	ハイフネーション、禁則処理、均等割、縦組み	<input checked="" type="radio"/>	-
拡張漢字	サロゲートペア、異体字セレクタ、グリフ直接指定	<input checked="" type="radio"/>	-
書式指定	数値書式、日付書式、文字参照	<input checked="" type="radio"/>	-

3.1.2 レンダリングパラメータ

レンダリングパラメータでは、ページの構成要素、フィールド名と可変データの対応、スタイル指定、リソース定義などを記述します。

表 3.3 にレンダリングパラメータの主な構成要素を示します。

表 3.3: レンダリングパラメータの構成要素

要素名	説明
テンプレート	名前空間を定義し、PDF テンプレートと関連付けます。
コンテキスト	フィールド名に設定する属性を 1 対 1 対応で指定します。
スタイル	複数のフィールドに対して、一括してフィールド属性を指定します。
リソース	フォント・画像リソースを定義します。

以降の節では、レンダリングパラメータの各構成要素について説明します。

3.2 テンプレート要素

テンプレート要素では、生成する PDF 帳票のページ構成を定義します。

3.2.1 単票の場合のテンプレート指定

1種類の PDF テンプレートを元に作成される帳票を単票と呼びます。

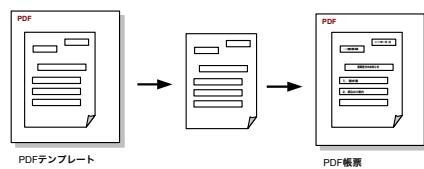


図 3.3: 単票

単票では、`template` 要素の値として、PDF テンプレートのパス名を直接指定します。

```
{  
    "template": "./mitumori.pdf"  
}
```

単票ではフィールド名の重複を考慮する必要がないので、名前空間を挿入する必要はありません。元々のフィールド名をそのまま使用します。

3.2.2 複合帳票の場合のテンプレート指定 (Standard 版のみ)

複数の PDF テンプレートを組み合わせて作成される帳票を複合帳票と呼びます (図 3.4)。

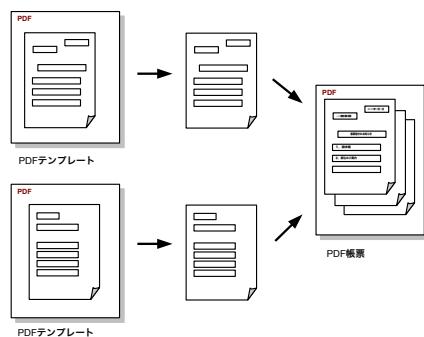


図 3.4: 複合帳票

複合帳票では、PDF テンプレートに任意の部分フィールド名を付けて名前空間を分けます。

部分フィールド名をキーにパス名を値とした辞書を作成して、出現順にリスト形式で並べます。

```
{
  "template": [
    {"header": "./hyousi.pdf"},
    {"body": "./mitumori.pdf"}
  ]
}
```

テンプレートに対応付けられた部分フィールド名は、名前空間の先頭に挿入されます（図 3.5）。例えば、`a.pdf`, `b.pdf` でそれぞれ「`title`」という同じ名称のフィールドが定義してあったとすると、名前空間 A と B をそれぞれ挿入することで、「`A.title`」「`B.title`」のように区別できるようになります。

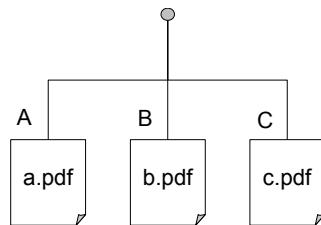


図 3.5: 複合帳票の名前空間

3.2.3 連続帳票の場合のテンプレート指定 (Standard 版のみ)

テーブル形式のデータの項目数によって、ページ数が変化する帳票を連続帳票と呼びます（図 3.6）。

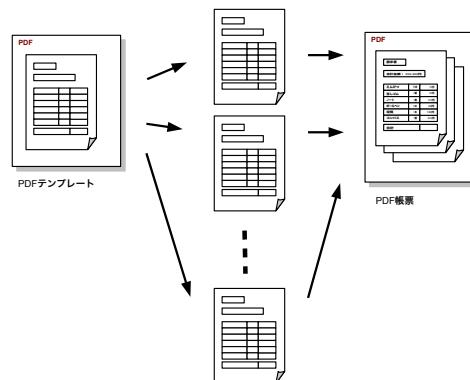


図 3.6: 連続帳票

連続帳票では、テンプレートの名前空間を “*” として、複合帳票と同様に `template` 要素を記述します。

ただし、連続帳票では PDF テンプレートのパス名以外に、テーブルの最大行数を指定する必要があります。以下の例では、テンプレートに関する情報を辞書形式で指定しています。

```
{
  "template": [
    {"*": {"src": "./mitumori.pdf", "rows": 10}}
  ]
}
```

連続帳票では、 $0, 1, 2, \dots$ のような 0 始まりの整数の名前空間が暗黙的に定義されますので、それらの名前空間を各ページに割り振ります図 3.7)。

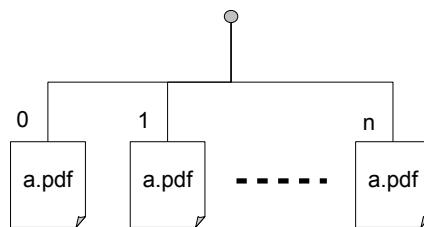


図 3.7: 連続帳票の名前空間

3.2.4 複合帳票 + 連続帳票でのテンプレート指定 (Standard 版のみ)

複合帳票の一部に連続帳票を含む形式の帳票も作成することができます (図 3.8)。

```
{
  "template": [
    {"header": "./hyousi.pdf"},
    {"body.*": {"src": "./mitumori.pdf", "rows": 10}}
  ]
}
```

上記の例では、連続帳票部分の部分フィールド名が “body.*” となり、連続帳票の例で示した “*” より名前空間の階層が 1 段深くなっています (図 3.8)。これは、コンテキスト要素のデータ構造上の都合のために必要となるものですが、このような書き方をすることで、複数の連続帳票を組み合わせることも可能になります。

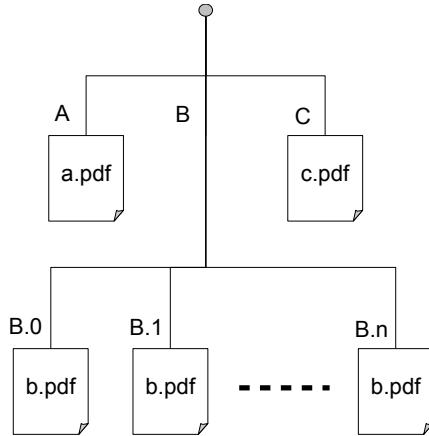


図 3.8: 複合帳票 + 連続帳票の名前空間

3.3 コンテキスト要素

コンテキスト要素では、PDF テンプレートに配置されたフィールドに設定するデータを宣言します。

フィールドに設定するデータとしては、主にテキスト（テキストフィールドの場合）や画像（ボタンフィールドの場合）などの「値」を指しますが、フォント・表示色・境界線色などの表示属性も同様に指定することができます。Field Reports では、フィールドに設定する値と表示属性を合わせて「フィールド属性」と呼んでいます。

各フィールドは「フィールド名」で一意に特定することができますので、コンテキスト要素では、フィールド名とフィールド属性の組を記述することになります。

3.3.1 単票でのフィールド値の指定

単票では、フィールド名をキーに、フィールド属性を値とした辞書形式のデータを context 要素直下の値として記述します。

フィールド名がテーブル形式の場合は、リストとして記述することもできます。

```
{
  "template": "./mitumori.pdf",
  "context": {
    "date": "平成 23 年 1 月 22 日",
    "number": "10R0001",
    "to": "△△△惣菜株式会社",
    "title": "肉じゃがの材料",
    "delivery_date": "平成 23 年 1 月 22 日",
    "delivery_place": "貴社指定場所",
    "payment_terms": "銀行振込",
    "expiration_date": "発行から 3 ヶ月以内",
    "stamp1": {"icon": "./stamp.png"},
    "table": [
      ...
    ]
  }
}
```

```

        ["1", "N001", "牛肉（切り落とし）", "200g", "250 円", "500 円"],
        ["2", "Y001", "じゃがいも（乱切り）", "3 個", "30 円", "90 円"],
        ["3", "Y002", "にんじん（乱切り）", "1 本", "40 円", "40 円"],
        ["4", "Y003", "たまねぎ（くし切り）", "1 個", "50 円", "50 円"],
        ["5", "Y004", "しらたき", "1 袋", "80 円", "80 円"],
        ["6", "Y005", "いんげん", "1 袋", "40 円", "40 円"]
    ],
    "sub_total": "800 円",
    "tax": "40 円",
    "total": "840 円"
}
}

```

3.3.2 複合帳票でのフィールド値の指定

複合帳票の場合は、`template` 要素で宣言した部分フィールド名を利用して、各帳票ごとに分離して記述します。

```

{
  "template": [
    {"header": "./hyousi.pdf"},
    {"body": "./mitumori.pdf"}
  ],
  "context": {
    "header": {
      "date": "${NOW}",
      "number": "10R0001",
      "to": "△△△惣菜株式会社",
      "title": "肉じゃがの材料",
      "delivery_date": "2011-03-01",
      "delivery_place": "貴社指定場所",
      "payment_terms": "銀行振込",
      "expiration_date": "発行から 3 ヶ月以内",
      "total": 840
    },
    "body": {
      "date": "${NOW}",
      "number": "10R0001",
      "to": "△△△惣菜株式会社",
      "title": "肉じゃがの材料",
      "delivery_date": "2011-03-01",
      "delivery_place": "貴社指定場所",
      "payment_terms": "銀行振込",
      "expiration_date": "発行から 3 ヶ月以内",
      "stamp1": {"icon": "./stamp.png"},
      "table": [
        ["1", "N001", "牛肉（切り落とし）", "200g", 250, 500],
        ["2", "Y001", "じゃがいも（乱切り）", "3 個", 30, 90],
        ["3", "Y002", "にんじん（乱切り）", "1 本", 40, 40],
        ["4", "Y003", "たまねぎ（くし切り）", "1 個", 50, 50],
        ["5", "Y004", "しらたき", "1 袋", 80, 80],
        ["6", "Y005", "いんげん", "1 袋", 40, 40]
      ],
      "sub_total": 800,
    }
  }
}

```

```
        "tax": 40,  
        "total": 840  
    },  
}
```

3.3.3 連続帳票でのフィールド値の指定

連続帳票の場合は、フィールド名と属性の組を記述した辞書を並べたリストを context 要素の値として宣言します。

以下の例では、1ページに配置できるテーブルの行数を最大 10 行としているのに対して、データは 14 行分あるので、1 ページには収まらないように見えますが、テーブル分割機能 (4.2) の働きにより自動的に 2 ページに分割されます。

```
{  
    "template": {"*": {"src": "./recipe.pdf", "rows": 10}},  
    "context": [  
        {  
            "title": "ビーフストロガノフ",  
            "num": "4",  
            "material": [  
                ["牛肉", "(バラ、肩ロースなど) 400 g"],  
                ["玉ねぎ", "1コ"],  
                ["マッシュルーム", "6コ"],  
                ["塩・こしょう", "少々"],  
                ["小麦粉", "大さじ2"],  
                ["バター", "20 g"],  
                ["トマトピューレー", "400 g (2びん)"],  
                ["水", "400 cc"],  
                ["コンソメ", "顆粒1袋 (5 g)"],  
                ["パブリカ", "小さじ1"],  
                ["塩", "小さじ半分"],  
                ["さとう", "大さじ2半"],  
                ["ブランデー", "大さじ1"],  
                ["生クリーム", "少々"]]  
        }  
    ]  
}
```

3.4 スタイル要素

context 要素ではフィールド名とフィールド属性の組を 1 体 1 対応で記述しますが、複数のフィールドの表示属性を一括して指定できると便利な場合もあります。そのような時には、style 要素を使用してください。

3.4.1 スタイル指定の例

以下にスタイル指定の例を示します。

```
"style": [
    {"header.date": {"datetime": "GGE 年 M 月 D 日"}},
    {"header.delivery_date": {"datetime": "GGE 年 M 月 D 日"}},
    {"header.total": {"format": "###,###円"}},
    {"header.sub_total": {"format": "###,###円"}},
    {"header.tax": {"format": "###,###円"}},
    {"header.table.*.[4:6]": {"format": "###,###円"}},
    {"body.*.date": {"datetime": "GGE 年 M 月 D 日"}},
    {"body.*.delivery_date": {"datetime": "GGE 年 M 月 D 日"}},
    {"body.*.total": {"format": "###,###円"}},
    {"body.*.sub_total": {"format": "###,###円"}},
    {"body.*.tax": {"format": "###,###円"}},
    {"body.*.table.*.[4:6]": {"format": "###,###円"}},
    {"body.[1:].stamp1": {"visible": False}},
    {"body.[1:].remark": {"visible": False}},
    {"body.*.table.[1::2]": {"background-color": [220, 220, 255]}}
]
```

3.4.2 セレクタとは

テンプレート上に配置されたフィールドの選択範囲を表現するための記法です。style 要素では、セレクタとフィールド要素の組としてスタイルを指定します。

完全修飾フィールド名によりフィールドを特定する方法と比較すると、以下の点が異なります。

- フィールド階層構造における共通の祖先を指定することで、その子孫のフィールドを一括して選択できます。
- ワイルドカードを使用することで、パターンマッチングに基づくフィールドの選択ができます。

例えば style 指定においては、以下のようなことが可能になります。

- 共通の親を持つフィールドのグループを一括して非表示にする。
- 連続帳票の 1 ページ目だけに「合計」欄を表示する。2 ページ目以降は「合計」欄を空欄にする。
- テーブルの偶数行と奇数行でテキスト表示色を変える。

セレクタの詳細については [5.3](#) を参照してください。

3.5 リソース要素

resource 要素では、フォントと画像のリソースを定義します。

3.5.1 フォントリソース

フォントリソース定義の例

以下にフォントリソース定義の例を示します。

ここで定義したフォント名は、font フィールド属性の値として使用します。

```
"resources": {
  "font": {
    "IPAmjMincho": {
      "src": "./ipamjm.ttf",
      "embed": true,
      "subset": true
    },
    "@KouzanBrushFontSousyoOTF": {
      "src": "./KouzanSoushoOTF.otf",
      "writing-mode": 1,
      "embed": true,
      "subset": true
    }
  }
}
```

対応フォント形式

Field Reports では、以下のフォント形式に対応しています。

- TrueType (拡張子 : *.ttf, *.ttc)
- OpenType (拡張子 : *.otf)

フォントの埋め込みについて

フォント埋め込みとは PDF ファイルにフォントの字形（グリフ）データを埋め込む機能です。PDF にフォントを埋め込むことで、字形を含むテキスト情報をより確実に伝達できるようになります。

例えば、毛筆体フォントなどのデザインを重視したフォント、ORC フォント・バーコードフォントのように正確な再現が必須となるフォントを使用する場合に特に役立ちます。

フォントを埋め込まない場合、PDF を受け取った相手のコンピュータに同じフォントがインストールされていないと、異なったフォント（代替フォント）で表示されます。適切な代替フォントが見つからない場合、文字位置のずれや文字化け等の問題が発生することがあります。

グリフデータを埋め込むことで PDF のファイルサイズが増えますが、「サブセット化」を有効にすることで、最小限の増加に留めることができます。

フォント埋込機能を有効にするには、embed フラグを有効にします。

おは、あけほの。やうやうふゝなりゆく「山さ
 は」うしゆりて眺だちたる雲のぬくたなびきたる。
 友は、友。月のぬはさらなり。夏もなほ。管の
 ゆへ飛び走ひたる。また、ただ一つ二つなど、
 ほのかにうち光りてゆくもをかし。おなじ降る
 もをかし。

村は、夕季。夕のさして、山の渓はいとら
 うなりたるに、鳥の宿どころへゆくとて、三つ
 四つ、二つ三つなど、飛びまぐせへあはれなり。
 まいて花などのむねたるがいとひそく見ゆるは、
 いとをかし。日入り果てて、風の音、虫の音な
 ど、はたい小べきにあらず。

夜は、つとめて。雪のはりたるはいとべきにも
 あらず。雪のいとふきも、またさらでも、いと
 空きに、火などさざ燃して、夜もて渡るも、い
 とつきづきし。夜になりて、ぬるべゆるびもて
 いけば、火桶の火も、ふき度がちになりて、わ
 ろし。

図 3.9: フォント埋込の例（毛筆体フォント）

3.5.2 画像リソース

画像リソース定義の例

以下に画像リソース定義の例を示します。

ここで定義した画像名称は、image フィールド属性の値として使用します。

```

"resources": {
  "image": {
    "photo1": "./kid0043-009.jpg",
    "photo2": "./kid0054-009.jpg",
    "pin": "./pin.pdf"
  }
}
  
```

対応画像形式

Field Reports では、以下の画像形式に対応しています。

- PNG
- BMP
- JPEG
- JPEG2000
- PDF

透過データ (α チャンネル) を持った PNG, BMP 形式に対応しています。

PNG 形式には以下の制限があります。

- インターレースモード形式には未対応です。
- α チャンネルを持ち、かつビット深度 4 以下の形式 (16 色以下のグレースケール・インデックスカラーなど) には未対応です。

3.5.3 リソース URL 指定について

画像・PDF テンプレートの取得先として、任意の URL を指定することができます。

- ローカルファイル
- data URI scheme
- URL

3.5.4 リソースのキャッシュについて

リソースは内部でキャッシュされますので、2 回目以降のロードは高速に処理できます。

キャッシュの階層構造

リソース・キャッシュは、3 階層の構造を持っています。

- 初期設定ファイルで定義されたリソース
- デフォルトパラメータとして定義されたリソース
- レンダリングパラメータで定義されたリソース

第一の階層には、初期設定ファイル (2.2.3) で定義されたリソースがキャッシュされます。初期設定ファイルで定義されたリソースは、Field Reports の実行モジュールがロードされてからアンロードされるまで保持されます。

第二の階層には、デフォルトのレンダリングパラメータで定義されたリソースがキャッシュされます。レンダリングパラメータのデフォルト値は、`set_defaults()` API (6) を使用して設定／変更することができます。デフォルト値を変更すると、過去の `set_defaults()` により作成されたキャッシュはクリアされます。

