

Oracle Package を DB2 UDB で実現する サンプル・プログラムの紹介

日本アイ・ビー・エム株式会社
ソフトウェア事業部
ソフトウェア・テクニカルサポート DM技術部
開発技術支援グループ

目次

1 . はじめに	1
1 . 1 目的	1
1 . 2 対象とする読者	1
1 . 3 検証環境	1
1 . 4 商標	1
1 . 5 変更履歴	2
2 . 移行の流れ	3
3 . PL/SQL で提供されている主なパッケージ	4
3 . 1 DBMS_SQL の移行について	4
3 . 2 DBMS_OUTPUT の移行について	4
3 . 3 DBMS_STANDARD の移行について	4
3 . 4 UTL_FILE の移行について	4
4 . サンプル	5
4 . 1 DBMS_PIPE Windows 版 C 言語	5
4 . 1 . 1 処理概要	5
4 . 1 . 2 サンプル・プログラムを用いた DBMS_PIPE 機能の使用方法的概要	5
4 . 1 . 3 制限事項	7
4 . 1 . 4 外部仕様	8
4 . 1 . 5 データ構成	15
4 . 2 DBMS_PIPE UNIX 版 C 言語	18
4 . 2 . 1 処理概要	18
4 . 2 . 2 サンプル・プログラムを用いた DBMS_PIPE 機能の使用方法的概要	18
4 . 2 . 3 制限事項	19
4 . 2 . 4 外部仕様	19
4 . 2 . 5 IPC 資源及びワークファイルについて	20
4 . 2 . 6 補足説明	21
4 . 3 DBMS_PIPE Windows 版 Java 言語	23
4 . 3 . 1 処理概要	23
4 . 3 . 2 サンプル・プログラムを用いた DBMS_PIPE 機能の使用方法的概要	23
4 . 3 . 3 制限事項	24
4 . 3 . 4 外部仕様	24
4 . 4 DBMS_PIPE UNIX 版 Java 言語	31
4 . 5 DBMS_ALERT Windows 版 C 言語	31
4 . 5 . 1 処理概要	31
4 . 5 . 2 サンプル・プログラムを用いた DBMS_ALERT 機能の使用方法的概要	31
4 . 5 . 3 制限事項	33
4 . 5 . 4 外部仕様	34
4 . 6 DBMS_ALERT UNIX 版 C 言語	40

4 . 7 DBMS_ALERT Windows 版 Java 言語	4 0
4 . 7 . 1 処理概要	4 0
4 . 7 . 2 サンプル・プログラムを用いた DBMS_ALERT 機能の使用方法的概要	4 0
4 . 7 . 3 制限事項	4 0
4 . 7 . 4 外部仕様	4 0
4 . 8 DBMS_ALERT UNIX 版 Java 言語	4 4

1 . はじめに

1 . 1 目的

Oracleで開発されたシステムをDB2 UDBへ移行する際、アプリケーションの移行も必須となります。PL/SQLで提供されているパッケージはOracle環境で開発された多くのアプリケーションで使用されていますが、DB2 UDBではそれに該当する機能は提供されていません。従って、その機能をDB2 UDB環境で使用する場合、該当する機能を独自に開発する必要があります。本ガイドではDB2 UDB環境でPL/SQLで提供されているパッケージに相当する機能を実現するサンプル・プログラムを紹介し、実際の移行作業の際に、DBMSなどのパッケージをすみやかに移行するための情報の提供を目的としています。

1 . 2 対象とする読者

本ガイドでは、PL/SQLで提供されているパッケージ機能をDB2 UDB環境へ移行するにあたり、その設計・開発に携わる方を対象とします。また、Oracle、DB2 UDB、PL/SQL、CおよびJavaに関する基礎的な知識があることを前提とします。

1 . 3 検証環境

本ガイドは以下の環境を前提としています。したがって掲載しているサンプル、手順等は環境が異なると動作しない可能性があります。

OS: Windows 2000 Professional + SP3 または、RedHat Linux 7.3

DB2: DB2 UDB EE 7.2 + FixPax7

Oracle: Oracle 8i

Cコンパイラ: MS Visual Studio V6 + SP5

Java: J2SDK 1.3.1

1 . 4 商標

Oracleは、米国オラクルの商標または登録商標です。

Windows 2000は、Microsoft Corp.の商標または登録商標です。

DB2 UDBは、IBM Corporationの商標または登録商標です。

本ガイドに使用されているその他の会社名、製品名は各社の商標または登録商標です。

2. 移行の流れ

Oracle 環境で稼動するシステムを DB2 UDB 環境で稼動するシステムへ移行するためには、大きく分けて以下の 3 つの移行を順番に行う必要があります。初めにデータベースの移行を行います。次に開発/実行環境の移行を行い、最後にアプリケーションの移行を行います。

それぞれの移行項目については、別ガイドを参照してください。

データベースの移行	Oracle 8i Java から DB2 UDB V7.2 Java への移行ガイド	3 章
開発/実行環境の移行	Oracle 8i Java から DB2 UDB V7.2 Java への移行ガイド	4 章
アプリケーションの移行	Oracle 8i Java から DB2 UDB V7.2 Java への移行ガイド	5 章
	PL/SQL から DB2 UDB v7.2Java への移行ガイド	2 章以降

本ガイドではアプリケーションの移行のなかでも PL/SQL で提供されているパッケージの機能を DB2 UDB 環境で実現する方法について、いくつかの具体的な移行サンプル・プログラムを通して、紹介します。

サンプル・プログラムは本ガイドと同じ以下の Web ページからダウンロードできます。

<http://www-6.ibm.com/jp/software/data/db2/migration/kanren.html>

入手できない場合は、下記に連絡してください。

日本アイ・ビー・エム株式会社

ソフトウェア事業部 ソフトウェア・テクニカルサポート DM技術部 開発技術支援グループ

3 . PL/SQL で提供されている主なパッケージ

PL/SQL で提供されているパッケージの主なものには以下があります。本章ではこれらの中から幾つかの移行について簡単な解説と、次章ではこれらの中から使用頻度が特に高い DBMS_ALERT と DBMS_PIPE の移行についてのサンプル・プログラムを紹介します。

パッケージ名	説明
DBMS_ALERT	データベース事象の非同期通知
DBMS_DDL	DDL コマンド (一部) の PL/SQL 版
DBMS_JAVA	Oracle J Server での Java VM
DBMS_JOB	PL/SQL のプロシージャのスケジューリング
DBMS_LOB	Oracle8 の LOB 操作
DBMS_OUTPUT	SQL*Plus またはサーバーマネージャーから、画面への出力
DBMS_PIPE	セッション間の非同期通信
DBMS_PROFILE	Probe への API
DBMS_SQL	動的な PL/SQL 及び SQL
DBMS_STANDARD	PL/SQL の組み込みファンクションとプロシージャ
DBMS_TRACE	PL/SQL のトレーシングの制御
UTL_COLL	SQL のコレクションロケータの操作
UTL_FILE	ファイル I/O
UTL_REF	PL/SQL の REF を通じのアクセス

3 . 1 DBMS_SQL の移行について

PL/SQL はデータ定義言語 (DDL) や動的 SQL を標準ではサポートしていません。これらの機能を実現するためには、Oracle 環境では DBMS_SQL パッケージを利用します。DB2 UDB 環境では組み込み SQL ステートメントで、動的 SQL を標準でサポートしていますので、そのままアプリケーションで用いることができます。DB2 コール・レベル・インターフェース (CLI) でも C と C++ の動的 SQL アプリケーションを作成することができます。Java プログラムでは JDBC API を使用しますと動的 SQL になります。SQLJ は静的 SQL ですので注意してください。

3 . 2 DBMS_OUTPUT の移行について

DBMS_OUTPUT パッケージを使用して、プロシージャやファンクションから値やメッセージを出力することができます。DB2 UDB 環境では、本ガイドの C 言語プログラムの様に、ユーザー定義関数を作成し、その中に「printf」の様なメッセージ出力機能を組み込んでも画面には出力されません。これは起動したコマンド・ウィンドウが直接呼び出していないからです。テキストファイルに書き出すなどの方法に切り替える等の検討が必要です。

3 . 3 DBMS_STANDARD の移行について

DBMS_STANDARD の「RAISE_APPLICATION_ERROR」というプロシージャを使用すると、アプリケーション固有のエラーを定義することができます。DB2 UDB への移行は、本ガイドの Java 言語プログラムに、「RaiseApplicationErrorException」クラスがありますので参照してください。

3 . 4 UTL_FILE の移行について

DB2 UDB 環境では、本ガイドの C 言語プログラムの様に、ユーザー定義関数を作成すればその中で「fopen」等のファイルのアクセス機能を組み込むことができます。

