「日医標準レセプトソフト」

ORCA Project

日レセプラグイン

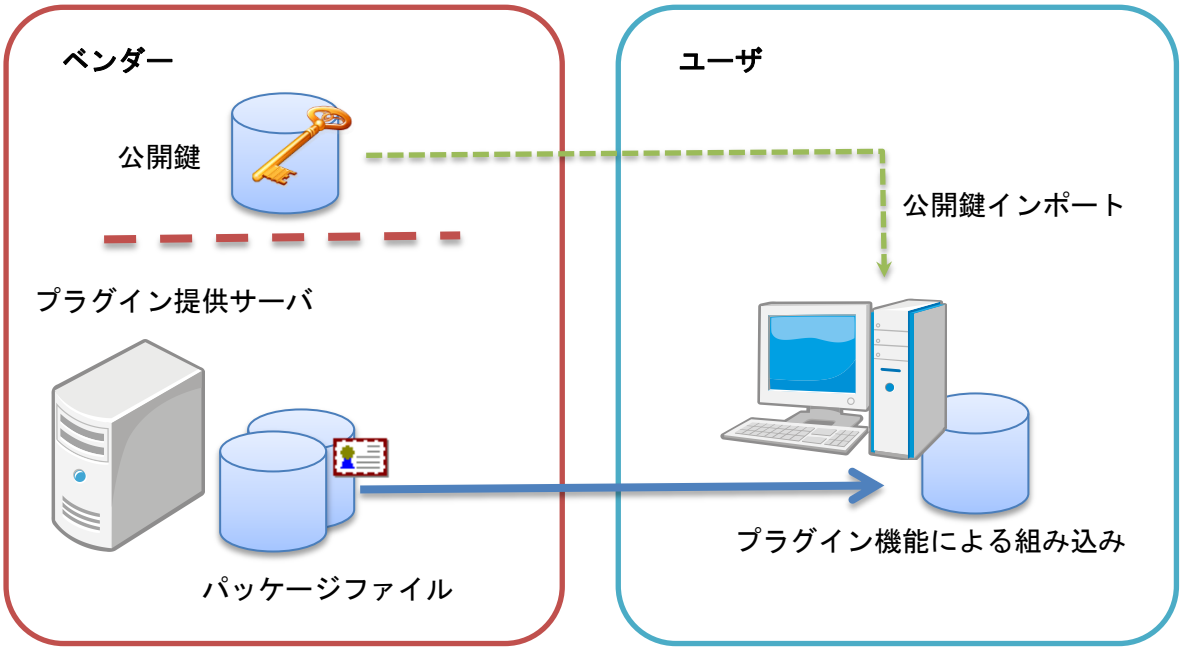
2013 年 9 月 19 日

公益社団法人日本医師会

１　日レセプラグインの概要

日レセプラグインとは、目的の機能を有する外部プログラムを日レセシステム内へ組み込みしたり、取り外したりすることを可能とする機能のことを指します。
Ver4.5.0 より試験的に実装し、Ver4.7.0 より本格的にサービス提供となりました。

ORCA プロジェクトからは、地方公費や公開帳票のプログラムをプラグインパッケージとして提供していますが、各ベンダーにおいては、サポートユーザへ提供するカスタマイズプログラムをプラグインパッケージ化して配布することを想定した機能です。