第三の階層には、レンダリングパラメータで定義されたリソースがキャッシュされます。この階層で保持されるリソースの寿命はトランザクション単位となります。

第 4 章

帳票生成

4.1 帳票生成処理の概要

図 4.1 は、Field Reports がレンダリングパラメータと PDF テンプレートを元に PDF 帳票を作成するまでの処理の流れを示したものです。

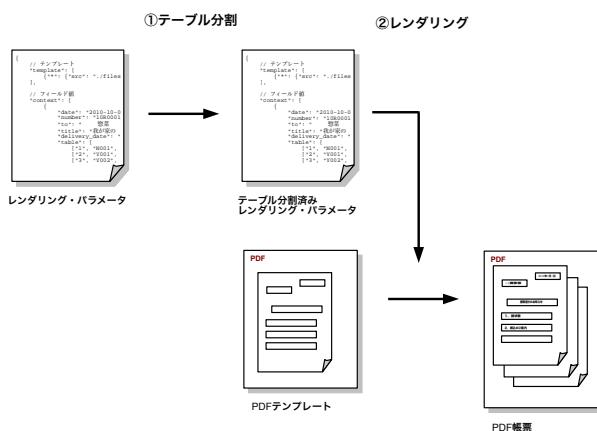


図 4.1: 処理の流れ (全体)

4.1.1 テーブル分割処理 (Standard 版のみ)

最初に前処理として、レンダリングパラメータにテーブル分割処理 (4.2) を施します。具体的には、レンダリングパラメータの中からテーブル形式のデータを探し出し、1 ページに収まる行数のテーブルに分割します。

ここまで処理で、連続帳票部分に必要なページ数が確定します。

4.1.2 レンダリング処理

次に、PDF テンプレートとテーブル分割処理済みのレンダリングパラメータを合成して、ひとつの PDF 帳票としてまとめます。

図 4.2 は、このレンダリング処理部分を詳細に図示したものです。

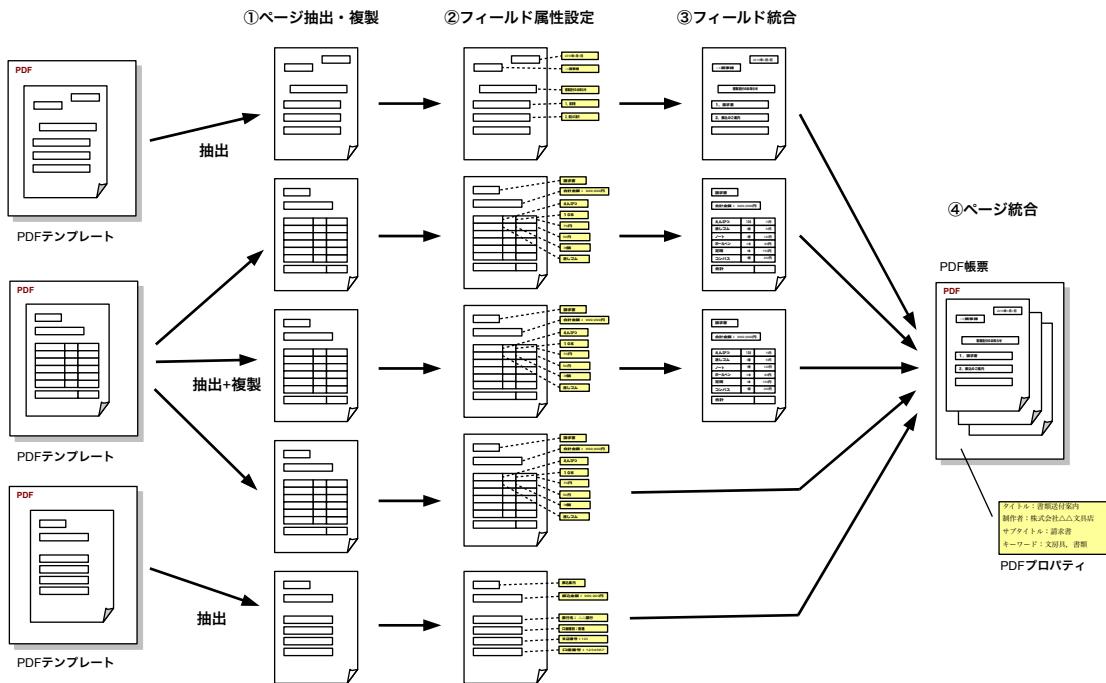


図 4.2: 処理の流れ（レンダリング部分）

ページ抽出・複製処理

レンダリングパラメータの定義にしたがって、PDF テンプレートから必要なページを抜き出します。連続帳票の場合は、さらに抽出したページをテーブル分割処理で確定したページ数分だけ複製します。

PDF テンプレートにあらかじめ配置されているフィールドは、この段階ではページ抽出・複製後もそのまま複製されます（ただし、複合帳票・連続帳票の場合は、フィールド名の名前空間の階層が深くなります）。

フィールドを動的に配置する場合は、この段階で新規にフィールドを生成します。

フィールド属性設定処理

各ページに配置されたフィールドに、テキスト・表示色・境界線などのフィールド属性を設定します。

フィールド属性は、コンテキスト要素もしくはスタイル要素としてレンダリングパラメータに記述されています。

フィールド属性には、PDF の仕様で規定された「基本フィールド属性」と、Field Reports が独自に拡張した「拡張フィールド属性」があります。

フィールド統合処理

フィールドに設定されたフィールド属性値を元にして、PDF 描画命令列（外観ストリーム）を生成します。

フィールド統合が有効な場合には、生成された外観ストリームはページのコンテンツ・ストリーム（テンプレートの下絵）と統合され、元のフィールドは削除されます。この処理によりフィールドの外観は恒久化され、ユーザによる変更ができなくなります。

フィールド統合が無効な場合はフィールドが残りますが、フィールドの新しい外観として、ここで生成された外観ストリームが設定されます。

フィールドに関連付けられた外観ストリームは、PDF ビューア（Acobe Reader 等）で開いた時の初期の外観として利用されますが、PDF ビューア依存の任意のタイミングで再構築されます。フィールド統合を行わない場合は、拡張フィールド属性を使用せずに基本フィールド属性の範囲内で使用してください。

ページ統合処理

各ページを統合して、ひとつの PDF 帳票にまとめます。

結合した PDF には、プロパティを設定することができます（表 4.1）。Adobe Acrobat の「プロパティ」に概ね対応するプロパティ項目を設定することができますが、実際に設定可能な項目については 5.10 を参照してください。

表 4.1: 設定可能なプロパティの一覧

種類	設定可能な値
文書情報	タイトル、作成者、サブタイトル、キーワード、アプリケーション、PDF 変換、作成日・更新日、XMP 形式メタデータ
セキュリティ	パスワードによるアクセス制限 暗号アルゴリズム：RC4(40 ビット/128 ビット)、AES(128 ビット)
開き方	レイアウトと倍率、ウインドウオプション、ユーザ・インターフェースオプション、表示領域、印刷領域、印刷ダイアログプリセット
その他	Flate 圧縮、WEB 表示用に最適化

4.2 テーブル分割処理の詳細 (Standard 版のみ)

レンダリングパラメータの中からテーブル形式のデータを探し出し、データの行数をカウントします。1ページに収まらないようであればテーブルを分割します。1ページ中の最大行数は、レンダリングパラメータの中に定数として記述されます。

以降に、1ページ10行でテーブル分割した場合の実行例を示します。テーブルの分割に連動して、他のフィールドも複製されることに注目してください。

一方、改ページの位置を手動で制御したい場合には、最初から分割後の様に複数ページに分けたデータを作成することで、自動テーブル分割処理を抑制することができます。

```
// テーブル分割前のレンダリングパラメータ
{
    "template": {"*": {"src": "./recipe.pdf", "rows": 10}},
    "context": [
        {
            "title": "ビーフストロガノフ",
            "num": "4",
            "material": [
                ["牛肉", "（バラ、肩ロースなど）400g"],
                ["玉ねぎ", "1コ"],
                ["マッシュルーム", "6コ"],
                ["塩・こしょう", "少々"],
                ["小麦粉", "大さじ2"],
                ["バター", "20g"],
                ["トマトピューレー", "400g (2びん)"],
                ["水", "400cc"],
                ["コンソメ", "顆粒1袋 (5g)"],
                ["パブリカ", "小さじ1"],
                ["塩", "小さじ半分"],
                ["さとう", "大さじ2半"],
                ["ブランデー", "大さじ1"],
                ["生クリーム", "少々"]
            ]
        }
    ]
}
```

```
// テーブル分割後のレンダリングパラメータ
{
    "template": {"*": {"src": "./recipe.pdf", "rows": 10}},
    "context": [
        {
            "title": "ビーフストロガノフ",
            "num": "4",
            "material": [
                ["牛肉", "（バラ、肩ロースなど）400g"],
                ["玉ねぎ", "1コ"],
                ["マッシュルーム", "6コ"],
                ["塩・こしょう", "少々"],
                ["小麦粉", "大さじ2"],
                ["バター", "20g"],
                ["トマトピューレー", "400g (2びん)"],
                ["水", "400cc"],
                ...
            ]
        }
    ]
}
```

```
        ["コンソメ", "顆粒 1 袋 (5 g)"],
        ["パプリカ", "小さじ 1"]
    },
{
    "title": "ビーフストロガノフ",
    "num": "4",
    "material": [
        ["塩", "小さじ半分"],
        ["さとう", "大さじ 2 半"],
        ["ブランデー", "大さじ 1"],
        ["生クリーム", "少々"]
    ]
}
```

4.3 レンダリング処理の詳細

4.3.1 テキスト・フィールドの外観生成

以下のフィールド属性を主なパラメータとして、テキストの外観を生成します。

- 値（テキスト）
- フォント
- フォントサイズ
- テキストの色
- 整列方法
- 行間隔
- 書式指定
- 複数行指定
- 座標（左下、右上）

フォント

システム定義フォント（[5.9.2](#)）またはリソースとして定義されたフォントが指定できます。

フォントサイズ

フォントサイズが自動（0pt）の場合には、テキストがフィールドの矩形に収まる最大のフォントサイズを自動計算します。ただし複数行指定が有効な場合は、10.5 ポイントに固定とします。

整列方法

左寄せ・中央寄せ・右寄せに加えて、均等割付の指定が可能です。



図 4.3: 横組みでのテキストの整列

縦組みでは、上寄せ・中央寄せ・下寄せ・均等割付の指定ができます。

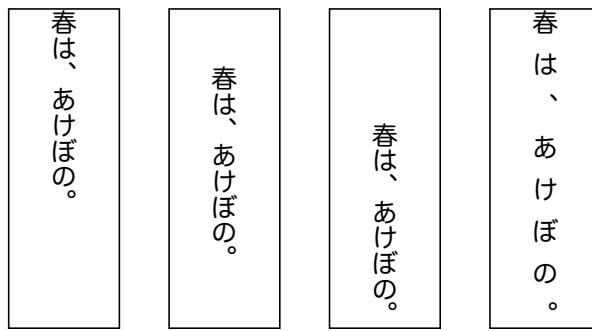


図 4.4: 縦組みでのテキストの整列

組版処理

複数行の指定が有効な場合には、フィールドの矩形幅に収まるように複数行に分割します。

行分割を行う際には、ハイフネーション処理・禁則処理を考慮した高度な割付処理を行います。詳細は、[4.4](#) を参照してください。

4.3.2 ボタン・フィールドの外観生成

以下のフィールド属性をパラメータとして、ボタン・フィールドの外観を生成します。

- アイコン（画像）
- 座標（左下、右上）

画像の縦横比を保存した上で、ボタン・フィールドの中央に画像を配置します。画像のサイズは、フィールドの矩形に収まる最大のサイズとします。「レイアウト」「アイコンの配置」等の属性が設定してあっても無視します。

また、透過情報（ α チャンネル）を持った PNG/BMP 画像に対応しています。



図 4.5: 透過画像の利用例

4.3.3 境界線の外観生成

以下のフィールド属性をパラメータとして、境界線と背景の塗りつぶしの外観を生成します。

- 境界線の幅
- 境界線の色
- 塗りつぶしの色
- 塗りつぶしのスタイル

図 4.6 に、境界線と背景色の描画例を示します。

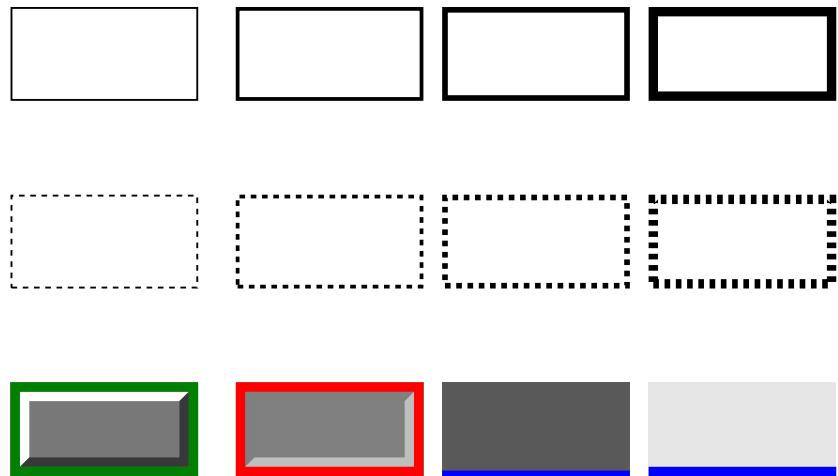


図 4.6: 境界線と背景色

4.3.4 回転角度

フィールドに任意の回転角度を設定することができます。

基本フィールド属性の「向き」を指定した場合、フィールドの内容物の描画方向が 90 度単位で回転します。

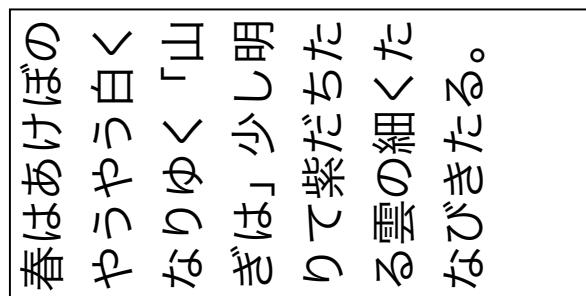


図 4.7: 基本フィールド属性の「向き」を 90 度に指定

4.3.5 座標変換

拡張フィールド属性の「回転角度」を指定した場合、フィールド自体が任意の角度で回転します。



図 4.8: 拡張フィールド属性の「回転角度」を指定

さらに座標変換行列を直接指定すれば、任意のアフィン変換を掛けることができます。

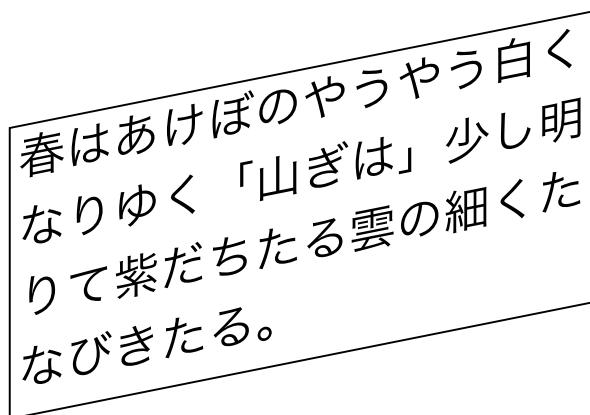


図 4.9: 拡張フィールド属性の「座標変換行列」を指定

4.3.6 透明度

透明度の設定により、フィールドを重ねて表示することができます。すかし・印影などの表現に利用できます。

春はあけぼの月の頃はさ
夜。やうやう白
夏は、
なりゆく「闇きは」少し明
らなり。
りで紫だちが邊ひた
の多く飛びたる雲の細くた
なびきたる。

図 4.10: 透明度を指定しての重ねあわせ表示

4.3.7 ブレンドモード

フィールドを重畠表示する際に、下記のブレンドモードを指定した合成ができます。

- 通常
- 乗算
- スクリーン
- オーバーレイ
- ソフトライト
- ハードライト
- 覆い焼きカラー
- 焼き込みカラー
- 比較（暗）
- 比較（明）
- 差の絶対値
- 除外
- 色相
- 彩度
- カラー
- 輝度



図 4.11: ブレンドモードを指定して背景画とテキストを合成

4.4 組版処理

フィールド統合処理（4.1.2）において、テキストの外観を生成する際に行われる組版処理について説明します。

4.4.1 行分割

明示的な行分割

テキストに改行文字が挿入されている場合は、その位置で分割します。

改行として認識する文字は以下のとおりです。

- LF (U+000A)
- CR (U+000D)

自動的な行分割

以下のポイントを分割位置の候補とし、最も矩形幅に近い位置で分割します。

- 単語境界（空白文字）
- 欧文と和文の間
- 和文文字の間
- ハイフン ‘-’ の後
- ハイフネーション可能な位置

単語境界として認識する空白文字は以下のとおりです。

- SPACE (U+0020)
- CHARACTER TABULATION (U+0009)
- EN QUAD (U+2000)
- EM QUAD (U+2001)
- EN SPACE (U+2002)
- EM SPACE (U+2003)
- THREE-PER-EM SPACE (U+2004)
- FOUR-PER-SPACE (U+2005)
- SIX-PER-EM SPACE (U+2006)
- PUNCTUATION SPACE (U+2008)
- THIN SPACE (U+2009)
- HAIR SPACE (U+200A)
- ZERO WIDTH SPACE (U+200B)
- IDEOGRAPHIC SPACE (U+3000)

いわゆる全角スペース (IDEOGRAPHIC SPACE (U+3000)) は、単語境界として利用しません。SPACE以外の空白文字がどのように表示されるかは、フォントの定義に依存します。

NO-BREAK SPACE (U+00A0) は SPACE に置き換えられますが、その位置での改行は行いません。

基本的に、欧文単語・数字列などの途中では行分割は起こりませんが、上記の分割候補位置が見つからない場合（フィールドの幅が狭い場合など）には、強制的に分割することがあります。