4 . サンプル

本章では PL/SQL で提供されているパッケージの中から使用頻度が特に高い、DBMS_PIPE と DBMS_ALERT の移行について、それぞれの機能を実現するための Windows と UNIX について C 言語と Java 言語のサンプルを以下の章で紹介します。

	Windows 版 C 言語	UNIX 版 C 言語	Windows 版 Java 言語	UNIX 版 Java 言語
DBMS_PIPE	4.1 章	4.2 章	4.3 章	4.4 章
DBMS_ALERT	4.5 章	4.6 章	4.7 章	4.8 章

4 . 1 DBMS_PIPE Windows 版 C 言語

ここでは PL/SQL で提供されているパッケージで、DBMS_PIPE を DB2 UDB、Windows 環境で稼動する C 言語アプリケーションで実現するサンプルについて紹介します。

4 . 1 . 1 処理概要

サンプル・プログラムでの内部処理について述べます。

- (1) Oracle の DBMS_PIPE の各機能を C 言語アプリケーションで作成し、ユーザー定義関数として使用します。
- (2) ユーザー単位でパイプに相当する共有メモリを割り付け、これを排他的に使用しながら、SEND と RECEIVE を行います。
- (3) 共有メモリはファイル・マッピング・オブジェクトとし、ファイルとして作成します。
- (4) 排他制御はメモリマップドファイルを非共有モードでマッピングし、他者が使用中の場合は、Wait することで対応しています。

4 . 1 . 2 サンプル・プログラムを用いた DBMS_PIPE 機能の使用方法的概要

(1) 送り手

```
CREATE_PIPE( ' session1 ' )
```

但し、' session1 ' はパラメータの例です。

- ・パラメータで渡された名前の先頭に"PIPE_"付与し「ファイル・マッピング・オブジェクト」を作成します。(データの構成は 4 . 1 . 1 (5) を参照)

例として、ファイル " C:¥Temp¥PIPE_session1 " を作成します。

- ・パイプ名の後ろに"_ATTR"を付与し、パイプ属性管理ファイルを作成します。(データの構成は 4 . 1 . 5 を参照)

例として、ファイル " C:¥Temp¥PIPE_session1_ATTR " を作成します。

```
PACK_MESSAGE
```

- ・ローカル・バッファ用ファイル・マッピング・オブジェクトが無ければファイルとして作成し、内部バッファとします。
- ・ローカル・バッファ用ファイル・マッピング・オブジェクトには管理用とデータ用があります。
- ・データを内部バッファに詰め込みます。
- ・PACK しただけでは名前付きパイプ(ファイル)には書き込みません。

```
SEND_MESSAGE
```

- ・データ別の型とデータ長を添付して、内部バッファから名前付きパイプ(ファイル)に書

き込みます。(データの構成は4 . 1 . 5を参照)

- ・ 内部バッファの管理用とデータ用をクリアします。
REMOVE_PIPE
- ・ 作成したパイプ用ファイル・マッピング・オブジェクト等を削除します。
- ・ ローカル・バッファ用ファイル・マッピング・オブジェクトも一括して削除します。
PURGE
- ・ 名前付きパイプ内のデータを全て読み出し、ファイル・マッピング・オブジェクトをクリアします。
RESET_BUFFER
- ・ バック及びアンパック用管理テーブルのリセットとバッファのクリアを行います。

(2) 受け手

RECEIVE_MESSAGE

- ・ パイプが存在しない場合、暗黙モードでパイプを作成します。
- ・ パラメータで渡されたユーザー名を使用してパック用と管理用の「ファイル・マッピング・オブジェクト」を作成します。
- ・ 名前付きパイプよりデータを SEND 単位で読み込みます。
- ・ データ型を順番に管理テーブルに記録します。
- ・ 指定時間内にデータがない場合はタイムアウトを戻しますが、暗黙モードで作られたパイプはこのタイミングで削除されます。

NEXT_ITEM_PIPE

- ・ ファイル・マッピング・オブジェクトのデータ型を読んで戻します。

UNPACK_MESSAGE

- ・ ファイル・マッピング・オブジェクトの先頭よりデータをコピーします。
- ・ コピーしたデータ部分をファイル・マッピング・オブジェクトより削除します。
- ・ データの型が一致しない場合は ORA-06559 エラー相当とします。
- ・ データが存在しない場合は ORA-06556 エラー相当とします。

(3) 補足

パラメータで渡された「ユーザー名」を使いますので、権限管理はアプリケーション側で管理します。

複数のパイプが作成されても名前が違えば平行に運用できます。

暗黙モードの場合は PACK 時に資源を作成し、全データを読み込んだタイミングで削除します。

他ユーザーが作成した同一名のパイプが存在した場合は ORA-23322 (重複) エラー相当とします。

4.1.3 制限事項

以下に、サンプル・プログラムの制限事項について述べます。

- (1) パイプ間の通信は同一マシン内に限定されます。
- (2) 各種メモリマップドファイル及びログファイルを作成するディレクトリは DB2 UDB を起動しているユーザー環境変数の “ TMP ” に依存します。上記環境変数の設定がなされていない場合は、デフォルトとして “ C:¥Temp ” が使用されます。
- (3) PL/SQL ではパラメータのデフォルト値、例えば CREATE_PIPE でのパイプサイズ、が設定可能ですが、ここでは各パラメータを明示的に設定する必要があります。
- (4) UNIQUE_SESSION_NAME ファンクションは名前の制限 (18 バイト以下) により、UNIQUE_SESSION となっています。
- (5) データの上限を 8K Byte とします。

4.1.4 外部仕様

ユーザー定義関数または、ストアード・プロシージャです。

(1) CREATE_PIPE

```
>>---CREATE_PIPE---(---pipename-----in-----varchar----->
>-----,-----maxpipesize----in-----integer----->
>-----,-----private-----in-----integer-----)-->
>-----RETURN-----integer-----<
```

スキーマは DBMS_PIPE です。

CREATE_PIPE ファンクションはパイプを作成します。

戻り値

- 0 : 成功。
- 2 : ファイル作成エラー。
- 3 : メッセージ ID 取得エラー。
- 23322 : パイプへのアクセス権限不足。

pipename

このセッションに作成するパイプ名。

maxpipesize

パイプの最大サイズ。メッセージの合計サイズは、このサイズを超えることはできません。

private

プライベート、パブリック・パイプの指定。

- 0 : パブリック・パイプ。
- 1 : プライベート・パイプ。

プライベート・パイプ

一度作成されると、REMOVE_PIPE ファンクションをコールして明示的に解除するまで存在します。

パブリック・パイプ

暗黙モードまたは、明示モードで作成されます。暗黙モードなパブリック・パイプはそのパイプの最初の参照時に自動的に作成され、データがなくなると消去されます。

(2) PACK_MESSAGE

```
>>---PACK_MESSAGE_VARCH-----(-item---in-----varchar----->  
>-----,-----ret---out-----integer-----)<-----<
```

```
>>---PACK_MESSAGE_INT-----(-item---in-----integer----->  
>-----,-----ret---out-----integer-----)<-----<
```

```
>>---PACK_MESSAGE_TIMEST-----(-item---in-----timestamp----->  
>-----,-----ret---out-----integer-----)<-----<
```

スキーマは DBMS_PIPE です。

PACK_MESSAGE プロシージャはユーザーのメッセージを送信バッファにため込みます。

戻り値

なし。

item

メッセージ。TimeSet の場合は例として ' 2002-12-20-13.59.555555 ' など。

ret

ステータス。

0 : 成功。

6558 : バッファオーバーフロー。

-2 : ファイル作成エラー。

-5 : 共有メモリ取得エラー。

(3) SEND_MESSAGE

```
>>---SEND_MESSAGE---(---pipename-----in-----varchar----->
>-----,-----timeout-----in-----integer----->
>-----,-----maxpipesize----in-----smallint-----)>
>-----RETURN---integer-----><
```

スキーマは DBMS_PIPE です。

SEND_MESSAGE ファンクションは蓄積したメッセージをパイプに送信します。指定したパイプがない場合、暗黙モードのパブリック・パイプを作成します。割り込みが発生した場合、暗黙モードのパブリック・パイプは削除します。

戻り値

- 0 : 成功。
- 1 : タイムアウト。
- 2 : ファイル作成エラー。
- 3 : メッセージ ID 取得エラー。
- 5 : 共有メモリ取得エラー。
- 23322 : パイプに対する権限不足。

pipename

メッセージを設定するパイプの名前。

timeout

パイプにメッセージを設定する間の待機時間 (秒)

maxpipesize

パイプの最大バイト長。

(4) RECEIVE_MESSAGE

```
>>---RECEIVE_MESSAGE---(---pipename-----in---varchar----->
>-----,-----timeout-----in---integer-----)>
>-----RETURN---integer-----<
```

