

Tmax HMSユーザガイド

Tmax v6.0



Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

Copyright Notice

Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, 13613, South Korea

Restricted Rights Legend

All TmaxSoft Software (Tmax®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd.

Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features. This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

このソフトウェア(Tmax®)マニュアルの内容とプログラムは、日本国の著作権法および国際条約によって保護されています。マニュアルの内容とプログラムは、TmaxSoft Co., Ltd.との使用許諾契約書の下でのみ使用することができ、マニュアルは使用許諾契約で許可されている範囲を除いては、配布または複製することができません。TmaxSoftの書面による事前の承諾を得ることなく、このマニュアルの全部または一部を電子的または機械的な方法を問わず、転送、複製、配布したり、または二次的著作物を作成する等の行為を一切禁じます。

このソフトウェアのマニュアルとプログラムの使用許諾契約は、いかなる場合においても、マニュアル及びプログラムと関連する知的財産権(登録の有無を問わず)を譲渡するものと解釈されず、TmaxSoftのブランド、ロゴ、商標等の使用権限を与えるものではありません。マニュアルは、情報を提供する目的でのみ提供しており、これに伴う契約上の直接的ないしは間接的な責任を負わず、マニュアルの内容は法律上もしくは商業的な特定の条件が満たされることを保証しません。マニュアルの内容は、製品のアップグレード及び修正により、その内容が予告なく変更されることがあり、内容上の誤りがないことを保証しません。

Trademarks

Tmax®, Tmax WebtoB® and JEUS® are registered trademark of TmaxSoft Co., Ltd. Other products, titles or services may be registered trademarks of their respective companies.

Tmax®, Tmax WebtoB®, JEUS® は、TmaxSoft Co., Ltd.の登録商標です。その他、記載されている会社名、製品名などは、各社の商標または登録商標です。

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : openssl-0.9.7.m, zlib-1.1.4, expat-2.0.0, net-snmp, DCE1.0, pthread, google-diff-match-patch, libevent, getopt

Detailed Information related to the license can be found in the following directory :
\${INSTALL_PATH}/license/oss_licenses

この製品の一部ファイルまたはモジュールは、openssl-0.9.7.m、zlib-1.1.4、expat-2.0.0、net-snmp、DCE1.0、pthread、google-diff-match-patch、libevent、getoptライセンスを遵守します。

詳細情報については、製品ディレクトリーの\${INSTALL_PATH}/license/oss_licensesに記載されている事項を参照してください。

文書情報

文書名: Tmax HMSユーザガイド

発行日: 2016年8月5日

ソフトウェアバージョン: Tmax v6.0

ガイドバージョン: v2.1.1

目次

このガイドについて	xi
第1章 紹介	1
1.1. 概要	1
1.2. HMSの構造	1
1.3. JMSとHMSプログラミング	2
1.3.1. コネクション	3
1.3.2. セッション	3
1.3.3. デスティネーション	4
1.3.4. メッセージ	6
1.3.5. キュー・ブラウザー	7
1.3.6. リカバリー	7
1.3.7. クラスタリング	7
第2章 HMSの環境設定	9
2.1. Tmaxの環境設定	9
2.1.1. DOMAINセクション	9
2.1.2. NODEセクション	9
2.1.3. SVRGROUPセッション	10
2.1.4. HMSセクション	13
2.2. 例	16
第3章 API関数	17
3.1. 概要	17
3.2. セッションAPI	19
3.2.1. hms_create_session	20
3.2.2. hms_create_xa_session	22
3.2.3. hms_create_async_session	24
3.2.4. hms_close_session	27
3.2.5. hms_unsubscribe_durable_subscriber	28
3.2.6. hms_commit	31
3.2.7. hms_rollback	32
3.2.8. hms_recover	33
3.3. メッセージAPI	35
3.3.1. hms_alloc	35
3.3.2. hms_free	37
3.3.3. hms_get_property	39
3.3.4. hms_set_property	41
3.3.5. hms_get_body	44
3.3.6. hms_set_body	46
3.3.7. hms_ack	47
3.4. プロデューサーAPI	50

3.4.1.	hms_create_producer	50
3.4.2.	hms_create_sender	52
3.4.3.	hms_create_publisher	54
3.4.4.	hms_close_producer	56
3.4.5.	hms_close_sender	57
3.4.6.	hms_close_publisher	58
3.4.7.	hms_sendex	59
3.4.8.	hms_send	61
3.5.	コンシューマAPI	63
3.5.1.	hms_create_consumer	63
3.5.2.	hms_create_receiver	66
3.5.3.	hms_create_subscriber	68
3.5.4.	hms_create_durable_subscriber	70
3.5.5.	hms_close_consumer	72
3.5.6.	hms_close_receiver	73
3.5.7.	hms_close_subscriber	74
3.5.8.	hms_close_durable_subscriber	75
3.5.9.	hms_recvex	77
3.5.10.	hms_rcv	79
3.6.	キュー・ブラウザーAPI	81
3.6.1.	hms_create_browser	81
3.6.2.	hms_close_browser	83
3.6.3.	hms_browser_nextmsg	85
3.6.4.	hms_browser_get_msgselector	87
3.6.5.	hms_browser_get_queue	89
第4章	HMS管理	91
4.1.	HMSのビルド	91
4.1.1.	事前準備	91
4.1.2.	Makefile	92
4.1.3.	HMSビルドおよび確認	92
4.2.	HMSの起動および終了	93
4.3.	HMSの状態照会	93
4.3.1.	デスティネーション情報の照会	93
4.3.2.	クライアント情報の照会	94
第5章	例	97
5.1.	メッセージ送信プログラム	97
5.1.1.	HMSの環境設定	97
5.1.2.	クライアント・プログラム	98
5.1.3.	サーバー・プログラム	101
5.1.4.	プログラムのコンパイル	102
5.2.	メッセージ保存プログラム	103
5.2.1.	HMSの環境設定	103

5.2.2.	クライアント・プログラム	104
5.2.3.	サーバー・プログラム	106
5.2.4.	データベース・スクリプト	110
5.2.5.	プログラムのコンパイル	110
付録 A.	メッセージ・セレクタの使用法	111
索引	113

図目次

[図 1.1]	HMSの動作構造	1
[図 1.2]	プログラミング・モデル	2
[図 5.1]	メッセージ送信プログラムのフロー	97
[図 5.2]	メッセージ保存プログラムのフロー	103

このガイドについて

対象読者

本書は、Tmax[®](以下、Tmax)でHMSを使用する開発者や管理者を対象としています。HMSの基本概念と環境設定方法およびAPIの使用方法和例について説明します。

前提知識

本書は、Tmaxシステムの概要とTmaxシステムが提供する各種機能や特性などを習得するための手引書です。

本書を理解するためには、以下の事項についての知識が必要です。

- ミドルウェア(Middleware)およびUNIXシステムについて
- Tmaxの基本概念について
- Java、Cプログラミングについて
- JMS(Java Messaging Service)仕様について

制限事項

本書では、HMSに関連する内容についてのみ説明しています。実務上の具体的な使用方法や管理・運用についての内容は、各製品ガイドを参照してください。

参考

Tmaxシステムの開発についての基本的な内容は、『Tmax 運用ガイド』および『Tmax アプリケーション開発ガイド』を、Tmaxが提供するコマンドとC APIについては、『Tmax リファレンスガイド』を参照してください。

本書の構成

本書は、計5章と付録で構成されています。

各章の主な内容は以下のとおりです。

- 第1章: 紹介

HMSの紹介と構造について説明します。

- 第2章: HMSの環境設定

HMS関連の設定について説明します。

- 第3章: API関数

HMSで使用するAPIについて説明します。

- 第4章: HMS管理

HMSを使用するためにHMSをビルドする手順、HMSの起動と終了および状態照会の方法について説明します。

- 第5章: 例

アプリケーションの例を通じて基本的な使用方法について説明します。

- 付録A メッセージ・セレクタの使用方法

メッセージ・セレクタの構成と使用方法について説明します。

表記上の規則

表記	意味
<AaBbCc123>	プログラム・ソースコードのファイル名、ディレクトリー
<Ctrl> + C	CtrlキーとCキーを同時に押す
[Button]	GUIのボタン、メニュー名
太字	強調
「」、『』（鍵カッコ）	関連文書、あるいはガイド内の他の章および節の表示
「入力項目」	画面UI上の入力項目
ハイパーリンク	メール・アカウント、Webサイト
>	メニューの実行順
+ ----	下位ディレクトリー/ファイル有り
----	下位ディレクトリー/ファイル無し
<div>参考</div>	参照/注意事項
[図1.1]	図の名前
[表1.1]	表の名前
AaBbCc123	コマンド、コマンド実行結果の画面出力、サンプル・コード
[]	オプション・パラメータ値
	選択パラメータ値

システム要件

	要求事項
プラットフォーム	IBM AIX 5.x / 6.1 / 7.1
	HP-UX 11.xx
	SunOS 5.7~5.9 / SunOS 5.10 / SunOS 5.11
ハードウェア	1GB以上のハードディスク空き容量
	512MB以上のメモリー空き容量
データベース	Oracle 9~12
	Tibero 4~5
	DB2
	Informix

関連文書

ガイド	説明
Tmax メッセージリファレンスガイド	Tmax製品の使用時に発生する可能性のあるメッセージ(エラー・メッセージを含む)と、その対応方法について説明しています
Tmax FDLリファレンスガイド	Tmax FDL関数の定義とサンプル・プログラムを利用して、FDLが提供する機能を活用する方法について説明しています
Tmax 運用ガイド	Tmaxを利用するための環境設定ファイルとシステム運用方法について説明しています
Tmax プログラミングガイド(ダイナミックライブラリ)	TmaxのTDL(Tmax Dynamic Library)を使用してプログラムを開発するユーザー向けに、TDLを使用するための環境設定と提供されるAPIの使用方法について説明しています
Tmax アプリケーション開発ガイド	Tmaxアプリケーション・プログラムの開発で使用するAPIの概念と使用方法および例について説明しています
Tmax ゲートウェイガイド(Webサービス)	Tmaxサービスを変更せずにWebサービスで使用するために提供されるWebサービス・ゲートウェイについて説明しています
Tmax プログラミングガイド(SQ)	Tmax SQ(Session Queue)の概念と使用方法について説明しています

お問合せ先

Korea

TmaxSoft Co., Ltd.
45, Jeongjail-ro, Bundang-gu,
Seongnam-si, Gyeonggi-do, 13613
South Korea
Tel: +82-31-8018-1000
Fax: +82-31-8018-1115
Email: info@tmax.co.kr
Web (Korean): <http://www.tmaxsoft.com>
TechNet: <http://technet.tmaxsoft.com>

USA

TmaxSoft Inc.
101 North Wacker Drive, Suite 2014,
Chicago, IL 60606
U.S.A
Tel: +1-312-525-8330
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/us_en/home

Japan

TmaxSoft Japan Co., Ltd.
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073
Japan
Tel: +81-3-5765-2550
Fax: +81-3-5765-2567
Email: info@tmaxsoft.co.jp
Web (Japanese): <http://www.tmaxsoft.co.jp>

China

Beijing TmaxSoft System Software Co., Ltd.
Room103, No.2 Huizhong Building, Seven Street Shangdi,
Haidian District, Beijing, 100085
P.R.China
Tel: +86-10-6298-8827
Email: info@tmaxsoft.com.cn
Web (Chinese): http://www.tmaxsoft.com/cn_en/home_cn_en

Brazil

Tmax Brasil Sistemas e Serviços Ltda.
Av. Copacabana, 177, sala 32~35 Empresarial 18 do Fortel
Alphaville Barueri, Sao Paulo, 06472-001
Brazil
Tel: +55-11-4191-3100
Fax: +55(11) 4191-3705 (extension#112)
Email: info.bra@tmaxsoft.com
Web (Portuguese): http://www.tmaxsoft.com/br_en/home_br_en

Russia

Tmax Rus L.L.C.
Leninsky prospekt, 113/1 (Park Place Moscow),
Office 318e, Moscow, 117198
Russia
Tel: +7(495)970-01-35
Email: info.rus@tmaxsoft.com
Web (Russian): http://www.tmaxsoft.com/ru_ru/home_ru_ru

Singapore

Tmax Singapore Pte. Ltd.
430 Lorong 6, Toa Payoh #10-02,
OrangeTee Building, 319402
Singapore
Tel: +65-6259-7223
Fax: +65-6258-7112
Email: info.sg@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/sg_en/home_sg_en

United Kingdom

TmaxSoft UK Ltd.
215 Knyvett House, Watermans Business Park,
The Causeway, Staines TW18 3BAB
United Kingdom
Tel: +44-1784-895005
Email: info.uk@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/gb_en/home_gb_en

Canada

TmaxSoft Canada, Inc.
2425 Matheson Blvd East, 8th floor,
Unit 824 Mississauga, ON, L4W 5K4
Canada
Tel: +1-905-361-2888
Email: info.canada@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/ca_en/home_ca_en

Australia

TmaxSoft Proprietary Limited
L32, 101 Miller Street, North Sydney 2060
Australia
Tel: +91-9845-330-704
Email: info.aus@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/au_en/home_au_en

India

TmaxSoft Technologies Private Limited
Sobha Alexander Plaza, 3rd Floor,
16/2 Commissariat Road, Bangalore-560025
India
Tel: +91-9845-330-704
Email: info.india@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/in_en/home_in_en

Turkey

TmaxSoft Co., Ltd. Turkey Liaison Office
Windowist Tower. Eski Buyukdere Cad. No:26,
Maslak 34467 Istanbul
Turkey
Tel: +90-544-553-6045
Email: cslee@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/tr_en/home_tr_en

第1章 紹介

本章では、HMSについての紹介と構造について記述し、メッセージング・システムに関する基本的な概念と用語について説明します。

1.1. 概要

メッセージング・システム(Messaging System)という送信元(Sender)と受信元(Receiver)間の疎結合(loosely coupled)を可能にする通信媒体です。メッセージング・システムを使用することで、送信元と受信元は互いの情報を取得する必要なく、仮想的チャネル、つまり、デスティネーション(Destination)に関する情報のみで互いに通信することができます。また、送信する時点と受信する時点を分離できるためデータの遅延処理が可能で、このためにメッセージング・システムはメッセージの送受信の信頼性(Reliability)を保証します。