なお、これら特殊文字をテキスト中に挿入するには、文字コードを直接埋め込むか、エスケープ文字 (5.7) もしくは文字実体参照 (5.32) を利用してください。

4.4.2 ハイフネーション処理

行末付近の欧文単語がハイフネーション可能な場合は、ハイフンを挿入した上で、行を分割します。

SOFT HYPHEN (U+00AD) の位置が分割ポイントになった場合は、ハイフン ‘-’ を表示して改行します。分割しない場合は、何も表示しません。

ハイフネーション位置の特定には、Frank Liang のアルゴリズム（パターンマッチングに基づく手法）を使用しています。

Alice was beginning to get
very tired of sitting by her sis-
ter on the bank, and of having
nothing to do: once or twice
she had peeped into the book
her sister was reading, but it
had no pictures or conversa-
tions in it, 'and what is the use
of a book,' thought Alice 'with-
out pictures or conversation?'
So she was considering in her

図 4.12: ハイフネーション処理の例

4.4.3 禁則処理

表 4.2, 4.3, 4.4 に示す禁則文字が行頭・行末にかかった場合などには、追い出し処理による調整を行います。

表 4.2: 行頭禁則文字

分類	文字
終わり括弧類	' ")] } > » (U+2986) » «
ハイフン類	- - = -
中点類	・ : ;
句点類	、 ,
繰り返し記号	、 ノ > ノ ノ
後置省略記号	。 ' " °C € % %o HP ℥

表 4.3: 行末禁則文字

分類	文字
始め括弧類	' " ([{ < 《 「 『 (U+2985) 』 』 « "
前置省略記号	\$ £ № € №

表 4.4: 分離禁止文字

分類	文字
ダッシュ	—
リーダー	… …
アラビア数字	0 1 2 3 4 5 6 7 8 9 0 (全角)

春は、あけぼの。やうやう白くなりゆく山
ぎは 少し明りて紫だちたる雲の細くたな
びきたる。

夏は、夜。月の頃はさらなり。闇もなほ。
螢の多く飛び違ひたる。また、ただ一つ二
つなど、ほのかにうち光りて行くもをか
し。雨など降るもをかし。

秋は、夕暮。夕日のさして、山の端（は）
いと近うなりたるに、鳥の寝どころへ行く
とて、三つ四つ、二つ三つなど、飛び急ぐ
さへあはれなり。まいて雁などの連ねたる
がいと小さく見ゆるは、いとをかし。日入
り果てて、風の音、虫の音など、はたいふ
べきにあらず。

冬は、つとめて。雪の降りたるはいふべき

図 4.13: 禁則処理の例

4.4.4 割付け処理

テキストの整列方法として「均等割付」が指定されている場合は、文字間隔の調整を行います。

行長が文字枠の幅より短い場合は「アキ」の部分の文字間隔を広げます。反対に行長が長い場合は「ツメ」の部分の文字間隔を狭めます。

文字間隔の調整が行われるポイントは、表 4.5 のとおりです。

表 4.5: 文字間隔の調整ポイント

分類	アキ	ツメ
欧文単語の区切り（空白文字）	○	×
和文と欧文の間	○	×
和文文字の間	○	○
括弧類	×	○
中点	×	○

春は、あけぼの。やうやう白くなりゆく山
ぎは 少し明りて紫だちたる雲の細くたな
びきたる。

夏は、夜。月の頃はさらなり。闇もなほ。
螢の多く飛び違ひたる。また、ただ一つ二
つなど、ほのかにうちちめりて行くもをか
し。雨など降るもをかし。

秋は、夕暮。夕日のさして、山の端（は）
いと近うなりたるに、鳥の寝どころへ行く
とて、三つ四つ、二つ三つなど、飛び急ぐ
さへあはれなり。まいて雁などの連ねたる
がいと小さく見ゆるは、いとをかし。日入
り果てて、風の音、虫の音など、はたいふ
べきにあらず。

冬は、つとめて。雪の降りたるはいふべき

図 4.14: 割付処理の例（割付前のイメージと重ねあわせ）

4.4.5 縦組みテキスト

単行テキスト

横書きテキストと同様に、「上寄せ」「下寄せ」「中央寄せ」「均等割」など、寄せの指定ができます。句読点・
かぎ括弧などの記号は、縦組用のグリフに差し替えられます。

複数行テキスト

横書きテキストと同様に、禁則処理を伴う寄せの指定ができます。

春は、あけぼの。やうやう白くなりゆく「山ぎ
は」少し明りて紫だちたる雲の細くなびきたる。
夏は、夜。月の頃はさらなり。闇もなほ。螢の
多く飛び違ひたる。また、ただ一つ二つなど、
ほのかにうち光りて行くもをかし。雨など降る
もをかし。

秋は、夕暮。夕日のさして、山の端（は）いと近
うなりたるに、鳥の寝どころへ行くとて、三つ
四つ、二つ三つなど、飛び急ぐさへあはれなり。
まいて雁などの連ねたるがいと小さく見ゆるは、
いとをかし。日入り果てて、風の音、虫の音な
ど、はたいふべきにあらず。

冬は、つとめて。雪の降りたるはいふべきにも
あらず。霜のいと白きも、またさらでも、いと
寒きに、火など急ぎ熾して、炭もて渡るも、い
とつきづきし。昼になりて、ぬるくゆるびもて
いけば、火桶の火も、白き灰がちになりて、わ
ろし。

図 4.15: 縦組テキストの例

4.5 拡張漢字の利用

4.5.1 追加面に格納された Unicode 文字の指定

Unicode による表現

Unicode 型文字列を利用可能なプログラミング言語を介して Field Reports を利用する場合は、 Unicode を用いて拡張漢字を含む文字列を表現することができます。16 ビット幅のワイド文字で追加面の文字を指定する場合は、2 文字分のサロゲートペアで 1 文字を表現します。

UTF-8 による表現

UTF-8 エンコードしたバイト列で文字列を表現する場合は、 BMP 領域の文字を 1 文字を 1~3 バイトの可変長で、追加面の文字を 4 バイトのバイト列で表現します。

エスケープシーケンスによる指定

プログラミング言語によっては、 Unicode コードポイントを表現するためのエスケープシーケンスが用意されています。

Python, C# では、 \uhhhh または \Uhhhhhhhh で 16 ビットまたは 32 ビットの Unicode 文字が指定できます。

Java では、 \uhhhh がありますが、 16 ビット Unicode 文字しか指定できません。追加面の文字を指定する場合は、2 文字のゲートペアに分解して指定してください。

Ruby, Perl, PHP では特に Unicode 用のエスケープシーケンスは用意されていないので、 \xhh などのバイト文字用のエスケープシーケンスを利用して UTF-8 バイト列を埋め込んでください。

レンダリングパラメータの表現形式のひとつである JSON の規格として用意されているのは 16 ビット形式の \uhhhh のみです。Field Reports では、 JSON の規格を独自拡張しているので、 \Uhhhhhhhh 形式の 32 ビット Unicode 文字も利用することができます。

文字参照による指定

文字参照により Unicode コードポイントを指定することもできます。 &#dddd; または &#xhhhhhh; という書式により、10 進数または 16 進数による指定を行うことができます。

文字参照はフィールド単位で有効・無効を切り替えます。デフォルトでは無効になっているので、利用する場合は明示的に有効にする必要があります。

実行例

以下にエスケープシーケンスと文字参照を利用して、サロゲートペアで表現される文字を表示する例を示します。

```
{  
  "resources": {  
    "font": {  
      "IPAmjMincho": {  
        "src": "../fonts/ipamjm.ttf",  
        "format": "opentype"  
      }  
    }  
  }  
}
```

```

        "embed": true,
        "subset": true
    },
},
{
"template": {"size": "A4"},

"context": {
"text": [
{
"new": "Tx",
"font": "IPAmjMincho",
"font-size": 24,
"multiline": true,
"rect": [50, 650, 550, 750],
"charref": true,
"value": "\u00002000B\u000020089\u0000200A2\u0000200A4&x201A2;&x20213;&x2032B;
&x20371;&x20381;&x203F9;&x2044A;&x20509;&x205D6;&x20628;&x2074F;
&x20807;&x2083A;&x208B9;&x2097C;&x2099D;\n
&x2A716;&x2A729;&x2a72a;&x2a72c;&x2a738;&x2a73d;&x2a746;&x2a752;
&x2a758;&x2a75f;\n
&x2B740;&x2B741;&x2B742;&x2B743;&x2B744;&x2B745;&x2B746;&x2B747;
&x2B749;&x2B74A;&x2B74C;&x2B74D;"}
]
}
}
}

```

丈ノ辰自ヘ倘俾集儻儻儻矣浴几劔剣劔勵斗卓
畠勾勾余併金傍假偒偒
五尹所爾久矣今金倉傳奐岡

図 4.16: サロゲートペアの使用例

4.5.2 異体字セレクタによる異体字の指定

異体字セレクタとは

Unicode の漢字統合の原則により、複数の自体のバリエーションを持つ文字であってもひとつのコードポイントに統合されるのが基本です。例えば多くの字体が存在することで有名な渡邊の「邊」で、実際にコードポイントが割り当たられてるのは、邊 (U+9089), 邊 (U+908A) の 2 文字だけです。

人名・地名などを扱うためには字体のバリエーションを実際に区別する必要がありますが、漢字統合の原則によりむやみに増やすことができません。そこで、漢字統合の原則を守りつつ字体を区別する方法として考えだされたのが異体字セレクタと呼ばれる特殊な文字コードです。

日本語の場合、U+E0100～U+E01EF の範囲のコードを異体字セレクタとして使用します。基本となる正体字の文字コードに続けて異体字セレクタを配置することで、何番目の異体字かを表現します。

異体字セレクタを扱う方法

以下に、IPAmj明朝フォントを使って邊の異体字を列挙した例を示します。U+E0100～U+E01EFは2バイトで表現できないので、独自形式のJSONで記述しています。

```
{
  "resources": {
    "font": {
      "IPAmjMincho": {
        "path": "./ipamjm.ttf",
        "embed": true,
        "subset": true
      }
    }
  },
  "template": {"paper": "A4"},

  "context": {
    "text": [
      {
        "new": "Tx",
        "font": "IPAmjMincho",
        "rect": [100, 450, 500, 750],
        "value": "\u0000E010F\u0000E0119
\u0000E011B\u0000E011A\u0000E0119
\u0000E011D\u0000E0117\u0000E0116
\u0000E0115\u0000E0114\u0000E0118
\u0000E0113\u0000E0112\u0000E0111
\u0000E0110"
      }
    ]
  }
}
```

邊邊邊邊邊邊邊邊邊邊邊邊
邊邊邊邊邊邊邊邊邊邊
葛飾区 葛城市／蓮田市 蓮田市

図 4.17: 異体字セレクタの使用例

4.5.3 グリフ直接指定

CIDによるグリフ指定

使用するフォントがOpenTypeのCJKフォント（拡張子：*.otf）であれば、CIDをキーとしてグリフを指定することができます。CIDとは、CIDフォントが内蔵するすべてのグリフを一意に識別するために、Adobe社が付与した番号です。その番号体系は、日本語CIDフォントであればAdobe-Japan1文字コレクションに

もとづいています。

CID によるグリフ指定は、Field Reports 独自の以下の実体参照形式により行います。

&#dddd; または &xhhhhh;

グリフ名によるグリフ指定

使用するフォントで「グリフ名」が定義されていれば、グリフ名によるグリフ指定を行うこともできます。Field Reports 独自で定義した実体参照形式によりグリフを指定することができます。

<グリフ名>;

グリフ名が定義されているかどうかは、ttfdump などのツールで、TrueType フォントの「post」テーブルの内容をダンプすると確認することができます。

```
{
  "resources": {
    "font": {
      "IPAmjMincho": {
        "src": "../fonts/ipamjm.ttf",
        "embed": true,
        "subset": true
      }
    }
  },
  "template": {"size": "A4"},

  "context": {
    "text": [
      {
        "new": "Tx",
        "font": "IPAmjMincho",
        "font-size": 24,
        "multiline": true,
        "rect": [50, 350, 550, 450],
        "charref": true,
        "value": "&@mj000007;&@mj000008;&@mj000012;&@mj000022;&@mj000023;&@mj000028;
          &@mj000029;&@mj000036;&@mj000037;&@mj000045;&@mj000046;&@mj000047;
          &@mj000048;&@mj000073;&@mj000074;&@mj000089;&@mj000105;&@mj000106;
          &@mj000129;&@mj000130;&@mj000143;&@mj000144;&@mj000145;&@mj000146;
          &@mj000156;&@mj000157;&@mj000175;&@mj000176;&@mj000183;&@mj000184;
          &@mj000185;&@mj000206;&@mj000207;&@mj000208;&@mj000209;&@mj000241;
          &@mj000242;&@mj000264;&@mj000265;&@mj000266;&@mj000267;&@mj000269;
          &@mj000270;&@mj000276;&@mj000277;&@mj000278;&@mj000302;&@mj000303;
          &@mj000309;&@mj000310;&@mj000311;&@mj000312;&@mj000317;&@mj000318;
          &@mj000332;&@mj000333;&@mj000380;&@mj000381;&@mj000405;&@mj000406;"}
    ]
  }
}
```

図 4.18: グリフ名指定の使用例

第 5 章

レンダリングパラメータ

5.1 基本データ

表 5.1 に示す 基本データを組み合わせてレンダリングパラメータを記述します。

表 5.1: 基本データ

基本データ型	例
数値	42, 3.14
真理値	True, False
文字列	“文字列”
列挙値	Left, CropBox
辞書	{"title": “請求書”, “合計”: 10000}
リスト	[1, 2, 3]

辞書の要素は、取り出す際に並び順が維持されないものとします。

5.1.1 Python から利用する場合

Python から Field Reports を利用する場合は、基本データ型と Python のデータ型を表 5.2 のように対応付けて、レンダリングパラメータとなるデータ構造を作成します。

表 5.2: Python データ型との対応

基本データ型	Python データ型
数値	整数 (int 型) または浮動小数点数 (float 型)
真理値	真偽値 (bool 型)
文字列	文字列 (string 型) または Unicode 文字列 (unicode string 型)
列挙値	文字列 (string 型) または Unicode 文字列 (unicode string 型)
辞書	辞書 (dict 型)
リスト	リスト (list 型) またはタプル (tuple 型)

辞書のキーとして、文字列 (string 型) または Unicode 文字列 (unicode string 型) が使用できます。
文字列 (string 型) の文字コードは、UTF-8 としてください。

5.1.2 Ruby から利用する場合

Ruby から Field Reports を利用する場合は、基本データ型と Ruby のデータ型を表 5.3 のように対応付けて、レンダリングパラメータとなるデータ構造を作成します。

表 5.3: Ruby データ型との対応

基本データ型	Ruby データ型
数値	整数 (Fixnum または Bignum) または浮動小数点数 (Float)
真理値	true または false
文字列	文字列 (String)
列挙値	文字列 (String) またはシンボル (Symbol)
辞書	ハッシュ (Hash)
リスト	配列 (Array)

ハッシュのキーとして、文字列 (String) またはシンボル (Symbol) が使用できます。

Ruby1.8 の場合、文字列の文字コードを UTF-8 としてください。

Ruby1.9 の場合は、文字列自身が持つ Encoding と文字コードが一致しているものとします。

5.1.3 Perl から利用する場合

Perl から Field Reports を利用する場合は、基本データ型と Perl のデータ型を表 5.4 のように対応付けて、レンダリングパラメータとなるデータ構造を作成します。

表 5.4: Perl データ型との対応

基本データ型	Perl データ型
数値	数値
真理値	文字列 ("True" または "False")
文字列	文字列
列挙値	文字列
辞書	ハッシュリファレンス
リスト	配列リファレンス

ハッシュのキーは、文字列としてください。

文字列の文字コードは、UTF-8 としてください。

5.1.4 PHP から利用する場合

PHP から Field Reports を利用する場合は、基本データ型と PHP のデータ型を表 5.5 のように対応付けて、レンダリングパラメータとなるデータ構造を作成します。

文字列の文字コードは、UTF-8 としてください。

配列中にキーを持たない要素とキーを持つ要素が混在している場合は、数字のキーを持つ連想配列とみなします。

表 5.5: PHP データ型との対応

基本データ型	PHP データ型
数値	数値
真理値	論理型
文字列	文字列
列挙値	文字列
辞書	連想配列 (配列要素がキーを持つ)
リスト	添字配列 (配列要素がキーを持たない)

5.1.5 JSON で記述する場合

JSON でレンダリングパラメータを記述する際には、基本データ型と JSON のデータ型を表 5.6 のように対応付けます。

表 5.6: JSON データ型との対応

基本データ型	JSON データ型
数値	整数または浮動小数点数
真理値	真理値
文字列	文字列
列挙値	文字列
辞書	オブジェクト
リスト	配列

整数は、10 進記法に限ります。8 進・16 進記法は使用できません。浮動小数点数としては、1.0e-10 のような指数表記も可能です。

真理値としては、true と false が使用できます。

文字列は、ダブルコーテーションでくくります。文字のエンコーディングは、UTF-8 とします。表 5.7 のエスケープ文字を含めることができます。

辞書は、オブジェクトに対応付けます。オブジェクトは、キーと値のペアをコロンで対にして、これらの対をコンマで区切ってゼロ個以上列挙し、全体を中カッコでくくることで表現します。キーとして使うデータの型は文字列に限ります。

リストは、配列に対応付けます。配列はゼロ個以上の値をコンマで区切って、角カッコくくることで表現します。

表 5.7: JSON で利用可能なエスケープ文字

エスケープ文字	意味
\\"	バックスラッシュ (\)
\\"	二重引用符 ("")
\\\	スラッシュ (/)
\b	バックスペース (BS)
\f	フォームフィード (FF)
\n	行送り (LF)
\r	復帰 (CR)
\t	水平タブ (TAB)
\uhhhh	16-bit の 16 進数値 hhhh を持つ Unicode 文字
\Uhhhhhhhh	32-bit の 16 進数値 hhhhhhh を持つ Unicode 文字 (Field Reports での拡張仕様)

5.2 共通データ構造

基本データ構造を組み合わせて構築される汎用のデータ構造がいくつかあり、レンダリングパラメータの各所で使用されます。

この節では、日付／時刻文字列・URL・色指定のデータ構造について説明します。

5.2.1 日付／時刻文字列

日付または時刻は、以下の書式の文字列で表現します (ISO 8601 のサブセット)。

日付 → YYYY[- MM[- DD]]

時刻 → hh[:mm[:ss]]]

日付と時刻 → YYYY[- MM[- DD]]T hh[:mm[:ss]]]

日付と時刻を同時に表記する場合は、日付と時刻を区切り記号 “T” で区切れます。ISO 8601 の規格からは外れますが、“T” の代わりに空白文字 “_” も許容します。