スキーマは DBMS_PIPE です。

RECEIVE_MESSAGE ファンクションはパイプからのメッセージをためます。またパイプにあるメッセージを削除します。暗黙モードで作成されたパブリック・パイプの場合、パイプは削除します。

戻り値

- 0 : 成功。
- 1 : タイムアウト。
- 2 : パイプにあるレコードがバッファに対して大きすぎます。
- 23322 : パイプに対する権限不足。
- 2 : ファイル作成エラー。
- 3 : メッセージ ID 取得エラー。
- 5 : 共有メモリ取得エラー。

pipename

メッセージを受信するパイプ名。

timeout

メッセージを待つ時間 (秒)。

(5) NEXT_ITEM_TYPE

```
>>---NEXT_ITEM_TYPE-(-)----->
>-----RETURN---integer-----<
```

スキーマは DBMS_PIPE です。

NEXT_ITEM_TYPE ファンクションは,RESEIVE_MESSAGE によってためられたメッセージの次の項目データ型を判別します。

戻り値

- 0 : 次項目がありません。
- 6 : integer。
- 9 : varchar。
- 12 : timestamp。
- 5 : 共有メモリ取得エラー。

(6) UNPACK_MESSAGE

```
>>---UNPACK_MESSAGE_VARCH---(---item---out-----varchar----->
>-----,-----ret-----out-----integer-----)>><

>>---UNPACK_MESSAGE_INT----- (---item---out-----integer----->
>-----,-----ret-----out-----integer-----)>><

>>---UNPACK_MESSAGE_TIMEST---(--tem-----out-----timestamp----->
>-----,-----ret-----out-----integer-----)>><
```

スキーマは DBMS_PIPE です。

UNPACK_MESSAGE_VARCH プロシージャは次の項目の型が VARCHAR の場合メッセージを取り出します。

UNPACK_MESSAGE_INT プロシージャは次の項目の型が INTEGER の場合メッセージを取り出します。

UNPACK_MESSAGE_TIMEST プロシージャは次の項目の型が TIMESTAMP の場合メッセージを取り出します。

戻り値

なし。

item

アンパックされた次のメッセージを受け取るための引数。

ret

0 : 成功。

6556 : バッファに次項目がない。

6559 : データ型が異なる。

-5 : 共有メモリ取得エラー。

(7) REMOVE_PIPE

```
>>---REMOVE_MESSAGE---(---pipename---in---varchar-----)>
>-----RETURN---integer-----<
```

スキーマは DBMS_PIPE です。

REMOVE_PIPE ファンクションは明示的に作成されたパイプを削除します。

戻り値

0 : 成功。

23322 : パイプに対する権限不足。

pipename

削除するパイプ名。

(8) PURGE

```
>>---PURGE---(---pipename---in---varchar----->
>-----,---ret-----out---integer-----)<
```

スキーマは DBMS_PIPE です。

PURGE プロシージャは名前付きパイプの内容を空にします。

戻り値

なし。

pipename

全てのメッセージを削除するパイプ名。

ret

ステータス。

0 : 成功。

23322 : パイプに対する権限不足。

(9) RESET_BUFFER

>>---RESET_BUFFER---(----ret-----out-----integer-----)-----><

スキーマは DBMS_PIPE です。

RESET_BUFFER プロシージャは PACK_MESSAGE、UNPACK_MESSAGE で蓄積したメッセージを削除し
ます。

戻り値

なし。

ret

ステータス。

0 : 成功。

-5 : 共有メモリ取得エラー。

(1 0) UNIQUE_SESSION

>>---UNIQUE_SESSION---(---)-----><

スキーマは DBMS_PIPE です。

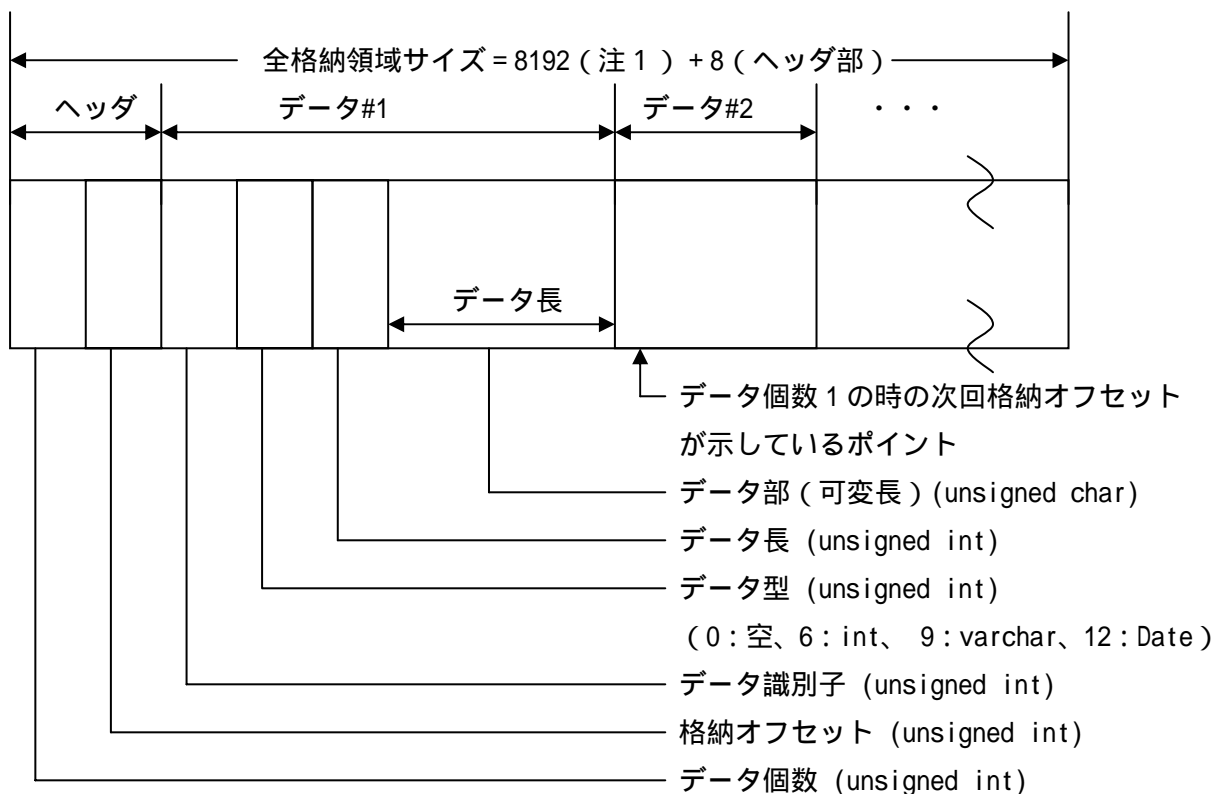
UNIQUE_SESSION ファンクションは現在接続している全てのセッション間での一意の名前を受け取
ります。この関数は DBMS_PIPE で内部的に呼ばれます。

戻り値

アプリケーション ID。

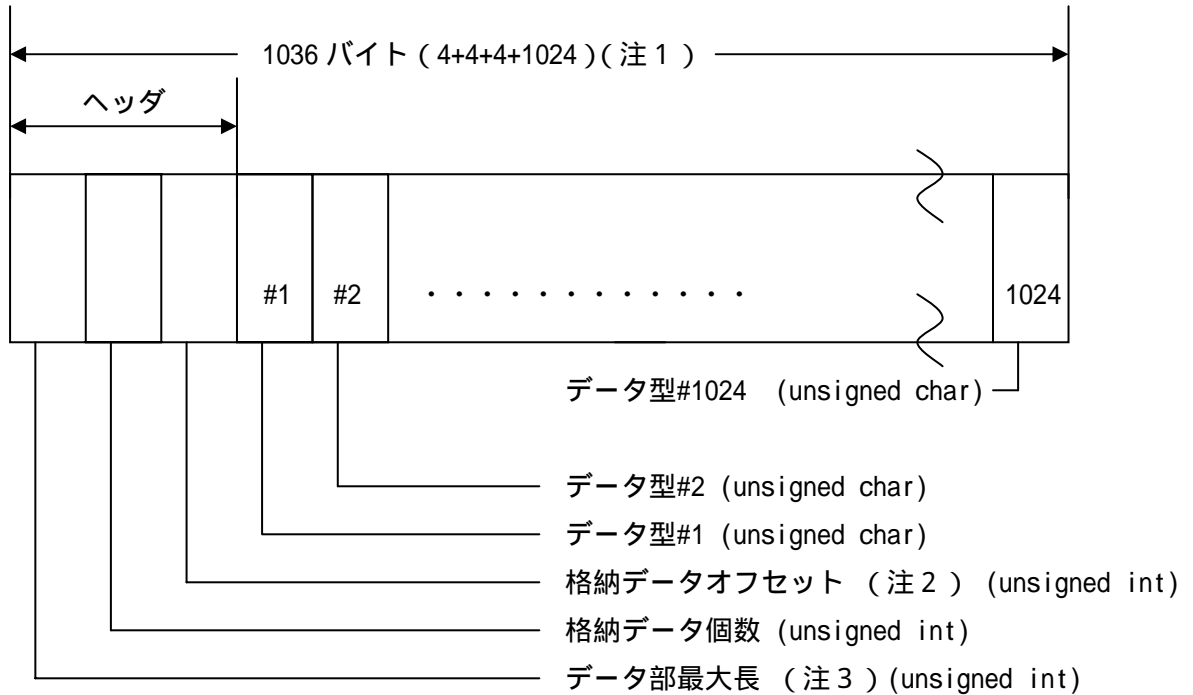
4.1.5 データ構成

(1) パイプ上のデータ構成



注1 : パイプ領域としては 8192 バイトを割り付けますが、実際に使用できるサイズはパイプの属性情報として指定 (PIPE 関数のパラメータにより) されたサイズに制約されます。

