ＰＳＬＸバージョン３
仕様書パート５

製造業情報連携プラットフォーム
実装マニュアル

業務ソフトウェア（簡易版）

２０１４年１１月

ＮＰＯ法人ものづくりＡＰＳ推進機構

目次

はじめに

本書の利用方法

本書は、PSLX バージョン３仕様書にしたがって、実際に業務アプリケーション・プログラムを連携させる情報システムを設計、構築するための手引書です。本手引書は、PSLX バージョン３仕様書に準拠していますが、本手引書で示す方法が、唯一の PSLX バージョン３仕様書に準拠した実装方法ではありません。

本手引書の利用者は、情報システムの構築を専門とする技術者です。ただし、XML 関連技術や、データベース設計などの高度な専門技術や知識は前提としません。一般的な業務アプリケーション・プログラムを開発あるいは設計した経験のある技術者であれば理解可能な記述となっています。

システム環境

業務アプリケーション・プログラムが動作する環境は、Windows 7 または Windows 8 上とし、オンプレミス型のソフトウェアとして稼働するものとします。ただし、これは、これは Windows 以外の OS であることを否定するものではなく、またサーバ上で稼働する Web アプリであることを否定するものではありません。こうした動作環境でも情報連携システムを構築することは可能ですが、本手引書では、Windows を前提とした解説となっています。

一方、連携のための連携オブジェクトや連携データがおかれる場所は、ネットワーク上のあらゆる場所が考えられます。本手引書では、それぞれの業務アプリケーション・プログラムが共通してアクセス可能なファイルサーバ、RDB、そして HTTP サーバの３種類を想定します。こうした外部のリソースには、連携データとは別に、情報連携のためのコントローラがインストールさせる場合があります。

連携システムのアーキテクチャー

連携方法の種類

この仕様書では、簡易型粗結合パターンを用いて連携を行うものとします。連携方法を、連携データの物理的な形式で分類すると、テキスト形式、RDB 形式、そして WebDb 形式の３種類があります。

表 1　連携方法の整理

分類	保存場所	接続方法	データ型式
テキスト形式	ファイルサーバ等	ファイルシステム	CSV
RDB 形式	DB サーバ	DBMS に依存	テーブル
WebDb 形式	HTTP サーバ	HTTP(REST)	KVS

CSV 形式

テキスト形式では、連携データをテキスト形式でファイルサーバ上に保存します。データ型式は CSV（カンマで区切られた形式）とします。データの同時アクセスの制御や排他処理ができない等の問題はありますが、もっとも簡易な方法であり、その都度、操作者を介した連携処理には向いています。

RDB 形式

RDB 形式は、連携データをリレーショナルデータベースのテーブル形式で保存します。RDB との接続は SQL を用います。信頼性や保守性が高く、一般的な多くの情報システムにおいて利用される方法といえます。

WebDB(KVS)形式

WebDB(KVS)形式は、連携データを HTTP サーバ上に保存します。HTTP サーバ上で実際にどのような形式（RDB 型や KVS 型など）で保存されるかは問いません。接続には REST（GET や PUT などのシンプルな HTTP 型接続）を用います。拠点間や企業間など異なるサイト間でデータ連携を行なう場合などに有効です。

連携コントローラ

業務アプリケーションが連携する場合、連携データとは別に、業務アプリケーション間で、通知メッセージの受け渡しが必要です。ここで通知メッセージは、たとえば、業務アプリケーションＡが、“業務アプリケーションＢに対して、ある連携データを共通領域に登録した”という事実を業務アプリケーションＢに通知するために利用します。こうした通知メッセージのやり取りを管理するのが連携コントローラです。連携コントローラが、連携データの送受信に関する情報を受信側の業務アプリケーションに通知する方法は、以下の３種類あります。

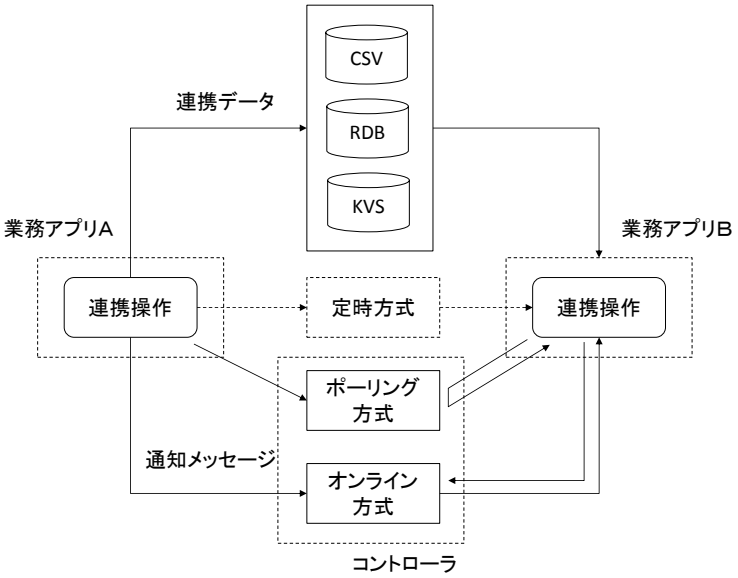

図 1 連携システムの基本構成

表 2　連携方式の種類

分類	データ型式
定時方式	前もって設定した時刻に連携操作が行われる際の連携方法
ポーリング方式	連携操作の通知を受け取る側が一定間隔でコントローラに通知の有無を問い合わせる連携方法
オンライン方式	連携操作の通知を受け取る側がコントローラに接続して通知を直ちに受け取る連携方法

通知メッセージのデータ構造

連携コントローラは、業務アプリケーションからの連携データの転送依頼に対応して、通知メッセージを作成し保存および相手の業務アプリケーションに通知します。通知メッセージは、連携操作１つに対して１つ生成され、以下の構造となります。

表 3　通知メッセージの構造

項目名	英語名	説明
通知 ID	ManageId	連携コントローラ上の ID
マシン ID	MachineId	送信者を識別する ID
マシン位置	Location	マシンの位置。URL/IP アドレスなど
保存形式	StoreType	データの保存形式
標準スキーマ名	SchemaName	連携データの標準スキーマ名
業務オブジェクト名	ObjectName	連携データの業務オブジェクト名
業務データキー	Key	連携データのレコードを識別する文字列
受信者または受信グループ	SendTo	宛先を表す文字列
受信者の権限	Grant	受信者の権限を表す文字列
送信日時	DateTime	通知が発生した日時
保管期限	Expires	通知の有効期限を表す日時
ステータス	Status	状態を表す文字列
説明	Remark	エラーなどの状況を説明する文字列
状態コード	Code	状態を表す整数

通知メッセージは、コントローラ内で通知 ID とマシン ID とでユニークに決まるものでなければなりません。

連携エラーと取消

連携操作をいくつかまとめて１つのトランザクションとして処理すること

は可能ですが、こうした連携操作によるアクションは、実行後に取り消すことはできません。また、もし連携操作がエラーとなった場合には、トランザクションの単位でもとの状態にもどしたうえで、エラーであることを依頼者側に通知しなければなりません。

なお、連携先の状態の照会が可能な場合には、過去の連携操作などのトランザクションを連携オブジェクト ID としてその実行結果を照会することを可能としてください。

連携の構成

CSV 形式連携の構成

CSV 形式連携では、次の構成が必要です。

- ファイルシステム
- データ連携コントローラ
- 業務アプリケーション

業務アプリケーションには、次の機能を有する必要があります。

- ファイルシステムに保存されている CSV ファイルにて連携データを取得・追加・更新するための機能
- 連携操作時、連携コントローラに通知メッセージを送信する機能。業務アプリケーションと連携コントローラ間は、HTTP によって通信します。

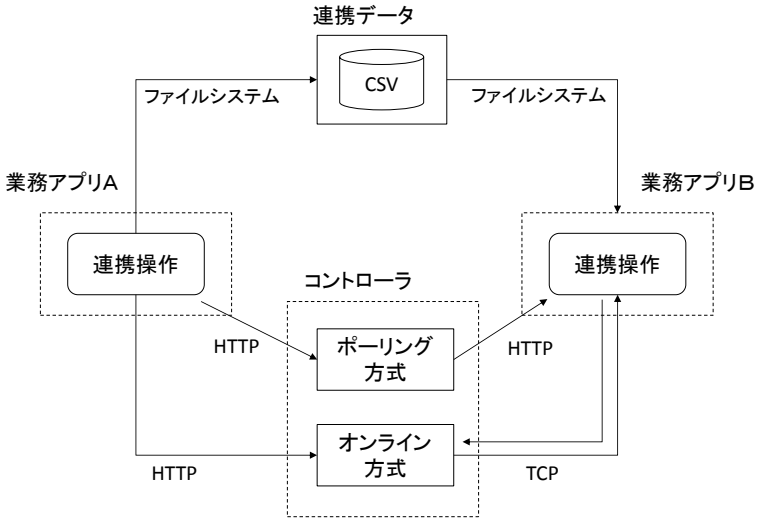

図 2 CSV 形式の連携

RDB 形式連携の構成

RDB 形式連携では、次の構成が必要です。

- データベースシステム
- データ連携コントローラ
- 業務アプリケーション

業務アプリケーションには、次の機能を有する必要があります。

- データベースに対して連携データを取得・追加・更新するための機能
- 連携操作時、連携コントローラに通知メッセージを送信する機能。業務アプリケーションと連携コントローラ間は、HTTP によって通信します。