5.2.2 色指定

色指定が必要な場面では、以下のデータ構造を使用します。

透明色 → []

グレースケール色 → 数値 | [数値]

RGB 色 → [数値 , 数値 , 数値]

CMYK 色 → [数値 , 数値 , 数値 , 数値]

色名 → 列挙値

グレースケール色では、0～255までの数値で階調を表現します。RGB 色では、赤・緑・青の各色成分を0～255までの数値のリストとして表現します。CMYK 色では、シアン・マゼンタ・黄・黒の各色成分を0～255までの数値のリストとして表現します。色名による指定では、表 5.8 にあげる列挙値が使用できます。

表 5.8: 色名の一覧

列挙値	RGB 値
Black	[0, 0, 0]
Gray	[128, 128, 128]
Silver	[192, 192, 192]
White	[255, 255, 255]
Maroon	[128, 0, 0]
Red	[255, 0, 0]
Purple	[128, 0, 128]
Fuchsia	[255, 0, 255]
Green	[0, 128, 0]
Lime	[0, 255, 0]
Olive	[128, 128, 0]
Yellow	[255, 255, 0]
Navy	[0, 0, 128]
Blue	[0, 0, 255]
Teal	[0, 128, 128]
Aqua	[0, 255, 255]

5.2.3 URL

PDF ファイル・画像ファイル・フォントファイルなどのファイルの場所を指定する場面では、URL によりリソースの場所を指定します。

URL は、以下の書式の文字列です。

URL → キーム名 + ドメイン名 + パス名

スキーム名としては、バックエンドで使用している libcurl ([B.2](#)) がサポートしているスキームが使用できます。

ローカルファイル

スキーム名・ドメイン名を省略しパス名のみを記述した場合は、ローカルファイルを指示していると解釈します。

ローカルファイルのパス名が "/" もしくはドライブ名 (Windows のみ) で始まる場合は、絶対パスによるファイルの指定として扱います。

パス名が "." もしくは ".." で始まる場合は、カレントディレクトリ相対のパス名によるファイル指定とみなします。Field Reports を実行しているプロセスのカレントディレクトリを基準として、ファイルを指定します。

パス名の先頭が “/”・“.”・“..” もしくはドライブ名以外で始まるディレクトリ名の場合は、template-root (5.12) 相対のパス名によるファイル指定として扱います。

data URI scheme 文字列

スキーム名が “data:” の場合は、以下の data URI scheme 文字列形式を用いてファイルの内容をレンダリングパラメータにインラインで埋めこむことができます。

data スキーム文字列 → **data:** MIME-type[;base64] , データ列

MIME-type として指定できる値は以下のとおりです。

- application/pdf
- image/jpeg
- image/jp2
- image/png
- image/x-bmp

“;base64” が付いている場合は、カンマ以降のデータ列 Base64 デコードしてから取り込みます。
“;base64” を省略した場合は、カンマ以降にバイナリデータが続いているものとします。

文字列の終端をヌル文字で判断する C タイプの文字列データを使用しているプログラミング言語では、文字列にバイナリデータを埋め込むことができませんので、Base64 エンコードが必要となります。また、JSON で記述する場合も Base64 エンコードが必要です。

5.3 セレクタ文字列

ピリオド “.” を区切り文字として部分セレクタを結合したものをセレクタと呼びます。

部分セレクタは、名前セレクタ・全称セレクタ・整数セレクタのいずれかです。名前セレクタは、リテラル文字列で指定し、同じ文字列を持つ部分フィールド名とマッチします。全称セレクタは、“*” で指定し、任意の部分フィールド名とマッチします。整数セレクタは、開始値、終了値、ステップ数 から生成される整数列のいずれかと同じ値を持つ部分フィールド名とマッチします。

セレクタと完全修飾フィールド名とのマッチング処理では、ルートの部分フィールド名から順にマッチングを試みていきます。途中でマッチングが失敗するか、すべての部分セレクタのマッチングが成功した時点で、マッチング処理は終了します。マッチング処理全体が成功した場合は、最後に検査したフィールドとその子フィールドすべてが選択対象となります。

以下にセレクタ文字列の書式を示します。

```
セレクタ → 部分セレクタ . ...
部分セレクタ → 名前セレクタ
          | 全称セレクタ
          | 整数セレクタ
名前セレクタ → リテラル文字列
```

全称セレクタ → *

整数セレクタ → [インデックス値]

→ [[開始値]:[終了値]]

→ [[開始値]:[終了値]:ステップ数]

整数セレクタの第一の書式では、インデックス値により要素を一つ選択します。第二・第三の書式では、開始値から終了値までの範囲の整数列にマッチする要素を選択します。ただし第三の書式では、整数列を生成する際にステップ数づつ加算していきます。

開始値・終了値はそれぞれ省略可能です。開始値を省略した場合は 0 と解釈します。終了値を省略した場合は、最後の要素までを範囲とします。

整数セレクタでは、図 5.1 のように、要素と要素の間にインデックスがあると考えます。終了値にマイナスの数値を使用した場合は、最後の要素から数えて何個目かを示します。

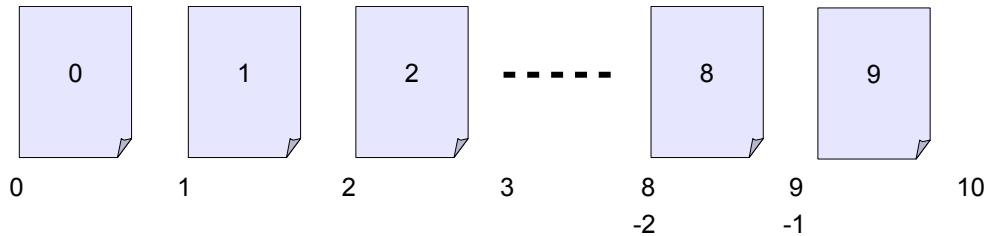


図 5.1: 開始値と終了値の考え方

整数セレクタの記述例を表 5.9 にいくつか示します（要素が 10 個の場合）。

表 5.9: 整数セレクタの使用例

名前パターン	意味
[1]	1 にマッチする。
[-2]	8 にマッチする。
[0:2]	0, 1 にマッチする。
[0:-1]	0, 1, 2, ..., 8 にマッチする。
[0:10]	0, 1, 2, ..., 9 にマッチする。
[::3]	0, 1, 2 にマッチする。
[8:]	8, 9 にマッチする。
[::]	0, 1, 2, ..., 9 にマッチする。
[1::2]	1, 3, 5, 7, 9 にマッチする。

5.4 レンダリング辞書

レンダリングパラメータとして、表 5.10 に示すレンダリング辞書を与えます。

表 5.10: レンダリング辞書のエントリ

キー	型	値
template	template 要素	(必須) 名前空間を定義し、PDF テンプレートと関連付けます。 複数の PDF テンプレートを連結して PDF 帳票を組み立てる際には、PDF テンプレートに元々存在するフィールドの名前とここで定義した名前空間を合わせて、新しいフィールド名とします。
resources	resources 辞書	(オプション) フォント・画像リソースを定義します。 リソースとして定義すると、データ読み込み時・PDF 出力時に処理の重複を避けることができる、処理時間とファイルサイズの節約になります。
context	context 要素	(オプション) フィールド名に対応する属性を 1 対 1 対応で指定します。 フィールドが階層構造を持つ場合は、辞書とリストを使って、相似形の木構造データとして記述します。
style	リスト	(オプション) 復数のフィールドの属性を一括して指定します。 セレクタにより適用対象となるフィールドをパターンマッチングにより特定します。
environ	辞書	(オプション) ユーザ定義の環境変数を定義します。 フィールド値の一部に “\${ 環境変数名 }” の並びが出現する箇所で、環境変数の値に置き換えられます。ページ番号・全ページ数などのシステム定義の環境変数については 5.11.2 を参照してください。
property	辞書	(オプション) 生成される帳票の PDF 属性を指定します。 文書情報・セキュリティ・文書の開きかたなどの属性を設定することができます。
settings	辞書	(オプション) Field Reports の共通設定情報を格納します。 シリアル番号、ライセンスキー、PDF テンプレート格納ディレクトリなどの情報を記述します。

フィールド属性に影響を与えるレンダリングパラメータは、以下の順序で適用されます。

1. style リスト
2. context 要素
3. 環境変数

style リストと context 要素の指定が重複した場合は、context 要素による指定が優先されます。

環境変数の展開は、style リストと context 要素の適用後の value 属性の値に対して行われます。

5.5 template 要素

帳票で使用する PDF テンプレートを宣言します。

ここでは名前空間を定義し、PDF 要素との対応付けを行ないます。template 要素の書式を表 [5.11](#) に示します。

表 5.11: template 要素の書式

書式	説明
PDF 要素	名前空間の定義が不要な場合の書式です。 単票で使用します。
〔 { 名前空間 : PDF 要素 } , ... 〕	複数の帳票を組み合わせた複合帳票を構成する場合の書式です。 連続帳票でも使用します (Standard 版のみ)。

5.5.1 名前空間

名前空間 では、名前空間の名称を文字列で定義します。英数字とアンダースコア “_” が使用できますが、連続帳票で数字の名称を使用しますので、数字始まりの名称を付ける場合はご注意ください。

連続帳票を作成する場合は、名前空間の名称を “*” 単独もしくは “.*” で終わるものとします。“*” の部分が “0,1,...” という整数列に置き換わった名前空間が、実際に展開されたページ数に応じて、暗黙的に定義されます。

```

名前空間 → リテラル名
| *
| リテラル名 .*
リテラル名 → 英字 {(英数字 | _)}

```

5.5.2 PDF 要素

PDF 要素の書式は、表 5.12 のとおりです。

表 5.12: PDF 要素の書式

書式	説明
URL 文字列	src 属性の指定だけでよい場合の省略記法です。
PDF 辞書	src 以外の属性も指定する場合は辞書形式とします (表 5.12 参照)。

単に PDF ファイルの URL を指定するだけで十分な場合は、URL 文字列を直接記述します。その他の属性を指定する必要がある場合は、PDF 辞書を記述します。

PDF 辞書として、表 5.13 のエントリを持つ辞書を与えます。

表 5.13: PDF 辞書のエントリ

キー	型	値
src	URL 文字列	(paper を指定しない場合は必須) PDF テンプレートの URL を指定します。
paper	数値リストまたは列挙値	(オプション; src を指定した場合は無効) ページを新規に生成し, PDF テンプレートとして利用します。生成されたページは空白ですが, 動的に生成したテキスト・画像フィールドを任意の位置に配置することができます。ここで指定した用紙サイズは, 出力する PDF の MediaBox (用紙サイズ) に反映されます。数値リスト形式で指定する場合は, 幅・高さを 2 要素のリストとしてポイント単位で指定します。列挙値で指定する場合は, 次の選択値のどれかを指定します。選択値 : A0, A1, … A10, B0, B1, … B10, Letter, Legal.
orientation	列挙値	(オプション; src を指定した場合は無効) 用紙の方向を指定します。paper と組で使用します。選択値 : Portrait (縦長), Landscape (横長). 省略値 : Portrait.
crop-box	数値リスト	(オプション) 出力する PDF の CropBox (トリミング範囲) を指定します。MediaBox の左下を原点とした座標系における左下・右上の座標を 4 要素の数値リストとして与えます。書式 : [<i>llx</i> , <i>lly</i> , <i>urx</i> , <i>ury</i>]. 単位 : ポイント.
bleed-box	数値リスト	(オプション) 出力する PDF の BleedBox (裁ち落としサイズ) を指定します。書式・単位は同上。
trim-box	数値リスト	(オプション) 出力する PDF の TrimBox (仕上がりサイズ) を指定します。書式・単位は同上。
art-box	数値リスト	(オプション) 出力する PDF の ArtBox (アートサイズ) を指定します。書式・単位は同上。
rows	整数または辞書	(連続帳票では必須) テーブル形式フィールドの最大の行数を指定します。テンプレート内にテーブルがひとつだけ, もしくはすべてのテーブルの行数が同一の場合は, 単に整数で指定します。複数のテーブルが存在し, かつそれぞれの行数が異なる場合は, それぞれのテーブルの行数を辞書形式で個別に指定します (書式 : {部分フィールド名 : 最大行数, ...})。rows エントリの値を元にテーブル分割処理を行います。テーブル行数が 0 の場合, テーブル分割処理は行いません。省略値 : 0.
pages	整数リスト	(オプション) PDF テンプレートとして実際に使用するページを列挙します。省略値 : [] (全ページ).

5.6 resources 辞書

繰り返し使用するフォント・画像データをリソースとして定義します。

表 5.14: resources 辞書のエントリ

キー	型	値
font	font 要素	(オプション) フォント・リソースを定義します。
image	image 要素	(オプション) 画像リソースを定義します。

5.6.1 font 要素

font 辞書の書式は、表 5.15 のとおりです。

表 5.15: font 辞書のエントリ

キー名	型	値
フォント・リソース名	URL 文字列または font 属性辞書	フォント属性として URL 指定のみで十分な 場合は、フォントファイルの URL を文字列で 指定します。その他のフォント属性も指定す る必要がある場合は、font 属性辞書 (表 5.16) を使用します。

表 5.16: font 属性辞書のエントリ

キー名	型	値
src	URL 文字列	フォントファイルの URL を指定します。
embed	真偽値	(オプション) フォントを PDF に埋め込 むかどうかの指定をします。省略値: true
subset	真偽値	(オプション; embed が true の場合のみ 有効) フォントを PDF に埋め込む際にサブ セット化を行うかどうかの指定をします。 省略値: true
ttc-index	整数	(オプション; TTC フォントの場合のみ有 効) TTC フォントのインデックス番号を指 定します。省略値: 0
writing-mode	列挙値または整数	(オプション) テキストの組み方向を規定 します。 HorizontalTb または 0 横組み。行送り方向は上から下。 VerticalRl または 1 縦組み。行送り方向は右から左。 省略値: HorizontalTb

フォントリソース名について

辞書要素のキーとして他のフォントと重複しないフォントリソース名を指定します。ここで定義したフォントリソース名は、`field` 辞書（表 5.24）でフォントを指定する際に利用します。

フォントリソース名は任意に決めることができますが、PDF テンプレートで使用しているフォントの名称と一致させると、`font` 辞書の定義が優先して使用されるようになります。これを利用して、`field` 要素において各フィールドの `font` 属性を明示的に指定することなく、フォントを埋め込む指定を行うことができます。

フィールドで使用しているフォントの名称は、`reports` コマンド（6.7.1 参照）の “info” サブコマンドにより確認することができます（以下の実行例の場合、 “title” フィールドに設定されているフォントは “/KozMinProVI-Regular” であることがわかる）。

【実行例】

```
$ reports info --field template.pdf
title: "テンプレート"
creator: "Acrobat Editor 8.0"
producer: "Adobe Acrobat 8.2.4"
creation-date: "D:20101122090233+09'00'"
mod-date: "D:20111217162540+09'00'"
num_pages: 1
page: #1
    field: "title"
        type: "Tx"
        font: "/KozMinProVI-Regular"
        font-size: 0
        value: ""
        rect: [106.908, 767.546, 496.36, 789.182]
```

横組みと縦組みを同時に使用する場合のフォント定義

一つのフォントファイルを縦組みと横組みで同時に使用する場合は、フォントリソース名と `writing-mode` を変えたりソースを 2 つ定義してください。

```
// 横組みと縦組みを同時に使う場合の設定例
"resources": {
    "font": {
        "HiraMaruPro-W4": {
            "src": "/usr/lib/fonts/HiraMaruGo_Pro_W4.otf",
            "writing-mode": 0
        },
        "@HiraMaruPro-W4": {
            "src": "/usr/lib/fonts/HiraMaruGo_Pro_W4.otf",
            "writing-mode": 1
        }
    }
}
```

```
}
```

TTC フォントを使用する場合のヒント

TTC フォントとは、複数の TrueType フォントを一つのファイルに格納した形式のフォントです。Field Reports では、0 始まりの ttc-index により TTC フォント内のフォントを識別します。

reports コマンドの “font” サブコマンドを使うと、フォントファイルの内部情報が調べられます。このコマンドにより、TTC フォントが内蔵しているフォントの数を知ることができます。

【実行例】

```
$ reports font msmin04.ttc
[msmin04.ttc]
type: TrueType
PostScript name: MS-Mincho
ttc index: 0

[msmin04.ttc]
type: TrueType
PostScript name: MS-PMincho
ttc index: 1
```

フォントリソースの雛形の作成

“font” サブコマンドに “—json” オプションを付けて実行することで、フォントファイルからフォントリソース定義の雛形を生成することができます。

5.6.2 image 要素

image 辞書の書式は、表 5.17 のとおりです。

表 5.17: image 辞書のエントリ

キー名	型	値
画像リソース名	URL 文字列または image 属性辞書	画像属性として URL 指定のみで十分な場合は、画像ファイルの URL を文字列で指定します。その他の画像属性も指定する必要がある場合は、image 属性辞書（表 5.18）を使用します。

表 5.18: image 属性辞書のエントリ

キー名	型	値
src	URL 文字列	画像ファイルの URL を指定します。
page	真偽値	(オプション; PDF 形式の場合のみ有効) 画像ファイルとして PDF を指定した際に使用するページを指定します。省略値: 0
enable-bmp-alpha	真偽値	(オプション; 32 ビット BMP 形式の場合のみ有効) 32 ビットモード BMP 形式の画像を指定した際に, RGB (24 ビット) 以外の残りの 8 ビットを α チャンネルとして使用するかどうかを指定します。省略値: false
transparent-range	整数リスト	(オプション; ラスタ画像形式の場合のみ有効) ラスタ画像を利用する際に, 透明色として扱う色の範囲を指定します。 $2 \times n$ 個の整数のリスト $[min_1, max_1, \dots, min_n, max_n]$ で指定します。ラスタ画像の色モードが RGB カラーであれば $n = 3$ となり, グレースケールであれば $n = 1$ となります。色の範囲はラスタ画像のビット深度 d に依存し, $0 \sim 2^d - 1$ となります。例えば RGB24 ビットカラー画像で赤を透過色にするには, $[255, 255, 0, 0, 0, 0]$ とします。インデックスカラーの場合は, $n = 1$ として 2 つのインデックス番号値で色の範囲を指定します。省略値: []