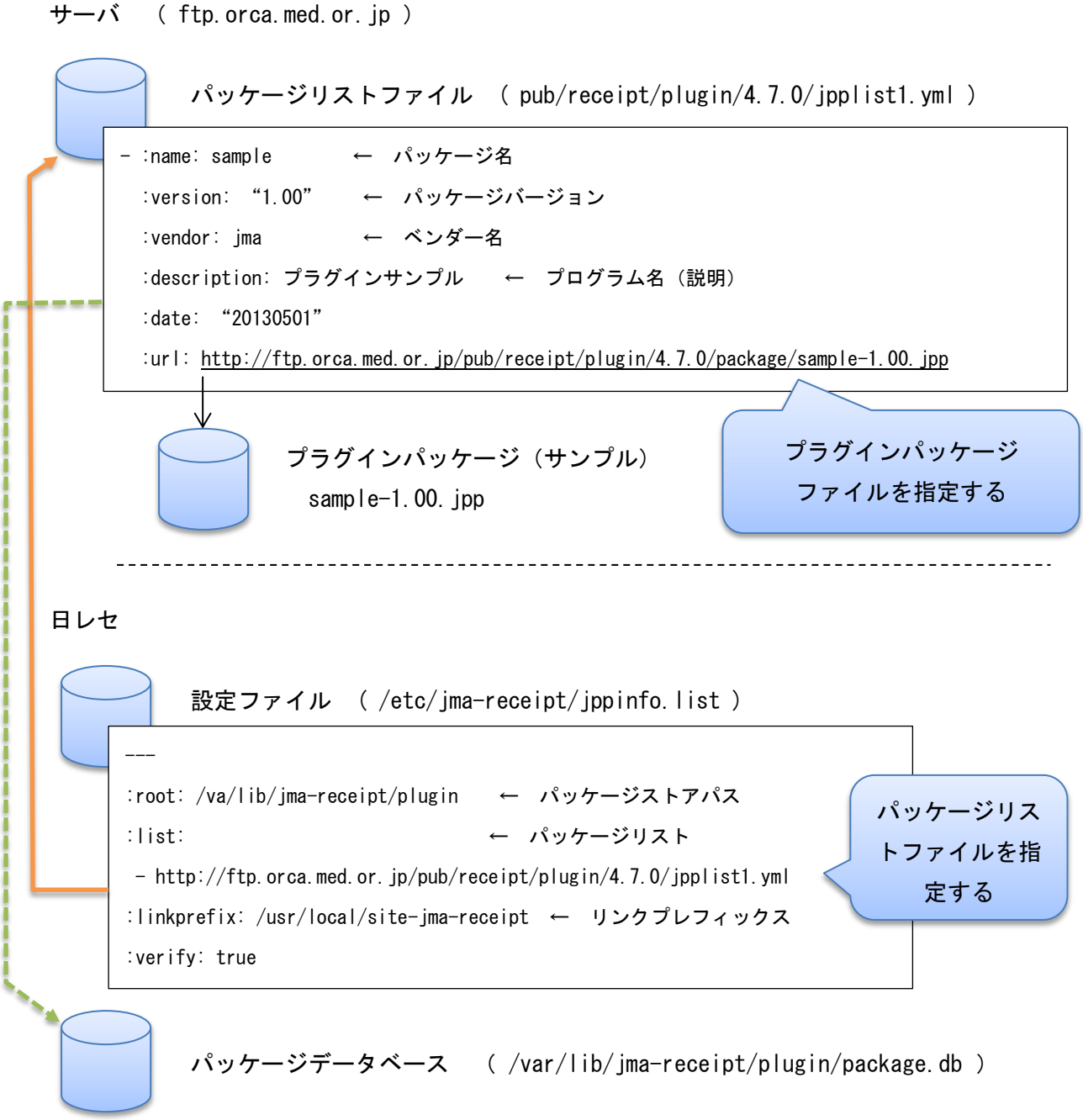


２　日レセプラグインの仕組み

日レセプラグインの仕組みについて説明します。

２．１　プラグイン情報の取得

日レセから、設定ファイルにあるパッケージリストよりサーバへ照会します。
サーバにあるパッケージリストファイルの内容から、日レセのパッケージデータベースを更新します。



２．２　プラグインパッケージの組み込み

パッケージリストの中から“プラグインサンプル”を組み込みます。

パッケージファイル（sample-1.00.jpp）をダウンロードし、設定ファイルにあるパッケージストアパスのディレクトリへ展開します。

```
/var/lib/jma-receipt/plugin/     ←　パッケージストアパス
  sample-1.00/
    cobol/
      SAMPLE1.CBL
    cobol/copy
      SAMPLE1.INC
    form/
      SAMPLE1.red
    meta/
      link
        :
```

設定ファイルで指定されている

展開された link ファイルに従い、設定ファイルにあるリンクプレフィックスで指定したディレクトリへ シンボリックリンクを作成します。

link ファイル（ /var/lib/jma-receipt/plugin/sample-1.00/meta/link ）

```
---
 - [cobol/SAMPLE1.CBL , cobol/]
 - [cobol/copy/SAMPLE1.INC , cobol/copy/]
 - [form/SAMPLE1.red , form/]
   :
```

リンクプレフィックス　（ /usr/local/site-jma-receipt ）

```
cobol/
 lrwxrwxrwx SAMPLE1.CBL -> /var/lib/jam-receipt/plugin/sample-1.00/cobol/SAMPLE1.CBL
cobol/copy/
 lrwxrwxrwx SAMPLE1.INC -> /var/lib/jam-receipt/plugin/sample-1.00/cobol/copy/SAMPLE1.INC
form/
 lrwxrwxrwx SAMPLE1.red -> /var/lib/jam-receipt/plugin/sample-1.00/form/SAMPLE1.red
  :
```