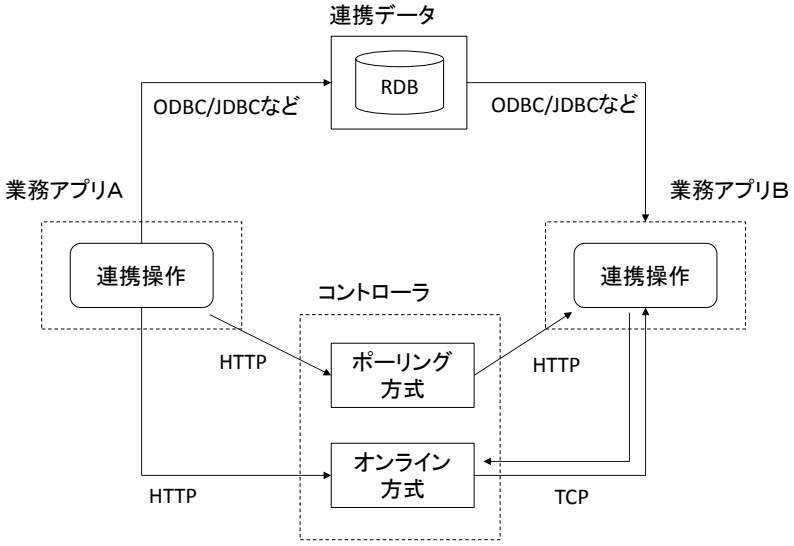

図 3　RDB 形式の連携

WebDB(KVS)形式連携の構成

WebDB(KVS)形式連携では、次の構成が必要です。

- KVS(Key Value Store)によるデータ管理システム
- データ連携コントローラ

- 業務アプリケーション

業務アプリケーションには、次の機能を有する必要があります。

- WebDB 管理システムに対して連携データを取得・追加・更新するための機能
- 連携操作時、連携コントローラに通知メッセージを送信する機能。業務アプリケーションと連携コントローラ間は、HTTP によって通信します。

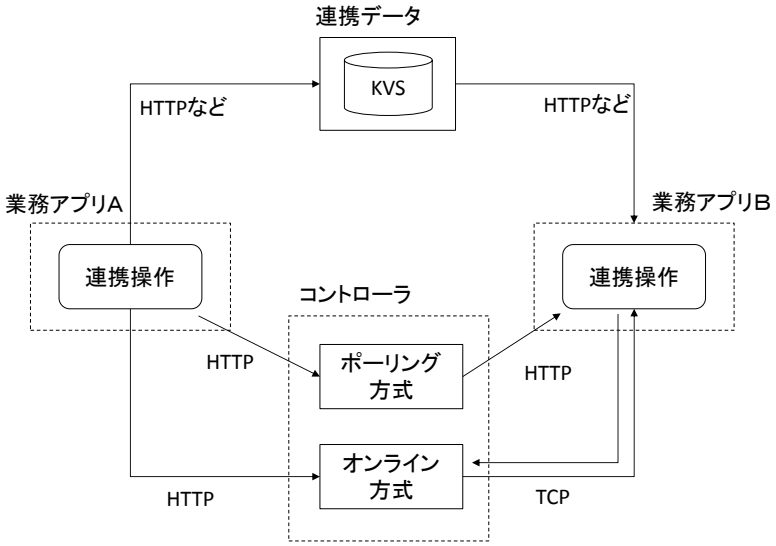

図 4　KVS 形式の連携

連携の方式

定時方式

定時方式は、業務アプリケーション間で、あらかじめ連携データの更新時刻を決定しておき、その時刻にあわせて連携データの送信側はデータを登録します。一方で、連携データの受信側は、その時刻から一定時間経過後に、データを受信します。したがって、実際の連携情報の送受信時には、通知メッセージのやり取りはありません。

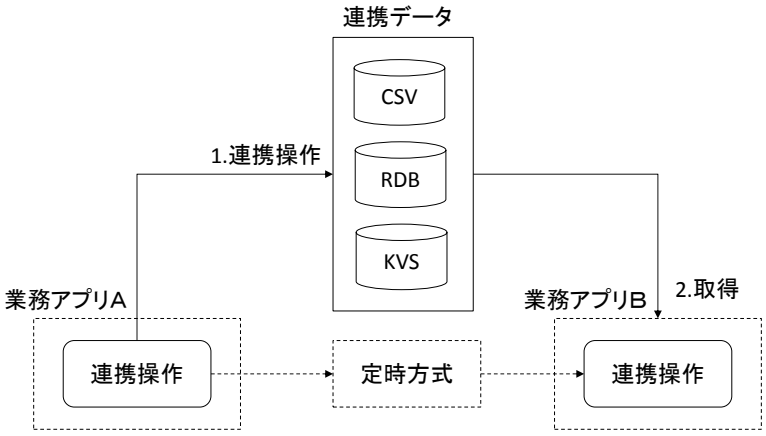

図 5　定時方式

業務アプリ A が連携データを連携操作し、その通知を業務アプリ B が受け取り連携データを取得する場合は、次の手順で行います。

1. 業務アプリ A が連携操作します。
2. 業務アプリ B は、一定間隔で連携データを取得します。

ポーリング方式

ポーリング方式は、連携データの受信側が、連携コントローラに対して、自

分宛ての連携データあるいは問い合わせ情報がないかを、定期的に確認にいきます。ポーリングのサイクルを短くすることによって、連携データを送信する時間をリアルタイムに近いものにすることができます。

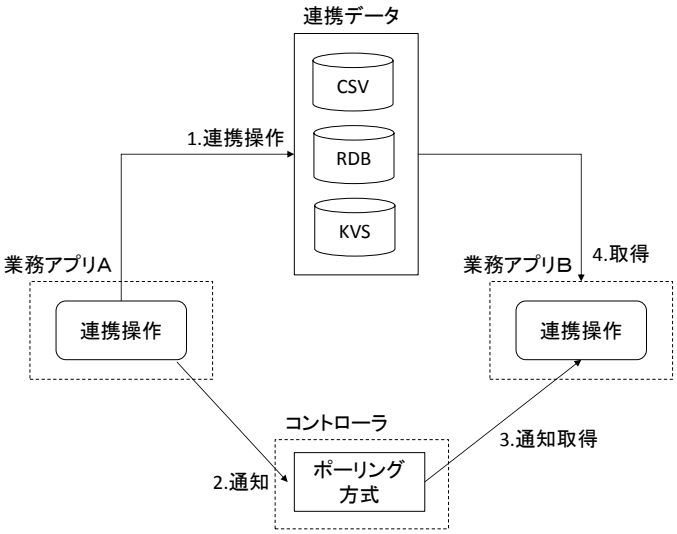

図 6　ポーリング方式

業務アプリ A が連携データを連携操作し、その通知を業務アプリ B が受け取り連携データを取得する場合は、次の手順で行います。

1. 業務アプリ A が連携操作します。
2. その後業務アプリ A が連携コントローラに連携操作の内容を通知します。
3. 連携アプリ B が連携コントローラから通知を受信します。
4. 連携アプリ B が連携データを取得します。

オンライン方式

オンライン方式は、連携コントローラが、データ連携を行なう業務アプリケーションとの間であらかじめ双方向の通信手段を確立しておき、連携データの送信側である業務アプリケーションが連携データを送信後、その事実を受信側の業務アプリケーションにリアルタイムで通知します。

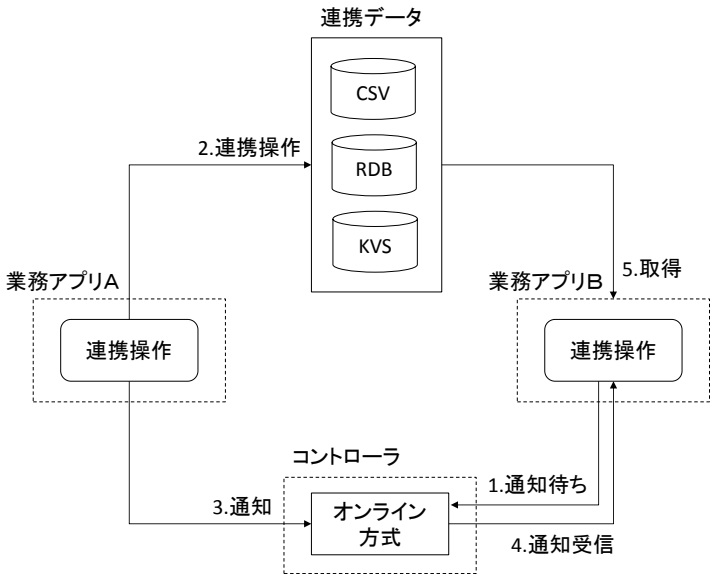

図 7 オンライン方式

業務アプリ A が連携データを連携操作し、その通知を業務アプリ B が受け取り連携データを取得する場合は、次の手順で行います。

1. 連携アプリ B が連携コントローラに接続して通知を待ち受けます。
2. 業務アプリ A が連携操作します。その後、連携コントローラに連携操作の内容を通知します。
3. 連携コントローラが待ち受け状態の業務アプリ B に通知を送り、業務アプリ B が通知を受信します。