辞書要素のキーとして任意の画像リソース名を指定します。ここで定義した画像リソース名は, field 辞書(表 5.24) で画像を指定する際に利用します。

対応画像形式

image 属性で利用可能な画像ファイルの形式は, 表 5.19 のとおりです。

表 5.19: 対応している画像形式

画像形式	拡張子	Media Type	制限事項
JPEG	*.jpg, *.jpeg	image/jpeg	・特になし。
JPEG2000	*.jp2	image/jpeg	・特になし。
PNG	*.png	image/png	・インターレースモードには対応していません。
BMP	*.bmp	image/x-bmp	・透過色の扱いについては表 5.18 を参照してください。
PDF	*.pdf	application/pdf	・セキュリティが有効な PDF は使用できません。 ・ページ指定の方法については表 5.18 を参照してください。

5.7 context 要素

context 要素では, 部分フィールド名とフィールド属性の組を列挙します。辞書とリストを組み合わせて, フィールドの階層構造に一致する木構造のデータを組み立てます。

context 要素の書式を表 5.20 に示します。

表 5.20: context 要素の書式

書式	説明
{ 部分フィールド名 : (context 要素 field 要素) }	辞書形式で記述する場合。 部分フィールド名 が親フィールドの場合, context 要素 を与えます。部分フィールド名 が末端の子フィールドの場合, field 要素 を与えます。
[(context 要素 field 要素), ...]	リスト形式で記述する場合。 リストが, 1次元テーブルにおける「行」, 2次元テーブルにおける「列」の位置にある場合, field 要素 を与える。それ以外では, context 要素 を与えます。

辞書形式では, 部分フィールド名をキーに, context 要素または field 要素を値として与えます。

部分フィールド名が0始まりの整数列の場合に限り, リスト形式で記述することができます。これは, 「{"0": context 要素₀, "1": context 要素₁, ... }」の形式の辞書を略した記法と解釈することができます。リスト形式の表記はまた, テーブル分割機能において, 分割ポイントを探すための目印にもなります。テーブル分割機能を利用する場合は, 「行」位置のデータをリスト形式で記述してください。

5.8 style リスト

style を適用するフィールドをセレクタで指定します。

表 5.21: style リストの書式

書式	説明
[{ セレクタ : field 属性 }, ...]	リストの要素として, セレクタ文字列と field 属性の組を列挙します。リストの先頭要素から順にセレクタにマッチするフィールドを探していく, マッチしたフィールドに対して field 属性を適用します。

セレクタの表記法については, [5.3](#) を参照してください。

field 属性の書式は, context 要素で説明した field 属性と同じです。表 [5.23](#), [5.24](#), [5.35](#) を参照してください。

5.9 filed 要素

field 要素の書式は, 表 [5.22](#) のとおりです。

表 5.22: field 要素の書式

書式	説明
文字列・数値または真理値	field 属性として value 属性のみで十分な場合の略記法です。
field 辞書	value 以外の属性を指定する場合の書式です。

5.9.1 共通フィールド属性

表 5.23 に示す共通 field 辞書では、テキスト・フィールドとボタン・フィールドに共通する属性を指定することができます。

表中の拡張欄が○の項目は拡張属性を示します。拡張属性は、permanent 属性が true の場合のみ有効となります。permanent 属性が false の場合は、単に無視されます。

表 5.23: field 辞書のエントリ (共通)

キー	拡張	型	値
new	-	列挙値	(新規作成の場合は必須) フィールドの種類を指定します。 Tx テキストフィールド Btn ボタンフィールド new 属性が指定された場合は、rect 属性で指定された座標にフィールドを追加します。new 属性を指定しない場合は、既存フィールドにその他の属性をセットします。new 属性は context 要素内のみ使用できます。style 要素では使用できません。
visible	-	真理値	(オプション) フィールドの表示／非表示属性を変更します。新規作成時の省略値 : true.
border-width	-	数値	(オプション) 境界線の太さを指定します。0 を指定すると、境界線が表示されません。単位：ポイント。新規作成時の省略値 : 0.
border-style	-	列挙値	(オプション) 境界線のスタイルを指定します。選択値 : Solid, Dashed, Beveled, Inset, UnderlineStyle. 新規作成時の省略値 : Solid.
border-dash	-	リスト	(オプション) 破線パターンを指定します。border-style が Dashed の場合のみ有効になります。交互に現れる破線と隙間の長さを 0 要素から 2 要素までの数値リストとして指定します。0 要素の場合、破線ではなく実線になります。1 要素の場合、破線と隙間の長さは同じになります。2 要素の場合、破線・隙間の順にそれぞれの長さを指定します。単位：ポイント。新規作成時の省略値 : [] (実線).
border-color	-	色指定	(オプション) 境界線の表示色を指定します。新規作成時の省略値 : [0] (黒).
border-join-style	○	列挙値	(オプション) 境界線の角のスタイルを指定します。 MiterJoin 境界線の角を直角に描きます。 RoundJoin 境界線の角を丸めます。 BevelJoin 境界線の角を切り落とします。 省略値 : MiterJoin.
background-color	-	色指定	(オプション) フィールドの背景色を指定します。新規作成時の省略値 : [] (背景色なし).
padding	○	数値またはリスト	(オプション) 境界線と内容物の間の余白の量を指定します。数値で指定した場合、左・下・右・上に同じパディングを指定します。4 要素の数値リストで指定した場合、左・下・右・上の順にパディングを指定します。書式 : [左 , 下 , 右 , 上]. 単位：ポイント。新規作成時の省略値 : border-style が Beveled または Inset の時は 2, それ以外は 1.
angle	-	数値	(オプション) フィールドの中身の描画方向を指定します。Adobe Acrobat でフィールドのプロパティを編集する際の「向き」に対応します。90 度単位の角度を設定してください (0, 90, ...).

表 5.23: field 辞書のエントリ (共通: 続き)

キー	拡張	型	値
rect	-	リスト	(新規作成の場合は必須) フィールドの矩形座標を指定します。ページの左下を原点として、左下・右上座標を 4 要素のリストで与えます。書式: <code>[llx , lly , urx , ury]</code> . 単位: ポイント.
llx	-	数値	(オプション) フィールドの X 座標を変更します。単位: ポイント.
lly	-	数値	(オプション) フィールドの Y 座標を変更します。単位: ポイント.
width	-	数値	(オプション) フィールドの幅を変更します。単位: ポイント.
height	-	数値	(オプション) フィールドの高さを変更します。単位: ポイント.
translation	-	リスト	(オプション) フィールドの位置を相対的に移動させます。現在のフィールドの座標からの移動量を 2 要素のリストとして与えます。単位: ポイント. 書式: <code>[dx , dy]</code> .
rotation	○	数値または 3 要素数値リスト	(オプション) フィールドの回転角度を指定します。angle と異なり境界線を含めたフィールド全体が回転します。単一の数値で指定した場合、回転の中心はフィールドの左下となります。3 要素の数値リストで指定した場合は、回転の中心位置をフィールドの左下からの相対座標で指定します。単位: 角度(度数法)。書式: 角度 または <code>[X 座標 , Y 座標 , 角度]</code> .
matrix	○	6 要素数値リスト	(オプション) 座標変換行列を指定します。境界線を含めたフィールド全体に対してアフィン変換を掛けます。書式: <code>[a , b , c , d , e , f]</code> .
opacity	○	数値	(オプション) フィールドの不透明度を 0~1 の数値で指定します。0 を指定した場合、完全に透明になります。省略値: 1.
blend-mode	○	列挙値	(オプション) フィールドを背景と合成する際のブレンドモードを指定します。選択値: Normal, Multiply, Screen, Overlay, Darken, Lighten, ColorDodge, ColorBurn, HardLight, SoftLight, Difference, Exclusion, Hue, Saturation, Color, Luminosity.
permanent	○	真理値	(オプション) true の場合、フィールドを削除して、代わりにコンテンツストリームをページに追加します。false の場合は、フィールドを残します。省略値: true.

5.9.2 テキストフィールド属性

テキストフィールドでは、表 5.24 に示す属性を設定することができます。

表中の拡張欄が○の項目は拡張属性を示します。△の項目は、一部の値が拡張属性扱いであることを意味します。拡張属性は、permanent 属性が true の場合のみ有効となります。permanent 属性が false の場合は、単に無視されます。

表 5.24: field 辞書のエントリ (テキストフィールド)

キー	拡張	型	値
value	-	文字列・数値 または真理値	(オプション) テキストフィールドの値を指定します。 値が数値または真理値の場合は、文字列に変換します。新規作成時の省略値：“” (空文字列)。
color	-	色指定	(オプション) テキストの塗りつぶし色を指定します。新規作成時の省略値：[0] (黒)。
text-stroke-color	○	色指定	(オプション) テキストの縁取り色を指定します。透明色を指定した場合、テキストの縁取りは表示しません。color に透明色を指定し、text-stroke-color に不透明色を指定した場合は、縁取りのみ表示します。省略値：[] (透明)。
text-stroke-width	○	数値	(オプション) テキストの縁取りの線幅を指定します。省略値：1.
font	-	文字列	(オプション) テキストを描画する際に使用するフォントを指定します。font 要素 (5.6.1) で定義されたフォント・リソース名もしくはシステム定義フォント名 (表 5.25) を指定します。新規作成時の省略値：“/KozGo-Medium”
font-size	-	数値	(オプション) フォントサイズを指定します。multiline が false の場合に 0 を指定すると、テキストが文字枠に収まるようにフォントサイズを自動で設定します。multiline が true の場合に 0 を指定すると、フォントサイズを 10.5 ポイントとします。単位：ポイント。新規作成時の省略値：0.
multiline	-	真理値	(オプション) テキストを複数行に分割することを指示します。新規作成時の省略値：false.
text-align	△	列挙値	(オプション) テキストの整列方法を指定します。 Left 左寄せ Center 中央寄せ Right 右寄せ Justify 均等割付 (拡張属性) 新規作成時の省略値：Left.
vertical-align	○	列挙値	(オプション) 縦方向の整列方法を指定します。multiline 属性が true の場合には無視されます。 Bottom 下寄せ Middle 中央寄せ Top 上寄せ Justify 均等割付 (縦組みの場合のみ有効) 新規作成時の省略値：Bottom.
line-height	○	数値	(オプション) 行の送り幅を指定します。縦組みの場合も行送り幅としてこの値を利用します。font-size に対する倍率で指定します。新規作成時の省略値：1.2.

表 5.24: field 辞書のエントリ（テキストフィールド；続き）

キー	拡張	型	値
hyphens	○	列挙値	(オプション) 欧文単語のハイフネーション処理を行う場合の挙動を指定します。 None ハイフネーション処理を行いません。 Manual ソフトハイフンの処理のみ行います。 Auto ハイフン挿入位置を自動決定します。 新規作成時の省略値：Manual.
format	○	文字列	(オプション) 数値の書式を指定します。value 属性の値が数値として解釈できる場合に限り有効となります。value 属性の値が文字列の場合は、数値への変換を試みます。新規作成時の省略値：書式変換を行わない。
datetime	○	文字列	(オプション) 日付／時刻の書式を指定します。value 属性の値が日付もしくは時刻として解釈できる場合に限り有効となります。新規作成時の省略値：書式変換を行わない。
replace	○	文字列リスト	(オプション) 正規表現による文字列置換を指定します。「正規表現」にマッチした部分文字列を「置換テンプレート」で置き換えます。置換テンプレートには、\1 や\2 を含めることができます。これらは正規表現中の対応するグループにマッチした部分文字列で置き換えられます。\\0 は、正規表現全体にマッチした部分文字列を表します。書式：[正規表現 , 置換テンプレート]. 新規作成時の省略値：文字列置換を行わない。例：{"replace": [".+", "\\\0様"]}
charref	○	真偽値	(オプション) テキスト文字列中に文字参照を利用します。省略値：false.
normalize	○	真偽値	(オプション) Unicode 正規化処理を行い、分解された文字を合成します（例：か+→が）。NFD(Normalization Form Canonical Decomposition) に準じた変換を行いますが、互換文字の置き換えは行いません。省略値：false.

format, datetime, replace の指定は排他的です。同時に指定された場合は、format → datetime → replace の優先順位で一つのみ適用されます。

システム定義フォント

フォント名として、表 5.25 のシステム定義フォントを使用することができます。

表 5.25: システム定義フォント

フォント名	説明
/Times-Roman	Times Roman 体フォント
/Times-Bold	Times ボールド体フォント
/Times-Italic	Times イタリック体フォント
/Times-BoldItalic	Times ボールド・イタリック体フォント
/Helvetica	Helvetica フォント
/Helvetica-Bold	Helvetica ボールド体フォント
/Helvetica-Oblique	Helvetica 斜体フォント
/Helvetica-BoldOblique	Helvetica ボールド斜体フォント
/Courier	Courier フォント
/Courier-Bold	Courier ボールドフォント
/Courier-Oblique	Courier 斜体フォント
/Courier-BoldOblique	Courier ボールド斜体フォント
/Symbol	Symbol フォント
/ZapfDingbats	ZapfDingbats フォント
/KozMin-Regular	小塚明朝体フォント
/KozGo-Medium	小塚ゴシック体フォント

数値書式指定文字列

数値書式指定文字列のプレースホルダとして利用可能な文字は、表 5.26 のとおりです。

表 5.26: 数値書式指定文字列

書式指定文字	意味
0	ゼロプレースホルダ
#	桁プレースホルダ
.	小数点
,	桁区切り記号、値の位取り
%	パーセントプレースホルダ
\ 文字	エスケープ文字
‘ 文字列 ’ “ 文字列 ”	リテラル文字列
;	セクション区切り記号（最大 3 セクション：正；負；ゼロ）
その他の文字	結果の文字列にコピーされます。

数値書式指定文字列で利用可能なエスケープ文字は、表 5.27 のとおりです。

表 5.27: 書式指定文字列で利用可能なエスケープ文字

エスケープ文字	意味
\b	バックスペース (BS)
\n	行送り (LF)
\r	復帰 (CR)
\t	水平タブ (TAB)
\u HHHH	16 進数値 HHHH を持つ Unicode 文字
\ 文字	文字自身

数値書式指定文字列の使用例を表 5.28 に示します。

表 5.28: 数値書式指定文字列の使用例

書式指定文字列	フィールド値	外観文字列
####	123	123
0	123	123
(###)###-###	1234567890	(123)456-7890
#.##	1.2	1.2
0.00	1.2	1.20
00.00	1.2	01.20
#,#	1234567890	1,234,567,890
#,,	1234567890	1235
#,,,	1234567890	1
#,##0,,	1234567890	1,235
[##-##-##]	123456	[12-34-56]
##;(##)	1234	1234
##;(##)	-1234	(1234)

日付/時刻書式指定文字列

日付/時刻書式指定文字列のプレースホルダとして利用可能な文字は表 5.29 に示すとおりです。

表 5.29: 日付/時刻書式指定文字列

書式指定文字	意味
YY	2桁年 (0パディングする)
YYYY	4桁年 (0パディングする)
M	月 (0パディングしない)
MM	月 (0パディングする)
B	月略名 (Jan., Feb., ... Dec.)
BB	月正式名 (1月, 2月, ... 12月)
D	日 (0パディングしない)
DD	日 (0パディングする)
A	曜日略名 (日, 月, ... 土)
AA	曜日正式名 (日曜日, 月曜日, ... 土曜日)
G	年号略名 (A.D., M, T, S, H)
GG	年号正式名 (西暦, 明治, 大正, 昭和, 平成)
E	和暦 (0パディングしない)
EE	和暦 (0パディングする)
h	時 (24時間表記, 0パディングしない)
hh	時 (24時間表記, 0パディングする)
H	時 (12時間表記, 0パディングしない)
HH	時 (12時間表記, 0パディングする)
m	分 (0パディングしない)
mm	分 (0パディングする)
s	秒 (0パディングしない)
ss	秒 (0パディングする)
t	午前/午後略名 (AM, PM)
tt	午前/午後正式名 (午前, 午後)
\ 文字	エスケープ文字。数値書式指定文字のエスケープ文字と同じ。
‘ 文字列 ’ “ 文字列 ”	リテラル文字列
;	セクション区切り記号 (最大3セクション: 正; 負; ゼロ)
その他の文字	結果の文字列にコピーされます。

日付/時刻書式指定文字列の使用例を表 5.30 に示します。

表 5.30: 日付/時刻書式指定文字列の使用例

書式指定文字列	フィールド値	外観文字列
YYYY 年 MM 月 DD 曜日	2010-10-23T15:21:10	2010 年 10 月 23 日
GGEE 年 MM 月 DD 曜日	1911-04-04	明治 44 年 04 月 04 日
AA	2040-01-01	日曜日

正規表現

文字列置換で利用できる正規表現は、表 5.31 のとおりです。

バックスラッシュ (\) を特殊文字として扱うプログラミング言語で使用する場合は、バックスラッシュ自身をエスケープする必要があります。

正規表現による文字列置換は、マルチバイト文字に対応していません。UTF-8において2バイト以上で表される文字は、1文字として扱われませんのでご注意ください。

表 5.31: 正規表現

特殊文字	説明
.	改行を除くすべての文字にマッチします。
*	(後置) 先行する正規表現の0回以上の繰り返しにマッチします。
+	(後置) 先行する正規表現の1回以上の繰り返しにマッチします。
?	(後置) 先行する正規表現の0回か1回の出現にマッチします。
[...]	文字集合。[a-z]のように-で範囲を表します。[\^0-9]のように先頭に^を書くと補集合を取ります。]を含めたい場合には]を最初に書きます。-を含めたい場合には最初か最後に書きます。
^	行頭にマッチします(マッチさせる文字の先頭か、改行文字の直後にマッチします)。
\$	行末にマッチします(マッチさせる文字の末尾か、改行文字の直前にマッチします)。
	(中置) ふたつの正規表現の選択です。
\(...\)	囲まれた正規表現をグループ化し、名前をつけます。
\1	\(...\)\1でマッチした最初のテキスト(\2は2番目の式で、同様に\9まであります)。
\b	語の境界にマッチします。
\	特殊文字をクオートします。“\$^.*+?[]”が特殊文字です。

文字参照

“charref”属性が有効な場合、テキスト文字列中に含まれる文字参照を文字に置き換えます。

文字参照には、文字実体参照と数値実体参照があります。

文字実体参照として、HTML4.0で定義されている表 5.32, 5.33, 5.34 が利用できます。

数値実体参照は、以下の書式で文字を指定します。

&#dddd; または &xhhh;

ここで “dddd” は10進数、“hhhh” は16進数の Unicode コードポイントであり、桁数は任意です。

表 5.32: 文字実体参照 (Latin 1 Characters)