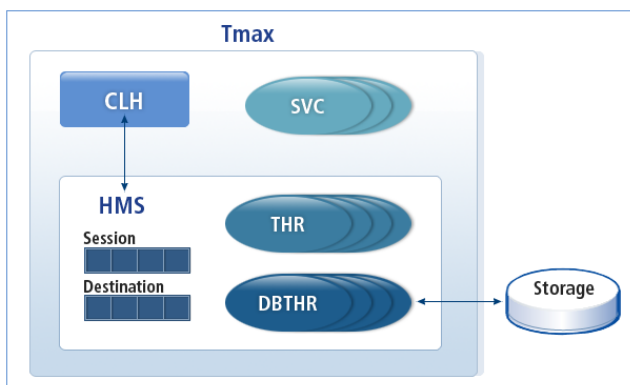
Hybrid Messaging System(以下、HMS)は、エンタープライズ・システム間で構成される情報交換の状態をメッセージという形式のデータとデスティネーションという仮想的なチャネルで抽象化します。ユーザーが作成したメッセージは、管理者によって作成された仮想チャネルであるデスティネーションに渡され、この情報が必要なアプリケーションはいつでもこのデスティネーションに自身を登録してメッセージを取得できます。HMSは、Sun Microsystemsで規定したJava基盤のメッセージング・システム標準であるJMS(Java Messaging Service)の概念と動作方式を反映しました。

HMSはTPモニターのTmaxシステムで動作するため、TPモニターの機能とメッセージング・システムの機能を柔軟に組み合わせて使用することができます。リソース・マネージャー(以下、RM)との2PC(2相コミット)機能やバックアップ・ノードを設定することにより、障害の発生時に柔軟な対処が可能です。

1.2. HMSの構造

HMSはTmaxシステム内でマルチスレッド・プロセスで動作します。

[図 1.1] HMSの動作構造



HMSのスレッドは、以下のように2種類に分けられます。

- メイン・スレッド(Main Thread)

メイン・スレッドは、クライアント、サービスなど、Tmaxとの通信部分を担当し、要求やメッセージをワーカー・スレッドに渡す役割をします。

- ワーカー・スレッド(Worker Thread)

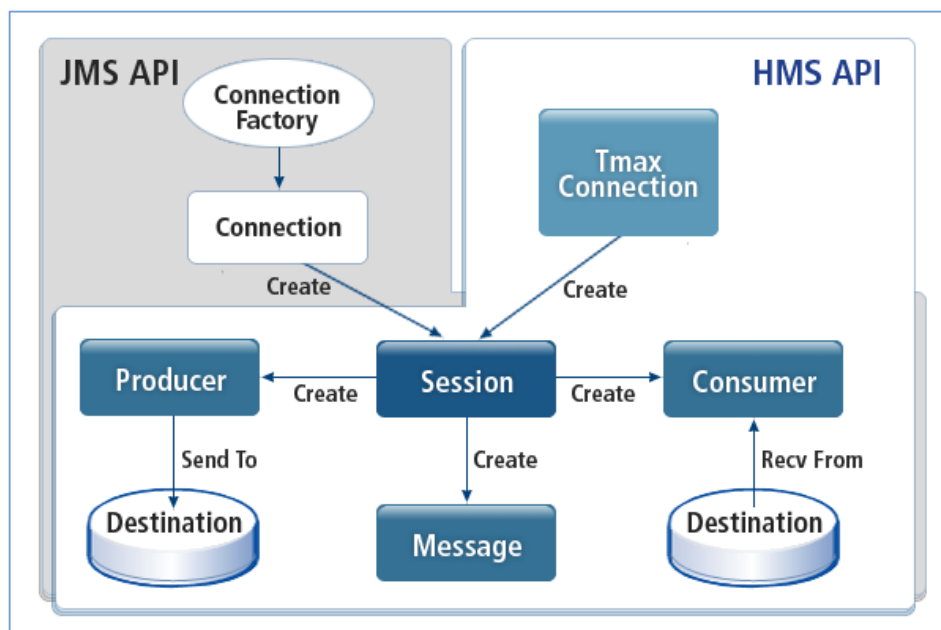
ワーカー・スレッドは一般スレッドとストレージ・スレッドに分けられます。メッセージや要求の種類によってストレージ関連作業が必要な場合はストレージ・スレッドが、それ以外の場合は一般スレッドが該当メッセージや要求を処理します。

各ワーカー・スレッドの数を環境ファイルにユーザーが直接設定できます。設定していない場合、基本的に一般スレッドは2つ、ストレージ・スレッドは4つで動作します。ストレージ・スレッドの数は多ければいいというわけではなく、運用システムおよびストレージの性能に従って適切な値を設定する必要があります。

1.3. JMSとHMSプログラミング

以下は、JMSとHMSで使用するプログラミング・モデルです。

[図 1.2] プログラミング・モデル



参考

HMSを使用するためには、メッセージング・システムの基本的な概念と用語についての知識が必要です。HMSはJMSの概念を使用するため、以降の説明ではJMSで使用する用語を使用します。

1.3.1. コネクション

HMS機能を使用するアプリケーションは、基本的にTmaxとの接続が必要です。起動すると自動的に接続するサーバー・アプリケーションとは異なり、クライアント・アプリケーションはHMS関連APIを使用する前にtpstart()を使用してTmaxと接続する必要があります。接続が終了した場合、クライアントがtpend()を実行するか、サーバーが終了した場合、アプリケーションで作成したセッションとクライアント情報がHMSからすべて削除されます。

1.3.2. セッション

セッションはHMS機能を使用するのに最も基本となる処理単位です。

HMSで使用するセッションは、以下の4つに分けられます。

- 一般的なセッション

セッションはTmaxとの接続が存在する状態で作成される必要があり、1つのアプリケーション内で多数のセッションを作成することもできます。セッションを作成するには対象HMSの名前が必要で、作成する場合はセッションの様々な属性を付与できます。

以下は、セッションの基本的な属性です。

属性	説明
Transacted	セッションのトランザクティッド属性の使用可否を決定します
Acknowledge Mode	メッセージ受信に対する応答方法の種類を決定します。 現在対応する応答方法は以下のとおりです – AUTO_ACK : 自動的に応答 – CLIENT_ACK : ユーザーが応答を直接送信

- トランザクティッド・セッション(Transacted Session)

トランザクティッド・セッションは、トランザクティッド属性が付与されて作成されたセッションです。主にローカル・トランザクションを使用するために使用します。トランザクティッド・セッションでは、メッセージの送受信の結果がHMSに即時反映されません。hms_commit()、hms_rollback() APIを使用して反映の可否を決定します。

たとえば、トランザクティッド・セッションで作成されたプロデューサーが多数のメッセージを送信し、hms_rollback()を行った場合、送信していたメッセージはデスティネーションに入らず、無視されます。送信した際の結果反映可否をhms_commit()、hms_rollback()で決定します。トランザクティッド・セッションではAUTO_ACKのみ許容します。

- XAセッション

XAセッションはHMSと他のRMとの2PCを実行するために存在します。トランザクションで結合する区間の開始と終わりにTmaxのトランザクション関連API(tx_beggin(), tx_commit(), tx_rollback())を使用して明示します。トランザクションが始まっていない状態では一般セッションのように動作します。

- 非同期セッション(Async Session)

HMSはメッセージに対する要求以降にメッセージが送信される同期的な方法だけでなく、メッセージに対する要求がなくてもメッセージを受信できる非同期セッションをサポートします。非同期セッションでコンシューマ(Consumer)を作成する場合、メッセージが到着する際に呼ばれるサービス名(Listener)を転送すると、メッセージが到着する度に指定したTmaxサービスが呼び出されます。

1.3.3. デスティネーション

デスティネーション(Destination)は、遅延処理のために送信されたメッセージがコンシューマに受信されるまで留まるストレージです。各デスティネーションは固有の名前を持ち、以下の2種類があります。

- トピック(Topic)

Publish-and-Subscribeメッセージングで使用されるデスティネーションです。1つのメッセージは、送信された時点で加入しているすべてのサブスクライバー(Subscriber)に渡されます。その時点で加入したサブスクライブが存在しない場合は、送信されたメッセージは破棄されます。

- キュー(Queue)

Point-to-Pointメッセージングで使用されるデスティネーションです。1つのメッセージは1つのレシーバー(Receiver)に送信されます。セnderが送信したメッセージは、レシーバーがメッセージを引き出すまでデスティネーションに存在します。

プロデューサー

プロデューサーはデスティネーションにメッセージを送信するサブジェクトを意味します。対象デスティネーションの種類に応じて、トピックのプロデューサーはパブリッシャー(Publisher)、キューのプロデューサーはセnder(Sender)と呼びます。

プロデューサーを作成するためには、Tmaxシステムと接続された状態で作成されたセッション、デスティネーションの名前、デスティネーションの種類が必要です。そして、必要に応じてプロデューサーの名前を設定できます。プロデューサーはメッセージを送信する際に該当メッセージの伝達についての属性を付与できます。

属性	説明
Delivery Mode	メッセージの永続的(Persistent)あるいは非永続的(Non-Persistent)属性を付与して送信します。永続的メッセージはデスティネーションへの到着と同時にストレージにも格納され、障害が発生した場合は復旧が可能です。非永続的メッセージはHMSのメモリーでのみ存在し、HMSを終了するか障害が発生すると消滅します

属性	説明
Priority	メッセージがデスティネーション内にキューイングされる際の優先順位を指定します。 現在HMSではサポートしていません
Time to Live	メッセージがデスティネーション内で有効な時間を指定します。指定時間が過ぎると、メッセージはコンシューマに送信されなくなります。HMSは、Time to Live(以下、TTL)を秒単位でサポートします

メッセージが送信に成功すると、各メッセージは固有のメッセージIDが発給されます。メッセージIDはプロパティとしてメッセージに存在し、ユーザーがhms_get_property()を使用して確認できます。

コンシューマ

コンシューマ(Consumer)は、デスティネーションからメッセージを受信するサブジェクトを意味します。

コンシューマを作成するためには、デスティネーションの名前、デスティネーションの種類、作成するコンシューマの名前が必要です。また、特定のプロパティを持つメッセージのみ選択して受信する場合はメッセージ・セレクトを設定でき、非同期的メッセージを受信する場合は受信する際に呼び出されるサービス名を設定します。コンシューマがメッセージを受信する際にタイムアウト値を指定できます。指定していない場合、メッセージが到着するまでブロックされます。タイムアウト値の単位は秒です。

- サブスクライバー(Subscriber)

対象デスティネーションの種類に応じたトピックのコンシューマをサブスクライバーと呼びます。

コンシューマのうち永続的属性を持つサブスクライバーを**恒久サブスクライバー**(Durable Subscriber)といいます。恒久サブスクライバーは、作成された後にサブスクライブの解除(unsubscribe)を行うまでデスティネーションに渡されたすべてのメッセージを受け取ることができます。恒久サブスクライバーがTmaxとの接続から解除されても、後に再接続すると以前のメッセージに続いて受信が可能であるという意味です。HMSが恒久サブスクライバーを区分する基準は、サブスクライバーの名前です。したがって、恒久サブスクライバーを作成する際には必ず固有の名前を付与します。

- レシーバー(Receiver)

キューのコンシューマをレシーバーと呼びます。

レシーバーを作成する際に**メッセージ・セレクト**(Message Selector)を設定できます。メッセージ・セレクトはユーザーが一定の条件を設定し、該当条件を充足するメッセージのみを受信できるようにします。メッセージ・セレクト構文はSQL条件文と同じです。

たとえば、メッセージにTypeとMonthというプロパティが存在する場合、以下のようなメッセージ・セレクトを設定したコンシューマは「4月以降のニュース」メッセージのみ受信します。

```
"(Type = 'News') AND (Month > 4)"
```

メッセージ・セクタについての詳細内容は、「[付録 A. メッセージ・セクタの使用方法](#)」を参照してください。

1.3.4. メッセージ

HMSで使用するメッセージはTmaxのフィールド・バッファーを使用するため、フィールド・バッファーの長所をそのまま使用できます。1つのメッセージ内には1つのBodyと多数のプロパティを設定できます。

- Body

CARRAYタイプに設定します。

- プロパティ

プロパティはフィールド・バッファーでサポートするHMS_CHAR、HMS_SHORT、HMS_INT、HMS_LONG、HMS_FLOAT、HMS_DOUBLE、HMS_STRING、HMS_CARRAYの8つのデータ型に設定します。プロパティ名はメッセージで一意である必要があります。同じ名前でプロパティを再設定した場合、以前の値が修正されます。

メッセージを作成するためにはセッションを使用し、各プロパティはデータ部分だけでなく、名前を示す空間も必要であるため、多くのプロパティを設定する場合はメッセージのサイズを考慮して十分に割り当てます。

以下は、HMSで基本的に定義した予約プロパティです。ユーザーはこれと同じ名前でプロパティを再設定できません。(これ以外の予約済みのプロパティがありますが、現在はサポートしていません)

プロパティ	説明
HMSMessageID (\$MessageID)	メッセージの固有IDです。HMSが発給するため、送信に成功後に確認できます。12文字の16進数の文字列で確認できます
HMSTimeStamp (\$TimeStamp)	プロデューサーがメッセージを送信し、その応答を受信するまでにかかった時間(秒)を示します
HMSDestination (\$Destination)	メッセージが送受信されたデスティネーションの名前です
HMSDeliveryMode (\$DeliveryMode)	メッセージが永続的(Persistent)であるのか、あるいは非永続的(Non-Persistent)であるのかを示します
HMSExpiration (\$Expiration)	プロデューサーがメッセージを送信しながら設定したTTL値です

1.3.5. キュー・ブラウザー

キュー・ブラウザー(QueueBrowser)は、キューのメッセージを照会できる一種のコンシューマです。キューからの確認用途のメッセージを受信しますが、実際にはメッセージが消費されません。基本的に最新メッセージから順に閲覧するように動作しますが、最も古いメッセージからの閲覧も可能です。キュー・ブラウザーの名前でのように、対象デスティネーションはキューのみ許可します。