(2) 管理用ローカル・バッファのデータ構成 (UNIX C 言語と共通構成)



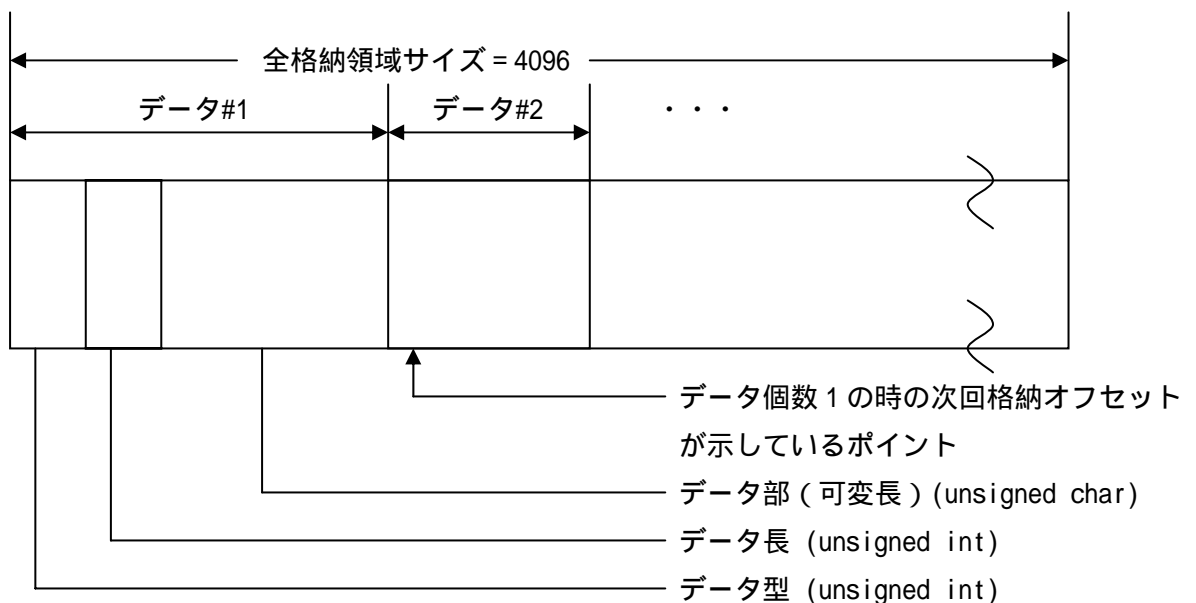
注 1 : 「データ型」を管理する配列の上限が 1024 の根拠は、パック及びアンパック用の共有メモリのサイズが 4096 固定なので、仮に各パックデータが 1 バイトの場合でも、各データに付属するヘッダ部 8 バイトを考慮すると、実際には 9 バイトの格納領域が必要となるので 1024 で十分なサイズとなります。(4096 ÷ 9) < 1024 となります。

1024 にヘッダ部分の 4+4+4=12 バイトが加算されます。

注 2 : データ用ローカル・バッファの格納データオフセットです。

注 3 : パック及びアンパック用の共有メモリのサイズですので、値は 4096 固定です。

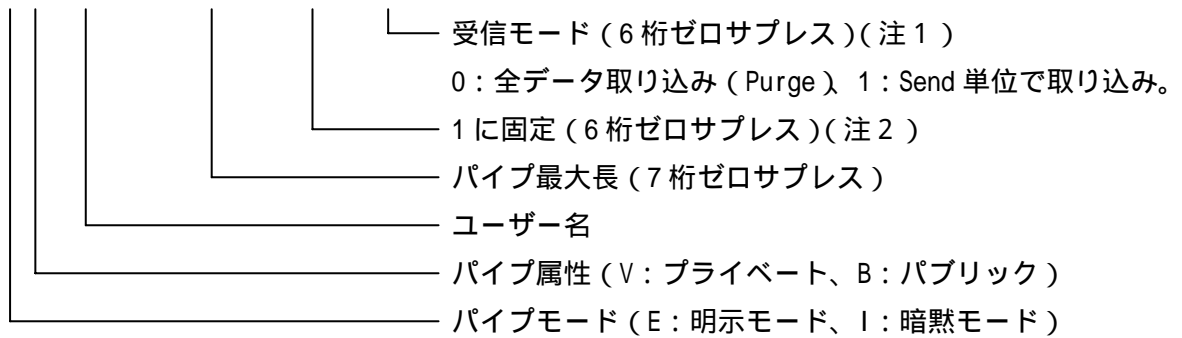
(3) データ用ローカル・バッファのデータ構成 (UNIX C 言語と共通構成)



(4) パイプ属性管理ファイルのデータ構成 (UNIX C 言語と共通構成)

テキストファイル形式です。区切り記号は “,” です。

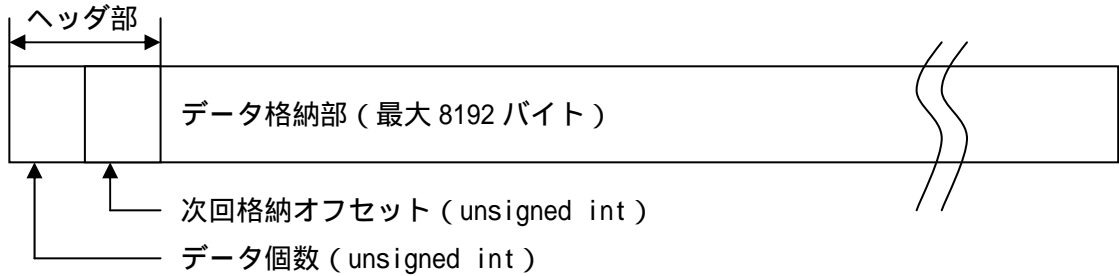
E,V,xxxxxxxx,008200,000001,000001



注1 : UNIX 版では Receive 回数 (6桁ゼロサプレス)

注2 : UNIX 版では Send 回数 (6桁ゼロサプレス)

(5) パイプ代行共有メモリ (ファイル・マッピング・オブジェクト) 上のデータ構成



4.2 DBMS_PIPE UNIX 版 C 言語

ここでは PL/SQL で提供されているパッケージで、DBMS_PIPE を DB2 UDB、UNIX 環境で稼動する C 言語アプリケーションで実現するサンプルについて紹介します。

4.2.1 処理概要

「4.1.1 処理概要 (Windows 版 C 言語)」と同様です。

4.2.2 サンプル・プログラムを用いた DBMS_PIPE 機能の使用法の概要

(1) 送り手

```
CREATE_PIPE( ' session1 ' )
```

```
/tmp/session1
```

- ・作成したファイル名をパラメータとして ftok() をコール。
 - ・ftok() で取得したキーを元にメッセージを 1 個割り付けます。
- ```
PACK_MESSAGE
```
- ・固定名称 (/tmp/PACK\_MESSAGE\_) + ユーザー名をキーパスとして共有メモリを 2 個 (データ用と管理テーブル用) 割り付けます。
  - ・データを共有メモリのデータ用に詰めます。(データの PACK 構成は 4.1.5 (3) を参照。)
  - ・データ型を順番に共有メモリの管理用に記録します。構成は 4.1.5 (2) を参照。

```
SEND_MESSAGE
```

- ・データ別にメッセージタイプを分けてメッセージを送信します。

```
9 : VARCHAR
```

```
6 : NUMBER
```

```
12 : TIMESTAMP
```

```
REMOVE_PIPE
```