文字	文字実体参照	説明
	 	改行禁止スペース
À	¡	反転させた感嘆符
¢	¢	セント
£	£	ポンド
Âd'	¤	通貨
¥	¥	円
Âe	¦	縦破線
฿	§	セクション
߱	¨	ウムラウト
Âl'	©	著作権
Âł	ª	女性序数
Âń	«	二重山括弧 (開始)
߱	¬	ノット、角ダッシュ
߱	­	ソフトハイフン
Âó	®	登録商標
Âŕ	¯	長音記号
߱	°	度
߱	±	プラスマイナス
߱	²	2乗 (2の上付き文字)
߱	³	3乗 (3の上付き文字)
߱	´	鋭 (揚音) アクセント
߱	µ	マイクロ
߱	¶	パラグラフ
߱	·	中黒
߱	¸	セディーユ
߱	¹	1乗 (1の上付き文字)
߱	º	男性序数
߱	»	二重山括弧 (終了)
߱ij	¼	4分の1
߱i	½	2分の1
߱	¾	4分の3
߱	¿	反転させた疑問符
߱A	À	低 (抑音) アクセントつき A
߱A	Á	鋭 (揚音) アクセントつき A
߱C	Â	曲折アクセントつき A
߱C	Ã	チルダつき A
߱D	Ä	ウムラウトつき A
߱E	Å	リングつき A
߱E	Æ	連字の AE
߱G	Ç	セディーユつき C
߱L	È	低 (抑音) アクセントつき E
߱L	É	鋭 (抑音) アクセントつき E
߱L	Ê	曲折アクセントつき E
߱N	Ë	ウムラウトつき E
߱N	Ì	低 (抑音) アクセントつき I
߱D	Í	鋭 (揚音) アクセントつき I

表 5.32: 文字実体参照 (Lantin 1 Characters : 続き)

文字	文字実体参照	説明
ÃÓ	Î	曲折アクセントつき I
ÃŔ	Ï	ウムラウトつき I
ÃŖ	Ð	古英語のエズ (ETH)
ÃŚ	Ñ	チルダつき N
ÃŚ	Ò	低（抑音）アクセントつき O
ÃŞ	Ó	鋭（揚音）アクセントつき O
ÃŖ	Ô	曲折アクセントつき O
ÃŖ	Õ	チルダつき O
ÃÚ	Ö	ウムラウトつき O
×	×	乗法、かけ算
ÃŶ	Ø	スラッシュつき O
ÃŽ	Ù	低（抑音）アクセントつき U
ÃŽ	Ú	鋭（揚音）アクセントつき U
ÃŽ	Û	曲折アクセントつき U
ÃĲ	Ü	ウムラウトつき U
ÃÍ	Ý	鋭（揚音）アクセントつき Y
Ãđ	Þ	古英語のソーン (THORN)
ÃŞ	ß	連字の sz (ドイツ語など)
Ãă	à	低（抑音）アクセントつき a
Ãą	á	鋭（揚音）アクセントつき a
Ãć	â	曲折アクセントつき a
Ãč	ã	チルダつき a
Ãđ	ä	ウムラウトつき a
Ãě	å	リングつき a
Ãę	æ	連字の ae
Ãğ	ç	セディーユつき c
Ãí	è	低（抑音）アクセントつき e
Ãł	é	鋭（揚音）アクセントつき e
Ãł	ê	曲折アクセントつき e
Ãń	ë	ウムラウトつき e
Ãń	ì	低（抑音）アクセントつき i
Ãń	í	鋭（揚音）アクセントつき i
Ãő	î	曲折アクセントつき i
Ãŕ	ï	ウムラウトつき i
Ãř	ð	古英語のエズ (eth)
Ãś	ñ	チルダつき n
Ãš	ò	低（抑音）アクセントつき o
Ãş	ó	鋭（揚音）アクセントつき o
ÃŖ	ô	曲折アクセントつき o
ÃŖ	õ	チルダつき o
Ãú	ö	ウムラウトつき o
÷	÷	除法、割り算
Ãÿ	ø	チルダつき o
Ãź	ù	低（抑音）アクセントつき u
Ãž	ú	鋭（揚音）アクセントつき u

表 5.32: 文字実体参照 (Lantin 1 Characters : 続き)

文字	文字実体参照	説明
Ãž	û	曲折アクセントつき u
Ãij	ü	ウムラウトつき u
Ãj	ý	鋭 (揚音) アクセントつき y
Ãč	þ	古英語のソーン (thorn)
Ã£	ÿ	ウムラウトつき y

表 5.33: 文字実体参照 (Special Characters)

文字	文字実体参照	説明
"	"	二重引用符
&	&	アンド (アンパサンド)
<	<	小なり
>	>	大なり
'	'	アポストロフィ
ÅŠ	Œ	連字の OE
Å§	œ	連字の oe
Åä	Š	キャロンつき S
Åä	š	キャロンつき s
Åÿ	Ÿ	ウムラウトつき Y
��	ˆ	曲折アクセント
~	˜	チルダ
	 	n 文字幅スペース
	 	m 文字幅スペース
	 	細いスペース
	‌	ゼロ幅ノンジョイナー
	‌	ゼロ幅ジョイナー
	‎	左から右マーク
	&rmlm;	右から左マーク
��	–	n 文字幅ダッシュ
—	—	m 文字幅ダッシュ
'	‘	引用符 (開始)
'	’	引用符 (終了)
��	‚	下付き引用符
“	“	二重引用符 (開始)
”	”	二重引用符 (終了)
��	„	下付き二重引用符
†	†	参照符
‡	‡	二重参照符
%o	‰	千分率 (パーセンテージ)
��	‹	山括弧 (開始)
��	›	山括弧 (終了)
��	€	ユーロ

表 5.34: 文字実体参照 (Symbols)

文字	文字実体参照	説明
A	Α	アルファ
B	Β	ベータ
Г	Γ	ガンマ
Δ	Δ	デルタ
Ε	Ε	イプシロン
Ζ	Ζ	ゼータ
Η	Η	イータ
Θ	Θ	シータ
Ι	Ι	イオタ
Κ	Κ	カッパ
Λ	Λ	ラムダ
Μ	Μ	ミュー
Ν	Ν	ニュー
Ξ	Ξ	クシー (クサイ)
Ο	Ο	オミクロン
Π	Π	パイ
Ρ	Ρ	ロー
Σ	Σ	シグマ
Τ	Τ	タウ
Υ	Υ	ユプシロン
Φ	Φ	ファイ
Χ	Χ	キー (カイ)
Ψ	Ψ	プシー (ブサイ)
Ω	Ω	オメガ
α	α	アルファ
β	β	ベータ
γ	γ	ガンマ
δ	δ	デルタ
ε	ε	イプシロン
ζ	ζ	ゼータ
η	η	イータ
θ	θ	シータ
ι	ι	イオタ
κ	κ	カッパ
λ	λ	ラムダ
μ	μ	ミュー
ν	ν	ニュー
ξ	ξ	クシー (クサイ)
ο	ο	オミクロン
π	π	パイ
ρ	ρ	ロー
ς	ς	ファイナルシグマ
σ	σ	シグマ
τ	τ	タウ
υ	υ	ユプシロン

表 5.34: 文字実体参照 (Symbols : 続き)

文字	文字実体参照	説明
ϕ	φ	ファイ
χ	χ	キー (カイ)
ψ	ψ	プシー (ブサイ)
ω	ω	オメガ
ϑ	ϑ	シータシンボル
(U+03D2)	ϒ	フックつきユーピシロン
(U+03D6)	ϖ	パイシンボル
$\text{â} \ddot{\text{A}}$	•	ブリット (中黒)
…	…	三点リーダー
'	′	プライム符号 (分またはフィート)
"	″	二重プライム符号 (秒またはインチ)
—	‾	オーバーライン (オーバースコア)
$\text{â} \ddot{\text{A}} \ddot{\text{D}}$	⁄	分数のスラッシュ
(U+2118)	℘	手書き風の P
(U+2111)	ℑ	手書き風の I (虚数の I)
(U+211C)	ℜ	手書き風の R (実数の R)
$\text{â} \ddot{\text{D}} \acute{\text{c}}$	™	商標
\aleph	ℵ	アーレフ (第一超限基数)
←	←	左矢印
↑	↑	上矢印
→	→	右矢印
↓	↓	下矢印
↔	↔	左右矢印
(U+21B5)	↵	改行キー (キャリッジリターン)
⇐	⇐	二重左矢印
(U+21D1)	⇑	二重上矢印
⇒	⇒	二重右矢印
(U+21D3)	⇓	二重下矢印
⇒⇒	⇔	二重左右矢印
∀	∀	すべての (数学記号)
∂	∂	偏微分 (数学記号)
∃	∃	存在する (数学記号)
∅	∅	空集合 (数学記号)
∇	∇	ナブラ (数学記号)
∈	∈	要素として含まれる (数学記号)
∉	∉	要素として含まれない (数学記号)
⊑	∋	元として含む (数学記号)
(U+220F)	∏	n の積 (数学記号)
Σ	∑	n の総和 (数学記号)
—	−	マイナス (数学記号)
*	∗	アスタリスク (数学記号)
√	√	平方根 (数学記号)
∞	∝	比例 (数学記号)
∞	∞	無限 (数学記号)
∠	∠	角度 (数学記号)

表 5.34: 文字実体参照 (Symbols : 続き)

文字	文字実体参照	説明
∧	∧	かつ (数学記号)
∨	∨	または (数学記号)
∩	∩	積集合 (数学記号)
∪	∪	和集合 (数学記号)
∫	∫	積分 (数学記号)
∴	∴	ゆえに (数学記号)
āLij	∼	チルダ (数学記号)
≈	≅	およそ等しい (数学記号)
≈	≈	ほぼ等しい (数学記号)
≠	≠	等しくない (数学記号)
≡	≡	合同 (数学記号)
(U+2264)	&le	小なりイコール (数学記号)
(U+2265)	≥	大なりイコール (数学記号)
⊂	⊂	部分集合 (含まれる) (数学記号)
⊃	⊃	部分集合 (含む) (数学記号)
⊄	⊄	部分集合 (含まれない) (数学記号)
⊆	⊆	部分集合 (含まれるか等しい) (数学記号)
⊇	⊇	部分集合 (含むか等しい) (数学記号)
⊕	⊕	丸つき加算記号
⊗	⊗	丸つき乗算記号 (ベクトル積)
⊥	⊥	垂直 (数学記号)
āNĚ	⋅	ドット
(U+2308)	⌈	左シーリング
(U+2309)	⌉	右シーリング
(U+230A)	⌊	左フロアー
(U+230B)	⌋	右フロアー
(U+2329)	⟨	左アングル
(U+232A)	⟩	右アングル
(U+25CA)	◊	ひし形
♠	♠	スペード
♣	♣	クラブ
♥	♥	ハート
♦	♦	ダイアモンド

グリフ参照

Field Reports 独自のグリフ参照形式により、フォントに内蔵されている字形（グリフ）を直接指定することができます。

グリフ参照には、CID/GID 参照とグリフ名参照があります。

使用するフォントがOpenTypeのCJKフォント（拡張子：*.otf）であれば、CIDをキーとしてグリフを指定することができます。CIDとは、CIDフォントが内蔵するすべてのグリフを一意に識別するために、Adobe社が策定した番号です。日本語CIDフォントの場合、Adobe-Japan1文字コレクションにもとづいています。

Adobe-Japan1-6 Character Collections for CID Keyed Fonts, Technical Note #5078 (<http://>

partners.adobe.com/public/developer/en/font/5078.Adobe-Japan1-6.pdf)

使用するフォントがTrueTypeフォント(拡張子: *.ttf または *.ttc)であれば、GID(グリフID)をキーとしてグリフを指定することができます。一般に、GIDはフォント固有の番号体系となっています。
CID/GID参照は、以下の書式で指定します。

&#dddd; または &#xhhhh;

ここで、“dddd”は10進数数、“hhhh”は16進数のCIDまたはGIDを示します。桁数は任意です。
TrueTypeフォントにおいて「グリフ名」が定義されている場合は、グリフ名によりグリフを指定することができます。グリフ名参照は、以下の書式で指定します。

<グリフ名>;

グリフ名が定義されているかどうかは、ttfdump(<http://support.microsoft.com/kb/84224/ja>)などのツールで、TrueTypeフォントの“post”テーブルの内容をダンプすると確認することができます。

現在、グリフ名が定義されていることが確認されているフォントとして、IPAフォント(<http://ossipedia.ipa.go.jp/ipafont/>)とIPAmj明朝フォント(<http://ossipedia.ipa.go.jp/ipamjfont/>)があります。

5.9.3 ボタンフィールド属性

ボタン(画像)フィールドでは、表5.35に示す属性を設定することができます。

表5.35: field辞書のエントリ(ボタンフィールド)

キー	型	値
icon	URL文字列	(オプション) ボタン・フィールドの「アイコン」として表示する画像ファイルのURLを指定します。
image	文字列	(オプション) ボタン・フィールドの「アイコン」として表示する画像のリソース名を指定します。画像リソースは、resources辞書(5.6)で定義します。

5.10 property辞書

PDF帳票に設定するプロパティを表5.36のproperty辞書により与えます。PDFテンプレートが元々持っているプロパティは、生成されるPDF帳票には引き継がれませんので、このproperty辞書で明示的に設定する必要があります。

表 5.36: property 辞書のエントリ

キー	型	値
docinfo	辞書	(オプション) 「文書のプロパティ」として設定する値を指定します。省略時：文書のプロパティの設定を行わない。
metadata	URL 文字列	(オプション) PDF に埋め込む metadata の URL を指定します。省略時：metadata の埋込みを行わない。
encryption	辞書	(オプション) セキュリティの設定を行う場合に、暗号化パラメータを設定します。省略時：セキュリティの設定を行わない。
linearized	真偽値	(オプション) 「Web 表示用に最適化」を行うかどうかを指定します。省略値：False
viewer-preferences	辞書	(オプション) PDF をビューア・アプリケーションで開いた際の「開きかた」を指示します。省略時：「開きかた」の指示を行わない。

metadata は、 docinfo の代わりに XML 形式の詳細な文書情報を埋め込む際に使用します。Adobe 社の定義した Extensible Metadata Platform (XMP) 規格に準拠した XML データを与えます。XMP の規格については下記サイトを参照してください。

<http://www.adobe.com/products/xmp/>

「Web 表示用に最適化」する必要がある場合に、 linearized に True を設定します。

以降では、残りの docinfo と encryption, viewer-preferences について、説明します。

5.10.1 docinfo 辞書

docinfo 辞書では、Adobe Reader の「文書プロパティ」で、主に「概要」として表示される情報を設定します。表 5.37 に docinfo 辞書のエントリを示します。

表 5.37: docinfo 辞書のエントリ

キー	型	値
title	文字列	(オプション) PDF の「文書情報辞書」の「Title」エントリの値を設定します。Adobe Reader の「文書のプロパティ」では「タイトル」として表示されます。
author	文字列	(オプション) PDF の「文書情報辞書」の「Author」エントリの値を設定します。「作成者」として表示されます。
subject	文字列	(オプション) PDF の「文書情報辞書」の「Subject」エントリの値を設定します。「サブタイトル」として表示されます。
keywords	文字列	(オプション) PDF の「文書情報辞書」の「Keywords」エントリの値を設定します。「キーワード」として表示されます。
creator	文字列	(オプション) PDF の「文書情報辞書」の「Creator」エントリの値を設定します。「アプリケーション」として表示されます。
producer	文字列	(オプション) PDF の「文書情報辞書」の「Producer」エントリの値を設定します。省略値: Field Reports
creation-date	文字列	(オプション) PDF の「文書情報辞書」の「CreationDate」エントリの値を設定します。「作成日」として表示されます。省略値: PDF 帳票を生成した時刻
mod-date	文字列	(オプション) PDF の「文書情報辞書」の「ModDate」エントリの値を設定します。「作成日」として表示されます。省略値: PDF 帳票を生成した時刻

5.10.2 encryption 辞書

encryption 辞書では、セキュリティの設定を指定します。表 5.38 に encryption 辞書のエントリを示します。

表 5.38: encryption 辞書のエントリ

キー	型	値
method	列挙値	(必須) 暗号アルゴリズムを指定します。選択値 : RC4, AES
key-length	整数	(オプション) 暗号化キーの長さ(ビット数)を指定します。暗号アルゴリズムに AES を選択した場合は、この項目は無視され、常に 128 が選択されます。選択値 : 40, 128 省略値 : 128.
owner-password	文字列	(オプション) オーナーパスワードを指定します。Adobe Acrobat では、「権限パスワード」と呼ばれます。owner-password または user-password のうちどちらかに 1 文字以上のパスワードを設定する必要があります。省略値 : "" (空文字列).
user-password	文字列	(オプション) ユーザーパスワードを指定します。Adobe Acrobat では、「文書を開くパスワード」と呼ばれます。owner-password または user-password のうちどちらかに 1 文字以上のパスワードを設定する必要があります。省略値 : "" (空文字列).
permissions	整数または列挙値のリスト	(オプション) 文書がユーザーパスワードで開かれるときに許可すべきアクセス権限の種類を指定するフラグのセットを設定します。整数で指定する場合、各アクセス権限は 32 ビット整数のビット位置に対応します。文字列のリストで指定する場合の選択可能な文字列の値は下表のとおりです。省略値 : 0xFFFF (すべて許可).