1.3.6. リカバリー

永続的メッセージ(Persistent Message)と恒久サブスクライバー(Durable Subscriber)の情報はストレージに格納されます。HMSは起動するとすぐに、ストレージに残っている処理されていない永続的メッセージと恒久サブスクライバーの情報を復旧できます。

復旧の可否は起動オプションで設定し、デスティネーション別に設定できます。

オプション	説明
WARM	起動する際に復旧します
COLD	起動する際にストレージの情報を削除します

恒久サブスクライバーが復旧後にメッセージを続けて受信するためには、以前作成された名前で再作成される必要があります。したがって、恒久サブスクライバーを作成する際には固有の名前を付与することが重要です。また、サブスクライブの解除(unsubscribe)を行わずに恒久サブスクライバーを削除した場合、該当の恒久サブスクライバーに渡されていないメッセージとそれ以降に到着するすべてのメッセージがデスティネーションに蓄積されるので注意します。

1.3.7. クラスターリング

Tmaxマルチノード環境で多数のHMSはクラスターリング(Clustering)構成が可能です。これにより、フェイルオーバーと負荷分散効果が得られます。名前が同じであり、GLOBAL属性を設定したデスティネーションは、論理的に1つのデスティネーションのように動作します。

HMSのクラスターリングを構成するためには、各セクションに以下のように設定します。

- SVRGROUPセクション

HMSPORTは必ず設定し、キューをクラスターリングするためには追加でGQINTを設定します。

- HMSセクション

クラスターリングするデスティネーションの名前を同じく設定し、GLOBAL=Yと設定します。

クラスターリングされたHMSは互いの障害を検知するためにHeartbeatメッセージを送受信します。ユーザーが設定したHMSHEARTBEAT時間内にHeartbeatメッセージが受信されない場合は障害が発生したと判断します。

HMSでは、クラスタリングが設定されたトピックをグローバル・トピック(Global Topic)、キューをグローバル・キュー(Global Queue)と呼びます。

- グローバル・トピック(Global Topic)

グローバル・トピックに送信されたメッセージは、他のノードのグローバル・トピックに渡されます。

たとえば、ノードA、B、CにTopic01という同じ名前を持つトピックにすべて「GLOBAL=Y」と設定した場合、ノードAのTopic01に受信されたメッセージはノードB、CのTopic01にも渡されます。

- グローバル・キュー(Global Queue)

設定されたすべてのノードにコピーが渡されるグローバル・トピックとは異なり、グローバル・キューでは1つのメッセージのみ存在するので、クラスタリングでは異なる方法を使用します。

各HMSはグローバル・キューの情報を一定時間(HMSGQINT)の周期で送受信を繰り返します。この情報には、現在のノードの各キューに存在するメッセージの数、接続しているクライアントの数などが含まれます。

送受信情報	説明
Global Queue Information Interval (GQINT)	HMSのグローバル・キューの状態情報をやり取りする周期です
Global Queue Threshold (GQTHR)	グローバル・キューのバッファのサイズを求める際に使用される定数値です (バッファ・サイズ＝該当するキューのコンシューマ数 X GQTHR)
Global Queue Max Request (GQMAXREQ)	現在のキューの長さがバッファ量を埋めることができない場合、他のノードから1度の要求で取得できるメッセージの最大数です
Global Queue Full Length (GQFULL)	キューの長さが不均衡になる現象を防ぐために設定するキューの境界の長さです。この長さ以上になると、より短い長さのキューにメッセージが移動します

各キューにはメッセージを円滑に送信するためにバッファリングを行います。そのサイズは現在のキューに接続されているコンシューマの数にユーザーが設定した定数値(GQTHR)を乗算して求めます。

キューのメッセージ数が計算されたバッファのサイズより小さい場合、他のノードのキューからメッセージを要求します。1度の要求で取得できるメッセージの数(GQMAXREQ)も設定できます。この値が大きい場合は少しの要求で必要なメッセージを埋めることができますが、大き過ぎる場合はノード間の不要なメッセージの移動が発生することがあります。

バッファのサイズ以外にも、キューの長さが不均衡的に大きく差が出る現象を防ぐために、メッセージが一定数以上(GQFULL)になればメッセージの要求がなくてもキューの長さを均衡的にする機能も提供します。

第2章 HMSの環境設定

HMSを使用するためには、Tmaxの環境ファイルにHMS関連の設定を追加します。

本章では、HMS関連の設定について説明します。

2.1. Tmaxの環境設定

HMSを使用するためには、DOMAIN、NODE、SVRGROUP、HMSセクションの設定が必要です。

2.1.1. DOMAINセクション

以下は、DOMAINセクションでHMSを使用するための環境設定形式と項目についての説明です。

```
*DOMAIN
Domain1      [MAXSESSION = numeric]
```

選択項目

- MAXSESSION = numeric
 - サイズ : 1 ~ 65535
 - デフォルト値 : 1024
 - ドメイン内のHMSで作成できるセッション数の最大値を設定します。

2.1.2. NODEセクション

以下は、NODEセクションの環境設定形式と項目についての説明です。

```
*NODE
Node1        [MAXSESSION = numeric]
```

選択項目

- MAXSESSION = numeric

- サイズ：1 ～ 65535
- デフォルト値：1024
- ノード内のHMSで作成できるセッションの最大数を設定します。

2.1.3. SVRGROUPセッション

以下は、SVRGROUPセクションの環境設定形式と項目についての説明です。

```
*SVRGROUP
ServergroupName      NODENAME = node-name,
                     SVGTYPE = HMS,
                     HMSNAME = string,
                     OPENINFO = literal,
                     HMSINDEX = numeric,
                     HMSMAXTHR = numeric,
                     HMSMAXDBTHR = numeric,
                     [HMSMAXBULKTHR = numeric,]
                     [HMSMAXBULKSIZE = numeric,]
                     [HMSOPT = literal,]
                     [HMSSUBSCFG = string,]
                     [HMSMSGLIVE = numeric,]
                     [HMSPORT = numeric,]
                     [HMSHEARTBEAT = numeric,]
                     [HMSGQINT = numeric]
```

必須項目

- *ServergroupName* = string
 - サイズ: 63文字以内
 - HMSサーバー・グループの論理的な名前として、SVRGROUPセクション内で一意の名前である必要があります。
- *NODENAME* = string
 - サイズ: 63文字以内
 - HMSサーバー・グループが存在するノードを定義します。使用される*NODENAME*は*NODE*セクションで定義したノード名である必要があります。

- SVGTYPE = string
 - 該当するサーバー・グループのタイプを定義します。HMSを使用するためには、必ずSVGTYPE=HMSと設定します。

- HMSNAME = string
 - サイズ: 63文字以内

 - ユーザーがビルドしたHMSプロセス名を定義します。

- OPENINFO = literal
 - HMSは基本的にデータベースをストレージとして使用します。したがって、HMSで接続するDBMSのOPENINFOを設定する必要があります。

 - データベースへの接続を初期化し、各データベースで提供する文法で定義します。

- HMSINDEX = numeric
 - サイズ : 0~65535

 - HMSのメッセージを処理するのに必要な設定値として、必ずドメイン内で一意の値で設定します。

- HMSMAXTHR = numeric
 - サイズ : 0~65535

 - ストレージ処理をしないスレッドの数を設定します。

 - 非永続的メッセージのみ送受信する場合、この項目の値を大きくします。非永続的メッセージ以外にもHMSの基本的な動作のために一定数以上設定します。

- HMSMAXDBTHR = numeric
 - サイズ : 0~65535

 - ストレージ処理をするスレッドの数を設定します。

 - 永続的メッセージが多い場合、この値を大きくします。永続的メッセージ処理以外にもHMSの基本的な動作のために一定数以上設定します。

選択項目

- HMSMAXBULKTHR = numeric
 - 範囲 : 1~65535
 - HMSメッセージに対するストレージを一括処理するスレッドの数を設定します。
- HMSMAXBULKSIZE = numeric
 - 範囲 : 2~65535
 - HMSメッセージに対するストレージを一括処理する際、一度に処理できるメッセージの最大数を設定します。
- HMSOPT = literal
 - サイズ : 255文字以内
 - HMSが起動される際、HMSプロセスに渡されるコマンド・オプションを定義します。

オプション	説明
-eファイル名	HMSの動作中に発生する標準エラー(standard error)をファイルに記録します
-oファイル名	HMSの動作中に発生する標準出力(standard output)をファイルに記録します

- HMSSUBSCFG = string
 - サイズ : 255文字以内
 - 永続的サブスクライバーを設定した環境ファイルのパスと名前を設定します。ファイル内の内容は、Labelを[TopicName]と設定した後、各行ごとにコロン(:)で区切ってClientName:Listener:Selectorの順に設定します。
 - 環境ファイルに定義された永続的サブスクライバーはHMSが起動すると自動的に登録されます。
- HMSMSGLIVE = numeric
 - サイズ : 0~65535(単位 : 時間)
 - HMSは永続的メッセージをストレージに格納します。ストレージに蓄積されたメッセージを周期的に削除する時間を設定します。

- すべてのサブスクライバーが受信したトピック・メッセージ、消費が完了したキュー・メッセージは、設定時間が経過するとストレージから削除されます。まだ受信していない永続的レシーバーが存在するメッセージは、設定時間が経過してもストレージから削除されません。
- HMSPORT = numeric
 - 多数のHMSをクラスタ化する場合、各HMSは互いに通信するために直接チャンネルを作成します。この際に使用するポート番号を設定します。
 - GLOBAL属性を設定したデスティネーションが存在する場合、HMSは設定されたポート番号で待機します。したがって、すでに使用しているポートを設定しないように注意します。
- HMSHEARTBEAT = numeric
 - サイズ : 0~65535(単位 : 秒)
 - 多数のHMSをクラスタ化する場合、HMS間のネットワーク障害を感知するために、周期的にHeartbeatメッセージを送受信します。そのメッセージを送信する周期を設定します。
 - 周期内のHeartbeatメッセージが他のHMSから来なかった場合は障害と感知し、該当HMSとの接続を切断します。
- HMSGQINT = numeric
 - サイズ : 0~65535(単位 : ミリ秒)
 - GLOBAL属性を設定したキュー間のメッセージを送受信するためには、各キューの状態情報を送受信する必要があります。この情報を送受信する周期を設定します。
 - クラスタ化されたキューを使用するためには必ず設定します。周期が短いほど正確な分配が可能です、その分ネットワークの負荷は大きいため、適切なサイズの値を設定する必要があります。

2.1.4. HMSセクション

以下は、HMSセクションの環境設定形式と項目についての説明です。

```
*HMS
DestinationName    SVGNAME = string,
                   TYPE = TOPIC | QUEUE,
                   [BOOT = WARM | (COLD),]
                   [GLOBAL = Y | (N),]
```

```
[GQTHR = numeric,]  
[GQMAXREQ = numeric,]  
[GQFULL = numeric]
```

必須項目

- *DestinationName* = string
 - サイズ : 63文字以内
 - デスティネーションの名前を設定します。プロデューサーとコンシューマはこの名前を使用して作成されます。
- *SVGNAME* = string
 - サイズ : 63文字以内
 - デスティネーションが存在するHMSサーバー・グループ名を定義します。
 - *SVGNAME*はSVRGROUPセクションで定義したHMSサーバー・グループ名(SVRGROP名)である必要があります。
- *TYPE* = TOPIC | QUEUE
 - デスティネーションのタイプを設定します。現在はTOPICとQUEUEのみサポートします。

選択項目

- *BOOT* = WARM | COLD
 - デフォルト値 : COLD
 - 該当するデスティネーションに対して起動する際に復旧するメッセージがある場合の復旧可否を設定します。
 - RQと同様、WARMとCOLDのみ設定可能であり、デフォルト値はCOLDです。

区分	説明
WARM	ストレージにある未処理のメッセージが復旧します
COLD	起動時に該当するメッセージをすべて削除してから起動します

- GLOBAL = Y | N

- デフォルト値 : N
- クラスタ化されたHMSの同じ名前を持つデスティネーションを論理的に1つのデスティネーションで動作するように設定します。
- GLOBAL設定をするためには、SVRGROUPセクションのHMS設定にHMSPORTが設定されている必要があります。
- キューのGLOBAL設定をするためには、次の項目で紹介するGQTHR、GQMAXREQ、GQFULLの追加設定が必要です。

- GQTHR = numeric

- サイズ : 0~65535
- クラスタ化されたキューが維持するバッファ・サイズを決定する係数値を設定します。

バッファサイズ = (コンシューマ数) X (GQTHRの値)

- GQTHR値です。現在のキューの長さがバッファのサイズより小さい場合、割り当てるメッセージが存在するクラスタ化された他のキューからメッセージを取得します。バッファのサイズより少ないメッセージを持っている場合は、メッセージの割り当て要求を受けてもメッセージを割り当てません。

- GQMAXREQ = numeric

- サイズ : 1~65535
- クラスタ化されたキューが他のキューにメッセージを要求する場合に、1度に要求できるメッセージの数を設定します。設定された値が大きい場合は少ない回数で必要なメッセージを埋めることができますが、大き過ぎる場合はノード間の不要なメッセージの移動が発生することがあるため、設定値を大きくし過ぎないことを推奨します。

- GQFULL = numeric

- サイズ : 0~65535
- クラスタ化されたキューのメッセージ消費が行われない場合やメッセージが不均衡に非常に多く蓄積される場合に備えて、設定した数以上のメッセージが蓄積した場合はその超過分に限って他のノードのクラスタリング・キューにメッセージを均等に分けます。