連携の手順

連携データ追加の通知

保存先へ連携データが追加されたことを通知する

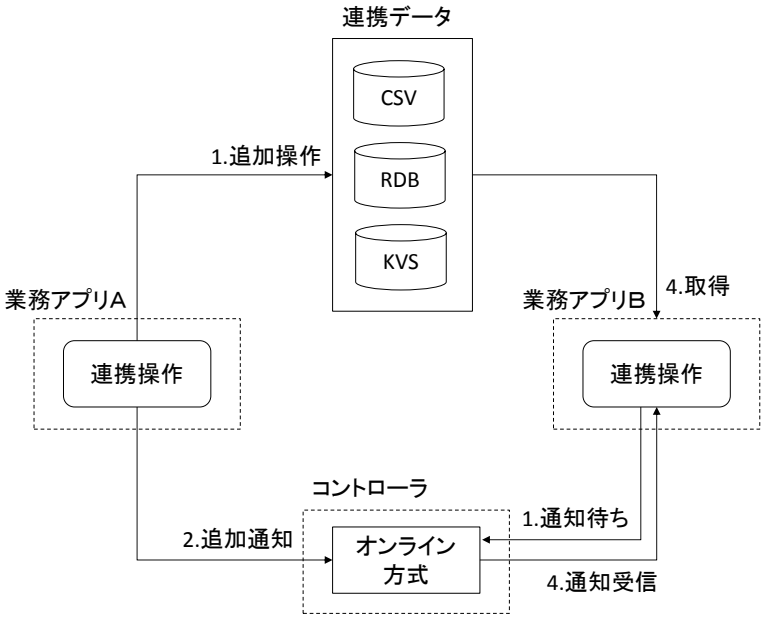

図 8　連携データ追加の通知

連携データ変更の通知

保存先の連携データが変更されたことを通知する

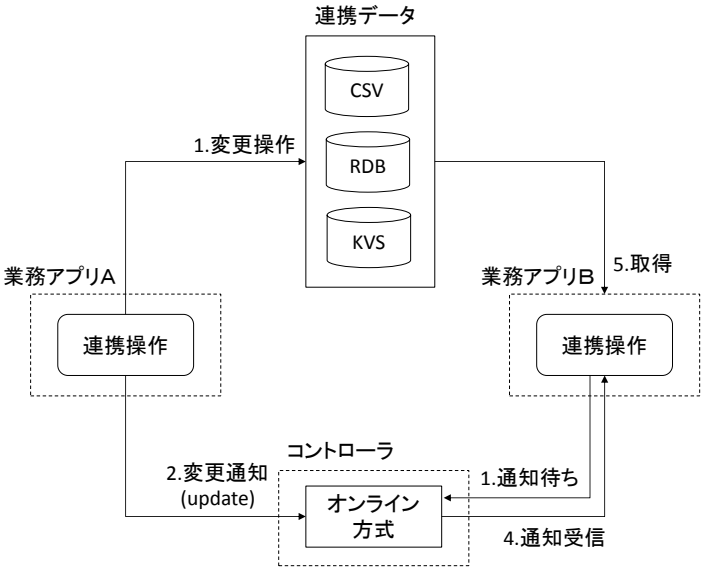

図 9　連携データ変更の通知

連携データ削除の通知

保存先の連携データが削除されたことを通知する

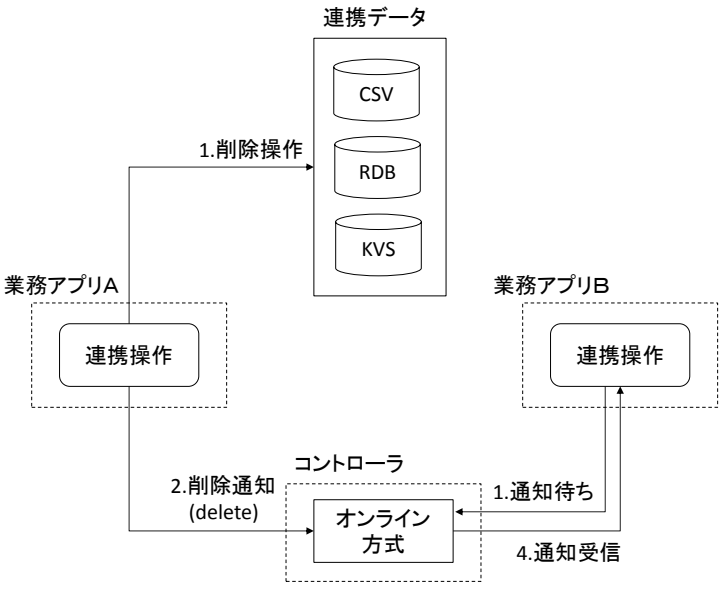

図 10　連携データ削除の通知

連携データの照会

保存先の連携データについて問い合わせ・回答を受け取る。

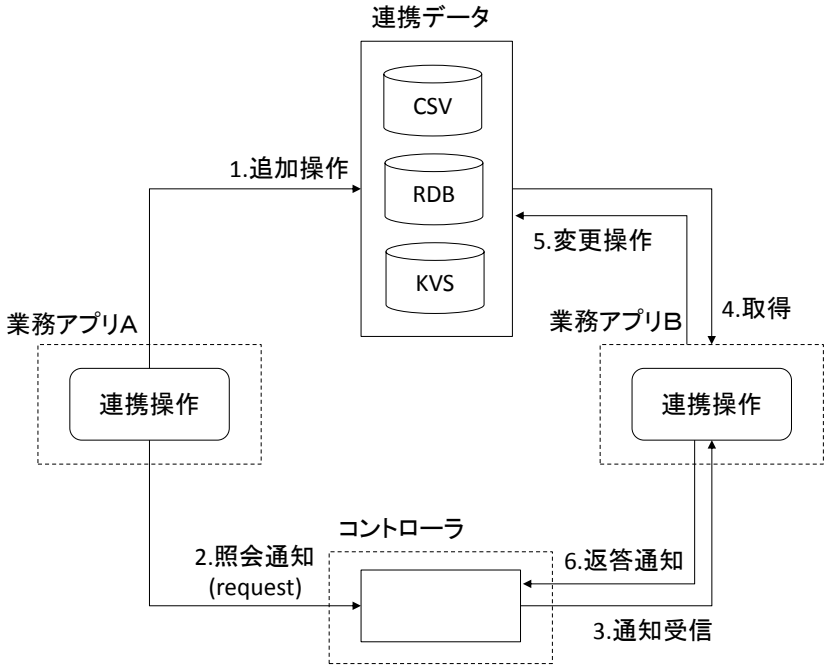

図 11　連携データの照会

連携データの通知

連携データを読み取ったことを通知する

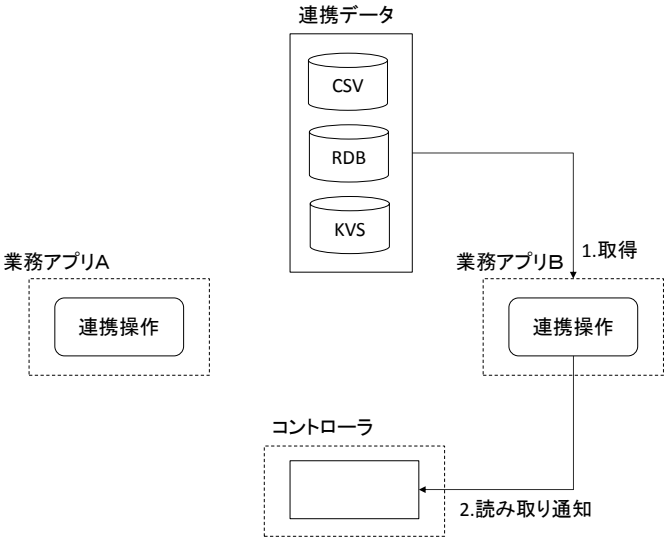

図 12　連携データの通知

相手の状態の照会

宛先における通知メッセージの状態を取得する

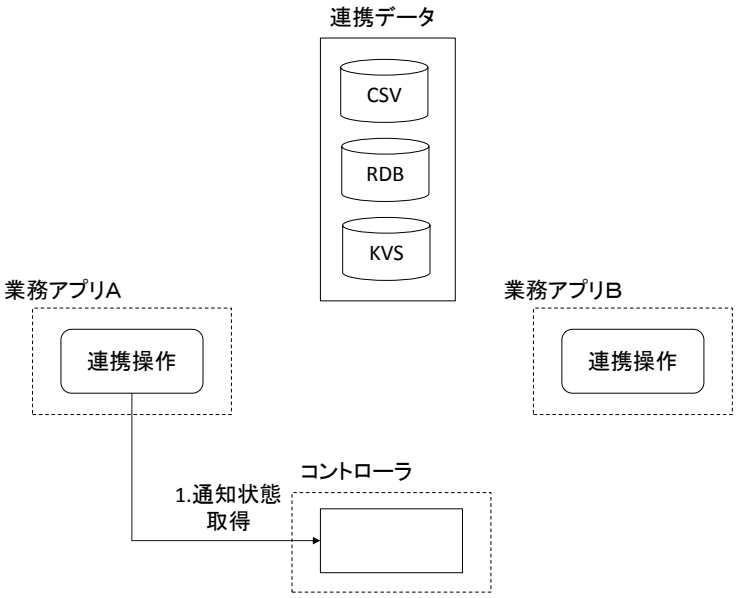

図 13　相手の状態の照会

連携コントローラ仕様

ユースケース

- 業務アプリケーションの ID と権限を管理する
- 業務アプリケーションが現在オンラインとなっているか把握している
- 業務アプリケーションから送信された連携データのログを蓄積する
- 送信先をグループとした場合に、グループに登録している業務アプリに配信してくれる。
- 連携データが保管されているサーバおよびアクセス方法（物理アドレス）を管理する。
- 自分宛ての連携データがあるかどうかを教えてくれる
- 送信した連携データを、相手アプリがもっていったかどうかを知る
- 送信した連携データを、誰がもっていったかを知る。
- 自分が起動したこと、またはシャットダウンすることを通知する。
- 送信者が送った連携データの通知について、連携コントローラに正常に到達、相手業務アプリに正常に到達、相手業務アプリからの返信を連携コントローラが受信、返信内容を送信者が正常に受信、といったステータスを管理する。
- 通信エラーとなった場合に、その事実を記録し相手に伝える。
- 返信の期限を設定し、それを超えた場合はタイムアウトとして終了する。