- ・作成したファイル、メッセージ・キュー、共有メモリ等を削除します。
- ・ローカル・バッファ用共有メモリも一括して削除します。

```
PURGE
```

- ・メッセージ・キューを NO\_WAIT で読み出します。

```
RESET_BUFFER
```

- ・バック及びアンパック用管理テーブルのリセットとバッファのクリアを行います。

#### (2) 受け手

```
RECEIVE_MESSAGE
```

- ・パイプが存在しない場合は、暗黙モードで作成します。
- ・固定名称 (/tmp/UNPACK\_MESSAGE\_) + ユーザー名をキーパスとして共有メモリを 2 個 (データ用と管理テーブル用) 割り付けます。
- ・メッセージ・キューよりデータを SEND 単位で読み込みます。
- ・データ型を順番に管理テーブルに記録します。

- ・指定時間内にデータがない場合はタイムアウトを戻しますが、暗黙モードで作られたパイプはこのタイミングで削除されます。

NEXT\_ITEM\_PIPE

- ・管理用共有メモリ上のデータ型を順に読んで戻します。  
UNPACK\_MESSAGE
- ・共有メモリの先頭よりデータをコピーします。
- ・データの型が一致しない場合は ORA-06559 エラー相当とします。
- ・データが存在しない場合は ORA-06556 エラー相当とします。

#### 4.2.3 制限事項

以下に、サンプル・プログラムの制限事項について述べます。

- (1) パイプ間の通信は同一マシン内に限定されます。
- (2) パイプの代行として IPC リソースの中のメッセージ機能を使用しているため、1度にパイプに送信できる上限はシステムの設定に依存します。
- (3) PL/SQL ではパラメータのデフォルト値 (例: CREATE\_PIPE のパイプサイズ) が設定可能であるが、本仕様では各パラメータを明示的に設定する必要があります。
- (4) 使用するマシンのカーネルパラメータを変更して、IPC 資源 (メッセージ、共有メモリ) を増やす必要があります。
- (5) UNIQUE\_SESSION\_NAME ファンクションは名前の制限 (18 バイト以下) により UNIQUE\_SESSION となっています。

#### 4.2.4 外部仕様

「4.1.4 外部仕様 (Windows 版 C 言語)」と同様です。

#### 4.2.5 IPC 資源及びワークファイルについて

##### (1) IPC 資源について

UNIX 版の DBMS\_PIPE 以下の IPC 資源を割り付けます。

##### ・共有メモリ (最大 4 個)

ipcs -ma で表示した場合、割り付けキー番号の先頭が **0x52**, **0x53**, **0x72**, **0x73** のものが DBMS\_PIPE で割り付けた資源です。

```
m 2417 0x52003007 --rw-r--r--
```

```
m 2418 0x53003008 --rw-r--r--
```

```
m 2419 0x72003009 --rw-r--r--
```

```
m 2420 0x73003010 --rw-r--r--
```

↑  
ここに着目

\* IPC の ID やキー番号の残りの部分は必ずしもこの番号にはなりません。

\* 毎回、4 個全てが作成される訳ではありません。

##### ・メッセージ ID (最大 1 個)

ipcs -qa で表示した場合、割り付けキー番号の先頭が **0x51** のもの DBMS\_PIPE で割り付けた資源です。

```
m 2417 0x51003007 --rw-r--r--
```

↑  
ここに着目

\* IPC の ID やキー番号の残りの部分は必ずしもこの番号にはなりません。

##### (2) ワークファイルについて

以下のワークファイルを作成します。

- /tmp/DBMS\_PIPE.LOG (ログファイル)
- /tmp/PIPE (パイプ用ファイル: 'PIPE' の部分はパイプ名)
- /tmp/UNPACK\_MESSAGE\_xxxxxxx (受信用共有メモリファイル: xxxxx はユーザー名)
- /tmp/PACK\_MESSAGE\_xxxxxxx (送信用共有メモリファイル: xxxxx はユーザー名)

#### 4.2.6 補足説明

##### (1) SEND\_MESSAGE 時のオーバーフロー・チェック処理

メッセージの送信時に行うオーバーフロー・チェックを実装されています。

これは、CREATE\_PIPE または、SEND\_MESSAGE 時にパラメータで指定された（指定が無い場合はデフォルトで 8192）パイプ最大サイズを超えてデータを送信した場合、指定の時間分 Wait します。Wait 時間は SEND\_MESSAGE のパラメータで指定され、この間にパイプに空きが発生しなければ「タイムアウト」が戻ります。

処理：「SEND\_MESSAGE」内で、メッセージ・キューに現在積まれているデータ量を求め（msgctl 使用）「パイプ最大サイズ」よりこの値を引いたものを送信可能な容量（capacity）として送信データ量と比較し、送信データ量の方が大きい場合は Wait します。

##### (2) パイプ（メッセージ・キューの）最大サイズ情報

上記の処理に対応するため、パイプの属性情報に「パイプ最大サイズ」があります。保存する場所は他の属性情報と同じです。

〔パイプの属性管理〕

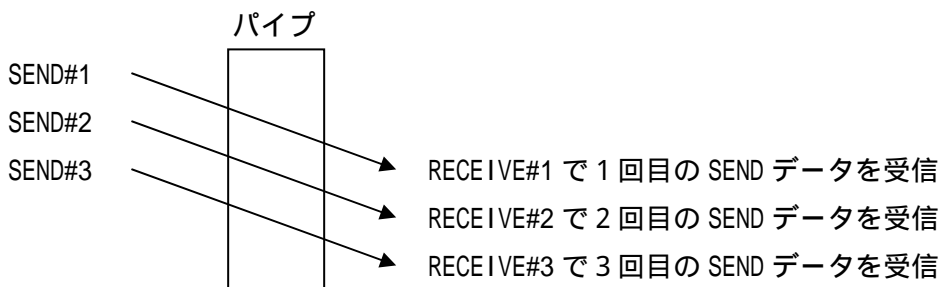
・パイプ用のメッセージ・キューを作成する為のファイル（概略図では"/tmp/session1"）に CSV 形式のアスキー・データとして書き込みます。

<例：パイプ属性が「プライベート」、「明示モード」、ユーザーが「dbinst1」、サイズが 8192 の場合>

E, V, db2inst1, 008192, 000001, 000001 <LF> ← この内容がファイルに書かれる

パイプ最大サイズを7桁ゼロサブレス  
で

##### (3) SEND\_MESSAGE と RECEIVE\_MESSAGE の同期処理



\* この動きは、PL/SQL パッケージの PIPE と同じ結果になります。

( 4 ) SEND 回数、RECEIVE 回数情報

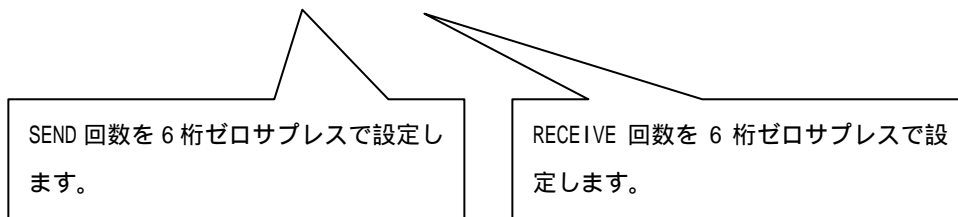
上記の処理に対応するため、パイプの属性情報に「SEND 回数」、「RECEIVE 回数」を追加しました。存在する場所は他の属性情報と同じです。

〔パイプの属性管理〕

- ・パイプ用のメッセージ・キューを作成するためのパイプ属性管理ファイル( 4 . 1 . 5 ( 4 ) 参照 ) に CSV 形式のアスキー・データとして書き込みます。

< 例 : SEND を 3 回、RECEIVE を 1 回行った場合 >

E, V, db2inst1, 008192, 000004, 000002 <LF> ← この内容がファイルに書かれる



- \* SEND 側はメッセージを送信する前に「SEND 回数」を取り出し、これをメッセージタイプとしてデータをキューに積む。SEND 関数から抜ける前にこの値を +1 して保存します。
- \* RECEIVE 側はメッセージを受信する前に「RECEIVE 回数」を取り出し、これをメッセージタイプとしてデータをキューから読み込みます。RECEIVE 関数から抜ける前にこの値を+1 して保存します。

( 5 ) 権限管理は、パラメータで渡された「ユーザー名」を使いアプリケーションが管理します。

( 6 ) 複数のパイプが作成されても名前が違えば平行に運用できます。( ftok 関数はファイル毎に、ユニークなキーを戻します。)