表 5.39: ユーザーパスワードによるアクセス権限

ビット位置	列挙値	意味
1-2	-	予約: 必ず 0 でなければなりません。
3	Print	高解像度での印刷
4	Edit	(テキスト注釈および対話フォームフィールド以外) 文書内容の変更
5	Copy	文書からのテキストとグラフィックスのコピー
6	Annot	テキスト注釈および対話フォームフィールドの追加または変更
7-8	-	予約: 必ず 1 でなければなりません。
9	Forms	対話フォームフィールドへの値の入力
10	Extract	スクリーンリーダーデバイスのテキストアクセスを有効にする。
11	Assemble	ページの挿入・削除・回転
12	HqPrint	低解像度での印刷
13-32	-	予約: 必ず 1 でなければなりません。

5.10.3 viewer-preferences 辞書

表 5.40 に示す viewer-preferences 辞書では、PDF をビューア・アプリケーションで開いた時の開きかたについての指示を設定します。

表 5.40: viewer-preferences 辞書のエントリ

キー	型	値
hide-toolbar	真偽値	(オプション) ビューア・アプリケーションのツールバーを隠すかどうかを指定します。省略値: false.
hide-menubar	真偽値	(オプション) ビューア・アプリケーションのメニューバーを隠すかどうかを指定します。省略値: false.
hide-window-ui	真偽値	(オプション) 文書ウィンドウのユーザインターフェース要素を隠すかどうかを指定します。省略値: false.
fit-window	真偽値	(オプション) 最初に表示されるページのサイズに適合するように文書ウィンドウのサイズを変更するかどうかを指定します。省略値: false.
center-window	真偽値	(オプション) 文書ウィンドウを画面の中央に配置するかどうかを指定します。省略値: false.
display-doc-title	真偽値	(オプション) 文書ウィンドウのタイトルバーに文書プロパティの title を表示するかどうかを指定します。省略値: false.
non-full-screen-page-mode	列挙値	(オプション) フルスクリーンモードでない時の文書のページモードを指定します。選択値: UseNone, UseOutlines, UseThumbs。省略値: UseNone.
direction	列挙値	(オプション) 文書を読む方向を指定します。選択値: L2R (左から右), R2L (右から左)。省略値: L2R.
view-area	列挙値	(オプション) ページの表示領域となる「境界」を指定します。選択値: MediaBox, CropBox, BleedBox, TrimBox, ArtBox. 省略値: CropBox.
view-clip	列挙値	(オプション) 画面表示時のクリッピング領域となる「境界」を指定します。選択値: MediaBox, CropBox, BleedBox, TrimBox, ArtBox. 省略値: CropBox.
print-area	列挙値	(オプション) 印刷領域となる「境界」を指定します。選択値: MediaBox, CropBox, BleedBox, TrimBox, ArtBox. 省略値: CropBox.
print-clip	列挙値	(オプション) 印刷時のクリッピング領域となる「境界」を指定します。選択値: MediaBox, CropBox, BleedBox, TrimBox, ArtBox. 省略値: CropBox.
print-scaling	列挙値	(オプション) 文書を印刷する際に表示されるプリントダイアログの「印刷倍率」を指定します。選択値: None (印刷倍率を変更しない), AppDefault (現在の印刷倍率を使用する)。省略値: AppDefault.

5.11 環境変数

5.11.1 ユーザ定義環境変数

表 5.41 に示す environ 辞書により、ユーザ定義の環境変数を設定します。

表 5.41: environ 辞書のエントリ

キー名	型	値
環境変数名	文字列または数値	環境変数を定義し、任意の値を与えます。

5.11.2 システム定義環境変数

システム定義されている環境変数の一覧を表 5.42 に示します。

表 5.42: 既定の環境変数

変数名	値
PAGE	現在のページ数（0 はじまり）
PAGE+	現在のページ数（1 はじまり）
NUM_PAGES	全ページ数
NOW	現在時刻

PAGE, NUM_PAGES を使って、帳票全体を通したページ番号を取得できます。Field Reports 内部では、ページ番号を 0 始まりの数値で管理しているので、1 始まりのページ番号を取得したい場合は、PAGE+ を使用してください。

5.12 settings 辞書

Field Reports の設定情報を表 5.43 の settings 辞書により与えます。

表 5.43: settings 辞書のエントリ

キー	型	値
template-root	文字列	PDF テンプレート・画像・フォント・metadata ファイルが格納されているディレクトリを設定します。省略値：“”。
serial-number	文字列	製品ご購入時に発行されたシリアル番号を設定します。
auth-code	文字列	ライセンス認証手続きにより発行されたライセンスキーを設定します。

第 6 章

API リファレンス

Field Reports で利用可能な API の一覧を表 6.1 に示します。

表 6.1: API 一覧

分類	提供形態	説明
LL 言語 Bridge	Python Bridge	Python 2.6 以上
	Ruby Bridge	Ruby 1.8.7 以上
	Perl Bridge	Perl 5.8 以上
	PHP Bridge	PHP 5.1 以上
VM 言語 Bridge	Java Bridge	JDK 1.6 以上
	.NET Framework Bridge	.NET Framework 2.0 以上 (Windows 版のみに付属)
コマンドライン I/F	実行ファイル	外部プロセスとして呼び出し可能
低レベル I/F	C ライブラリ	C 共有ライブラリ
	OCaml ライブラリ	OCaml 3.12

なお、「低レベル I/F」に関しましては、サポート対象外の扱いとなります。ご了承願います。

6.1 Python Bridge

6.1.1 field.reports モジュール

version()

バージョン番号を取得します。

set_log_level(*n*)

ログ出力のレベルを設定します。有効な値の範囲は 0~4 です (0: ログを出力しない, 1: ERROR ログを出力する, 2: WARN ログを出力する, 3: INFO ログを出力する, 4: DEBUG ログを出力する)。1 以上の値を設定した場合、標準エラー出力にログを出力します (初期値: 0)。

set_defaults(*param*)

レンダリングパラメータのデフォルト値を設定します。*param* の型が辞書の場合は、レンダリングパラメータが Python ネイティブのデータ・オブジェクトで表現されているものとします。*param* が文字列の場合は、JSON 形式の文字列で記述されているものとします。バイト文字列を使用する場合は、エンコーディングを UTF-8 してください。

```
renders( param )
レンダリングパラメータ param を元にレンダリングを実行し、結果をバイト文字列として返します。
param の型として辞書または文字列を受け付けます。

render( param , filename )
レンダリングパラメータ param を元にレンダリングを実行します。処理結果は、filename で指定した
ファイルに出力されます。param の型として辞書または文字列を受け付けます。
```

6.2 Ruby Bridge

6.2.1 Field::Reports モジュール

```
version
バージョン番号を取得します。

set_log_level( n )
ログ出力のレベルを設定します。有効な値の範囲は 0~4 です (0: ログを出力しない, 1: ERROR ログ
を出力する, 2: WARN ログを出力する, 3: INFO ログを出力する, 4: DEBUG ログを出力する)。1
以上の値を設定した場合、標準エラー出力にログを出力します (初期値 : 0)。

set_defaults( param )
レンダリングパラメータのデフォルト値を設定します。param の型がハッシュの場合は、レンダリン
グパラメータがRuby ネイティブのデータ・オブジェクトで表現されているものとします。param が
文字列の場合は、JSON 形式の文字列で記述されているものとします。Ruby1.8 の場合は、文字列デー
タのエンコーディングを UTF-8 してください。

renders( param )
レンダリングパラメータ param を元にレンダリングを実行し、結果をバイト文字列として返します。
param の型としてハッシュまたは文字列を受け付けます。

render( param , filename )
レンダリングパラメータ param を元にレンダリングを実行します。処理結果は、filename で指定した
ファイルに出力されます。param の型としてハッシュまたは文字列を受け付けます。
```

6.3 Perl Bridge

6.3.1 Field::Reports モジュール

```
version
バージョン番号を取得します。

set_log_level( n )
ログ出力のレベルを設定します。有効な値の範囲は 0~4 です (0: ログを出力しない, 1: ERROR ログ
を出力する, 2: WARN ログを出力する, 3: INFO ログを出力する, 4: DEBUG ログを出力する)。1
以上の値を設定した場合、標準エラー出力にログを出力します (初期値 : 0)。

set_defaults( param )
```

レンダリングパラメータのデフォルト値を設定します。*param* の型がハッシュリファレンスの場合は、レンダリング・パラメータが Perl ネイティブのデータ・オブジェクトで表現されているものとします。*param* が文字列の場合は、JSON 形式の文字列で記述されているものとします。文字列データのエンコーディングは、UTF-8 としてください。

renders(*param*)

レンダリングパラメータ *param* を元にレンダリングを実行し、結果をバイト文字列として返します。*param* の型としてハッシュリファレンスまたは文字列を受け付けます。

render(*param* , *filename*)

レンダリングパラメータ *param* を元にレンダリングを実行します。処理結果は、*filename* で指定したファイルに出力されます。*param* の型としてハッシュリファレンスまたは文字列を受け付けます。

6.4 PHP Bridge

6.4.1 php_reports モジュール

fr_version()

バージョン番号を取得します。

fr_set_log_level(*int \$loglevel*)

ログ出力のレベルを設定します。有効な値の範囲は 0~4 です (0: ログを出力しない, 1: ERROR ログを出力する, 2: WARN ログを出力する, 3: INFO ログを出力する, 4: DEBUG ログを出力する)。1 以上の値を設定した場合、標準エラー出力にログを出力します (初期値: 0)。

fr_set_defaults(*mixed \$param*)

レンダリングパラメータのデフォルト値を設定します。*\$param* の型が配列場合は、レンダリングパラメータが PHP ネイティブのデータ・オブジェクトで表現されているものとします。*\$param* が文字列の場合は、JSON 形式の文字列で記述されているものとします。文字列データのエンコーディングは、UTF-8 としてください。

fr_renders(*mixed \$param*)

レンダリングパラメータ *\$param* を元にレンダリングを実行し、結果をバイト文字列として返します。*\$param* の型として配列または文字列を受け付けます。

fr_render(*mixed \$param* , *string \$filename*)

レンダリングパラメータ *\$param* を元にレンダリングを実行します。処理結果は、*\$filename* で指定したファイルに出力されます。*\$param* の型として配列または文字列を受け付けます。

6.5 Java Bridge

6.5.1 jp.co.field_works.Reports クラス

public static String version() throws ReportsException;

バージョン番号を取得します。

public static void setLogLevel(*int n*) throws ReportsException;

ログ出力のレベルを設定します。有効な値の範囲は 0~4 です (0: ログを出力しない, 1: ERROR ログを出力する, 2: WARN ログを出力する, 3: INFO ログを出力する, 4: DEBUG ログを出力する)。1 以上の値を設定した場合、標準エラー出力にログを出力します (初期値 : 0)。

```
public static void setDefaults(String param) throws ReportsException;
```

レンダリングパラメータのデフォルト値を設定します。レンダリングパラメータは、JSON 形式の文字列で与えます。

```
public static byte[] renders(String param) throws ReportsException;
```

レンダリングパラメータ param を元にレンダリングを実行し、結果をバイト文字列として返します。

レンダリングパラメータは、JSON 形式の文字列で与えます。

```
public static void render(String param, String filename) throws ReportsException;
```

レンダリングパラメータ param を元にレンダリングを実行します。処理結果は、filename で指定したファイルに出力されます。レンダリングパラメータは、JSON 形式の文字列で与えます。

6.6 .NET Framework Bridge

6.6.1 Field.Reports クラス

```
public unsafe String Version();
```

バージョン番号を取得します。

```
public unsafe void SetLogLevel(int n);
```

ログ出力のレベルを設定します。有効な値の範囲は 0~4 です (0: ログを出力しない, 1: ERROR ログを出力する, 2: WARN ログを出力する, 3: INFO ログを出力する, 4: DEBUG ログを出力する)。1 以上の値を設定した場合、標準エラー出力にログを出力します (初期値 : 0)。

```
public unsafe void SetDefaults(String param);
```

レンダリングパラメータのデフォルト値を設定します。レンダリングパラメータは、JSON 形式の文字列で与えます。

```
public unsafe byte[] Renders(String param);
```

レンダリングパラメータ param を元にレンダリングを実行し、結果をバイト文字列として返します。

レンダリングパラメータは、JSON 形式の文字列で与えます。

```
public unsafe void Render(String param, String filename);
```

レンダリングパラメータ param を元にレンダリングを実行します。処理結果は、filename で指定したファイルに出力されます。レンダリングパラメータは、JSON 形式の文字列で与えます。

6.6.2 COM コンポーネント

Field.Reports クラスは、COM 参照可能な設定でビルドされています。Field Reports 本体のインストール時に COM コンポーネントとして登録されますので、COM 呼び出し可能なプログラミング言語から呼び出すことができます。

以下に JScript でのプログラミング例を示します。この例ではレンダリングパラメータを文字列として記述していますが、WSH 5.8 以降で利用可能な JSON.stringify メソッドを用いれば、JSON Object を JSON 文字列に簡単に変換できます。

```
var reports = new ActiveXObject("Field.Reports");

var param = '{ \
    "template": "./mitumori.pdf", \
    \
    "context": { \
        "date": "平成 23 年 1 月 22 日", \
        "number": "10R0001", \
        "to": "△△△惣菜株式会社", \
        "title": "肉じゃがの材料", \
        "delivery_date": "平成 23 年 1 月 22 日", \
        "delivery_place": "貴社指定場所", \
        "payment_terms": "銀行振込", \
        "expiration_date": "発行から 3 ヶ月以内", \
        "stamp1": {"icon": "./stamp.png"}, \
        "table": [ \
            ["1", "N001", "牛肉（切り落とし）", "200g", "250 円", "500 円"], \
            ["2", "Y001", "じゃがいも（乱切り）", "3 個", "30 円", "90 円"], \
            ["3", "Y002", "にんじん（乱切り）", "1 本", "40 円", "40 円"], \
            ["4", "Y003", "たまねぎ（くし切り）", "1 個", "50 円", "50 円"], \
            ["5", "Y004", "しらたき", "1 袋", "80 円", "80 円"], \
            ["6", "Y005", "いんげん", "1 袋", "40 円", "40 円"] \
        ], \
        "sub_total": "800 円", \
        "tax": "40 円", \
        "total": "840 円" \
    } \
};';

reports.SetLogLevel(3);
reports.Version();

reports.Render(param, "out.pdf");
```

実行する際は、文字コードを Shift JIS でファイルに保存し、cscript の引数として与えます（ここでは、ファイル名を ‘test.js’ とした）。

```
> cscript test.js
```

6.7 コマンドライン I/F

6.7.1 reports コマンド

Field Reports のコマンドライン・プログラムは、以下の書式で呼び出します。

```
reports <サブコマンド>[<オプション>][<引数>]
```

以下のサブコマンドが利用できます。

```
create <入力ファイル名> <出力ファイル名>
```

PDF 帳票を生成します。<入力ファイル名> には、JSON 形式で記述したレンダリングパラメータのファイル名を指定します。“-” を指定した場合、標準入力からレンダリングパラメータを取得します。<出力ファイル名> には、PDF 帳票を保存するファイル名を指定します。“-” を指定した場合、生成した PDF の内容を標準出力へ出力します。

```
info <入力ファイル名>
```

PDF の情報を解析して表示します。<入力ファイル名> には、解析対象となる PDF のファイル名を指定します。

```
font <フォントファイル名> ...
```

指定したフォントファイルの情報を表示します。<フォントファイル名> は、スペースで区切って複数指定することができます。-j オプションを指定すると、リソース定義要素の雛形を JSON 形式で出力します。

```
activate <シリアル番号>
```

ライセンス認証時に必要となるチェックコードを生成します。

```
check
```

登録されたライセンスキードが有効かどうか確認します。

各サブコマンドで有効なオプションについては、以下のコマンドを実行して確認してください。

```
$ reports <サブコマンド> ----help
```

6.8 C I/F

6.8.1 API 一覧

Field Reports の共有ライブラリでは、以下の API を公開しています。

```
void rp_init(void)
```

Field Reports を初期化します。最初に一度だけ呼び出してください。

```
caml_value* rp_version(void)
```

Field Reports のバージョンを文字列で取得します。返り値：OCaml 文字列。

caml_value* rp_set_log_level(value n)

ログ出力のレベルを整数で設定します。引数 n として有効な値の範囲は 0~4 です（0: ログを出力しない、1: ERROR ログを出力する、2: WARN ログを出力する、3: INFO ログを出力する、4: DEBUG ログを出力する）。1 以上の値を設定した場合、標準エラー出力にログを出力します。初期値:0。返り値：なし。

caml_value* rp_set_defaults(caml_value* param)

レンダリングパラメータのデフォルト値を設定します。引数 json には、レンダリングパラメータを JSON オブジェクトとして渡します。返り値：なし。

caml_value* rp_pdf_of_json(caml_value* param)

引数で与えたレンダリングパラメータを元に PDF 帳票のレンダリングを行います。結果は、OCaml 文字列として返されます。引数 json には、OCaml 形式の JSON オブジェクトとして、レンダリングパラメータを渡します。返り値：OCaml 文字列。

caml_value* rp_write_pdf_from_json(caml_value* param, const char* filename)

引数で与えたレンダリングパラメータを元に PDF 帳票のレンダリングを行います。結果は、引数 filename で与えたファイル名のファイルを作成して出力します。引数 json には、OCaml 形式の JSON オブジェクトとして、レンダリングパラメータを渡します。返り値：なし。

caml_value* rp_encode_json(rp_callback* cb, rp_val v)

プログラミング言語固有のデータオブジェクトを OCaml 形式の JSON オブジェクトに変換します。cb には、データ型を判定したり、データ変換を行うためのコールバック関数を登録します。返り値：OCaml 形式 JSON オブジェクト。

caml_value* rp_json_of_string(const char* param)

JSON 文字列を OCaml 形式の JSON オブジェクトに変換します。返り値：JSON オブジェクト。

void rp_free(caml_value* val)

OCaml 形式のデータオブジェクトを開放します。

int rp_is_error(caml_value* val)

OCaml 形式のデータオブジェクトを元に例外の発生有無を判定します。戻り値：真偽値（0：正常終了、0 以外：例外発生）。

const char* rp_get_error(caml_value* val, int* err, char* msg)

例外オブジェクトからエラー情報を抽出します。引数 val には、各 API の返り値を渡します。err にはエラーレベル番号、msg にはエラーメッセージ、返り値にはエラーの種別を示す文字列（タグ）が設定されます。msg には、呼び出し側で RP_MAX_ERR_MSG_LEN バイト以上の文字列バッファを確保してください。

unsigned int rp_get_string_val(caml_value* val, char* buf)

OCaml 文字列より、C 形式のバイト文字列を抽出します。引数 val には、OCaml 文字列のポインタを設定します。引数 buf で指定した文字列バッファにバイト文字列がコピーされ、返り値には文字列バッファに必要なサイズ（バイト数）が返されます。buf に NULL を設定すると、必要なサイズのみ取得できます。

6.8.2 caml_value 型について

caml_value は、Objective Caml(OCaml) オブジェクトを格納するための型です。

各 API の返り値として受け取った OCaml オブジェクトは、呼び出し元の責任で rp_free() を使って解放する必要があります。各 API の処理中にエラー（例外）が発生した場合は、戻り値として OCaml 形式の例外オブジェクトが返されます。処理結果がエラーかどうかを判定するには、rp_is_error() 関数を使用してください。