2.2. 例

以下は、HMSを使用する環境ファイルの例です。

```
*DOMAIN
Domain1      SHMKEY = 0x11936,
              TMMLOGLVL = "DEBUG1",
              CLHLOGLVL = "DEBUG1",
              MINCLH = 1,
              MAXCLH = 1,
              TPORTNO = 7783,
              MAXSESSION = 1024

*NODE
Node1        TMAXDIR = "/data/tmax",
              APPDIR  = "/data/tmax/appbin/",
              MAXSESSION = 1024

*SVRGROUP
svghms       NODENAME = "node1", CPC=1, SVGTYPE="HMS", RESTART=N,
              OPENINFO = "ORACLE_XA+Acc=P/scott/tiger+SesTm=60+Threads=true",
HMSSUBSCFG = "/data/tmax/config/hmssubconfig",
              HMSNAME = "hms_ora", HMSSINDEX = 1,
              HMSMSGLIVE = 1, HMSMAXTHR = 4, HMSMAXDBTHR = 10

*SERVER
...
*SERVICE
...

*HMS
queue1       SVGNAME = svghms,
              TYPE = "QUEUE",
              BOOT = "WARM",
              GLOBAL = N
topic1       SVGNAME = svghms,
              TYPE = "TOPIC",
              BOOT = "WARM",
              GLOBAL = N
```

以下は、**HMSSUBSCFG**に設定した環境設定の例です。

```
[TOPIC1]
Durable0:listener:
Durable1:listener:
Durable2:listener:
```

第3章 API関数

本章では、HMSを使用してメッセージを送信するために必要な様々なAPIについて説明します。

3.1. 概要

HMS API関数はTmax APIと一緒に提供されるサーバー・ライブラリーとクライアント・ライブラリーを通じて提供されます。したがって、Tmax APIとHMS APIと一緒に使用してアプリケーションを作成できます。UNIX環境でのサーバー・ライブラリーはlibsvr.a、クライアント・ライブラリーはlibcli.a、libclithr.aで提供され、アプリケーションをコンパイルする際に該当ライブラリーを含んでコンパイルします。

Tmax APIは基本的にhmsapi.hヘッダー・ファイルを使用し、Tmax APIと一緒に使用するためには、atmi.h、tmaxapi.h、tx.hヘッダー・ファイルと一緒に使用します。クライアント・アプリケーションを作成する場合、HMSを使用するためにはtpstart()を使用してTmaxに必ず接続します。したがって、この場合はhmsapi.hと一緒にatmi.hヘッダー・ファイルを使用します。

ヘッダー・ファイル	説明
hmsapi.h	HMSを使用するためのセッションおよびプロデューサー、コンシューマ、メッセージなどに関連するAPIが定義されています
atmi.h	バッファー管理および通信と関連するAPIが定義されています
tmaxapi.h	非標準APIのプロトタイプが定義されています
tx.h	トランザクション関数のプロトタイプが定義されています

HMS APIは、セッション、メッセージ、プロデューサー、コンシューマ、キュー・ブラウザーの5つに分けられます。

● セッションAPI

API	説明
hms_create_session	HMSのセッションを作成し、セッション・ハンドルを返します
hms_create_xa_session	HMSのXAセッションを作成し、ハンドルを返します
hms_create_async_session	非同期セッションを作成してハンドルを返します
hms_close_session	セッションを終了します
hms_unsubscribe_durable_subscriber	恒久サブスクライバーの登録を解除します
hms_commit	セッションを使用して送受信していたメッセージのローカル・トランザクションをコミットします

API	説明
hms_rollback	セッションを使用して送受信していたメッセージのローカル・トランザクションをロールバックします
hms_recover	セッションを使用して送受信していたメッセージのうちACKを受信できなかったメッセージから送信を再度開始します

- メッセージAPI

API	説明
hms_alloc	メッセージング・サービスに使用するメッセージ・バッファを割り当て、ポインタを返します
hms_free	メッセージ・バッファに割り当てられたメモリを解除します
hms_get_property	メッセージに設定された、該当する名前のプロパティの情報を取得します
hms_set_property	該当する名前でメッセージにプロパティを設定します
hms_get_body	メッセージのデータ部分を取得します
hms_set_body	メッセージのデータ部分を設定します
hms_ack	受信したメッセージのACKを送信します

- プロデューサーAPI

API	説明
hms_create_producer	デスティネーションにメッセージを送信するためにプロデューサーを作成し、ハンドルを返します
hms_create_sender	キュータイプのデスティネーションにメッセージを送信するセNDERを作成します
hms_create_publisher	トピックタイプのデスティネーションにメッセージを送信するパブリッシャーを作成します
hms_close_producer	プロデューサーを閉じ、割り当てられていたリソースを返します
hms_close_sender	キュータイプのセNDERを閉じ、割り当てられていたリソースを返します
hms_close_publisher	トピックタイプのパブリッシャーを閉じ、割り当てられていたリソースを返します
hms_sendex	プロデューサーを作成する際に指定したデスティネーションにメッセージを送信します
hms_send	基本的な設定のみでメッセージを送信します

- コンシューマAPI

API	説明
hms_create_consumer	デスティネーションからメッセージを受信するコンシューマを作成し、ハンドルを返します
hms_create_receiver	キュータイプのデスティネーションからのレシーバーを作成します
hms_create_subscriber	トピックタイプのデスティネーションからのサブスクライバーを作成します
hms_create_durable_subscriber	トピックタイプのデスティネーションから受信する恒久サブスクライバーを作成します
hms_close_consumer	コンシューマを閉じ、割り当てられていたリソースを返します
hms_close_receiver	キュータイプのレシーバーを閉じ、割り当てられていたリソースを返します
hms_close_subscriber	トピックタイプのサブスクライバーを閉じ、割り当てられていたリソースを返します
hms_close_durable_subscriber	恒久サブスクライバーを閉じ、割り当てられていたリソースを返します
hms_recvex	デスティネーションからメッセージを受信します
hms_recv	デスティネーションからのメッセージが到着するまで待機する関数です

- キュー・ブラウザーAPI

API	説明
hms_create_browser	デスティネーションに存在するメッセージを照会できるキュー・ブラウザーを作成し、ハンドルを返します
hms_close_browser	キュー・ブラウザーを閉じ、割り当てられていたリソースを返します
hms_browser_nextmsg	最後に届いたメッセージから順番にメッセージを照会します
hms_browser_get_msgselector	ブラウザーを作成する際に設定したメッセージ・セクタの構文を取得します
hms_browser_get_queue	ブラウザーを作成する際に設定したデスティネーションの名前をバッファーにコピーし、その長さをlenに設定します

3.2. セッションAPI

セッションAPIは、セッションを作成および終了するためのAPIおよびトランザクションと関連するAPIで構成されています。

3.2.1. hms_create_session

HMSのセッションを作成し、セッション・ハンドルを返す関数です。セッション・ハンドルを使用して、セッション、プロデューサー、コンシューマー、メッセージ、キュー・ブラウザーを作成できます。セッションを作成する際、ローカル・トランザクション可否とACKモードと一緒に設定できます。ローカル・トランザクション・モードでは、hms_commitまたはhms_rollback APIを呼び出さなければトランザクションが完了しません。

クライアント・アプリケーションでhms_create_session APIを使用するためには、まずTmaxへの接続が必要です。tpstart()を使用してTmaxと接続を行った後にセッションを作成します。サーバー・アプリケーションで使用する場合は、tpstart()を使用して接続を行う必要がありません。

● プロトタイプ

```
#include <usrinc/hmsapi.h>
HMS_SHND * hms_create_session (char *hms, int transacted, int ackmode, int flags)
```

● パラメータ

パラメータ	説明
hms	設定ファイルのSVRGROUPセクションで指定したHMSに該当するSVGの名前です
transacted	ローカル・トランザクション可否です – 0 : ローカル・トランザクションを使用しません – 1 : ローカル・トランザクションを使用します
ackmode	– HMS_AUTO_ACK : 内部的にメッセージごとにACKを送信します – HMS_CLIENT_ACK : クライアントが直接ACKを送信します – HMS_DUPS_OK_ACK : 多数のメッセージを受信し、一度にACKを送信します (現在はサポートしていません)
flags	現在は使用されていません

● 戻り値

戻り値	説明
作成されたセッション・ハンドラのポインタ	関数の呼び出しに成功しました
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

● エラー

以下のような状況において、hms_create_session()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。hmsがNULLであるか、指定された文字列の長さを超過したか、ackmodeが有効でない値に設定されました
[TPEOS]	クライアントでAPIを使用する際に発生することがあります。HMSの運用システムにエラーが発生したか、または、環境変数の設定が正しくありません。 たとえば、TMAX_HOST_ADDRやTMAX_HOST_PORTの設定が正しくなく、接続が失敗した場合に発生します
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPELIMIT]	HMSに接続されたセッションの数が設定ファイルのMAXSESSIONに到達し、それ以上セッションを作成できない状態です
[TPENOREADY]	HMSに指定した名前のHMSがNOT READY状態です
[TPENOENT]	HMSに指定した名前のHMSが存在しません

● 例

```

...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    char *hms, *dest;
    char *buf;
    int ret;
    long size = 1024;
    ...
    if (argc != 3)
        return;
    hms = argv[1];
    dest = argv[2];
    /* トランザクション・モードのセッションを作成する */
    session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                               NULL, 0);
    if (cons == NULL) { error processing }
    msg = hms_alloc(session, size);

```

```

    if (msg == NULL) { error processing }
    ret = hms_recv(cons, &msg, 0);
    if (ret < 0) { error processing }
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL) { error processing }
    ret = hms_get_body(msg, buf, &size);
    if (ret < 0) { error processing }
    ...
    ret = hms_commit(session, 0);
    if (ret < 0) { error processing }
    ...
    hms_free(msg);
    hms_close_consumer(cons, 0);
    /* セッションを終了する */
    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}

```

- 関連関数

hms_close_session(), hms_create_producer(), hms_create_consumer(), hms_create_browser(), hms_commit(), hms_rollback(), hms_recover()

3.2.2. hms_create_xa_session

HMSのXAセッションを作成し、ハンドルを返す関数です。このセッションで作成したクライアント(プロデューサーまたはコンシューマー)のすべてのメッセージはトランザクションで処理されます。トランザクションの範囲はtx_begin()を呼び出し、tx_commit()またはtx_rollback()が呼び出されるところまでです。tx_begin()とtx_commit()の範囲以外でメッセージを処理した場合、該当メッセージはトランザクションの影響を受けません。hms_create_sessionで作成した場合、transacted = 0、ackmode = HMS_AUTO_ACKに設定した場合と同一に作動し、hms_send()、hms_recv() APIを呼び出す即時呼び出し結果が反映されます。

クライアント・アプリケーションでhms_create_xa_session APIを使用するためには、まずTmaxへの接続が必要です。tpstart()を使用してTmaxと接続を行った後にセッションを作成します。サーバー・アプリケーションで使用する場合は、tpstart()を使用して接続する必要がありません。

- プロトタイプ

```

#include <usrinc/hmsapi.h>
HMS_SHND * hms_create_xa_session (char *hms, int flags)

```

- パラメータ

パラメータ	説明
hms	設定ファイルで指定したHMSに該当するサーバー・グループ名です
flags	現在は使用されていません

- 戻り値

戻り値	説明
作成されたセッション・ハンドラのポインタ	関数の呼び出しに成功しました
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_create_xa_session()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。hmsがNULLあるいは指定された文字列の長さを超過しました
[TPEOS]	クライアントでAPIを使用する際に発生することがあります。HMSの運用システムにエラーが発生したか、または、環境変数の設定が正しくありません。 たとえば、TMAX_HOST_ADDRやTMAX_HOST_PORTの設定が正しくなく、接続が失敗した場合に発生します
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPELIMIT]	HMSに接続されたセッションの数が設定ファイルのMAXSESSIONに到達し、それ以上セッションを作成できない状態です
[TPENOREADY]	HMSに指定した名前のHMSがNOT READY状態です
[TPENOENT]	HMSに指定した名前のHMSが存在しません

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_PHND *prod;
    hms_msg_t *msg;
```

```

char *hms, *dest, *buf;
int ret;
long size = 1024;
...
if (argc != 3)
    return;
hms = argv[1];
dest = argv[2];
/* グローバル・トランザクション・モードのセッションを作成する */
session = hms_create_xa_session(hms, 0);
if (session == NULL) { error processing }

ret = tx_begin();
if (ret < 0) { error processing }

prod = hms_create_producer(session, dest, HMS_QUEUE, "producer_1", 0);
if (prod == NULL) { error processing }
msg = hms_alloc(session, size);
if (msg == NULL) { error processing }
...
ret = hms_set_body(msg, buf, size);
if (ret < 0) { error processing }
ret = hms_send(prod, msg, 0);
if (ret < 0) { error processing }
...
ret = tx_commit();
if (ret < 0) { error processing }
...
hms_free(msg);
hms_close_producer(prod, 0);
/* セッションを終了する */
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_close_session()

3.2.3. hms_create_async_session

非同期セッションを作成してハンドルを返す関数です。このセッションでメッセージを非同期的に処理できます。

非同期セッションでコンシューマを作成し、メッセージを受信するためには、hms_recv()を呼び出す代わりにメッセージを受信し、これを処理するサービスを指定します。サービス名はコンシューマを作成する際に指定できます。非同期セッションを通じてコンシューマを作成する際は必ずsvcnameを指定し、メッセージを処理するサービスを指定します。該当サービスは新しいメッセージがデスティネーションに送信されると、即時該当メッセージを受信し、この処理を実行します。クライアントやサーバー・アプリケーションのどこでも非同期セッションによるコンシューマを作成できます。

非同期セッションが終了した場合やHMSで障害が発生した場合は、Abendfuncコールバック関数が実行されます。ユーザーはこの関数を定義し、障害が発生する場合を回避できます。

● プロトタイプ

```
#include <usrinc/hmsapi.h>
HMS_SHND * hms_create_async_session (char *hms, Abendfunc *func, int flags)
```

● パラメータ

パラメータ	説明
hms	設定ファイルで指定したHMSに該当するサーバー・グループ名です
func	HMSに障害が発生した場合に呼ばれるコールバック関数です
flags	現在は使用されていません

● 戻り値

戻り値	説明
作成されたセッション・ハンドラのポインタ	関数の呼び出しに成功しました
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