次に、展開された postlink コマンドを実行します。
postlink コマンドでは、リンクプレフィックス（ /usr/local/site-jma-receipt ）から必要なファイルを日レセのサイトディレクトリ（ /usr/lib/jma-receipt/site-lib ）へ複写する処理、COBOL ソースについてはコンパイル処理を行い、実行モジュールをサイトディレクトリへ作成する処理などを記述しています。

postlink ファイル
（ /var/lib/jma-receipt/plugin/sample-1.00/meta/postlink ）

```
～
# compile COBOL programs    ←  COBOL ソースをコンパイルしモジュールを site-lib へ
COBMEM=`module CBL`
for f in ${COBMEM}; do
  if test "x`echo -n $f | grep 'CBL$'`" != "x"; then
    m=`echo $f | sed 's/CBL$/so/'`
    echo -n "Building ${m}..." | logger
    ${COBOL} ${COBOLFLAGS} -o ${SITELIBDIR}/${m} ¥
         -I ${PATCHCOPYDIR} ¥
         -I ${COPYDIR} ¥
         -I ${SITESRCDIR}/cobol/copy ¥
        ${SITESRCDIR}/cobol/${f}
  fi
done

# copy scripts file    ←  スクリプトファイルを site-lib へ
SCRIPTMEM=`module scripts`
for f in ${SCRIPTMEM}; do
  if test -f "${SITESRCDIR}/${f}"; then
    echo -n "Copying ${SITESRCDIR}/${f}..." | logger
    sed -e 's,¥@jma-receipt-env¥@,/etc/jma-receipt/jma-receipt.env,g' ¥
        < "${SITESRCDIR}/${f}" > "${SITEDIR}/${f}"
    chmod +x "${SITEDIR}/${f}"
  fi
done
echo "done"

～
```

２．３　プラグインパッケージの取り外し

組み込まれたプラグインパッケージ“プラグインサンプル”を取り外します。

展開された preunlink コマンドを実行します。
preunlink コマンドでは、日レセのサイトディレクトリ(/usr/lib/jma-receipt/site-lib)からパッケージが作成したファイルを削除する処理などを記述しています。

preunlink コマンド
(/var/lib/jma-receipt/plugin/sample-1.00/meta/preunlink)

```
～
# remove .so
echo -n "Remove .so files..."
COBMEM=`module CBL`
echo -n "${COBMEM}"
for f in ${COBMEM}; do
  if test "x`echo -n $f | grep 'CBL$'`" != "x"; then
    m=`echo $f | sed 's/CBL$/so/'`
    echo -n "Remove ${m}..." | logger
    rm -rf ${SITELIBDIR}/${m}
  fi
done
echo "done"

# remove file
for d in scripts lddef data doc form init record screen ; do
  echo -n "Remove ${d}/ files..."
  m=`module ${d}`
  for f in ${m}; do
    if test -f "${SITEDIR}/${f}"; then
      echo -n "Remove ${f}..." | logger
      rm -rf ${SITEDIR}/${f}
    fi
  done
done
echo "done"
～
```

次に、link ファイルに従い、リンクプレフィックスディレクトリへ作成したシンボリックリンクを削除します。

３　プラグインパッケージの作成

３．１　鍵の生成

署名検証のため公開鍵暗号方式による鍵を作成します。

```
$ gpg --gen-key
gpg (GnuPG) 1.4.10; Copyright (C) 2008 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

ご希望の鍵の種類を選択してください:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (署名のみ)
   (4) RSA (署名のみ)
選択は? 1  ← [1]を入力し、[Enter]キーを入力します
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)   ← [Enter]キーを入力します
要求された鍵長は 2048 ビット
鍵の有効期限を指定してください。
         0 = 鍵は無期限
      <n>  = 鍵は n 日間で満了
      <n>w = 鍵は n 週間で満了
      <n>m = 鍵は n か月間で満了
      <n>y = 鍵は n 年間で満了
鍵の有効期間は? (0)   ← [Enter]キーを入力します
Key does not expire at all
これで正しいですか? (y/N) y  ← [y]を入力し、[Enter]キーを入力します

あなたの鍵を同定するためにユーザーID が必要です。
このソフトは本名、コメント、電子メール・アドレスから
次の書式でユーザーID を構成します:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"
```