OCaml と C とのインターフェースについての詳細は、OCaml のユーザーズ・マニュアルを参照してください (<http://caml.inria.fr/pub/docs/manual-ocaml/index.html>)。

6.8.3 コールバック関数について

rp_encode_json() 関数の引数として、以下に示すコールバック関数を格納した構造体へのポインタを設定します。

```
typedef int (*t_fn_type)(rp_val v);
typedef int (*t_fn_val_bool)(rp_val v, rp_val* opt);
typedef int (*t_fn_val_int)(rp_val v, rp_val* opt);
typedef double (*t_fn_val_float)(rp_val v, rp_val* opt);
typedef char* (*t_fn_val_string)(rp_val v, rp_val* opt);
typedef rp_pos (*t_fn_array_head)(rp_val v, rp_val* opt);
typedef int (*t_fn_array_next)(rp_val v, rp_pos* pos, rp_val* entry, rp_val* opt);
typedef rp_pos (*t_fn_object_head)(rp_val v, rp_val* opt);
typedef int (*t_fn_object_next)(rp_val v, rp_pos* pos, char** key, rp_val* entry, rp_val* opt);
typedef void (*t_fn_release)(rp_val opt);

typedef struct {
    t_fn_type fn_type;
    t_fn_val_bool fn_val_bool;
    t_fn_val_int fn_val_int;
    t_fn_val_float fn_val_float;
    t_fn_val_string fn_val_string;
    t_fn_array_head fn_array_head;
    t_fn_array_next fn_array_next;
    t_fn_object_head fn_object_head;
    t_fn_object_next fn_object_next;
    t_fn_release fn_release;
} rp_callback;
```

`rp_val` は変換元データオブジェクトへのポインタを格納するための型であり、`rp_pos` は配列・辞書型オブジェクトを列挙する際の位置情報を格納するための型です。

各コールバック関数の定義は以下のとおりです。

int fn_type(rp_val v)

変換元データオブジェクト `v` の型を判定します。返り値：列挙値（NULL 型：RP_TYPE_NULL, 真偽型：RP_TYPE_BOOL, 整数型：RP_TYPE_INT, 実数型：RP_TYPE_FLOAT, 文字列型：RP_TYPE_STRING, 配列型：RP_TYPE_ARRAY, 辞書型：RP_TYPE_OBJECT, その他：RP_TYPE_OTHER）。RP_TYPE_OTHER を返すと、データ変換処理は失敗し変換処理を中断します。

int fn_val_bool(rp_val v, rp_val* opt)

変換元データオブジェクト `v` の真偽値としての値を取得します。`opt` には、メモリ解放などの後処理が必要なオブジェクトへのポインタをセットします。後処理が不要な場合は NULL をセットします。返り値：真偽値（0：偽, 0以外：真）。

int fn_val_int(rp_val v, rp_val* opt)

変換元データオブジェクト `v` の整数としての値を取得します。`opt` には、後処理が必要なオブジェクトへのポインタをセットします。返り値：整数值。

double fn_val_float(rp_val v, rp_val* opt)

変換元データオブジェクト `v` の実数としての値を取得します。`opt` には、後処理が必要なオブジェクトへのポインタをセットします。返り値：実数値。

char* fn_val_string(rp_val v, rp_val* opt)

変換元データオブジェクト `v` の文字列としての値を取得します。`opt` には、後処理が必要なオブジェクトへのポインタをセットします。返り値：文字配列へのポインタ。

rp_pos fn_array_head(rp_val v, rp_val* opt)

配列型データオブジェクト `v` の先頭位置を取得します。`opt` には、メモリ解放などの後処理が必要なオブジェクトへのポインタをセットします。返り値：位置情報。

int fn_array_next(rp_val v, rp_pos* pos, rp_val* entry, rp_val* opt)

配列型データオブジェクト `v` の現在位置の値を取得し、`entry` にセットします。現在位置をひとつ進め、`pos` にセットします。進めます。位置情報が最後の位置を超えた場合、`pos` に NULL をセットし、返り値に 0（偽）を返します。`opt` には、後処理が必要なオブジェクトへのポインタをセットします。返り値：真偽値（0：偽, 0以外：真）。

rp_pos fn_object_head(rp_val v, rp_val* opt)

辞書型データオブジェクト `v` の先頭位置を取得します。`opt` には、後処理が必要なオブジェクトへのポインタをセットします。返り値：位置情報。

int fn_object_next(rp_val v, rp_pos* pos, char key, rp_val* entry, rp_val* opt)**

辞書型データオブジェクト `v` の現在位置の値を取得し、`entry` にセットします。現在位置をひとつ進め、`pos` にセットします。進めます。位置情報が最後の位置を超えた場合、`pos` に NULL をセットし、返り値に 0（偽）を返します。`opt` には、後処理が必要なオブジェクトへのポインタをセットします。返り値：真偽値（0：偽, 0以外：真）。

void fn_release(rp_val opt)

opt の後処理を行います。opt が NULL の場合にも fn_release() は呼び出されます。

6.9 OCaml I/F

6.9.1 Field.Reports モジュール

```
val reports_version : string
  バージョン番号を取得します。
val set_log_level : int -> unit
  ログ出力のレベルを設定します。有効な値の範囲は 0~4 です (0: ログを出力しない, 1: ERROR ログ
  を出力する, 2: WARN ログを出力する, 3: INFO ログを出力する, 4: DEBUG ログを出力する)。1
  以上の値を設定した場合, 標準エラー出力にログを出力します。初期値: 0.
val set_defaults : Field.Jsonwheel.Json_type.t -> unit
  レンダリングパラメータのデフォルト値を設定します。
val pdf_of_json : Field.Jsonwheel.Json_type.json_type -> string
  JSON オブジェクトを元に PDF を生成します。
val pdf_of_jsons : string -> string
  JSON 文字列を元に PDF を生成します。
val write_pdf_from_json : Field.Jsonwheel.Json_type.json_type -> string -> unit
  JSON オブジェクトを元に PDF を生成し, ファイルに出力します。
val write_pdf_from_jsons : string -> string -> unit
  JSON 文字列を元に PDF を生成し, ファイルに出力します。
val json_of_string : string -> Field.Jsonwheel.Json_type.json_type
  JSON 文字列を JSON オブジェクトへ変換します。
```

OCaml I/F でのご使用はサポート対象外となります。

付録 A

言語 Bridge のビルド手順

インストール媒体には拡張モジュールのソースファイルが添付されています。ソースファイルからビルドすることで、ご使用の動作環境に適合した拡張モジュールを構築することができます。

A.1 前提条件

- Field Reports 本体のインストールならびに必要な環境変数等の設定が完了していること。

なお、想定している言語処理系ならびに Java 実行環境、開発環境については、各言語 Bridge のビルド手順を参照してください。

A.2 Python

ここでは、MinGW を用いて Python Bridge をビルドする手順を説明します。

A.2.1 準備

1. MinGW

MinGW 開発環境を入手してインストールしてください（公式サイト：<http://www.mingw.org/>）。

A.2.2 環境変数の設定

MinGW のインストールディレクトリ（ここでは “C:\mingw” とする）を環境変数 PATH へ追加してください。また、Field Reports のヘッダファイルとインポートライブラリの格納ディレクトリを環境変数 CPATH と LIBRARY_PATH へそれぞれ設定してください。

```
> set MINGW_ROOT=C:\mingw
> set PATH=%MINGW_ROOT%\bin;%PATH%
> set CPATH=%PROGRAMFILES%\Field Works\Field Reports x.x\include
> set LIBRARY_PATH=%PROGRAMFILES%\Field Works\Field Reports x.x\lib
```

A.2.3 ビルド手順

ソースファイル媒体 “field.reports-x.x.zip” を展開してから、以下のコマンドを実行してください。

```
> cd field.reports-x.x  
> python setup.py build ----compiler=mingw32
```

A.2.4 インストール手順

管理者権限で、以下のコマンドを実行してください。

```
> python setup.py install
```

注意事項

- 動作確認を行う場合は、ビルドを実行した場所とは別のディレクトリで行ってください。

A.2.5 アンインストール手順

Python のインストールディレクトリ以下にある “site-package” ディレクトリに作成された “field” ディレクトリを削除してください。

“site-package” ディレクトリの場所が不明な場合は、以下のコマンドで確認してください。

```
> python  
>>> import sys  
>>> sys.path
```

A.3 Ruby

ここでは、RubyInstaller でインストールした Ruby 処理系に、RubyInstaller Development Kit (DevKit) を用いて Ruby Bridge をインストールする手順を説明します。

A.3.1 準備

1. DevKit

RubyInstaller のサイト (<http://rubyinstaller.org/>) より、Development Kit をダウンロードしてください。

2. DevKit をインストールしてください（ここでは “C:\devkit” に展開したものとします）。

A.3.2 DevKit の初期設定

DevKit のトップディレクトリで、以下のコマンドを実行してください。

```
> ruby dk.rb init  
> ruby dk.rb install
```

A.3.3 環境変数の設定

Field Reports のヘッダファイルとライブラリの格納ディレクトリを環境変数 CPATH と LIBRARY_PATH へそれぞれ設定してください（“x.x” は Field Reports のバージョンを示します）。

```
> set CPATH=%PROGRAMFILES%\Field Works\Field Reports x.x\include  
> set LIBRARY_PATH=%PROGRAMFILES%\Field Works\Field Reports x.x\lib
```

A.3.4 ビルドならびにインストールの手順

管理者権限で、以下のコマンドを実行してください。

```
> gem install reports-x.x.x.gem -V
```

A.3.5 アンインストール手順

管理者権限で、以下のコマンドを実行してください。

```
> gem uninstall reports
```

A.4 Perl

ここでは、Strawberry Perl (32 ビット) を用いて Perl Bridge をビルドする手順を説明します。

A.4.1 準備

1. Field Reports 本体

Field Reports 本体のインストールと環境変数 PATH の設定を適切に行なってください (2.2 参照)。

2. Strawberry Perl

Strawberry Perl のサイト (<http://strawberryperl.com/>) より、インストール媒体をダウンロードしてインストールしてください。

A.4.2 環境変数の設定

Strawberry Perl の実行ファイル格納ディレクトリ (ここでは Strawberry Perl のインストールディレクトリを “C:\strawberry” とした) を環境変数 PATH へ追加してください。また、Field Reports のヘッダファイルとインポートライブラリの格納ディレクトリを環境変数 CPATH と LIBRARY_PATH へそれぞれ設定してください。

```
> set PERL_ROOT=C:\strawberry  
> set PATH=%PERL_ROOT%\perl\bin;%PERL_ROOT%\c\bin;%PATH%  
> set CPATH=%PROGRAMFILES%\Field Works\Field Reports x.x\include  
> set LIBRARY_PATH=%PROGRAMFILES%\Field Works\Field Reports x.x\lib
```

A.4.3 ビルド手順

ソースファイル媒体 “Field-Reports-x.x.x.tar.gz” を解凍してから、以下のコマンドを実行してください。

```
> cd Field-Reports-x.x.x  
> perl Makefile.PL  
> dmake
```

A.4.4 インストール手順

管理者権限で、以下のコマンドを実行してください。

```
> dmake install
```

A.4.5 アンインストール手順

管理者権限で、以下のコマンドを実行してください。

```
> dmake uninstall
```

下記のようなメッセージが表示される場合は、メッセージにしたがって手動でファイルを削除してください。

```
Uninstall is unsafe and deprecated, the uninstallation was not performed.  
We will show what would have been done.
```

```
unlink C:\strawberry\perl\site\lib\Field\Reports.pm  
unlink C:\strawberry\perl\site\lib\auto\Field\Reports\Reports.bs  
unlink C:\strawberry\perl\site\lib\auto\Field\Reports\Reports.dll  
unlink C:\strawberry\perl\site\lib\auto\Field\Reports\.packlist
```

```
Uninstall is unsafe and deprecated, the uninstallation was not performed.  
Please check the list above carefully, there may be errors.  
Remove the appropriate files manually.  
Sorry for the inconvenience.
```

A.5 PHP

PHP では、PHP 本体をビルドしたものと同じバージョンの C コンパイラを用いて拡張モジュールをビルドする必要があります。

ここでは、PHP5.3 用の拡張モジュールを Visual Studio 2008 でビルドする手順を説明します。

A.5.1 準備

1. Visual C++ 2008

Visual C++ 2008 (Express も可) を入手してインストールしてください。

2. PHP ソース

PHP 実行環境と同じバージョンの PHP ソースを入手してください (<http://www.php.net/downloads.php>)。

3. 作業用フォルダ

ビルドを行うための作業用フォルダを作成してください (ここでは “C:\php-sdk” とする)。

4. binary-tools

binary-tools (<http://windows.php.net/downloads/php-sdk/>) を入手して、作業用フォルダ内に展開してください (“C:\php-sdk\bin” と “C:\php-sdk\script” が作成されます)。また、deps-5.3-vc9-x86.7z もダウンロードして、C:\php-sdk\deps に展開してください。

5. ソース展開用ディレクトリ

スタートメニューより “Visual Studio Tools/Visua Studio 2008 コマンドプロンプト” を開いて、以下のコマンドを実行してください。

```
> cd c:\php-sdk  
> bin\phpsdk_setvars.bat  
> bin\phpsdk_buildtree.bat php53dev
```

6. PHP ソースの展開

PHP ソースを “C:\php-sdk\php53dev\vc9\x86\php-5.3.x” 配下に展開してください。

7. ヘッダファイルの生成

“Visua Studio 2008 コマンドプロンプト” の中で、以下のコマンドを実行してください。

```
> cd c:\php-sdk\php53dev\vc9\x86\php-5.3.x  
> buildconf.bat  
> configure.bat
```

ヘッダファイル “main\config.w32.h” が作成されれば成功です。

A.5.2 ビルド手順

1. PHP Bridge ソースファイルの展開

ソースファイル媒体 “php_reports-1.x.x.zip” を任意の場所に展開してください。

2. 環境変数の設定

REPORTS_HOME に Field Reports のインストールディレクトリ(通常“C:\Program Files\Field Reports 1.x”)を設定してください。次に、PHP_SRC に PHP ソースを展開したディレクトリ（上の例では “C:\php-sdk\php53dev\vc9\x86\php-5.3.x”）を、PHP_HOME に PHP のインストールディレクトリ (“C:\php” など）を設定してください。

3. ソリューションファイルのオープン

ソリューションファイル “msvc-5.3\reports.sln” を Visual Studio 2008 で開いてください。

4. ビルド

「ビルド/バッチ ビルド」メニューを選択し、必要な構成にチェックを入れてください。「ビルド」ボタンを押せばビルドが実行されます。

A.5.3 インストール手順

Release(x86) もしくは Release_TS(x86) ディレクトリに作成された DLL ファイルを拡張モジュール格納場所にコピーし、php.ini の設定を行ってください。

A.5.4 アンインストール手順

拡張モジュール格納場所へコピーした “php_reports_ts.dll” または “php_reports.dll” ファイルを削除してください。

既定の拡張モジュール格納場所が不明な場合は、コマンドプロンプトから以下のコマンドを実行して確認してください。

```
> php -i | find "extension_dir"
```

さらに、PHP 設定ファイル (php.ini) に追加した “extension = ...” の行を削除してください。

A.6 Java

ここでは、Ruby Bridge のビルドの際にも使用した DevKit を流用して Java Bridge をビルドする手順を説明します。

すでにビルド環境が整っている場合は、そちらを使用してください。

A.6.1 準備

1. DevKit

RubyInstaller のサイト (<http://rubyinstaller.org/>) より、Development Kit をダウンロードしてインストールしてください。

A.6.2 環境変数の設定

JDK のインストールディレクトリ (ここでは、“C:\JDK” とする) を環境変数 JAVA_HOME に設定し、MinGW の実行ファイル格納ディレクトリ (ここでは “C:\mingw\bin” とする) を環境変数 PATH へ追加してください。また、Field Reports のヘッダファイルとインポートライブラリの格納ディレクトリを環境変数 CPATH と LIBRARY_PATH へそれぞれ設定してください。

```
> set JAVA_HOME=C:\JDK  
> set DEVKIT_ROOT=C:\devkit  
> set PATH=%JAVA_HOME%\bin;%DEVKIT_ROOT%\bin;%DEVKIT_ROOT%\mingw\bin;%PATH%  
> set CPATH=%PROGRAMFILES%\Field Works\Field Reports x.x\include  
> set LIBRARY_PATH=%PROGRAMFILES%\Field Works\Field Reports x.x\lib
```

A.6.3 ビルド手順

ソースファイル媒体 “jni_reports-x.x.zip” を展開してから、以下のコマンドを実行してください。

```
> cd jni_reports-x.x.x  
> make
```

A.6.4 インストール手順

管理者権限で、以下のコマンドを実行してください。

```
> make install
```

A.6.5 アンインストール手順

拡張ディレクトリへコピーした JINI ライブラリ “libReports.dll” と jar ファイル “reports.jar” を削除してください。

付録 B

依存ライブラリ

B.1 OCaml ライブラリ

本ソフトウェアの本体はプログラミング言語 OCaml で実装されており、表 B.1 のライブラリを利用しています。

表 B.1: 利用している OCaml ライブラリの一覧

名称	版数	開発元情報・ライセンス条件等
OCaml 標準ライブラリ	4.02.1	Institut National de Recherche en Informatique et en Automatique (INRIA) http://caml.inria.fr/ocaml/ LGPL with linking exceptions
CamlPDF	0.5	Coherent Graphics Ltd. http://www.coherentpdf.com/ BSD licence with special exceptions
ExtLib	1.5.1	Nicolas Cannasse 他 http://code.google.com/p/ocaml-extlib/ LGPL with linking exceptions
json-wheel	1.0.6	Wink Technologies, Inc., Martin Jambon http://martin.jambon.free.fr/json-wheel.html BSD license
cryptokit	1.4	Xavier Leroy. http://forge.ocamlcore.org/projects/cryptokit/ GNU Lesser General Public with static compilation exception

B.2 C ライブライ

本ソフトウェアの本体または言語 Bridge は、表 B.2 のライブラリを利用しています。

表 B.2: 利用している C ライブライの一覧

名称	版数	開発元情報・ライセンス条件等
zlib	1.2.5	Jean-loup Gailly and Mark Adler. http://zlib.net/ zlib License
libcurl	7.12.1 以上	http://curl.haxx.se/ MIT/X derivate license.