● エラー

以下の状況下で、hms_create_async_session()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。hmsがNULLあるいは指定された文字列の長さを超過しました
[TPEOS]	クライアントでAPIを使用する際に発生することがあります。HMSの運用システムにエラーが発生したか、または、環境変数の設定が正しくありません。 たとえば、TMAX_HOST_ADDRやTMAX_HOST_PORTの設定が正しくなく、接続が失敗した場合に発生します

エラーコード	説明
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPELIMIT]	HMSに接続されたセッションの数が設定ファイルのMAXSESSIONに達し、それ以上セッションを作成できない状態です
[TPENOREADY]	HMSに指定した名前のHMSがNOT READY状態です
[TPENOENT]	HMSに指定した名前のHMSが存在しません

- 例

```

...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

HMS_SHND *session;
HMS_PHND *cons;
HMS_PHND *prod;

void abend_callback(HMS_SHND *sess)
{
    hms_close_session(sess, 0);
    printf("hms_close_session success\n");
    while (1) {
        session = hms_create_async_session(hms, abend_callback, 0);
        if (session == NULL) {
            printf("hms_create_session() : FAIL [%d]\n\n", tperrno);
            if (tperrno == TPENOREADY) {
                sleep(2);
                continue;
            }
        }
        return;
    }
    break;
}

printf("hms_create_session success\n");
cons = hms_create_receiver(session, dest, "consumer_1", NULL, "MYSVC", 0);

if (cons == NULL) {
    printf("hms_create_receiver() : FAIL tperrno = %d\n\n", tperrno);
    return;
}
}

main(int argc, char* argv[])
{

```

```

hms_msg_t *msg;
char *hms, *dest, *buf;
int ret;
long size = 1024;
...
if (argc != 3)
    return;
hms = argv[1];
dest = argv[2];
/* 非同期セッションを作成する */
session = hms_create_async_session(hms, 0);
if (session == NULL) { error processing }

cons = hms_create_receiver(session, dest, "consumer_1", NULL, "MYSVC", 0);

if (cons == NULL) { error processing }

prod = hms_create_producer(session, dest, HMS_QUEUE, "producer_1", 0);
if (prod == NULL) { error processing }
msg = hms_alloc(session, size);
if (msg == NULL) { error processing }
...
ret = hms_set_body(msg, buf, size);
if (ret < 0) { error processing }
ret = hms_send(prod, msg, 0);
if (ret < 0) { error processing }
...
hms_free(msg);
hms_close_producer(prod, 0);
/* セッションを終了する */
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_close_session()

3.2.4. hms_close_session

作成していたセッションを終了し、HMSに割り当てられていたリソースを返す関数です。hms_create_session() またはhms_create_xa_session()、hms_create_async_session()に作成されていたセッションを閉じ、HMS内に割り当てられていたリソースを返します。セッションがそれ以上必要でない場合はセッションを終了しま

す。セッションが終了する場合、該当セッションで作成されていたすべてのクライアント(プロデューサーとコンシューマ)も一緒に終了します。

- プロトタイプ

```
#include <usrinc/hmsapi.h>
int hms_close_session (HMS_SHND *sess, int flags)
```

- パラメータ

パラメータ	説明
sess	閉じるセッション・ポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_close_session()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessセッション・ポインタがNULLあるいは正しくないポインタです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPENOREADY]	HMSに指定した名前のHMSがNOT READY状態です

- 関連関数

hms_create_session(), hms_create_xa_session(), hms_create_async_session()

3.2.5. hms_unsubscribe_durable_subscriber

恒久サブスクライバー(Durable Subscriber)の登録を解除する関数です。デスティネーションに接続されたクライアントのうち恒久サブスクライバーが存在する場合、TOPICタイプのメッセージはすべての恒久サブスクライバーがメッセージを受信するまでその情報が維持されます。

また、恒久サブスクライバーがhms_unsubscribe_durable_subscriber() APIを呼び出さずにセッションを終了する場合でも、クライアント情報(durable subscriberの名前)がデスティネーションに残っているため、再

度同じ名前の恒久サブスクライバーがセッションを新規作成すると、コンシューマが終了後にパブリッシュされたトピック・メッセージを紛失せず、すべて受信できるようになります。恒久サブスクライバーがトピック・タイプのメッセージをそれ以上受信する必要がない場合、APIを呼び出すことが望ましいです。そうすると、以降に入ってきたメッセージが該当の恒久サブスクライバーが受信するまで待機せず、メモリーから削除されます。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_unsubscribe_durable_subscriber (HMS_SHND *sess, char *des, char *name,
                                         int flags)
```

- パラメータ

パラメータ	説明
sess	セッション・ポインタです
des	登録していたデスティネーションの名前です
name	登録していた恒久サブスクライバーの名前です
flags	まだ使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tpernoにエラーコードが設定されます

- エラー

以下の状況下で、hms_unsubscribe_durable_subscriber()は失敗し、tpernoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessやdesがNULLであるか、des、nameが指定された文字列の長さを超過した場合に発生します
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPETIME]	タイムアウトが発生しました
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
#include <stdio.h>...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    /* durable_subscriberを作成する */
    cons = hms_create_durable_subscriber(session, dest, "consumer_1", NULL,
                                         NULL, 0);

    if (cons == NULL) { error processing }
    ...
    ret = hms_rcv(cons, &msg, 0);
    if (ret < 0) { error processing }
    ...

    hms_free(msg);
    ret = hms_close_durable_subscriber(cons, 0);
    if (ret < 0) { error processing }
    ...

    /* durable_subscriberの登録を解除する */
    ret = hms_unsubscribe_durable_subscriber(session, dest, "consumer_1", 0);
    if (ret < 0) { error processing }
    ...

    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}
```

- 関連関数

hms_create_durable_subscriber(), hms_close_durable_subscriber()

3.2.6. hms_commit

セッションを使用して送受信していたメッセージのローカル・トランザクションをコミットする関数です。セッションを作成する際、hms_create_session() APIで「transacted = 1」に設定し、作成したローカル・トランザクション・モードのセッションでのみ使用可能です。トランザクションがROLLBACK_ONLY状態の場合はロールバックされます。XAセッションではtx_commit() APIを使用する必要があります。XAセッションとは異なり、トランザクションの開始を明示的に宣言するためのtx_begin()と同じAPIは呼び出す必要がありません。

● プロトタイプ

```
#include <usrinc/hmsapi.h>
int hms_commit (HMS_SHND *sess, int flags)
```

● パラメータ

パラメータ	説明
sess	該当するセッション・ポインタです
flags	現在は使用されていません

● 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

● エラー

以下の状況下で、hms_commit()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessセッション・ポインタがNULLあるいは正しくないポインタです
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPEPROTO]	関数が不適切な位置で呼び出されました。たとえば、hms_create_session()を使用してセッションを作成する際、transactedを0に設定した場合に発生します
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPETIME]	タイムアウトが発生しました
[TPENOENT]	関数が呼び出される前にメッセージを送信していたプロデューサーあるいはコンシューマが終了しました
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    ...
    /* ローカル・トランザクション・モードのセッションを作成する */
    session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    prod = hms_create_producer(session, dest, HMS_QUEUE, "producer_1", 0);
    if (prod == NULL) { error processing }
    ...
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }
    ret = hms_set_body(msg, buf, size);
    if (ret < 0) { error processing }
    ret = hms_send(prod, msg, 0);
    if (ret < 0) { error processing }
    ...

    /* 今までのトランザクション作業を処理する */
    ret = hms_commit(session, 0);
    if (ret < 0) { error processing }
    ...
}
```

- 関連関数

hms_create_session(), hms_rollback()

3.2.7. hms_rollback

セッションを使用して送受信していたメッセージのローカル・トランザクションをロールバックする関数です。

セッションが作成する際にhms_create_session() APIで「transacted = 1」に設定して作成したローカル・トランザクション・モードのセッションでのみ使用可能です。XAセッションではtx_rollback() APIを使用します。

- プロトタイプ

```
#include <usrinc/hmsapi.h>
int hms_rollback (HMS_SHND *sess, int flags)
```

- パラメータ

パラメータ	説明
sess	該当するセッション・ポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

3.2.8. hms_recover

セッションを使用して送受信していたメッセージのうちACKを受信できなかったメッセージから送信を再度開始する関数です。

セッションが作成される際、hms_create_session() APIで「transacted = 0」に設定して作成したメッセージでのみ使用可能です。

コンシューマがメッセージを受信中に障害が発生するか、受信したメッセージのACKをHMSに渡さずに終了した場合、再度セッションが接続した際にhms_recover() APIを呼び出すことで、ACKを受信できなかったメッセージを再度受信できるようになります。

- プロトタイプ

```
#include <usrinc/hmsapi.h>
int hms_recover (HMS_SHND *sess, int flags)
```

- パラメータ

パラメータ	説明
sess	該当するセッション・ポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_recover()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessセッション・ポインタがNULLです
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して、現在のセッションを終了し、新しいセッションで作成して再度行います
[TPEPROTO]	関数が不適切な位置で呼び出されました。たとえば、hms_create_session()を使用してセッションを作成する際、transactedを0ではない値に設定して作成したセッションの場合に発生します
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPETIME]	タイムアウトが発生しました
[TPENOENT]	関数が呼び出される前にメッセージを送信していたプロデューサーあるいはコンシューマが終了しました
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    ...
    /* セッションを作成する */
    session = hms_create_session(hms, 0, HMS_CLIENT_ACK, 0);

    if (session == NULL) { error processing }
    cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                               NULL, 0);
    if (cons == NULL) { error processing }
    ...

    /* HMSからメッセージを受信する */
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }
    ret = hms_rcv(cons, &msg, 0);
    if (ret < 0) { error processing }
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL) { error processing }
    ret = hms_get_body(msg, buf, &size);
}
```

```

    if (ret < 0) { error processing }
    ...

    if ( condition ) {
        ret = hms_ack(msg, HMS_ACK_ONE);
        if (ret < 0) { error processing }
    } else {
        ret = hms_recover(session, 0);
        if (ret < 0) { error processing }
    }
    ...
}

```

- 関連関数

hms_create_session()

3.3. メッセージAPI

メッセージAPIは、メッセージの作成と削除、メッセージの情報を格納および照会するためのAPIで構成されています。

3.3.1. hms_alloc

メッセージング・サービスに使用するメッセージ・バッファを割り当て、バッファのポインタを返す関数です。

使用していないメッセージ・バッファを解除するためには、必ずhms_free() APIを使用し、free()を使用してメッセージ・バッファを解除してはなりません。メッセージ・バッファはプロパティとBodyで構成されており、hms_set_property()、hms_set_body()などのAPIを使用してメッセージ・バッファに情報を格納できます。プロパティはメッセージについての基本的な情報とクライアントがメッセージ・セクタを使用してメッセージを選別し取得する際、その基準を提供します。プロパティはName-Valueペアの情報で構成され、複数のプロパティを持つことができます。Bodyはメッセージの主要コンテンツを格納します。詳細情報は各APIのリファレンスを参照してください。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
hms_msg_t * hms_alloc (HMS_SHND *sess, long size)

```

- パラメータ

パラメータ	説明
sess	HMSと接続されたセッション・ポインタです

パラメータ	説明
size	割り当てられたメッセージ・バッファのサイズです

- 戻り値

戻り値	説明
割り当てられたメッセージ・バッファのポインタ	関数の呼び出しに成功しました
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_alloc()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	sessポインタがNULLです
[TPESYSTEM]	HMSシステム・エラーが発生しました

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    long size;
    ...

    /* 一般モードのセッションを作成する */
    session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }
    cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                               NULL, 0);
    if (cons == NULL) { error processing }

    /* メッセージ・バッファを割り当てる */
    size = 1024;
    msg = hms_alloc(session, size);
```



```

    if (msg == NULL) { error processing }

    ret = hms_recv(cons, &msg, 0);
    if (ret < 0) { error processing }
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL) { error processing }
    ret = hms_get_body(msg, buf, &size);
    if (ret < 0) { error processing }
    ...
    ret = hms_ack(msg, HMS_ACK_ONE);
    if (ret < 0) { error processing }
    ...

    /* メッセージ・バッファを解除する */
    hms_free(msg);
    hms_close_consumer(cons, 0);
    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}

```

- 関連関数

hms_free()

3.3.2. hms_free

メッセージ・バッファに割り当てられたメモリーを解除する関数です。hms_alloc()を使用して割り当てられたメッセージ・バッファは必要ない場合、必ずAPIを使用して解除します。メッセージ・バッファをfree()で解除してはならず、すでに解除されているメッセージ・バッファを再度解除する場合はその結果を知ることできません。

- プロトタイプ

```

#include <usrinc/hmsapi.h>
void hms_free (hms_msg_t * msg)

```

- パラメータ

パラメータ	説明
msg	解除するメッセージ・バッファ・ポインタです

- 戻り値

戻り値はありません。

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    /* 一般モードのセッションを作成する */
    session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                                NULL, 0);
    if (cons == NULL) { error processing }

    /* メッセージ・バッファを割り当てる */
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }

    ret = hms_rcv(cons, &msg, 0);
    if (ret < 0) { error processing }
    buf = tmalloc("STRING", NULL, 0);
    if (buf == NULL) { error processing }
    ret = hms_get_body(msg, buf, &size);
    if (ret < 0) { error processing }
    ...
    ret = hms_ack(msg, HMS_ACK_ONE);
    if (ret < 0) { error processing }
    ...
    /* メッセージ・バッファを解除する */
    hms_free(msg);
    hms_close_consumer(cons, 0);
    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}
```

- 関連関数

hms_alloc()

3.3.3. hms_get_property

メッセージに設定された該当する名前のプロパティの情報を取得する関数です。該当する名前のプロパティがある場合、タイプ(type)、値(value)、サイズ(len)を取得します。存在しない場合は-1を返し、tperrnoはTPENOENTに設定されます。データを格納するバッファ(value)はあらかじめ割り当て(または宣言)されている必要があり、バッファのサイズと一緒に渡される必要があります。受信したデータがバッファのサイズより大きい場合、TPEINVALエラーが発生します。