本名: Nichii Taro　← 名前を入力し、[Enter]キーを入力します
電子メール・アドレス: nichii-taro@hoge.jp　← メールアドレスを入力し、[Enter]キーを入力します
コメント: jppsample　← コメントを入力し、[Enter]キーを入力します
次のユーザーID を選択しました:
　　"Nichii Taro (jppsample) <nichii-taro@hoge.jp>"

名前(N)、コメント(C)、電子メール(E)の変更、または OK(O)か終了(Q)? O　← [O]を入力し、[Enter]キーを入力します
秘密鍵を保護するためにパスフレーズがいります。

パスフレーズを入力:　← パスフレーズを入力し、[Enter]キーを入力します
パスフレーズを再入力:　← 再度パスフレーズを入力し、[Enter]キーを入力します

今から長い乱数を生成します。キーボードを打つとか、マウスを動かす
とか、ディスクにアクセスするとかの他のことをすると、乱数生成子で
乱雑さの大きないい乱数を生成しやすくなるので、お勧めいたします。
（コマンドから抜けるまでマウスを動かします）

+++++
gpg: 鍵 XXXXXXXX を絶対的に信用するよう記録しました
公開鍵と秘密鍵を作成し、署名しました。

gpg: 信用データベースの検査
gpg: 最小の「ある程度の信用」3、最小の「全面的信用」1、PGP 信用モデル
gpg: 深さ: 0　有効性:　1　署名:　0　信用: 0-, 0q, 0n, 0m, 0f, 1u
pub　 00000/XXXXXXXX 2012-10-24
　　指紋 = XXXX 0000 XXXX 0000 XXXX　0000 XXXX 0000 XXXX 0000
uid　　　　Nichii Taro (jppsample) <nichii-taro@hoge.jp>
sub　 00000/XXXXXXXX 2012-10-24

３．２　鍵の確認

生成した鍵を確認します。

```
$ gpg --list-keys
/home/oruser/.gnupg/pubring.gpg
--------------------------------
pub   00000/XXXXXXXX 2012-10-24
uid                  Nichii Taro (jppsample) <nichii-taro@hoge.jp>
sub   00000/XXXXXXXX 2012-10-24
```

３．３　公開鍵ファイルの作成

公開鍵をエクスポートします。

```
$ gpg -o nichii-taro.pub -a --export nichii-taro@hoge.jp

$ cat nichii-taro.pub  ← 公開鍵ファイル
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.10 (GNU/Linux)

mQENBFCHsrIBCAC/x/7oGCjAXhWExYfdssnaCfbDapipIjjWyZAiFjpApOul12u3
（省略）
=yDNX
-----END PGP PUBLIC KEY BLOCK-----
```

ここでは、nichii-taro.pub ファイルが公開鍵ファイルとなります。
プラグインの提供先には、この公開鍵ファイルを渡す必要があります。

３．４　プラグインパッケージ作成ディレクトリの準備

プラグインパッケージのサンプルをダウンロードします。

サンプルには、ORCA プロジェクトホームページで公開しています、『日次・月次統計帳票ユーザカスタマイズについて』（ http://www.orca.med.or.jp/receipt/tec/report2.html ）で使用していますサンプルプログラムを参考として含めています。

```
$ wget http://ftp.orca.med.or.jp/pub/etc/tools/jma-plugin-sample.tgz
```

（ 705f2523b0959db480cb33b7be5be28c　jma-plugin-sample.tgz ）

解凍します。

```
$ tar xvzf jma-plugin-sample.tgz
```

jma-plugin-sample ディレクトリが作成されますので、ディレクトリ名を自由な名前に変更します。(plugin に名前変更して以下説明します)

```
$ mv jma-plugin-sample plugin
```

plugin ディレクトリの内容は以下となります。

```
plugin/        網掛け部分がプラグイン機能自体のファイルとなります。
  build.conf   build.rb
  sample/
    cobol/
      SCSAMPLE01.CBL
      copy/
        SCSAMPLEH01.INC
    data/
    doc/
    etc/
    form/
      SCSAMPLEH01.red
    init/
    lddef/
    meta/
      control   link
      module    postinst   postlink   postrm      postunlink
      preinst   prelink    prerm      preunlink
    record/
    screen/
    scripts/
      allways/
      daily/
      kaisei/
      kentan/
      monthly/
      tools/
```