( 7 ) 暗黙モードの場合は PACK 時に資源を作成し、全データを読み込んだタイミングで削除します。

( 8 ) 同一のファイルが存在した場合は ORA-23322 ( 重複 ) エラー相当とします。

#### 4.3 DBMS\_PIPE Windows 版 Java 言語

ここでは PL/SQL で提供されているパッケージで、DBMS\_PIPE を DB2 UDB、Windows 環境で稼動する Java 言語アプリケーションで実現するサンプルについて紹介します。

##### 4.3.1 処理概要

- (1) PL/SQL で提供されているパッケージで DBMS\_PIPE の各機能を Java 言語アプリケーションで作成し、ユーザー定義関数として登録します。
- (2) サンプル・プログラムでは、パイプ機能を実現するためにテンポラリーファイルを作成し、メッセージは別途メッセージファイルを作成して、メッセージの送受信を行います。

##### 4.3.2 サンプル・プログラムを用いた DBMS\_PIPE 機能の使用法の概要

###### (1) 送り手

###### CREATE\_PIPE

- ・パラメータで渡された名前の先頭に "PIPE\_OTHER" 付与しファイルを作成します。

/tmp/PIPE\_OTHERsession1

###### PACK\_MESSAGE

- ・メッセージオブジェクトにメッセージをセットします。

###### SEND\_MESSAGE

- ・メッセージオブジェクトをパイプファイルに書き出し、メッセージファイル PIPE\_OTHERsession1\_RECV に書き換えます。

###### REMOVE\_PIPE

- ・メッセージオブジェクトをクリアします。また、メッセージファイルを削除します。

###### PURGE

- ・内部で REMOVE\_PIPE コールします。

###### RESET\_BUFFER

- ・テンポラリー内の全てのパイプファイル及び、メッセージオブジェクトファイルを削除します。

###### UNIQUE\_SESSION\_NAME

- ・クラス作成時にランダムな値をとり変数にセットします。

###### (2) 受け手

###### RECEIVE\_MESSAGE

- ・メッセージファイルの存在をチェックします。
- ・無い場合 1 秒おきに存在をチェックします。TIMEOUT 秒後、メッセージファイルが無い場合 1 を戻します。

###### LOAD\_PIPE

- ・オブジェクトファイルを読み込みメッセージオブジェクトにセットします。

###### NEXT\_ITEM\_TYPE

- ・メッセージオブジェクトより次のメッセージの型を取り出す。メッセージが無い場合は 0 を

返します。

UNPACK\_\_MESSAGE

- ・メッセージオブジェクトよりメッセージを取り出します。

#### 4.3.3 制限事項

サンプル・プログラムの制限事項ですが、同一パイプ名の READ&WRITE はクラス内で同期を取っていないため、上位のクラスで同期を取る必要があります。これはクラス間の同期を上位でとる方がソースの可視性及び汎用性が高まるためです。

#### 4.3.4 外部仕様

##### (1) すべてのパッケージの階層

クラス階層

```
class java.lang.Object
 class DbmsPipe
 class PipeObject (implements java.io.Serializable)
 class java.lang.Throwable (implements java.io.Serializable)
 class java.lang.Exception
 class java.io.IOException
 class RaiseApplicationErrorException
```

説明

DbmsPipe クラス：オラクル DBMS\_PIPE パッケージの実装。

PipeObject クラス：Pipe メッセージ保存用直列化オブジェクト。オブジェクトデータをリストとして保持します。

RaiseApplicationErrorException：オラクル RAISE\_APPLICATION\_ERROR プロシージャの代用として使用する例外クラス。

## ( 2 ) DbmsPipe コンストラクタの概要

---

DbmsPipe()

テンポラリーファイル抽象パス名、識別セッションにデフォルト値を使用して、DbmsPipe オブジェクトを構築します。

DbmsPipe(java.lang.String temporary)

識別セッションにデフォルト値を使用して、DbmsPipe オブジェクトを構築します。

DbmsPipe(java.lang.String temporary, java.lang.String session)

DbmsPipe オブジェクトを構築します。

## ( 3 ) DbmsPipe メソッドの概要

---

int create\_pipe(java.lang.String pipename)

int create\_pipe(java.lang.String pipename, int maxpipesize)

int create\_pipe(java.lang.String pipename, int maxpipesize, boolean privateflg)

指定メッセージファイルの作成 pipename が null の場合は例外を発生、命名競合の場合はエラーを返却します。

---

private java.lang.String getBufferName()

テンポラリーファイル相対パス名を返却します。

private java.lang.String getBufferName(java.lang.String pipename)

pipename が指定されている場合は完全な SerializableObject 名を、指定されていない場合は SessionObject 名を返却します。

---

java.lang.String getPipename()

パイプ名を返却します。

---

java.lang.String getSession()

セッション名を返却します。

---

```
java.lang.String getTemporary()
```

テンポラリー抽象パス名を返却します。

---

```
void load_pipe(java.lang.String pipename)
```

指定メッセージファイルを直列化復元します。別インスタンスで receive\_message 後に呼び出します。

---

```
int next_item_type()
```

メッセージバッファの型を返却します。

---

```
void pack_message(java.util.Date item)
```

テンポラリーバッファに Date クラス型で追加します。

```
void pack_message(java.lang.Integer item)
```

テンポラリーバッファに Integer クラス型で追加します。

```
void pack_message(java.lang.String item)
```

テンポラリーバッファに String クラス型で追加します。

---

```
void purge(java.lang.String pipename)
```

指定メッセージファイルを消去します。

---

```
private java.lang.Object readObject(java.lang.String pathname)
```

オブジェクトを直列化復元します。

---

```
int receive_message(java.lang.String pipename, int timeout)
```

指定メッセージファイルを受信 ( 検査 ) します。

---

```
int remove_pipe(java.lang.String pipename)
```

指定メッセージファイルを消去します。

---

```
private void renameCompletionFile(java.lang.String pathname)
```

指定メッセージファイル名を送信確定に変更します。

---

```
void reset_buffer()
```

テンポラリー内を全ファイル消去します。

---

```
int send_message(java.lang.String pipename, int timeout, int maxpipesize)
```

指定メッセージファイルの送信し（ファイル書出し）、メソッドのクリティカルセクションを保つため、終了後に送信名でリネームします。

---

```
java.lang.String unique_session_name()
```

ユニーク(固定)値を返却します。

---

```
java.util.Date unpack_message_date()
```

メッセージバッファから Date クラス型で返却します。

---

```
java.lang.Integer unpack_message_integer()
```

メッセージバッファから Integer クラス型で返却します。

---

```
java.lang.String unpack_message_string()
```

メッセージバッファから String クラス型で返却します。

---

```
java.lang.Object unpack_message()
```

メッセージバッファから Object クラス型で返却します。

---

```
private int validatePipe(java.lang.String pipename)
```

pipename が指定されている場合は明示的なパイプを、指定されていない場合は暗黙のパイプ作成を保証します。但し、既に同じ pipename で作成済みの場合は Exception を発行します。

---

```
private void writeObject(java.lang.String pathname, java.lang.Object obj)
```

オブジェクトを直列化します。

---

#### ( 4 ) PipeObject コンストラクタの概要

---

PipeObject()

PipeObject クラスを生成します。

#### ( 5 ) PipeObject メソッドの概要

---

void add(java.util.Date date)

Date クラス型をリストに挿入します。

void add(int index, java.util.Date date)

Date クラス型をリストに挿入します。

void add(java.lang.Integer num)

Integer クラス型をリストに挿入します。

void add(int index, java.lang.Integer num)

Integer クラス型をリストに挿入します。

void add(int index, java.lang.String str)

String クラス型をリストに挿入します。

void add(java.lang.String str)

String クラス型をリストに挿入します。

---

private void addObject(int index, java.lang.Object object)

private void addObject(java.lang.Object object)

オブジェクトリストにオブジェクトを追加します。

---

private void addType(java.lang.Integer typ)

型名リストにオブジェクト型名を追加します。

private void addType(int index, java.lang.Integer typ)

---

`void clear()`

オブジェクトリストの内容をリセットします。

---

`java.lang.Object getCurrentObject()`

カレントのオブジェクトを返却します。

---

`int getCurrentType()`

カレントのオブジェクトタイプを返却します。

---

`java.util.Date getDate()`

オブジェクトリスト要素を Date クラス型で返却します。

---

`int getInt()`

オブジェクトリスト要素を int 型で返却します。

---

`java.lang.Integer getInteger()`

オブジェクトリスト要素を Integer クラス型で返却します。

---

`java.lang.Object getObject()`

オブジェクトリスト要素を Object クラス型で返却します。

---

`java.lang.String getString()`

オブジェクトリスト要素を String 型で返却します。

---

`int next()`

オブジェクトリストの参照を次に設定します。

---

`private void readObject(java.io.ObjectInputStream in)`

オブジェクト直列化復元メソッド `Serializable` インターフェースのメソッドをオーバーライド処理します。

---

```
int size()
```

オブジェクトリストのサイズを返却します。

---

```
private boolean validateCount()
```

カレント・インデックスの正当性をチェックします。

---

```
private void writeObject(java.io.ObjectOutputStream out)
```

オブジェクト直列化メソッド `Serializable` インターフェースのメソッドのオーバーライドを処理します。

#### ( 6 ) `RaiseApplicationErrorException` コンストラクタの概要

`java.io.IOException` を継承し、例外機能により処理を代替しています。

---

```
RaiseApplicationErrorException()
```

詳細メッセージを持たない、`RaiseApplicationErrorException` を構築します。

```
RaiseApplicationErrorException(int n, java.lang.String s)
```