プロパティはName-Valueのペアで構成され、1つのメッセージ・バッファでNameは一意です。したがって、すでに存在するNameについて再度hms_set_property()を使用してValueを格納すると、既存のValueは新しいValueに代替されます。プロパティはhms_send()を使用してメッセージをデスティネーションに送信する前とhms_rcv()を使用してデスティネーションで受信した後にも設定できます。これを通じて受信したメッセージを加工し、再度送信することができます。プロパティは、メッセージを受信するコンシューマがメッセージ・セクタを使用してメッセージを選択的に受信する際、その基準で使用されます。

メッセージ・セクタはコンシューマを作成する際に設定でき、詳細情報は「[3.5.1. hms_create_consumer](#)」や「[付録 A. メッセージ・セクタの使用方法](#)」を参照してください。

内部的に使用される予約済みの名前についてはマクロが提供されており、以下のような予約済みのプロパティはhmsapi.hに定義されています。予約済みの名前のプロパティをユーザーが任意で修正してはなりません。予約済みのプロパティについての説明は「[1.3.4. メッセージ](#)」を参照してください。

● プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_get_property (hms_msg_t *msg, char *name, int *type, char *value,
                     long *len)
```

● パラメータ

パラメータ	説明
msg	プロパティを取得するメッセージ・ポインタです
name	取得するプロパティの名前です
type	プロパティのタイプを示すポインタです – HMS_CHAR – HMS_SHORT – HMS_INT – HMS_LONG – HMS_FLOAT – HMS_DOUBLE – HMS_STRING

パラメータ	説明
	– HMS_CARRAY
value	プロパティを取得するポインタで、プロパティのタイプによって適切な変数タイプのポインタを渡します
len	プロパティを格納するバッファのサイズとプロパティのサイズを示すポインタです

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_get_property()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。msgやname、valueがNULLである場合やmsgポインタが正しくない場合、またはlenが取得するプロパティの長さより小さい場合に発生します
[TPESYSTEM]	HMSシステムエラーが発生しました。詳細情報はログファイルに記録されます
[TPENOENT]	nameに該当するプロパティがmsgに存在しません

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    int type;
    long len = 256;
    char buf[256];      /* hms_get_property()に使用 */
    int myval;          /* hms_get_property()に使用 */
}
```

```

/* 一般モードのセッションを作成する */
session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
if (session == NULL) { error processing }
cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                           NULL, 0);
if (cons == NULL) { error processing }

/* メッセージ・バッファを割り当てる */
msg = hms_alloc(session, size);
if (msg == NULL) { error processing }
ret = hms_recv(cons, &msg, 0);
if (ret < 0) { error processing }
ret = hms_get_property(msg, HMSDestination, &type, buf, &len);
if (ret < 0) { error processing }
ret = hms_get_property(msg, HMSDestination, &type, &myval, sizeof(int));
if (ret < 0) { error processing }
...

ret = hms_ack(msg, HMS_ACK_ONE);
if (ret < 0) { error processing }
...
/* メッセージ・バッファを解除する */
hms_free(msg);
hms_close_consumer(cons, 0);
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_set_property(), hms_alloc(), hms_free()

3.3.4. hms_set_property

該当する名前でメッセージにプロパティを設定する関数です。すでに該当する名前のプロパティが存在する場合、新しい値に変更されます。プロパティに設定できるタイプはフィールド・バッファでサポートするタイプと同一です。

プロパティはName-Valueのペアで構成され、1つのメッセージ・バッファでNameは一意です。したがって、すでに存在するNameについて再度hms_set_property()を使用してValueを格納すると、既存のValueは新しいValueに代替されます。プロパティはhms_send()を使用してメッセージをデスティネーションに送信する前とhms_recv()を使用してデスティネーションで受信した後にも設定できます。これを通じて受信したメッセージを加工し、再度送信することができます。プロパティは、メッセージを受信するコンシューマがメッセージ・セクタを使用してメッセージを選択的に受信する際、その基準で使用されます。メッセージ・セクタに

についての詳細情報は「[3.5.1. hms_create_consumer](#)」や「[付録 A. メッセージ・セレクトタの使用法](#)」を参照してください。

内部的に使用される予約済みの名前についてはマクロが提供されており、以下のような予約済みのプロパティはhmsapi.hに定義されています。予約済みの名前のプロパティをユーザーが任意で修正してはなりません。予約済みのプロパティについての説明は「[1.3.4. メッセージ](#)」を参照してください。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_set_property (hms_msg_t * msg, char *name, int type, char *value,
                     long len)
```

- パラメータ

パラメータ	説明
msg	プロパティを設定するメッセージ・ポインタです
name	設定するプロパティの名前として、内部で予約済みの名前は使用できません
type	設定するプロパティのタイプです – HMS_CHAR – HMS_SHORT – HMS_INT – HMS_LONG – HMS_FLOAT – HMS_DOUBLE – HMS_STRING – HMS_CARRAY
value	プロパティを示すポインタです
len	設定するプロパティのサイズです

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_set_property()は失敗し、tpernoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。 msgやname、valueがNULLである場合やmsgポインタが正しくない場合に発生します。またはlenが負数の場合、nameが予約済みのプロパティの場合、msgにプロパティを設定するだけの十分なメモリー容量がない場合などに発生します

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    char buf[256];
    ...
    if (argc != 2 ) return 1;
    strncpy(buf, argv[1], 255);

    /* ローカル・トランザクション・モードのセッションを作成する */
    session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    prod = hms_create_producer(session, dest, HMS_QUEUE, "producer_1", 0);
    if (prod == NULL) { error processing }
    ...
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }
    ret = hms_set_property(msg, "NAME", HMS_STRING, buf, strlen(buf));
    if (ret < 0) { error processing }
    ret = hms_send(prod, msg, 0);
    if (ret < 0) { error processing }
    ...
    /* 今までのトランザクション作業を処理する */
    ret = hms_commit(session, 0);
    if (ret < 0) { error processing }
    ...
}
```

- 関連関数

hms_get_property(), hms_alloc(), hms_free()

3.3.5. hms_get_body

メッセージのデータ部分を取得するための関数です。内部的に「\$DATA」というプロパティ名を使用します。

データを取得するバッファはtpalloc()で割り当てられたバッファである必要があります。データを格納するバッファはあらかじめ割り当てられている必要があり、バッファのサイズが一緒に送信される必要があります。受信したデータがバッファのサイズより大きい場合、TPEINVALエラーが発生します。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_get_body (hms_msg_t *msg, char *data, long *len)
```

- パラメータ

パラメータ	説明
msg	メッセージ・ポインタです
data	メッセージのデータを格納するためのバッファです
len	メッセージのデータを格納できるデータ・バッファの最大サイズです

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_get_body()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。msgやdata、lenポインタがNULLである場合やmsgポインタが正しくない場合、またはlenがメッセージのデータ・サイズより小さい場合に発生します
[TPENOENT]	メッセージ・データが存在しません

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>
```



```

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    char *buf;
    ...

    /* 一般モードのセッションを作成する */
    session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }
    cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                               NULL, 0);
    if (cons == NULL) { error processing }

    /* メッセージ・バッファを割り当てる */
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }
    ret = hms_recv(cons, &msg, 0);
    if (ret < 0) { error processing }

    /* メッセージ・データを取得する */
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL) { error processing }
    ret = hms_get_body(msg, buf, &size);
    if (ret < 0) { error processing }
    ...

    ret = hms_ack(msg, HMS_ACK_ONE);
    if (ret < 0) { error processing }
    ...

    /* メッセージ・バッファを解除する */
    hms_free(msg);
    hms_close_consumer(cons, 0);
    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}

```

- 関連関数

hms_set_body()

3.3.6. hms_set_body

メッセージのデータ部分を設定する関数です。内部的に「\$DATA」というプロパティ名を使用します。

データのバッファはtpalloc()で割り当てられたバッファである必要があります。Bodyはプロパティとは異なり、Typeを持っておらず、CARRAYタイプでlenに指定された長さ分の情報を格納します。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_set_body (hms_msg_t *msg, char *data, long len)
```

- パラメータ

パラメータ	説明
msg	メッセージ・ポインタです
data	メッセージのデータに格納する情報を持っているバッファです
len	メッセージのデータの長さです

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_set_body()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。msgやdataがNULLである場合やmsgポインタが正しくない場合やlenが負数の場合、msgにデータを設定するだけの十分なメモリー容量がない場合などに発生します

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
```

```

HMS_PHND *prod;
hms_msg_t *msg;
char *buf;
...
if (argc !=2 )
    return 1;

/* ローカル・トランザクション・モードのセッションを作成する */
session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
if (session == NULL) { error processing }
prod = hms_create_producer(session, dest, HMS_QUEUE, "producer_1", 0);
if (prod == NULL) { error processing }
...
msg = hms_alloc(session, size);
if (msg == NULL) { error processing }

/* メッセージ・データを設定する */
buf = tpalloc("STRING", NULL, 0);
if (buf == NULL) { error processing }
strcpy(buf, argv[1]);
ret = hms_set_body(msg, buf, strlen(buf));
if (ret < 0) { error processing }

ret = hms_send(prod, msg, 0);
if (ret < 0) { error processing }
...
/* 今までのトランザクション作業を処理する */
ret = hms_commit(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_get_body()

3.3.7. hms_ack

受信したメッセージのACKを送信します。HMSでメッセージ受信完了のACKが送信された場合、HMSで該当メッセージの情報を処理できます。ACKを送信しない場合、該当メッセージは引き続きHMSに残るようになります。メッセージの受信後、ACKを送信する必要がある場合は、hms_create_session() APIでackmodeがHMS_DUPS_OK_ACKまたはHMS_CLIENT_ACKで作成されたセッションでメッセージを受信した場合です。

ACKを送信しなくても自動的に送信される場合がありますが、セッションを作成する際にhms_create_session() APIのパラメータであるackmodeがHMS_AUTO_ACKの場合、hms_rcv()を使用してメッセージを受信した際に即時ACKが送信され、トランザクションをサポートするセッションではhms_commit()またはtx_commit() APIを呼び出す際に自動的にACKが送信されるため、hms_ack()を呼び出す必要がありません。ACKはメッセージを受信した際に必要であるため、hms_send() APIでメッセージを送信する際はhms_ack()を呼び出しません。

flags設定がHMS_ACK_ONEの場合は該当メッセージについてのみACKが送信され、HMS_ACK_UPTHROUGHの場合はそれ以前に受信したメッセージまですべてACK処理されます。

● プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_ack (hms_msg_t *msg, int flags)
```

● パラメータ

パラメータ	説明
msg	ACKを送信する対象となるメッセージ・ポインタです
flags	以下のいずれかを設定します <ul style="list-style-type: none"> – HMS_ACK_ONE : 該当する1つのメッセージのみをACK処理します – HMS_ACK_UPTHROUGH : それ以前のメッセージまで一度にACK処理します

● 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

● エラー

以下の状況下で、hms_ack()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。flagsに有効な値が設定されていない場合やHMS_ACK_ONEとHMS_ACK_UPTHROUGHを一緒に設定した場合に発生します
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います

エラーコード	説明
[TPEPROTO]	関数が不適切な状況で呼び出されました。 たとえば、XAセッションやトランザクションをサポートするセッションで呼び出された場合やackmodeがHMS_DUPS_OK_ACK、HMS_CLIENT_ACKではないセッションで呼び出された場合です
[TPEITYPE]	正しくないmsgが引数として渡されました

● 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...

    /* 一般モードのセッションを作成する */
    session = hms_create_session(hms, 0, HMS_CLIENT_ACK, 0);
    if (session == NULL) { error processing }
    cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                               NULL, 0);
    if (cons == NULL) { error processing }
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }

    ret = hms_rcv(cons, &msg, 0);
    if (ret < 0) { error processing }

    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL) { error processing }
    ret = hms_get_body(msg, buf, &size);
    if (ret < 0) { error processing }

    /* 受信したメッセージのACKを送信する */
    ret = hms_ack(msg, HMS_ACK_ONE);
    if (ret < 0) { error processing }
    ...

    hms_free(msg);
}
```

```

hms_close_consumer(cons, 0);
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_recvx(), hms_rcv()

3.4. プロデューサーAPI

プロデューサーAPIは、トピック、キュータイプのプロデューサーを作成および終了するためのAPIで構成されています。

3.4.1. hms_create_producer

デスティネーションにメッセージを送信するためにプロデューサーを作成し、そのプロデューサーのハンドルを返す関数です。デスティネーションは管理者によって作成された仮想チャネルで、メッセージを作成および送信する際の目的地であり、またメッセージを受信する際にメッセージを提供するソースのことをいいます。

デスティネーションは、Point-to-Point方式のメッセージをサポートするキュー方式と、Publish-and-Subscribe方式でメッセージ送信をサポートするトピック方式に分けられます。1つのアプリケーション内でこの2つの方式のメッセージを両方処理できます。

プロデューサーを作成するためには、まずセッションが作成されている必要があります。プロデューサーが作成されると、hms_send() APIを使用してデスティネーションにメッセージを送信できます。プロデューサーはキュータイプまたはトピックタイプのデスティネーションにメッセージを送信することで、該当デスティネーションに接続した他のコンシューマが該当メッセージを受信できるようになります。

- プロトタイプ