REST API

連携コントローラは、HTTP サーバを利用した Web アプリケーションとして動作します。連携コントローラへの各操作は、REST 形式によってリクエストを送信することで行います。

認証

連携コントローラへリクエストを送信できるのは、事前に連携コントローラに登録されたマシンのみです。マシンは、連携コントローラ接続時にマシン ID およびパスワードを Digest 認証方式にて認証します。

通知メッセージのデータ項目

(1)通知メッセージ

項目名	説明	データ型
notifyid	通知メッセージ ID	文字列
machineid	マシン ID	文字列
storeat	連携データの保存位置	文字列
storetype	連携データの保存形式	文字列(csv,rdb,webdb)
schemaname	標準スキーマ名	文字列
objectname	業務オブジェクト名	文字列
key	業務データキー	文字列 RDB 形式/WebDB 形式の場合は主キー CSV 形式の場合は CSV ファイル名
sendto	受信者または受信グループ	文字列
datetime	送信日時	日付日時 (yyyy/MM/dd hh:mm:ss)
expires	保管期限	日付日時 (yyyy/MM/dd hh:mm:ss)
status	ステータス	文字列

```
{
   "notifyid": "NM1111",
   "machineid": "ClientX",
   "dataid" : 12343,
   "storetype": "csv",
   "storeat": "192.112.64.50",
   "schemaname": "PSLX3",
   "objectname": "設計オーダ",
   "key": "DO-1234",
   "sendto": "GroupX",
   "grant": "",
   "datetime": "2015-01-15 12:34:56",
   "expires": "2015-01-25 12:34:56",
   "status": "Write"
}
```


(2)マシンデータ

マシンは、業務アプリケーションソフトウェアの単位を表します。マシンは、論理的な単位であり、一般的には連携コントローラに接続された業務アプリケーションソフトウェアを指します。物理的に 1 台の機器に複数のマシンを存在させることも可能です。

項目名	説明	データ型
machineid	マシン ID	文字列
storeat	連携データの保存位置	文字列
storetype	連携データの保存形式	文字列(csv,rdb,webdb)
group	グループ名	文字列
grant	受信者の権限	文字列 (read,write,readwrite,admin)
expires	保管期限	日付日時 (yyyy/MM/dd hh:mm:ss)
status	ステータス	文字列


```
{
```

```
    "machineid": "ClientF",
    "storetype": "csv",
    "storeat": "192.112.64.50",
    "group": "GroupX",
    "grant": "readwrite",
    "datetime": "2015-01-15 12:34:56",
    "expires": "2015-01-25 12:34:56"
}
```

通知メッセージ管理

URL 例	http://hostname/notification
URL 例(ID 指定)	http://hostname/notification/(通知 ID)
対応 HTTP メソッド	GET/POST/PUT

リクエストパラメータ

項目名		GET	POST	DELETE
id	通知 ID	○	-	*
machineid	送信者のマシン ID	*	*	*
schemaname	スキーマ名	○	*	-
objectname	業務オブジェクト名	○	*	-
key	業務データのキー	○	*	-
expires	有効期限	○	○	-
sendto	通知先のマシン ID またはグループ	○	*	-
status	read/insert/update/delete/request/response	○	*	-

*:必須, ○:省略可能, -:指定不可能

status には、取得(read), 挿入(insert), 更新(update), 削除(delete), 問い合わせ(request),回答(response)のいずれかを指定します。

POST メソッドにおいて schemaname, storetype は、省略するとマシンデータで設定された値が使用されます。

GET メソッド

通知メッセージを照会します。

自マシン宛ての通知メッセージを取得する

自マシン宛ての通知メッセージを取得するには、次のようなリクエストを送信します。

リクエスト内容

```
GET /notification?machineid=ClientB
```

レスポンス内容

```
{
   [
       "id": "CN54312",
       "machineid" : "ClientA",
       "sendto": "ClientB",
       "schemaname": "PSLX3",
       "storetype": "csv",
       "objectname": "受注オーダ",
       "key": "OD-1201",
       "status": "insert"
       "datetime": "2015-01-15 12:34:56",
       "expires": "2015-01-25 12:34:56",
   ] , [
       "id": "CN54315",
       "machineid" : "ClientA",
       "sendto": "GroupX",
       "schemaname": "PSLX3",
       "storetype": "csv",
       "objectname": "設計オーダ",
       "key": "DO-1234",
       "status": "update"
       "datetime": "2015-01-15 12:34:56",
       "expires": "2015-01-25 12:34:56",
   ]
}
```

POST メソッド

通知メッセージを連携コントローラに登録して、対象のマシンに通知します。通知が登録されると、連携コントローラは、相手先の通知ログにこの通知が登録されたことを記録します。

通知メッセージを登録する

指定したマシンに対して指定したキーの連携データが変更されたことを通知するには、次のようなリクエストを送信します。

リクエスト内容

```
POST /notification

{
   "machineid": "ClientA",
   "sendto": "ClientB",
   "objectname": "受注オーダ",
   "key": "OD-1002",
   "status": "insert"
   "datetime": "2015-01-15 12:34:56"
}
```

レスポンス内容

通知 ID と状態(値はリクエストと同値)を返します。

```
{
   "id": "CN54312",
   "status": "update"
}
```

指定したグループに所属するすべてのマシンに通知する

連携データが変更されたことを指定したグループ(GroupX)に属するすべてのマシンに通知するには、次のようなリクエストを送信します。

リクエスト内容

```
{
```

```
   "machineid": "ClientA",
   "sendto": "GroupX",
   "objectname": "設計オーダ",
   "key": "DO-1234",
   "status": "update"
   "datetime": "2015-01-15 12:34:56",
   "expires": "2015-01-25 12:34:56",
}
```

レスポンス内容

成功すると、通知 ID と状態(値はリクエストと同値)が返されます。

```
{
    “id": "CN54315",
   "status": "update"
}
```

返信の期限を設定する

通知した連携データに対する返答期限付きの要求する場合に指定した期限を超えた場合はタイムアウトとして通知が無効(none)となります。返信の期限を設定するには、次のようなリクエストを送信します。

リクエスト内容

```
POST /notification

{
   "machineid": "ClientA",
   "sendto": "ClientC",
   "objectname": "設計オーダ",
   "key": "DO-1234",
   "status": "request"
   "expires": "2015-01-25 12:34:56",
}
```

レスポンス内容

成功すると、通知 ID と状態(値はリクエストと同値)が返されます。

```
{
```

```
   "id": "CN54315",
   "status": "request"
}
```

DELETE メソッド

すでに連携コントローラに登録された通知メッセージを取り消します。この操作では、通知メッセージは削除されず、取り消すことが各マシンに通知されます。

リクエスト内容

```
DELETE /notification/CN54315
```

レスポンス内容

通知 ID と状態(値は canceled)を返します。

```
{
   "id": "CN54312",
   "status": "canceled"
}
```

リクエストに失敗した場合のレスポンス

失敗した場合は、通知 ID(DELETE メソッドのみ、POST,GET メソッドのでは空白)、マシン ID および状態(値は error)を返します。必要に応じてコード(code), 理由(remark)が返される場合があります。

```
{
   "id": "CN54315",
   "machineid" : "ClientX",
   "status": "error",
   "code": "101",
   "remark": "指定したマシン ID が見つかりません。"
}
```

マシン管理

マシンを管理します。

URL 例	http://hostname/machine
URL 例(ID 指定)	http://hostname/machine/(マシン ID)
対応 HTTP メソッド	GET/POST/PUT/DELETE

リクエストパラメータ

項目名	値	GET	POST	PUT	DELETE
machineid	マシン ID	*	*	*	*
storetype	連携データ形式	-	○	○	-
storeat	連携データの位置	-	○	○	-
group	グループ名	○	*	○	-
grant	read/write/readwrite/admin	-	*	○	-
status	none/ready/off/busy/error	○	-	○	-
remark	文字列	-	-	○	-
code	整数	-	-	○	-

*:必須, ○:省略可能, -:指定不可能

grant は、読み取り可能(read)、書き込み可能(write)、読み書き可能(readwrite)、管理者(admin)のいずれかを指定します。

status は、状態不明(none)、待ち受け状態(ready)、停止状態(off)、処理状態(busy)、エラー(error)のいずれかを指定します

GET メソッド

マシンの状態、連携データの保存位置を照会します。

指定したマシンが現在オンラインかどうか確認する

指定したマシンが現在オンライン状態かどうか確認するには、次のようなリクエストを送信します。

リクエスト内容

```
GET /machine?machineid=ClientA
```

レスポンス内容

```
{
   "machineid": "ClientA",
   "storeat": "192.112.64.50",
   "storetype": "csv",
   "grant": "readwrite",
   "status": "off"
}
```

POST メソッド

マシンを追加して、連携データの保存位置を登録します。

マシンとその権限を連携コントローラへ追加する

マシンとその権限を連携コントローラへ追加するには、次のようなリクエストを送信します。

リクエスト内容

```
POST /machine

{
   "machineid": "ClientF",
   "storeat": "192.112.64.50",
   "storetype": "csv",
   "grant": "readwrite"
}
```