指定したエラー番号、詳細メッセージを持つ、`RaiseApplicationErrorException` を構築します。

```
RaiseApplicationErrorException(java.lang.String s)
```

指定した詳細メッセージを持つ、`RaiseApplicationErrorException` を構築します。

#### 4.4 DBMS\_PIPE UNIX 版 Java 言語

「4.3 DBMS\_PIPE Windows 版 Java 言語」と同様です。

#### 4.5 DBMS\_ALERT Windows 版 C 言語

ここでは PL/SQL で提供されているパッケージで、DBMS\_ALERT を DB2 UDB、Windows 環境で稼動する C 言語アプリケーションで実現するサンプルについて紹介します。

##### 4.5.1 処理概要

サンプル・プログラムでの内部処理について述べます。

(1) PL/SQL で提供されているパッケージで DBMS\_ALERT を実現するための主だった処理は、DB のテーブルとプロシージャを使用した処理として実装してあります。

(2) DB2 UDB に以下のアラート管理テーブルを作成 (Create table) し、テーブルのアクセスによってアラート機能を実現します。

アラート管理テーブル仕様

```
CREATE TABLE DBMS_ALERT.DBMS_ALERT_INFO
```

```
(NAME VARCHAR(30) NOT NULL, ← アラート名
```

```
SID VARCHAR(33) NOT NULL, ← アプリケーション ID
```

```
CHANGED VARCHAR(1), ← メッセージフラグ
```

(N:メッセージなし。Y:メッセージあり。)

```
MESSAGE VARCHAR(1800), ← メッセージ
```

```
PRIMARY KEY (NAME, SID)
```

```
)
```

(3) 受け手はアラート名とアプリケーション ID をアラート管理テーブルに書き込み、メッセージフラグが 'Y' になるのを待ちます。送り手はアラート名をキーにして、メッセージを書き込み、メッセージフラグを 'Y' にします。受け手はアラート (メッセージ) を受け取るとメッセージフラグを 'N' にします。

##### 4.5.2 サンプル・プログラムを用いた DBMS\_ALERT 機能の使用法の概要

(1) 受け手

```
REGISTER
```

- アプリケーション ID を固有の文字列 ('xxxx') とし、name に ALT1,ALT2,ALT3 を登録します。

```
WAITONE('ALT1',?,?,10)
```

- NAME, アプリケーション ID をキーにし検索します。

- CHANGED 'Y' になるまで LOOP します。

- 10 秒間 LOOP 後 'Y' にならない場合 timeout します。

- メッセージを受け取ることができたならば、CHANGED を 'N' に UPDATE し、正常終了の場合

合内部で COMMIT します。

WAITANY(?,?,?,10)

- ・アプリケーション ID をキーに検索します。
- ・検索結果が 0 件の場合エラーを返す。CHANGED が 'N' ならばポーリング秒待機し LOOP します。
- ・10 秒間 LOOP 後 'Y' にならない場合 timeout します。
- ・この時、CHANGE が 'Y' ならばメッセージを取得し、CHANGED を 'N' に UPDATE します。正常終了の場合内部で COMMIT します。

SET\_DEFAULTS

- ・ポーリング時間を設定。

## ( 2 ) 送り手

REGISTER( ' ALT1 ' )

- ・テーブルにアラートを登録します。
  - ・正常終了の場合内部で COMMIT します。

SIGNAL( ' ALT2 ' , ' メッセージ ' )

- ・name をキーに CHNGED を ' Y ' 、MESSAGE に ' メッセージ ' を UPDATE。内部では COMMIT はしません。

COMMIT

- ・COMMIT をすることによって確定させます。

ROLLBACK

- ・SIGNAL(UPDATE)を取りやめたいときは、ROLLBACK によってテーブルを戻します。

REMOVE

指定したアラート、自分のアプリケーション ID をキーにテーブルから削除します。正常終了の場合内部で COMMIT します。

REMOVEALL

- ・自分のアプリケーション ID をキーに削除します。正常終了の場合内部で COMMIT します。

ALLDELETE

- ・全件を削除します。

#### 4.5.3 制限事項

以下に、サンプル・プログラムの制限事項について述べます。

##### (1) C言語により実現した処理

以下の3つの処理はコーディングの容易さ等を考慮してC言語により実装しました。

set\_defaults (公開関数)

- ・ポーリング間隔秒を設定するこの関数は、環境変数 (UNIX版は /tmp 固定) で指定されたディレクトリに、パラメータで与えられたポーリング間隔秒をファイルとして保存します。ファイル名の一部にアプリケーション ID を使用するので、DB のアプリケーション毎にユニークなファイルとなります。(下記参照)

"C:\tmp\pol longsec\_アプリケーション ID"

\*パスは参考です。実際は環境変数 "TMP" により指定されます。

get\_default (内部関数)

- ・ で保存したポーリング間隔秒を取り出す関数として用意しました。アラート機能の Waitany はこの関数を使用してポーリング間隔を取得します。ファイルが存在しない場合 (set\_defaults が呼ばれていない場合) はデフォルト値を返します。

sleepSec (内部関数)

- ・ 指定時間 (秒単位) スリープする関数で、windows 版は 『Sleep』 を、UNIX 版は 『sleep』 を使用しています。(共にシステムコール) Windows 版の 『Sleep』 はパラメータが "ミリ秒" 指定なので、入力値を 1000 倍して与えています。

##### (2) DB の LOCKTIMEOUT パラメータを 0 にします。

##### (3) コネクションした直後に REMOVEALL を実行します。

(同じアプリケーション ID でのアラートが存在している可能性があるため)

##### (4) SET\_DEFAULTS でポーリング間隔の管理

ポーリング間隔の管理はアプリケーション ID 毎にファイルで行っています。作成されるファイルは POLILINGSEC\_アプリケーション ID となっています。また、作成されるディレクトリは UNIX 版では '/tmp', WINDOWS 版は環境変数 'TMP' となっています。ファイルが無い場合ポーリング間隔は 5 秒となります。

#### 4 . 5 . 4 外部仕様

##### ( 1 ) REGISTER

```
>>---REGISTER---(--- in_name----- in-----varchar-----)-----><
```

スキーマは DBMS\_ALERT です。

REGISTER プロシージャはアラートを登録します。

戻り値

なし。

in\_name

アラート名。30 バイト以内。

##### ( 2 ) REMOVE

```
>>---REMOVE---(----- in_name----- in-----varchar-----)-----><
```

スキーマは DBMS\_ALERT です。

REMOVE プロシージャは登録したアラート削除します。

戻り値

なし。

in\_name

削除するアラート名

##### ( 3 ) REMOVEALL

```
>>---REMOVEALL---(---)-----><
```

スキーマは DBMS\_ALERT です。

REMOVEALL プロシージャはセッションに関連するアラートを登録リストから全て削除します。

戻り値

なし。

( 4 ) SET\_DEFAULTS

```
>>---SET_DEFAULTS---(-----pollingsec-----IN-----integer-----)-----><
```

スキーマは DBMS\_ALERT です。

SET\_DEFAULTS プロシージャは WAITANY で使用するポーリング間隔を設定します。

戻り値

なし。

pollingsec

ポーリング間のスリープ時間 ( 秒 )

( 5 ) SIGNAL

```
>>---SIGNAL---(---in_name-----in-----varchar----->
>-----,-----in_message-----in-----varchar-----)-----><
```

スキーマは DBMS\_ALERT です。

DBMS\_ALERT プロシージャはアラートを通知します。SIGNAL コールはコールしたトランザクションがコミットされたときのみ有効になります。

戻り値

なし。

in\_name

通知するアラートの名前。

in\_timeout

このアラートに関連付けるメッセージ ( 1800 バイト以下 )。

## ( 6 ) WAITANY

```
>>---WAITANY-----(-out_name-----out-----varchar----->
>-----,-----out_message-----out-----varchar----->
>-----,-----out_status-----out-----integer----->
>-----,-----in_timeout-----in-----integer-----)>
```