```

#include <usrinc/hmsapi.h>
HMS_PHND * hms_create_producer (HMS_SHND *sess, char *des, int destype,
                                char *name, int flags)

```

- パラメータ

パラメータ	説明
sess	メッセージの送信に使用するセッション・ポインタです
des	メッセージを送信するデスティネーションの名前です
destype	デスティネーションのタイプで、デスティネーションのタイプと一致する必要があります

パラメータ	説明
	<ul style="list-style-type: none"> – HMS_QUEUE – HMS_TOPIC
name	作成するプロデューサーの名前です
flags	現在は使用されていません

- 戻り値

戻り値	説明
該当プロデューサーのハンドル	関数の呼び出しに成功しました(プロデューサーの作成に成功した場合)
NULL	関数の呼び出しに失敗しました(tperrnoにエラーコードが設定されます)

- エラー

以下の状況下で、hms_create_producer()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessセッション・ポインタがNULLである場合やdes名が指定されていない場合、またはdesやnameがそれぞれ指定された長さを超過した場合、あるいはdestypeが有効でない値に設定されている場合に発生します
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行う必要があります
[TPESYSTEM]	HMSシステムエラーが発生しました。詳細情報はログファイルに記録されます
[TPENOENT]	メッセージを送信するデスティネーションが存在しない場合や、活性化していないデスティネーションの場合に発生します
[TPEITYPE]	デスティネーションのタイプと作成するプロデューサーのデスティネーションのタイプが一致しません
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリ割り当てに失敗した場合です
[TPEMATCH]	引数として渡されたプロデューサー名がすでにHMSに登録されています
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
#include <usrinc/hmsapi.h>
```

```

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_PHND *prod;
    hms_msg_t *msg;

    ...
    session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }
    /* キュータイプのプロデューサーを作成する */
    prod = hms_create_producer(session, dest, HMS_QUEUE, "producer_1", 0);
    if (prod == NULL) { error processing }
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }
    ...

    ret = hms_set_body(msg, buf, size);
    if (ret < 0) { error processing }
    ret = hms_send(prod, msg, 0);
    if (ret < 0) { error processing }
    ...

    hms_free(msg);
    hms_close_producer(prod, 0);
    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}

```

- 関連関数

hms_close_producer()

3.4.2. hms_create_sender

キュータイプのデスティネーションにメッセージを送信するセNDERを作成するための関数です。動作方式は hms_create_producer() と同一です。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
HMS_PHND * hms_create_sender (HMS_SHND *sess, char *des, char *name, int flags)

```

- パラメータ

パラメータ	説明
sess	メッセージの送信に使用するセッション・ポインタです
des	メッセージを送信するデスティネーションの名前です
name	作成するプロデューサーの名前です
flags	現在は使用されていません

- 戻り値

戻り値	説明
プロデューサー・ハンドル	関数の呼び出しに成功しました(プロデューサーの作成に成功した場合)
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_create_sender()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessセッション・ポインタがNULLである場合やdes名が指定されていない場合、またはdesやnameがそれぞれ指定された長さを超過した場合に発生します
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行う必要があります
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPENOENT]	メッセージを送信するデスティネーションが存在しない場合や活性化されていないデスティネーションの場合に発生します
[TPEITYPE]	デスティネーションのタイプと作成するプロデューサーのデスティネーションのタイプが一致しません
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリー割り当てに失敗した場合です
[TPEMATCH]	引数として渡されたセNDER名がすでにHMSに登録されています
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
```

```

HMS_SHND *session;
HMS_PHND *prod;
hms_msg_t *msg;
...
session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
if (session == NULL) { error processing }

/* キュータイプのプロデューサーを作成する */
prod = hms_create_sender(session, dest, "producer_1", 0);
if (prod == NULL) { error processing }
msg = hms_alloc(session, size);
if (msg == NULL) { error processing }
...
ret = hms_set_body(msg, buf, size);
if (ret < 0) { error processing }
ret = hms_send(prod, msg, 0);
if (ret < 0) { error processing }
...
hms_free(msg);
hms_close_sender(prod, 0);

/* セッションを終了する */
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_close_sender()

3.4.3. hms_create_publisher

トピックタイプのデスティネーションにメッセージを送信するパブリッシャーを作成する関数です。動作方式は hms_create_producer() と同一です。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
HMS_PHND * hms_create_publisher (HMS_SHND *sess, char *des, char *name, int
flags)

```

- パラメータ

パラメータ	説明
sess	メッセージの送信に使用するセッション・ポインタです
des	メッセージを送信するデスティネーションの名前です
name	作成するプロデューサーの名前です
flags	現在は使用されていません

- 戻り値

戻り値	説明
プロデューサー・ハンドル	関数の呼び出しに成功しました(プロデューサーが正常に作成された場合)
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_create_publisher()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessセッション・ポインタがNULLである場合やdes名が指定されていない場合、またはdesやnameがそれぞれ指定された長さを超過した場合に発生します
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行う必要があります
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPENOENT]	メッセージを送信するデスティネーションが存在しない場合や活性化されていないデスティネーションの場合に発生します
[TPEITYPE]	デスティネーションのタイプと作成するプロデューサーのデスティネーションのタイプが一致しません
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリー割り当てに失敗した場合です
[TPEMATCH]	引数として渡されたパブリッシャー名がすでにHMSに登録されています
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
```

```

HMS_SHND *session;
HMS_PHND *prod;
hms_msg_t *msg;
...
session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
if (session == NULL) { error processing }

/* トピックタイプのプロデューサーを作成する */
prod = hms_create_publisher(session, dest, "producer_1", 0);
if (prod == NULL) { error processing }
msg = hms_alloc(session, size);
if (msg == NULL) { error processing }
...
ret = hms_set_body(msg, buf, size);
if (ret < 0) { error processing }
ret = hms_send(prod, msg, 0);
if (ret < 0) { error processing }
...
hms_free(msg);
hms_close_publisher(prod, 0);

/* セッションを終了する */
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_close_publisher()

3.4.4. hms_close_producer

プロデューサーを閉じ、割り当てられていたリソースを返す関数です。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
int hms_close_producer (HMS_PHND *prod, int flags)

```

- パラメータ

パラメータ	説明
prod	閉じるプロデューサーのハンドル・ポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_close_producer()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。prodポインタがNULLであるか正しくないポインタです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPENOREADY]	HMSがNOT READY状態です

- 関連関数

hms_create_producer()

3.4.5. hms_close_sender

キュータイプのセNDERを閉じ、割り当てられていたリソースを返す関数です。動作方式は、hms_close_producer()と同じです。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_close_sender (HMS_PHND *prod, int flags)
```

- パラメータ

パラメータ	説明
prod	閉じるプロデューサーのハンドル・ポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_close_sender()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。prodがNULLです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPENOREADY]	HMSがNOT READY状態です

- 関連関数

hms_create_sender()

3.4.6. hms_close_publisher

トピック・タイプのパブリッシャーを閉じ、割り当てられたリソースを返す関数です。動作はhms_close_producer()と同一です。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_close_publisher (HMS_PHND *prod, int flags)
```

- パラメータ

パラメータ	説明
prod	閉じるプロデューサーのハンドル・ポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_close_publisher()は失敗し、tpernoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。prodがNULLです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成し、再度行います
[TPENOREADY]	HMSがNOT READY状態です

- 関連関数

hms_create_publisher()

3.4.7. hms_sendex

プロデューサーを作成する際に指定したデスティネーションにメッセージを送信します。

送信方式には、Persistent modeとNon-persistent modeが存在します。Persistent modeの場合、送信されたメッセージはデータベースに格納されます。HMSに障害が発生した場合、デスティネーションに送信されていたメッセージを完全に復旧できます。該当メッセージをコンシューマが受信すれば、データベースに格納されていたPersistent modeのメッセージは、それ以降一定時間が過ぎるとデータベースから削除されます。受信したメッセージがデータベースから削除される時間は、設定ファイルで指定できます。

メッセージがデスティネーション内でスケジューリングされる優先順位や有効時間を指定して送信することもできます。有効時間が過ぎてもコンシューマによってメッセージが受信されていない場合は該当メッセージは破棄され、それ以降にメッセージ受信要求が来ても該当メッセージを受信できなくなります。

正常に送信された場合、該当メッセージにメッセージIDと送信/応答時間の差がそれぞれHMSMessageID、HMSTimeStampの名前のプロパティに設定されます。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_sendex (HMS_PHND *prod, hms_msg_t *msg, int dlvmode, int priority,
               int ttl, int flags)
```

- パラメータ

パラメータ	説明
prod	メッセージを送信するプロデューサーのハンドル・ポインタです
msg	送信するメッセージのポインタです

パラメータ	説明
dlvmode	送信方式を設定します – HMS_DLV_NON_PERSISTENT – HMS_DLV_PERSISTENT
priority	メッセージの優先順位で、現在はサポートしていません
ttl	time to liveの略で、メッセージの有効時間です(デフォルト値: 0、単位: 秒)
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_sendex()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。prodまたはmsgがNULLである場合やttlが0より小さい場合、dlvmodeが有効ではない値に設定されている場合、または正常に送信後に該当msgのサイズが小さくて追加的なプロパティを設定できない場合に発生します
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリー割り当てが失敗した場合です
[TPETIME]	タイムアウトが発生しました
[TPEPROTO]	関数が不適切な位置で呼び出されました
[TPENOENT]	すでに終了したプロデューサーを使用してメッセージを送信しました
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
```



```

{
    HMS_SHND *session;
    HMS_PHND *prod;
    hms_msg_t *msg;
    ...
    session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    /* トピックタイプのプロデューサーを作成する */
    prod = hms_create_publisher(session, dest, "producer_1", 0);
    if (prod == NULL) { error processing }
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }
    ...
    ret = hms_set_body(msg, buf, size);
    if (ret < 0) { error processing }

    /* PERSISTENTメッセージをデスティネーションに送信する */
    ret = hms_sendex(prod, msg, HMS_DLV_PERSISTENT, 0, 60, 0);
    if (ret < 0) { error processing }
    ...
    hms_free(msg);
    hms_close_publisher(prod, 0);

    /* セッションを終了する */
    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}

```

- 関連関数

hms_send()

3.4.8. hms_send

基本的な設定のみでメッセージを送信します。送信モードはNon-persistentで、優先順位と有効時間は設定せずに送信します。それ以外の動作方式はhms_sendex()と同じです。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
int hms_send (HMS_PHND *prod, hms_msg_t *msg, int flags)

```

- パラメータ

パラメータ	説明
prod	メッセージを送信するプロデューサーのハンドル・ポインタです
msg	送信するメッセージのポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	メッセージの送信が成功しました
-1	メッセージの送信が失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_send()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。prodまたはmsgがNULLである場合やttlが0より小さい場合、dlvmodeが有効ではない値に設定されている場合、または正常に送信後に該当msgのサイズが小さくて追加的なプロパティを設定できない場合に発生します
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリ割り当てに失敗した場合です
[TPETIME]	タイムアウトが発生しました
[TPEPROTO]	関数が不適切な位置で呼び出されました
[TPENOENT]	すでに終了したプロデューサーを使用してメッセージを送信しました
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_PHND *prod;
    hms_msg_t *msg;
    ...
}
```

```

session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
if (session == NULL) { error processing }

/* トピックタイプのプロデューサーを作成する */
prod = hms_create_publisher(session, dest, "producer_1", 0);
if (prod == NULL) { error processing }
msg = hms_alloc(session, size);
if (msg == NULL) { error processing }
ret = hms_set_body(msg, buf, size);
if (ret < 0) { error processing }

/* メッセージをデスティネーションに送信する */
ret = hms_send(prod, msg, 0);
if (ret < 0) { error processing }
...
hms_free(msg);
hms_close_publisher(prod, 0);

/* セッションを終了する */
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_sendex()

3.5. コンシューマAPI

コンシューマAPIは、トピック、キュータイプのコンシューマを作成および終了するためのAPIで構成されています。

3.5.1. hms_create_consumer

デスティネーションからメッセージを受信するコンシューマを作成し、そのコンシューマのハンドルを返す関数です。メッセージ・セレクトを設定すると、条件に合うメッセージをフィルタリングして受信できます。

非同期セッションでは、プロデューサーからメッセージが送信された際に即時で該当メッセージを受信して処理するサービスを指定します。非同期セッションでコンシューマを作成する際は必ずsvcnameを指定します。反対に、非同期セッションではないセッションでコンシューマを作成する際はsvcnameを指定してはなりません。また、クライアントはメッセージ・セレクトを設定することで該当条件を満たすメッセージのみを受信できます。

メッセージ・セレクトについての詳細は、「[付録 A. メッセージ・セレクトの使用方法](#)」を参照してください。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
HMS_CHND * hms_create_consumer (HMS_SHND *sess, char *des, int destype,
                                char *name, char *msgselector, char *svcname, int flags)
```

- パラメータ

パラメータ	説明
sess	メッセージの受信に使用するセッション・ポインタです
des	メッセージを受信するデスティネーションの名前です
destype	デスティネーションのタイプです – MS_QUEUE – HMS_TOPIC
name	作成するコンシューマの名前です
msgselector	メッセージ・セレクトの構文です
svcname	非同期セッションでメッセージを受信するサービス名として、非同期セッションで作成する場合は必ず設定します
flags	現在は使用されていません

- 戻り値

戻り値	説明
コンシューマのハンドル	関数の呼び出しに成功しました
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_create_consumer()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessやdesがNULLの場合、desやname、msgselector、svcnameが指定された長さを超過する場合、非同期セッションでsvcnameが指定されていない場合、destypeに有効でない値が設定されている場合に発生します
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います

エラーコード	説明
[TPEPROTO]	関数が不適切な位置で呼び出されました。たとえば、非同期セッションですでに svcname を使用する他のコンシューマが作成されている場合、非同期セッションではないにもかかわらず svcname が設定されている場合、durable subscriber を作成しながらコンシューマが名前を指定しない場合に発生します
[TPESYSTEM]	HMS システム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPENOENT]	メッセージを受信するデスティネーションが存在しない場合や活性化されていないデスティネーションの場合に発生します
[TPEITYPE]	デスティネーションのタイプと作成するコンシューマのデスティネーションのタイプが一致しません
[TPEOS]	HMS または運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリ割り当てに失敗した場合です
[TPEMATCH]	引数として渡された コンシューマ名がすでに HMS に登録されています
[TPENOREADY]	HMS が NOT READY 状態です

● 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    /* キューQUEUEタイプのコンシューマ(Consumer)を作成する */
    cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                               NULL, 0);
    if (cons == NULL) { error processing }
    ...
    ret = hms_rcv(cons, &msg, 0);
    if (ret < 0) { error processing }
    ...
    hms_free(msg);

    /* コンシューマ(Consumer)を終了する */
    hms_close_consumer(cons, 0);
}
```

```

    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}

```

- 関連関数

hms_close_consumer()

3.5.2. hms_create_receiver

キュータイプのデスティネーションからのレシーバーを作成する関数で、動作方式はhms_create_consumer()と同一です。

- プロトタイプ

```

#include <usrinc/hmsapi.h>
HMS_CHND * hms_create_receiver (HMS_SHND *sess, char *des, char *name,
                                char *msgselector, char *svcname, int flags)