レスポンス内容

成功すると、マシン ID と状態(値は none)を返します。

```
{
   "machineid": "ClientF",
   "status": "none"
}
```

リクエストに失敗した場合のレスポンス

失敗した場合は、マシン ID および状態(値は error)を返します。必要に応じてコード(code)，理由(remark)が返される場合があります。

```
{
   "machineid": "ClientA",
   "status": "error",
   "code": "101",
   "remark": "指定したマシン ID は既に登録されています。"
}
```

PUT メソッド

マシンの情報を更新します。

自マシンが起動したことを通知する

自マシンが起動したことを通知するには、次のようなリクエストを送信します。

リクエスト内容

```
PUT /machine/ClientB

{
   "status": "ready"
}
```

レスポンス内容

成功した場合は、マシン ID と状態(値はリクエストと同値)を返します。

```
{
   "machineid": "ClientB",
   "status": "ready"
}
```

連携データが保管されているサーバおよびアクセス方法（物理アドレス）を変更する

連携データが保管されているアクセス方法を変更するには、次のようなリクエストを送信します。

リクエスト内容

```
PUT /machine


{
   "machineid": "ClientD",
   "storetype": "csv",
   "storeat": "192.112.64.50"
}
```

レスポンス内容

成功した場合は、マシン ID と状態を返します。

```
{
   "machineid": "ClientD",
   "status": "ready"
}
```

リクエストに失敗した場合のレスポンス

失敗した場合は、マシン ID および状態(値は error)を返します。必要に応じてコード(code), 理由(remark)が返される場合があります。

```
{
   "machineid": "ClientX",
   "status": "error",
   "code": "101",
   "remark": "指定したマシン ID のマシンが存在しません。"
}
```

DELETE メソッド

マシンを連携コントローラから削除します。

マシンを連携コントローラから削除する

マシンを連携コントローラから削除するには、次のようなリクエストを送信します。

リクエスト内容

```
DELETE /machine/ClientF
```

レスポンス内容

成功すると、マシン ID と状態(値は none)を返します。

```
{
   "machineid": "ClientF",
   "status": "none"
}
```

リクエストに失敗した場合のレスポンス

失敗した場合は、マシン ID および状態(値は error)を返します。必要に応じてコード(code), 理由(remark)が返される場合があります。

```
{
   "machineid": "ClientX",
   "status": "error",
   "code": "101",
   "remark": "指定したマシン ID のマシンが存在しません。"
}
```

ログ管理

ログ管理では、業務アプリケーションから送信された連携データのログを蓄積します。

URL 例	http://hostname/log
対応 HTTP メソッド	GET/POST

リクエストパラメータ

項目名	値	GET	POST
notification id	通知メッセージ ID	○	*
machineid	対象のマシン ID	*	*
datetime	登録された日時	○	-

read	true/false	－	－
status	none/unread/read/replied/completed/error	○	*
code	エラーコード(整数)	○	○
remark	状態を表す文字列	－	○

*:必須, ○:省略可能, －:指定不可能

GET メソッド

通知メッセージの履歴を取得します。

通知が読み取られたかどうかを取得する

指定した通知メッセージ(CT12143)について、マシン ID ごとに読み取られたかどうか、状態を取得するには、次のようなリクエストを送信します。

リクエスト内容

```
GET /log?notificationid=CT12143
```

レスポンス内容

```
{
   [
       "notificationid": "CT12143",
       "machineid": "ClientB",
       "read": "true",
       "status": "replied",
       "datetime": "2015/01/15 12:34:56"
   ] , [
       "notificationid": "CT12143",
       "machineid": "ClientC",
       "read": "false",
       "status": "unread",
       "datetime": ""
   ] , [
       "notificationid": "CT12143",
       "machineid" : "ClientD",
       "read": "true",
       "status": "completed",
       "datetime": "2015/01/15 12:34:56"
```

```
    ]
}
```

エラーが発生したマシン ID を取得する

　指定したトランザクション(CT12143)について、エラーが発生したマシン ID とその原因を取得するには、次のようなリクエストを送信します。

リクエスト内容

```
GET /log?notificationid=CT12143&status=error
```

レスポンス内容

```
{
    [
        "notificationid": "CT12143",
        "machineid": "ClientE",
        "read": "true",
        "status": "error",
        "code": "104",
        "remark": "データベースへ接続できませんでした。"
    ]
}
```

POST メソッド

　通知メッセージの履歴を追加します。

通知に応答したことを反映する

　指定した通知メッセージ(CT12143)について、ClientB が返答したことを連携コントローラに更新するには、次のようなリクエストを送信します。

リクエスト内容

```
PUT /log


{
    "notificationid": "CT12143",
    "machineid": "ClientB",
    "status": "replied"
```

```
}
```

レスポンス内容

通知 ID,マシン ID および状態(リクエストと同値)を返します。

```
{
   "notificationid": "CT12143",
   "machineid": "ClientB",
   "status": "replied"
}
```

リクエストに失敗した場合のレスポンス

失敗した場合は、通知 ID,マシン ID および状態(値は error)を返します。必要に応じてコード(code), 理由(remark)が返される場合があります。

```
{
   "notificationid": "XT12143",
   "machineid": "ClientB",
   "status": "error",
   "code": "102",
   "remark": "指定した通知 ID が存在しません。"
}
```

エラーメッセージ

業務アプリケーションからのリクエストの処理に問題が発生した場合、連携コントローラは、リクエストをエラーして扱い、その旨を返します。

リクエストがエラーとなった場合は、status 項目が「error」となります。また、必要に応じて code(整数)および remark(文字列)を返します。

code に指定される番号とそれに対応する意味は、次の通りです。

コード	意味
101	データベースに接続できません。
201	マシン ID が指定されていません。
202	通知 ID が指定されていません。
203	objectname が指定されていません。

204	key が指定されていません。
205	sendto が指定されていません。
301	マシン ID 指定されていません。
302	指定されたマシンが登録されていません。
303	status がありません。
401	マシン ID または通知 ID が指定されていません。
402	通知 ID が指定されていません。
403	マシン ID が指定されていません。
404	status がありません。
405	指定された通知 ID が登録されていません。

参考実装

動作環境

連携コントローラには、ポーリング方式、オンライン方式の２種類があります。

ポーリング方式

ポーリング方式の参考実装では、連携コントローラは PHP 言語で実装されています。HTTP サーバには、Apache HTTP Server を使用して、業務アプリケーションからの問い合わせに対して PHP プログラムを実行します。

連携データの保存は RDB 形式を用いて、データベースシステムには MySQL を利用します。

ポーリング方式では、HTTP 通信できる標準的なクライアントによって連携コントローラと通信できます。

オンライン方式

オンライン方式の参考実装では、連携コントローラは Windows 上で動作する単独のアプリケーションとして実装されています。この連携コントローラは、業務アプリケーションソフトウェアと常時接続することで通知メッセージを受け取ります。

連携コントローラの起動と終了

システム動作環境

ポーリング方式版

- 連携操作の通知を受け取る側が一定間隔で連携コントローラに通知の有無を問い合わせる連携方法
- Apache2.0/PHP5.5/MySQL5.5 にて構築
- Web サーバ上で動作する PSLX3 連携コントローラへアクセスする (HTTP REST)

参考実装ファイル構成

参考実装のファイル構成は次の通りです。

```
PSLX3Controller
├──pslx3controller.sql              --データベースキーマ
├──PSLX3Controller                  --連携コントローラ本体
|   ├──config.php                   --設定ファイル
|   └──manager
|       ├──css
|       ├──images
|       ├──js
|       ├──smarty
|       |   ├──plugins
|       |   └──sysplugins
|       ├──templates
|       └──templates_c
└──PSLX3ControllerClient            --クライアント接続サンプル
    ├──PSLX3ClientASample           --サンプル業務アプリ A
    ├──PSLX3ClientBSample           --サンプル業務アプリ B
    ├──PSLX3ClientLibrary           --クライアント接続ライブラリ
    ├──PSLX3MachineStatusSample     --マシン状態送信サンプル
    ├──PSLX3SendNotificationSample  --通知送信サンプル
    └──PSLX3ShowNotificationsSample --通知受信サンプル
```

連携コントローラのインストール

連携コントローラのインストールには、次の作業が必要です。

- 連携コントローラ本体の配置
- データベースへのスキーマ登録
- データベース環境の設定

連携コントローラ本体の配置

連携コントローラは、Web サーバのドキュメントルートに、

「PSLX3Controller」フォルダ内に含まれるすべてファイルをコピーすることで配置できます。

データベースへのスキーマ登録

連携コントローラは、通知データやマシン情報を格納するために MySQL のデータベースを使用します。連携コントローラが使用するデータベースのテーブルのスキーマは、「/pslx3controller.sql」に定義されています。MySQL のデータベース管理ツール(phpMyAdmin など)にて、「pslx3controller.sql」をインポートしてください。

データベース環境の設定

連携コントローラが接続するデータベースを設定ファイル(/PSLX3Controller/config.php)にて指定しておきます。

「DB_CONNECTION_STRING」定数では、MySQL データベースが起動しているホスト名、使用するデータベースのデータベース名を設定します。

また、「DB_USER」定数および「DB_PASSWORD」定数には、データベース接続時のユーザ名とパスワードを指定します。

```
<?php
/**************************************
 * PSLX3 Controller Server 1.0
```

```
 * Copyright(C) 2014 APSOM
 */
   define('DB_CONNECTION_STRING', 'mysql:host=localhost;dbname=pslx3con
troller;charset=utf8');
   define('DB_USER', 'pslx3');
   define('DB_PASSWORD', 'pslx3');

   /* 管理画面のパスワード */
   define('ADMIN_PASSWORD', 'pslx3');
?>
```