３．５　プラグインパッケージのビルド

プラグインパッケージを提供するサイトを決めます。
原則として http プロトコルでアクセス可能なサイトとします。

提供サイトを以下として説明します。
　サーバ：localhost
　ディレクトリ：plugin/4.7.0/package

```
$ cd plugin
```

build.conf ファイルを編集します。

```
:baseurl: http://localhost/plugin/(version)/package
:distdir: dist
:list: jma-sample.yml
:keyuser: sample
:passphrase: sample
```

最初はこのような内容になっています。
baseurl は、パッケージを提供するサイトを指定しますので (version)を 4.7.0 に変更します。
distdir は、生成するパッケージファイルを格納するディレクトリを指定します。
list は、パッケージリストファイル名を指定します。
サンプルでは jma-sample.yml としています。
実際にはプラグインパッケージを提供するベンダーが識別できるような名前としてください。
keyuser は、『３．１　鍵の生成』で作成中に指定した本名（ここでは Nichii Taro）に変更します。
passphrase は、『３．１　鍵の生成』で作成中に指定したパスフレーズ（ここでは sample とします）に変更します。

```
:baseurl: http://localhost/plugin/4.7.0/package
:distdir: dist
:list: jma-sample.yml
:keyuser: Nichii Taro
:passphrase: sample
```

サンプルプログラムのプラグインパッケージを作成します。

```
$ ruby build.rb -c build.conf
```

ビルドが終了したら、build.conf ファイルの distdir で指定したディレクトリが作成されその中にパッケージファイルが生成されます。

```
$ cd dist
$ ls
jma-sample.yml   jma-sample.yml.asc   sample-1.00.jpp   sample-1.00.jpp.asc
```

４　プラグインパッケージのテスト

４．１　プラグインパッケージのアップロード

パッケージファイルのテストを行うために提供サイトへアップロードします。

jma-sample.yml と jma-sample.yml.asc は、http://localhost/plugin/4.7.0 から引けるようにします。
sample-1.00.jpp と sample-1.00.jpp.asc は、http://localhost/plugin/4.7.0/package から引けるようにします。

実際には、公開する提供サイトとテストのための提供サイトは別になるでしょう。
テストを完了して公開する前には、build.conf の baseurl を公開する提供サイトへ変更してパッケージを再ビルドし、生成されたパッケージファイルをアップロードします。

４．２　公開鍵のインポート

テストするための日レセマシンを準備し、『３．３　公開鍵ファイルの作成』で作成した公開鍵ファイル（例 nichii-taro.pub）を orca ユーザとなってインポートします。

```
$ sudo su
# su orca
$ gpg --import nichii-taro.pub
確認します
$ gpg --list-keys
$ exit
# exit
```

作成したプラグインを配布するには、予めユーザへ公開鍵を渡し、同様の操作にて公開鍵をインポートしていただく必要があります。

４．３　パッケージリストの追加

日レセの設定ファイル（ /etc/jma-receipt/jppinfo.list ）を編集します。

```
---
:root: /var/lib/jma-receipt/plugin
:list:
 - http://ftp.orca.med.or.jp/pub/receipt/plugin/4.7.0/jpplist1.yml
 - http://ftp.orca.med.or.jp/pub/receipt/plugin/4.7.0/jpplist2.yml
 - http://localhost/plugin/4.7.0/jma-sample.yml  ← 追加します
:linkprefix: /usr/local/site-jma-receipt
:verify: true
```

list に提供サイトにアップロードしたパッケージリストファイルの指定を追加します。

４．４　プラグインの組み込み

日レセのプラグイン画面に作成したプログラム名“プラグインサンプル”が表示されることを確認します。
表示されたら選択して「組込」ボタンをクリックします。
「インストール」に“○”が表示されたら組み込み処理は正常終了したことになりますが、以下の確認をします。

パッケージがパッケージストアパスにインストールされているか確認します。

```
$ ls /var/lib/jma-receipt/plugin
cache package.db sample-1.00 version
```

リンクプレフィックスの該当ディレクトリに必要なソースのシンボリックリンクが作成されているか確認します。

```
$ ls -l /usr/local/site-jma-receipt/cobol
lrwx～  SCSAMPLE01.CBL -> /var/lib/jma-receipt/plugin/sample-1.00/cobol/SCSAMPLE0
1.CBL
drwx～  copy
```