スキーマは DBMS\_ALERT です。

WAITANY プロシージャは現行セッションが登録されている全てのアラートを対象として、そのいずれかの発生を待機する場合に WAITANY をコールします。

アラートが登録していない場合エラーを返します。

例として、SQL0438N アプリケーションで、診断テキスト "there are no alert registered" のエラーが 発生しました。 SQLSTATE=75000

### 戻り値

なし。

out\_name

発生したアラート名を戻します。

out\_message

アラートに関連したメッセージを返します。

out\_status

戻されるステータスの値。

0 : アラート発生。

1 : タイムアウト。

in\_timeout

アラートの最大待機時間 ( 秒 )。

( 7 ) WAITONE

```
>>---WAITONE-----(-----in_name-----in-----varchar----->
>-----,-----out_message---out-----varchar----->
>-----,-----out_status----out-----integer----->
>-----,-----in_timeout----in-----integer-----)>
```

スキーマは DBMS\_ALERT です。

WAITONE プロシージャは特定のアラートの発生を待機します。

戻り値

なし。

in\_name

待機するアラート名。

out\_message

アラートに関連したメッセージを返します。

out\_status

戻されるステータスの値。

0 : アラート発生。

1 : タイムアウト。

in\_timeout

アラートの最大待機時間 ( 秒 )

( 8 ) GET\_DEFAULT

```
>>---GET_DEFAULTS---(------pollingsec-----out-----integer-----)><
```

スキーマは DBMS\_ALERT です。

GET\_DEFAULT プロシージャは WAITANY で使用するポーリング間隔を取得します。

このプロシージャは WAITANY プロシージャに呼ばれます。

戻り値

なし。

polling\_interval

ポール間のスリープ時間 ( 秒 )

( 9 ) SLEEP

```
>>---SLEEP---(-----sec-----in-----integer-----)<<-----><
```

スキーマは DBMS\_ALERT です。

SLEEP プロシージャは sec 秒待機します。

戻り値

なし。

sec

待機時間 ( 秒 )。

( 1 0 ) ALLDELETE

```
>>---ALLDELETE---(--)<<-----><
```

スキーマは DBMS\_ALERT です。

ALLDELETE プロシージャはアラート管理テーブルを初期化します。

戻り値

なし。

( 1 1 ) UNIQUE\_SESSION

```
>>---UNIQUE_SESSION---(----sessionmae-----out-----varchar-----)-----<
```

スキーマは DBMS\_ALERT です。

UNIQUE\_SESSION プロシージャは現在接続している全てのセッション間での一意の名前を受け取ります。このプロシージャは DBMS\_ALERT で内部的に呼ばれます。

戻り値

なし。

sessionmae

セッション間での一意の名前 (アプリケーション ID)。

( 1 2 ) RM\_TMPFILE

```
>>---RM_TMPFILE---(----)-----<
```

スキーマは DBMS\_ALERT です。

RM\_TMPFILE プロシージャは SET\_DEFAULTS で作成したファイルを削除します。

このプロシージャは REMOVEALL プロシージャ内部でコールされます。

戻り値

なし。

#### 4.6 DBMS\_ALERT UNIX 版 C 言語

「4.5 DBMS\_ALERT Windows 版 C 言語」と同様です。

#### 4.7 DBMS\_ALERT Windows 版 Java 言語

ここでは PL/SQL で提供されているパッケージで、DBMS\_ALERT を DB2 UDB、Windows 環境で稼動する Java 言語アプリケーションで実現するサンプルについて紹介します。

##### 4.7.1 処理概要

「4.5.1 処理概要 (DBMS\_ALERT Windows 版 C 言語)」と同様です。

##### 4.7.2 サンプル・プログラムを用いた DBMS\_ALERT 機能の使用法の概要

「4.5.2 サンプル・プログラムを用いた DBMS\_ALERT 機能の使用法の概要 (DBMS\_ALERT Windows 版 C 言語)」と同様です。

\* アプリケーションは Java の UID を使用します。UID はクラスをインスタンス化した時点で決定します。

##### 4.7.3 制限事項

「4.5.3 制限事項 (DBMS\_ALERT Windows 版 C 言語)」と同様です。

##### 4.7.4 外部仕様

(1) すべてのパッケージの階層

```
class java.lang.Object
 class DbmsAlert
 class DbmsAlertM
 class DbmsAlertSql
 class java.lang.Throwable (implements java.io.Serializable)
 class java.lang.Exception
 class java.io.IOException
 class RaiseApplicationErrorException
```

##### 説明

DbmsAlert クラス : DBMS\_ALERT パッケージ機能の実装。

DbmsAlertM クラス : アラートメッセージ保存用直列化オブジェクト。

DbmsAlertSql クラス : テーブルにアクセスします。

RaiseApplicationErrorException クラス : オラクル RAISE\_APPLICATION\_ERROR プロシージャの代用として使用する例外クラス。

## ( 2 ) DbmsAlert コンストラクタの概要

---

DbmsAlert()

セッションの所有者となるユーザーにデフォルト値 ( サンプル ) を使用して、 DbmsAlert オブジェクトを構築します。

DbmsAlert(java.sql.Connection con)

DbmsAlert オブジェクトを構築します。

DbmsAlert(java.lang.String instance, java.lang.String userid, java.lang.String passwd)

DbmsAlert オブジェクトを構築します。

## ( 3 ) DbmsAlert メソッドの概要

---

void allDelete()

アラートを全て削除します。

void closeConnection()

コネクションを閉じます。

void commit()

コミットをします。

void openConnection()

コネクションを作成します。

void register(java.lang.String name)

セッションにアラートを登録 name が null 及びサイズ 0 の場合は例外を発生させます。

void remove(java.lang.String name)

セッションの登録リストから指定アラートを削除 name が null 及びサイズ 0 の場合は例外を発生させます。

---

`void removeall()`

セッションの登録リストから全てのアラートを削除します。

---

`void rollback()`

ロールバックをします。

---

`void set_defaults(int interval)`

WAITANY でポーリング・ループする待機時間を指定します。

---

`void signal(java.lang.String name, java.lang.String message)`

アラートの通知 `name` が `null` 及びサイズ 0 の場合は例外を発生させます。

---

`DbmsAlertM waitany(int timeout)`

セッションの登録リストに全てのアラートを待機 `timeout` がサイズ 0 以下の場合は例外を発生させます。

---

`DbmsAlertM waitone(java.lang.String name, int timeout)`

指定アラートを待機 `timeout` がサイズ 0 以下の場合は例外を発生させます。

---

#### ( 4 ) DbmsAlertM コンストラクタの概要

---

`DbmsAlertM()`

#### ( 5 ) DbmsAlertM メソッドの概要

---

`java.lang.String getMessage()`

メッセージの取得。

---

`java.lang.String getName()`

アラート名の取得。

---

`int getStatus()`

ステータスの取得。

---

---

`void setMessage(java.lang.String message)`

メッセージの登録。

---

`void setName(java.lang.String name)`

アラート名の登録。

---

`void setStatus(int status)`

ステータスの登録。

#### ( 6 ) DbmsAlertSql コンストラクタの概要

---

`DbmsAlertSql(java.sql.Connection con, java.lang.String unique_name)`

DbmsAlertSql オブジェクトを構築します。

#### ( 7 ) DbmsAlertSql メソッドの概要

---

`void allDelete()`

DBMS\_ALERT.DBMS\_PIPE\_INFO テーブルを全件削除します。

---

`void commit()`

コミットを発行します。

---

`void register(java.lang.String name)`

テーブルにアラートを登録します。

---

`void remove(java.lang.String name)`

セッションに関連したアラートをテーブルから削除します。

---

`void removeAll()`

セッションに関連したアラートを全てテーブルから削除します。

---

`void rollback()`

ロールバックを発行します。

---

`void signal(java.lang.String name, java.lang.String message)`

指定したアラートを全て更新します。

---

`DbmsAlertM waitany(int timeout, int interval)`

アラートのメッセージを受け取ります。

---

`DbmsAlertM waitone(java.lang.String name, int timeout)`

指定したアラートのメッセージを受け取ります。

---

#### ( 6 ) RaiseApplicationErrorException コンストラクタの概要

`java.io.IOException` を継承し、例外機能により処理を代替しています。

---

`RaiseApplicationErrorException()`

詳細メッセージを持たない、`RaiseApplicationErrorException` を構築します。

`RaiseApplicationErrorException(int n, java.lang.String s)`

指定したエラー番号、詳細メッセージを持つ、`RaiseApplicationErrorException` を構築します。

`RaiseApplicationErrorException(java.lang.String s)`

指定した詳細メッセージを持つ、`RaiseApplicationErrorException` を構築します。

#### 4 . 8 DBMS\_ALERT UNIX 版 Java 言語

「4 . 7 DBMS\_ALERT Windows 版 Java 言語」と同様です。