連携コントローラ管理ページ

連携コントローラの管理ページよりマシンの追加・削除および通知の確認、連携ログが確認できます。

ログイン方法

Web ブラウザにて、連携コントローラを配置した URL へアクセスします。

```
<連携コントローラの配置場所>/manager/
```

ログインするためにユーザ ID とパスワードの入力が求められますので、管理者の場合は、ユーザ ID「admin」、パスワード「pslx3」(既定値)を入力します。

図 14

通知管理

連携コントローラに登録された通知を確認するには、メニューの「通知」をクリックして「マシン状態」にて、通知の宛先となっているマシンを選択します。

連携ログの管理

連携コントローラ上の連携ログを確認できます。メニューより「ログ」をクリックし右側の「マシン状態」からマシンを選択すると、そのマシンの連携ログが表示されます。

図 15

業務アプリケーション(マシン)の登録

連携コントローラでは、業務アプリケーションを 1 マシンとして認識します。コントローラの管理画面でマシンを管理できます。

新たにマシンを登録する場合は、「マシン追加」よりマシン ID および接続用パスワードを指定します。接続用パスワードは、業務アプリケーション(マシン)が連携コントローラへ接続する際に必要となります。

図 16

なお、pslx3controller.sql にて、テーブルを生成した場合には、既定値でマシン ID として ClientA,ClientB,ClientC が登録されており、パスワードも同一の文字列が設定されています。

グループ管理

通知の送信先としてグループを指定することができます。マシンは、複数のグループに所属することができ、通知の宛先としてグループを指定した場合は、そのグループに属する各マシンへ一度に通知を送ることができます。

メニューより「グループ管理」を選ぶことでグループの追加・削除ができます。グループを追加するには、グループ追加項目にて追加するグループ名を入力して[追加]ボタンをクリックします。

図 17 グループの追加

マシンが所属するグループを設定するには、「マシン選択」欄にてマシンをクリックし、グループ一覧にて所属させるグループにチェックを入れて[決定]ボタンをクリックします。

図 18 マシンの所属するグループ

連携ログの管理

連携コントローラ上の連携ログを確認できます。メニューより「ログ」をクリックし右側の「マシン状態」からマシンを選択すると、そのマシンの連携ログが表示されます。

図 19

連携コントローラの操作方法

業務アプリケーションの登録

連携コントローラでは、業務アプリケーションを 1 マシンとして認識します。連携コントローラの管理画面の「マシン」登録にてマシンを登録できます。

マシン状態の更新サンプル

業務アプリケーションが起動したり終了したりするなどして状態が変化した際に、その旨を連携コントローラに登録することができます。連携コントローラに接続する業務アプリケーションは、マシン ID によって他の業務アプリケーションの状態を取得できます。

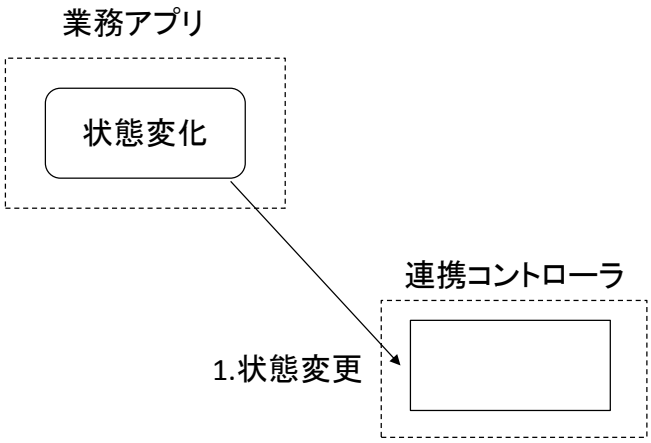

図 20 登録サンプルの動作概要図

業務アプリ側のサンプルコード例

業務アプリケーションの状態が変化したことを連携コントローラに通知するには、NotificationClient クラスの ChangeMachineState メソッドを用います。

なお、マシンの状態は、連携コントローラの「管理ページ」にて確認できます。

```
public partial class Form1 : Form
{
    NotificationClient client = new NotificationClient();

    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        client.Server = "http://localhost/PSLX3Controller/index.php";
        client.SchemaName = "PSLX3";
    }

    private void lunchButton_Click(object sender, EventArgs e)
    {
```

```
        client.MachineId = machineIdTextBox.Text;
        client.Password = client.MachineId;
        client.ChangeMachineState(MachineState.Ready);
    }

    private void offButton_Click(object sender, EventArgs e)
    {
        client.MachineId = machineIdTextBox.Text;
        client.Password = client.MachineId;
        client.ChangeMachineState(MachineState.Off);
    }

    private void busyButton_Click(object sender, EventArgs e)
    {
        client.MachineId = machineIdTextBox.Text;
        client.Password = client.MachineId;
        client.ChangeMachineState(MachineState.Busy);
    }

    private void errorButton_Click(object sender, EventArgs e)
    {
        client.MachineId = machineIdTextBox.Text;
        client.Password = client.MachineId;
        client.ChangeMachineStateError(101, remarkTextBox.Text);
    }
}
```

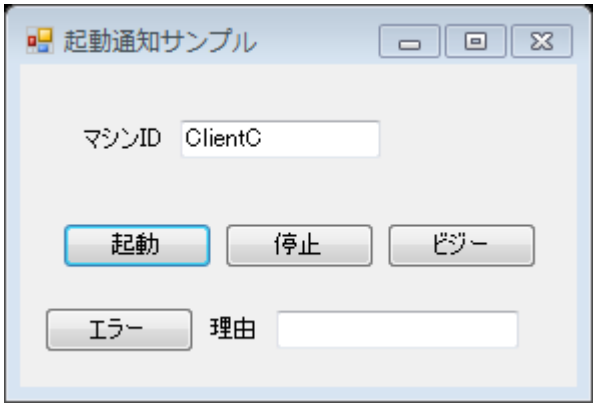

図 21 起動通知サンプル

通知の登録サンプル

業務アプリケーションが連携データを操作した際には、連携コントローラに対して通知を登録します。通知には、通知を送る相手先のマシン ID またはグループ ID を指定します。連携コントローラに通知が登録されると、連携コントローラが通知 ID を発行して、通知登録要求の戻り値として返します。通知相手の他の業務アプリケーションは、連携コントローラに接続して通知を確認し、通知の内容に応じて連携データを操作します。

なお、通知を登録すると、相手先の通知ログに通知が登録されたことが記録されます。

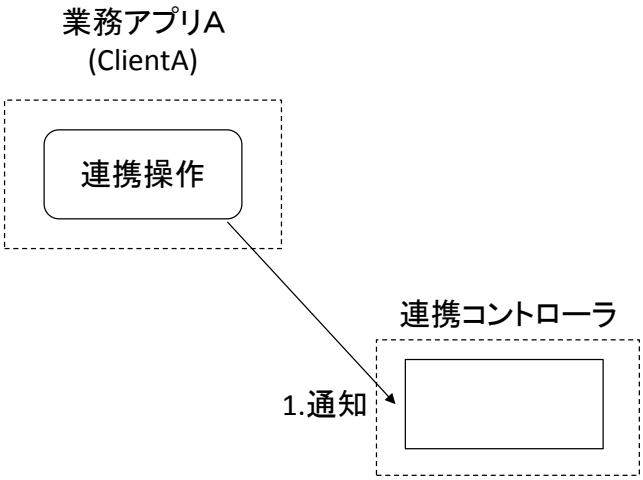

図 22 通知登録サンプルの動作概要図

業務アプリ A 側のサンプルコード例

通知を登録するには、NotificationClient クラスの AddNotification メソッドを使います。通知を登録すると戻り値として通知 ID が返されます。サンプルでは、連携データの挿入・更新・削除それぞれの通知をボタンによって送信できます。

```
public partial class Form1 : Form
{
    NotificationClient client = new NotificationClient();

    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        client.Server = "http://localhost/PSLX3Controller/index.php";
        client.SchemaName = "PSLX3";
    }

    private void insertNotifyButton_Click(object sender, EventArgs e)
    {
        client.MachineId = machineIdTextBox.Text;
        client.Password = client.MachineId;
        try
        {
            string notificationId = client.AddNotification(sendToTextBo
x.Text, objectNameTextBox.Text, keyTextBox.Text, DateTime.MinValue, Not
ificationTypeState.Insert);
            notificationIdTextBox.Text = notificationId;
        }
        catch (Exception ex)
        {
            MessageBox.Show("失敗しました" + ex.Message);
        }
    }
```

```
    private void updateNotifyButton_Click(object sender, EventArgs e)
    {
        client.MachineId = machineIdTextBox.Text;
        client.Password = client.MachineId;
        try
        {
            string notificationId = client.AddNotification(sendToTextBo
x.Text, objectNameTextBox.Text, keyTextBox.Text, DateTime.MinValue, Not
ificationTypeState.Update);
            notificationIdTextBox.Text = notificationId;
        }
        catch (Exception ex)
        {
            MessageBox.Show("失敗しました" + ex.Message);
        }
    }

    private void deleteNotifyButton_Click(object sender, EventArgs e)
    {
        client.MachineId = machineIdTextBox.Text;
        client.Password = client.MachineId;
        try
        {
           string notificationId = client.AddNotification(sendToTexBox.
Text, objectNameTextBox.Text, keyTextBox.Text, DateTime.MinValue, Notif
icationTypeState.Delete);
            notificationIdTextBox.Text = notificationId;
        }
        catch (Exception ex)
        {
            MessageBox.Show("失敗しました" + ex.Message);
        }
    }
```