サイトディレクトリの該当ディレクトリに必要なファイルが作成されているか確認します。

```
$ ls /usr/lib/jma-receipt/site-lib
SCSAMPLE01.so data form ～
```

パッケージの取り外しのテストなどを行ってください。
組み込みによって作成されたファイルやシンボリックリンクが削除されたことを確認します。

４．５　デバッグ

プラグイン処理のログを書き出す設定をします。
plugin.sh を編集します。

plugin.sh はパッチで提供する場合がありますので、
/usr/lib/jma-receipt/patch-lib/scripts/allways に存在する場合はそのファイルを、存在しない場合は /usr/lib/jma-receipt/scripts/allways にあるファイルを編集してください。

```
#!/bin/bash
(省略)
PACKAGE=$5
VERSION=$6

PLUGIN_SHELL=" ${ORCA_DIR}/bin/jma-plugin -d -L /tmp/plugin.log"
JPPINFO=${SYSCONFDIR}/jppinfo.list
```

変数 PLUGIN_SHELL に赤字部分のようなオプションを追加します。
プラグインの処理をして /tmp/plugin.log を確認してデバッグの参考にしてください。

プラグイン処理のテストを繰り返しているとパッケージデータベースのつじつまが合わなくなることがあります。
リセットするには、/var/lib/jma-receipt/plugin/package.db を削除してください。
また、すでに組み込みされた /usr/local/site-jma-receipt/ や
/usr/lib/jma-receipt/site-lib/ の各ディレクトリのリンクやファイルを削除してテストを再開してください。

５　パッケージメタファイル

パッケージメタファイルについて説明します。

詳しくは、『日レセプラグイン　リファレンス』を参照してください。

ファイル	説明
control	build.rb によりパッケージリストファイル及びパッケージファイルを作成する時の情報 :name: sample　← パッケージ名 :version: “1.00”　← バージョン :vendor: jma-sample　← ベンダー名 :description: プラグインサンプル　← パッケージの説明 バージョンアップのためのパッケージを作成する場合は version を大きい値に設定してビルドします。
link	リンクプレフィックスにシンボリックリンクを作成するための情報 - [cobol/SAMPLE1.CBL , cobol/]
module	link ファイルからメンバーリストを返却するスクリプト COBMEM=`module CBL` SCRIPTMEM=`module scripts`
postinst	インストール実行後に実行されるスクリプト （空）
postlink	リンク実行後に実行されるスクリプト サイトディレクトリへファイルを作成する処理を記述します。
postrm	パッケージ削除実行後に実行されるスクリプト （空）
postunlink	アンリンク実行後に実行されるスクリプト （空）
preinst	インストール実行前に実行されるスクリプト （空）
prelink	リンク実行前に実行されるスクリプト （空）
prerm	パッケージ削除実行前に実行されるスクリプト （空）
preunlink	アンリンク実行前に実行されるスクリプト postlink によりサイトディレクトリへ作成されたファイルなどを削除する処理を記述します。

６　複数のプラグインパッケージを作成

サンプルでは、パッケージ sample を単一で提供する場合で説明しましたが、複数のパッケージを提供することも可能です。
plugin ディレクトに複数のパッケージ名ディレクトリを作成してプログラム収容すれば、一括して作成できます。

```
plugin/
  build.conf
  build.rb
  package1/
    cobol/
      :(省略)
    meta/                    パッケージ名（ package1 ）のディレクトリ
      control
      link
      module
      postinst
      :(省略)
  package2/
    cobol/
      :(省略)
    meta/                    パッケージ名（ package2 ）のディレクトリ
      control
      link
      module
      postinst
      :(省略)
  package3/
    cobol/
      :(省略)
    meta/                    パッケージ名（ package3 ）のディレクトリ
      control
      link
      module
      postinst
      :(省略)
```

７　注意事項

７．１　プログラム名について

プラグイン機能で提供するプログラムファイルの名前については、ORCA プロジェクトホームページで公開しています、『カスタマイズプログラムの命名規約について』（ http://www.orca.med.or.jp/receipt/tec/customize_name.html ）での説明に従ってください。

７．２　共通ファイルについて

パッケージにより組み込みをするファイルについては、他のパッケージに含まれるファイル名と同一ファイル名を含めていけません。

共通で使用するファイルだからといって複数のパッケージに含めても、プラグインで組み込んだ後、他方の組み込まれたパッケージを取り外しした時に、ファイルが削除されてしまいます。