連携コントローラへ登録した通知を取り消すには、DeleteNotification メ

ソッドを使います。取り消しの際には、通知 ID が必要です。

```
    private void cancelNotifyButton_Click(object sender, EventArgs e)
    {
        client.MachineId = machineIdTextBox.Text;
        client.Password = client.MachineId;
        try
        {
            client.DeleteNotification(notificationIdTextBox.Text,  sendT
oTextBox.Text);
        }
        catch (Exception ex)
        {
            MessageBox.Show("失敗しました" + ex.Message);
        }
    }
}
```

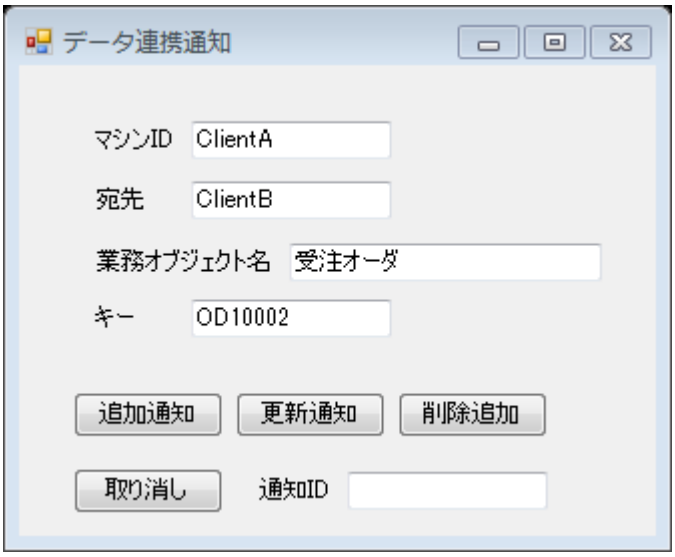

図 23 データ連携通知サンプル

連携通知の取得サンプル

業務アプリケーションは、他のアプリケーションからの連携データがある旨を連携コントローラからの通知によって認識します。通知を認識するため

に業務アプリは、ポーリング方式の場合一定間隔で連携コントローラに対して新着の通知があるかどうかを照会します。照会の結果、新着通知が存在する場合は、その通知をすべて取得して、通知を処理します。

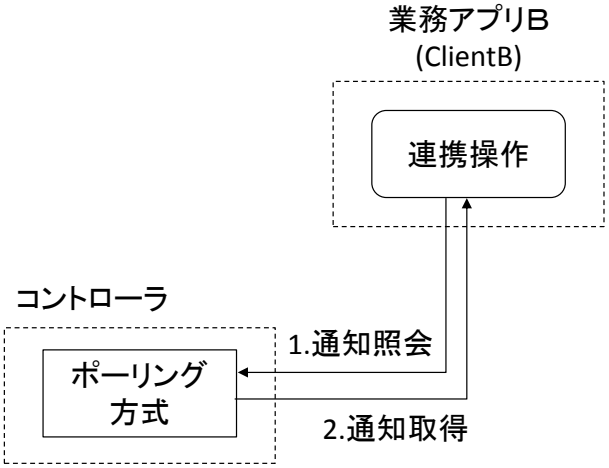

図 24 通知取得サンプルの動作概要図

業務アプリ B 側のサンプルコード例

マシンから登録された通知をコントローラに問い合わせるには、NotificationClient クラスの GetNotification メソッドを使います。次に示すサンプルコードでは、3 秒間隔で連携コントローラに対して新着の通知があるかどうかを照会し、新着通知が存在する場合は、それをすべて取得して、リストビューに表示します。

サンプルでは、[開始]ボタンをクリックすると、タイマーが動作して一定間隔で連携コントローラに新着通知が存在するかどうか照会します。

```
public partial class Form1 : Form
{
    NotificationClient client = new NotificationClient();
    DateTime last;

    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
```

```
    {
        client.Server = "http://localhost/PSLX3Controller/index.php";
        client.SchemaName = "PSLX3";
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        timer1.Interval = 3000;
        var list = client.GetNotification(last);
        foreach (var item in list)
        {
            var listviewItem = listView1.Items.Insert(0, item.Id);
            listviewItem.SubItems.AddRange(new string[] { item.MachineI
d, item.ObjectName, item.Key, item.Registered.ToString() });
            if (last < item.Registered) last = item.Registered;
        }
    }

    private void startButton_Click(object sender, EventArgs e)
    {
        if (!timer1.Enabled)
        {
            client.MachineId = machineIdTextBox.Text;
            client.Password = client.MachineId;
            timer1.Start();
            startButton.Text = "停止";
        }
        else
        {
            timer1.Stop();
            startButton.Text = "開始";
        }
    }
}
```

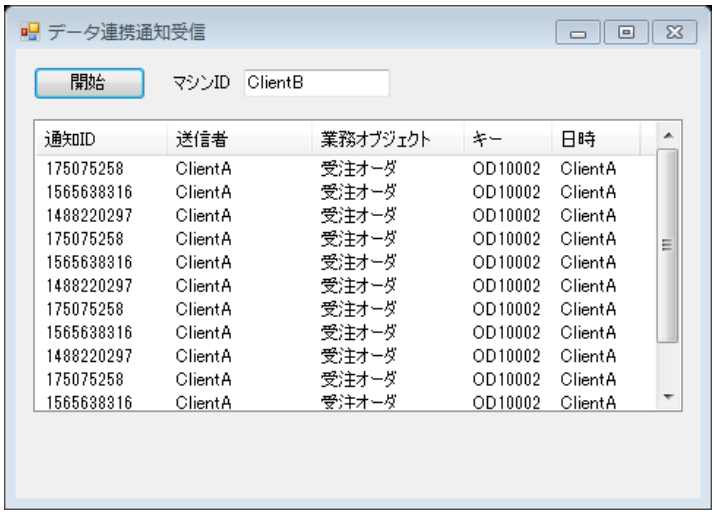

図 25 サンプルコード実行画面(連携通知の取得画面)

2 つの業務アプリケーション間でのデータ連携実装サンプル

2 つの業務アプリケーション間でデータ連携する例として、CSV 形式で連携する実装例を説明します。この実装例では、連携データをファイルサーバに保存しておき、連携コントローラでは通知のやりとりのみを行います。

CSV 形式の場合は、通知のキーに連携データが含まれる CSV ファイル名を記録します。

業務アプリケーション間でデータ連携するには、連携データを保存した業務アプリ(A)が、相手の業務アプリ(B)に対して、通知を行う必要があります。この際、業務アプリ A が業務アプリ B への通知を連携コントローラへ登録します。また業務アプリ B は、連携コントローラから新着通知がないかポーリング方式によって一定間隔で照会して、新着通知があれば、その通知の内容に応じて、保存先より連携データを取得して処理します。

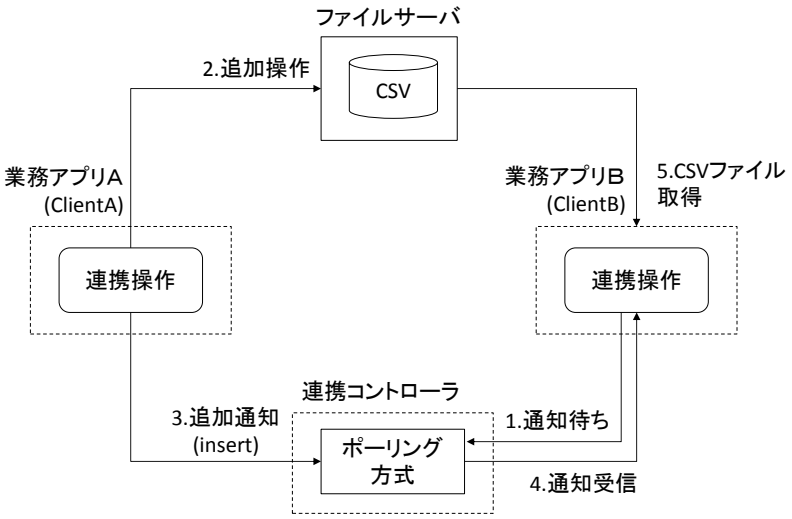

図 26 業務アプリ間の連携サンプルの動作概要図

業務アプリ A 側のサンプルコード例

業務アプリ A は、WriteCSV メソッド(仮称)によって連携データを保存場所に保存します。それと同時に NotificationClient クラスの AddNotification メソッドで、連携コントローラを通じて業務アプリ B に、連携データが挿入された旨を通知します。

通知登録後、業務アプリ A ではタイマーが作動してポーリング形式によって、業務アプリ B が連携データを受け取ったかどうかを通知ログから確認します。この確認処理は、業務アプリ B が連携データを受け取った、または読み取りに失敗した旨を通知ログが登録されるまで行われます。

```
public partial class Form1 : Form
{
    NotificationClient client = new NotificationClient();
    int counter = 1;
    string dataFolder;
    string notificationId;

    public Form1()
    {
        InitializeComponent();
    }
```

```
    private void Form1_Load(object sender, EventArgs e)
    {
        client.Server = "http://localhost/PSLX3Controller/index.php";
        client.SchemaName = "PSLX3";
        client.MachineId = "ClientA";
        client.Password = client.MachineId;

       dataFolder = Path.Combine(Environment.GetFolderPath(Environment.
SpecialFolder.MyDocuments), "csv") + "¥¥";
        client.ChangeMachineInfo(StoreType.CSV, dataFolder);
    }

    private void sendButton_Click(object sender, EventArgs e)
    {
        // 保存場所へ CSV ファイルを書き出します。
        WriteCSV();

        // 相手へ通知します。
        try
        {
            notificationId = client.AddNotification(sendToTextBox.Text,
 objectNameTextBox.Text, keyTextBox.Text, DateTime.MinValue, Notificati
onTypeState.Insert);
        }
        catch (Exception ex)
        {
            MessageBox.Show("失敗しました" + ex.Message);
        }
        timer1.Start();
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        timer1.Interval = 1000;
        PSLX3Log[] logs = client.GetLog(notificationId, sendToTextBox.T
```

```
ext, NotificationReadState.None, 0, DateTime.MinValue);
        if (logs == null || logs.Length == 0) return;

        foreach (var log in logs)
        {
            if (log.Status == NotificationReadState.Unread) continue;

            var item = new ListViewItem(new string[] { notificationId,
log.MachineId, log.Status.ToString(), log.Registered.ToString() });
            listView1.Items.Insert(0, item);
            if (log.Status == NotificationReadState.Read)
            {
                //ClientB が要求データを読み込んだ
                timer1.Stop();
            }
            else if (log.Status == NotificationReadState.Error)
            {
                //ClientB からエラーが返答された
                timer1.Stop();
            }
        }
    }
}
```

業務アプリ B 側のサンプルコード例

業務アプリ B は、業務アプリ B の通知ログから未読の通知がないかどうかを 1 秒間隔で照会します。通知ログから未読通知を取得するために、NotificationClient クラスの GetLog メソッドを呼び出します。

未読通知がある場合は、通知ログに含まれる通知 ID によって通知内容を取得して、連携データのファイル名を取得します。そして ReadCSV メソッド(仮称)によってファイルサーバから連携データを読み取り、画面に表示します。

通知を受け取り、その内容が処理されると、通知 ID に対応する通知ログとして、読み取り(Read)または失敗(Error)のログを連携コントローラに登録します。

```
public partial class Form1 : Form
{
    NotificationClient client = new NotificationClient();

    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        client.Server = "http://localhost/PSLX3Controller/index.php";
        client.SchemaName = "PSLX3";
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        timer1.Interval = 1000;

        //ClientB が未読のデータがあれば取得する
        PSLX3Log[] logs = client.GetLog(NotificationReadState.Unread, D
ateTime.MinValue);
        if (logs == null || logs.Length == 0) return;

        foreach (PSLX3Log log in logs)
        {
            string notificationId = log.NotificationId;

            PSLX3Notification notification = client.GetNotification(not
ificationId);
            if (notification == null) continue;

            PSLX3MachineInfo machine = client.GetMachineInfo(notificati
on.MachineId);

            //指定した保存場所から CSV ファイルを読み取ります
```

```
            if (ReadCSV(Path.Combine(machine.StoreAt, notification.Key))
            {
                //ClientB が要求データを読み込んだことをログに記録する
                client.AddLog(notificationId, NotificationReadState.Read);
                var item = new ListViewItem(new string[] { notificationId, log.MachineId, notification.Status.ToString(), notification.Registered.ToString() });
                listView1.Items.Insert(0, item);
            }
            else
            {
                //ClientB で要求データを読み込めなかったことをログに記録する
                client.AddLogError(notificationId, 100, "読み込みに失敗しました。");
            }

            break;
        }
    }
}
```

連携コントローラ通信ライブラリ仕様

連携コントローラ通信ライブラリは、業務アプリから連携コントローラへ接続するためのライブラリです。.NET Framework2.0 版および Java 版が提供されています。この章の説明では、C#上での利用を前提した.NET Framework2.0 版のクラスライブラリについて記載します。

PSLX3NotificationClient クラス

業務アプリケーションソフトウェアから連携コントローラへ接続するためのクラスです。

メソッド

PSLX3Notification GetNotification(string id)
通知を受け取ります。 id:通知ID
PSLX3Notification[] GetNotification(DateTime registered)
指定した日時以降の通知を受け取ります。 last:取得対象の開始日時
PSLX3Notification[] GetNotification(string objectName, string key, DateTime expires, string sendTo, NotificationTypeState status, DateTime registered)
通知を受け取ります。 objectName:業務オブジェクト名 key:キー expires:有効期限 sendTo:宛先 status:状態 last:取得対象の開始日時
PSLX3Notification[] GetNotification(string id, string objectName, string key, DateTime expires, string sendTo, NotificationTypeState status, DateTime registered)
通知を受け取ります。 id:通知ID objectName:業務オブジェクト名 key:キー expires:有効期限 sendTo:宛先 status:状態 registered:取得対象の開始日時
string AddNotification(string sendTo, string objectName, string key, DateTime expires, NotificationTypeState status)
連携コントローラへ通知します。 objectname:業務オブジェクト名 key:キー expires:有効期限 sendTo:宛先 status:状態

bool DeleteNotification(string id, string sendTo)
通知を取り消します。 id:通知ID sendTo:宛先
PSLX3MachineInfo GetMachineInfo(string machineId)
指定したマシンIDの状態を取得します。 machineId:取得するマシンのID
string AddMachine(string id, StoreType storeType, string storeAt, string group, MachineGrant grant)
マシンを追加します。 id:マシンID storeType:保存方法 storeAt:保存場所 group:所属するグループ(複数指定はカンマ区切り) grant:権限
bool ChangeMachineInfo(StoreType storeType, string storeAt)
マシンの連携データの保存先や権限を変更します。 storeType:保存方法 storeAt:保存場所
bool ChangeMachineState(MachineState status)
マシンの状態が変化したことを通知します。 status:状態
bool ChangeMachineStateError(int code, string remark)
マシンの状態が変化したことを通知します。 code:エラーコード remark:説明
bool DeleteMachine(string machineId)
指定したマシンを削除します。 machineId:マシンID
PSLX3Log[] GetLog(NotificationReadState state, DateTime registered)
指定した状態にある履歴を取得します。 state:状態 registered:取得対象の開始日時
PSLX3Log[] GetLog(string notificationId, NotificationReadState state, DateTime registered)
指定した状態にある履歴を取得します。

notificationId:通知ID state:状態 registered:取得対象の開始日時
PSLX3Log[] GetLogFor(string machineid, NotificationReadState state, DateTime registered)
指定したマシンIDへの履歴を取得します。 machineid:マシンID state:状態 registered:取得対象の開始日時
PSLX3Log[] GetLog(string notificationId, string machineid, NotificationReadState state, int code, DateTime registered)
指定した通知IDの履歴を取得します。 notificationId:通知ID state:状態 code:エラーコード registered:取得対象の開始日時
AddLog(string notificationId, NotificationReadState state)
履歴を連携コントローラへ登録します。 notificationId:通知ID state:状態
AddLogError(string notificationId, int code, string remark)
エラーが発生したことを示す履歴を連携コントローラへ登録します。 notificationId:通知ID code:エラーコード remark:説明

プロパティ

プロパティ名	値
ConnectionType ConnectionType	連携コントローラとの接続方法
string SchemaName	標準スキーマ
string MachineId	業務アプリケーションソフトウェアを識別するマシン ID
string Password	連携コントローラ接続時のパスワード

PSLX3Notification クラス

通知メッセージを表すクラスです。PSLX3ControllerResult クラスを継承します。

プロパティ

プロパティ名	値
string Id	通知メッセージ ID
string MachineId	マシン ID
string SendTo	宛先
string SchemaName	スキーマ名
StoreType StoreType	保存形式
string ObjectName	業務オブジェクト名
string Key	業務データのキー
NotificationTypeState Status	通知の種類
DateTime Updated	更新された日時
DateTime Registered	登録された日時
DateTime Expires	通知の有効期限

PSLX3MachineInfo クラス

マシン情報を表すクラスです。PSLX3ControllerResult クラスを継承します。

プロパティ

プロパティ名	値
string MachineId	マシン ID
MachineState Status	マシンの状態
string StoreAt	連携データの保存位置
StoreType StoreType	連携データの保存形式
MachineGrant Grant	マシンの権限
DateTime Registered	マシンの登録日

PSLX3Log クラス

通知の履歴を表すクラスです。PSLX3ControllerResult クラスを継承します。

プロパティ

プロパティ名	値
string NotificationId	通知メッセージ ID
string MachineId	マシン ID
bool Read	既読かどうか
NotificationReadState Status	通知の状態
DateTime Registered	マシンの登録日

PSLX3ControllerResult クラス

プロパティ

プロパティ名	値
int Code	状態コード
string Remark	メッセージ
bool IsError	エラー状態かどうか

ConnectionType 列挙体

値

プロパティ名	値
None	なし、定時方式
Online	オンライン方式
Polling	ポーリング方式

MachineState 列挙体

値

値名	値
None	不明
Ready	受付状態
Off	停止

Busy	処理中
Error	エラー状態

MachineGrant 列挙体

値

値名	値
Read	読み取りのみ
Write	書き込みのみ
ReadWrite	読み取りと書き込み
Admin	管理者

StoreType 列挙体

値

値名	値
CSV	CSV 形式
RDB	RDB 形式
WebDb	WebDB 形式

NotificationTypeState 列挙体

値

値名	値
None	なし
Read	読み取り操作
Insert	追加操作
Update	更新操作
Delete	削除操作
Request	他マシンへの問い合わせ
Response	他マシンからの回答

Canceled	通知の取り消し
Error	エラー

NotificationReadState 列挙体

値

値名	値
None	不明
Unread	未読
Read	既読
Replied	返信済み
Completed	完了
Error	エラー