```

- パラメータ

パラメータ	説明
sess	メッセージの送信に使用するセッション・ポインタです
des	メッセージを送信するデスティネーションの名前です
name	作成するプロデューサーの名前です
msgselector	メッセージ・セレクトタの構文です
svcname	非同期セッションでメッセージを受信するサービス名として、非同期セッションで作成する場合は必ず設定します
flags	現在は使用されていません

- 戻り値

戻り値	説明
コンシューマのハンドラ	関数の呼び出しに成功しました
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_create_receiver()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessやdesがNULLの場合、desやname、msgselector、svcnameが指定された長さを超過した場合、非同期セッションでsvcnameが指定されていない場合に発生します
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPEPROTO]	非同期セッションですのにsvcnameを使用する他のコンシューマが作成されています
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPENOENT]	メッセージを受信するデスティネーションが存在しない場合や活性化されていないデスティネーションの場合に発生します
[TPEITYPE]	デスティネーションのタイプと作成するレシーバーのデスティネーションのタイプが一致しません
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリ割り当てに失敗しました
[TPEMATCH]	引数として渡されたレシーバー名がすでにHMSに登録されています
[TPENOREADY]	HMSがNOT READY状態です

● 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    /* キュータイプのレシーバーを作成する */
    cons = hms_create_receiver(session, dest, "consumer_1", NULL, NULL, 0);
    if (cons == NULL) { error processing }
    ...

    ret = hms_rcv(cons, &msg, 0);
    if (ret < 0) { error processing }
    ...
}
```

```

hms_free(msg);

/* レシーバーを終了する */
hms_close_receiver(cons, 0);
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_close_receiver()

3.5.3. hms_create_subscriber

トピックタイプのデスティネーションからのサブスクライバーを作成する関数で、動作方式は hms_create_consumer()と同じです。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
HMS_CHND * hms_create_subscriber (HMS_SHND *sess, char *des, char *name,
                                char *msgselector, char *svcname, int flags)

```

- パラメータ

パラメータ	説明
sess	メッセージ送信に使用するセッション・ポインタです
des	メッセージを送信するデスティネーションの名前です
name	作成するプロデューサーの名前です
msgselector	メッセージ・セレクトの構文です
svcname	非同期セッションでメッセージを受信するサービス名として、非同期セッションで作成する場合は必ず設定します
flags	現在は使用されていません

- 戻り値

戻り値	説明
コンシューマのハンドラ	関数の呼び出しに成功しました
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_create_subscriber()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessやdesがNULLの場合、desやname、msgselector、svcnameが指定された長さを超過した場合、非同期セッションでsvcnameが指定されていない場合に発生します
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPEPROTO]	非同期セッションですすでにsvcnameを使用するほかのコンシューマが作成されています
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPENOENT]	メッセージを受信するデスティネーションが存在しない場合や活性化していないデスティネーションの場合に発生します
[TPEITYPE]	デスティネーションのタイプと作成するコンシューマのデスティネーションのタイプが一致しません
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリ割り当てに失敗しました
[TPEMATCH]	引数として渡されたサブスクライバー名がすでにHMSに登録されています
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    /* トピックタイプのサブスクライバーを作成する */
    cons = hms_create_subscriber(session, dest, "consumer_1", NULL, NULL, 0);
    if (cons == NULL) { error processing }
    ...
    ret = hms_rcv(cons, &msg, 0);
```

```

    if (ret < 0) { error processing }
    ...
    hms_free(msg);

    /* サブスクライバーを終了する */
    hms_close_subscriber(cons, 0);
    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}

```

- 関連関数

hms_close_subscriber()

3.5.4. hms_create_durable_subscriber

トピック・タイプのデスティネーションから受信する恒久サブスクライバーを作成する関数です。

恒久サブスクライバーはメッセージ受信に対する信頼性を保証し、恒久サブスクライバーがメッセージを正常に受信するまでHMSは該当メッセージを維持します。HMSは恒久サブスクライバーを名前で区別するため、各恒久サブスクライバーは必ず一意の名前を使用します。

hms_close_durable_subscriberとは別途で登録を解除するアンサブスクライブのプロセスが必ず必要です。アンサブスクライブをしなければ、恒久サブスクライバーが閉じられた後にも次の該当サブスクライバーによってメッセージが受信されるまで情報を維持するため、サブスクライブを行う必要がなくなればアンサブスクライブをする必要があります。

- プロトタイプ

```

# include <usrinc/hmsapi.h>

HMS_CHND * hms_create_durable_subscriber (HMS_SHND *sess, char *des, char *name,
                                           char *msgselector, char *svcname, int flags)

```

- パラメータ

パラメータ	説明
sess	メッセージの送信に使用するセッション・ポインタです
des	メッセージを送信するデスティネーションの名前です
name	作成するプロデューサーの名前です
msgselector	メッセージ・セレクタの構文です
svcname	非同期セッションでメッセージを受信するサービス名として、非同期セッションで作成する場合は必ず設定します

パラメータ	説明
flags	現在は使用されていません

- 戻り値

戻り値	説明
コンシューマのハンドル	関数の呼び出しに成功しました
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_create_durable_subscriber()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessやdesがNULLの場合、desやname、msgselector、svcnameが指定された長さを超過する場合、または非同期セッションでsvcnameが指定されていない場合に発生します
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPEPROTO]	非同期セッションですすでにsvcnameを使用する他のコンシューマが作成されています
[TPESYSTEM]	HMSシステムエラーが発生しました。詳細情報はログファイルに記録されます
[TPENOENT]	メッセージを受信するデスティネーションが存在しない場合や活性化されていないデスティネーションの場合に発生します
[TPEITYPE]	デスティネーションのタイプと作成するサブスクライバーのデスティネーションのタイプが一致しません
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリ割り当てに失敗した場合です
[TPEMATCH]	引数として渡したdurable subscriber nameがすでにHMSに登録されています
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>
```

```

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    /* durable_subscriberを作成する */
    cons = hms_create_durable_subscriber(session, dest, "consumer_1", NULL,
                                         NULL, 0);

    if (cons == NULL) { error processing }
    ...
    ret = hms_recv(cons, &msg, 0);
    if (ret < 0) { error processing }
    ...
    hms_free(msg);

    /* durable_subscriberを終了する */
    ret = hms_unsubscribe_durable_subscriber(session, dest, "consumer_1", 0);
    if (ret < 0) { error processing }
    ret = hms_close_consumer(cons, 0);
    if (ret < 0) { error processing }

    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}

```

- 関連関数

hms_close_durable_subscriber()

3.5.5. hms_close_consumer

コンシューマを閉じ、割り当てられたリソースを返す関数です。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
int hms_close_consumer (HMS_CHND *cons, int flags)

```

- パラメータ

パラメータ	説明
cons	閉じるコンシューマ・ハンドルのポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_close_consumer()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。consがNULLです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPENOREADY]	HMSがNOT READY状態です

- 関連関数

hms_create_consumer()

3.5.6. hms_close_receiver

キュータイプのレシーバーを閉じ、割り当てられていたリソースを返す関数で、動作方式はhms_close_consumer()と同じです。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_close_receiver (HMS_CHND *cons, int flags)
```

- パラメータ

パラメータ	説明
cons	閉じるコンシューマ・ハンドルのポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_close_receiver()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。consがNULLです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPENOREADY]	HMSがNOT READY状態です

- 関連関数

hms_create_receiver()

3.5.7. hms_close_subscriber

トピックタイプのサブスクライバーを閉じ、割り当てられていたリソースを返す関数で、動作方式はhms_close_consumer()と同じです。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_close_subscriber (HMS_CHND *cons, int flags)
```

- パラメータ

パラメータ	説明
cons	閉じるコンシューマ・ハンドルのポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました

戻り値	説明
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_close_subscriber()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。consがNULLです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPENOREADY]	HMSがNOT READY状態です

- 関連関数

hms_create_subscriber()

3.5.8. hms_close_durable_subscriber

恒久サブスクライバーを閉じ、割り当てられていたリソースを返す関数で、動作方式はhms_close_consumer()と同じです。使用後にhms_unsubscribe_durable_subscriber()を使用してunsubscribeのプロセスが必要です。unsubscribeを行わない場合、該当恒久サブスクライバーが受信していないすべてのメッセージは該当サブスクライバーが受信するまで、またはunsubscribeを実行するまで、メモリーから削除されずに受信を待ちます。

- プロトタイプ

```
# include <usrinc/hmsapi.h>
int hms_close_durable_subscriber (HMS_CHND *cons, int flags)
```

- パラメータ

パラメータ	説明
cons	閉じるコンシューマ・ハンドルのポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました

戻り値	説明
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_close_durable_subscriber()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。consがNULLです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
#include <stdio.h>...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    session = hms_create_session(hms, 1, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    /* durable_subscriberを作成する */
    cons = hms_create_durable_subscriber(session, dest, "consumer_1", NULL,
                                         NULL, 0);

    if (cons == NULL) { error processing }
    ...
    ret = hms_recv(cons, &msg, 0);
    if (ret < 0) { error processing }
    ...
    hms_free(msg);
    /* durable_subscriberを終了する */
    ret = hms_close_consumer(cons, 0);
    if (ret < 0) { error processing }
```



```

...
ret = hms_unsubscribe_durable_subscriber(session, dest, "consumer_1", 0);
if (ret < 0) { error processing }
...
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

`hms_create_durable_subscriber()`

3.5.9. hms_recvex

デスティネーションからメッセージを受信する関数です。タイムアウトを設定でき、0に設定した場合はメッセージが到着するまで待機します。コンシューマを作成する際にメッセージ・セクタを設定した場合、その基準に合うメッセージのみを選択的に受信できます。メッセージを受信するためには、`hms_alloc()` APIを使用してメッセージ・バッファをあらかじめ作成します。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
int hms_recvex (HMS_CHND * cons, hms_msg_t **msg, long timeout, int flags)

```

- パラメータ

パラメータ	説明
cons	メッセージを受信するコンシューマ・ハンドルのポインタです
msg	<code>hms_alloc()</code> で割り当てられたメッセージ・バッファです
timeout	タイムアウトです(デフォルト値: 0、単位: 秒)
flags	まだ使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、`hms_recvex()`は失敗し、`tperrno`に以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。consやmsgがNULLであるか、timeout値が負数です
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPEOTYPE]	正しくないmsgが引数として渡されました
[TPEITYPE]	正しくないmsgを受信しました
[TPETIME]	タイムアウトが発生しました
[TPENOENT]	すでに終了しているコンシューマを使用してメッセージを送信しました。セッションが終了し、セッションで作成していたプロデューサーとコンシューマが終了していることがあります
[TPENOREADY]	HMSがNOT READY状態です

● 例

```

...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    /* 一般モードのセッションを作成する */
    session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }
    cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                               NULL, 0);
    if (cons == NULL) { error processing }
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }

    /* メッセージの受信 (timeoutを20秒に設定) */
    ret = hms_rcvex(cons, &msg, 20, 0);
    if (ret < 0) { error processing }
    ret = hms_ack(msg, HMS_ACK_ONE);
    if (ret < 0) { error processing }
    ...
    /* メッセージ・バッファを解除する */

```

```

hms_free(msg);

hms_close_consumer(cons, 0);
ret = hms_close_session(session, 0);
if (ret < 0) { error processing }
...
}

```

- 関連関数

hms_recv(), hms_create_consumer(), hms_create_receiver(), hms_create_subscriber(),
hms_create_durable_subscriber(), hms_ack(), hms_commit(), hms_rollback()

3.5.10. hms_recv

デスティネーションからのメッセージが到着するまで待機する関数で、動作方式はhms_recvex()と同一です。
(timeout = 0)

- プロトタイプ

```

# include <usrinc/hmsapi.h>
int hms_recvex (HMS_CHND * cons, hms_msg_t **msg, int flags)

```

- パラメータ

パラメータ	説明
cons	メッセージを受信するコンシューマ・ハンドルのポインタです
msg	hms_alloc()で割り当てられたメッセージ・バッファです
flags	まだ使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_recv()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。consあるいはmsgがNULLです

エラーコード	説明
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を現在のセッションを使用して終了し、新しいセッションで作成して再度行います
[TPEOTYPE]	正しくないmsgが引数として渡されました
[TPEITYPE]	正しくないmsgを受信しました
[TPETIME]	タイムアウトが発生しました
[TPENOENT]	すでに終了しているコンシューマを使用してメッセージを送信しました。セッションが終了し、セッションで作成していたプロデューサーとコンシューマが終了していることがあります
[TPENOREADY]	HMSがNOT READY状態です

● 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_CHND *cons;
    hms_msg_t *msg;
    ...
    /* 一般モードのセッションを作成する */
    session = hms_create_session(hms, 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }
    cons = hms_create_consumer(session, dest, HMS_QUEUE, "consumer_1", NULL,
                               NULL, 0);
    if (cons == NULL) { error processing }
    msg = hms_alloc(session, size);
    if (msg == NULL) { error processing }

    /* メッセージの受信 */
    ret = hms_recv(cons, &msg, 0);
    if (ret < 0) { error processing }
    ret = hms_ack(msg, HMS_ACK_ONE);
    if (ret < 0) { error processing }
    ...
    /* メッセージ・バッファを解除する */
    hms_free(msg);
    hms_close_consumer(cons, 0);
}
```

```

    ret = hms_close_session(session, 0);
    if (ret < 0) { error processing }
    ...
}

```

- 関連関数

hms_recv(), hms_create_consumer(), hms_create_receiver(), hms_create_subscriber(),
hms_create_durable_subscriber(), hms_ack(), hms_commit(), hms_rollback()

3.6. キュー・ブラウザーAPI

キュー・ブラウザーAPIは、キュー・ブラウザーの作成と終了、メッセージの照会などのためのAPIで構成されています。

3.6.1. hms_create_browser

現在デスティネーションに存在するメッセージを照会できるキュー・ブラウザーを作成し、該当ブラウザーのハンドルを返す関数です。キュータイプのデスティネーションに対してのみ作成が可能です。ブラウザーを作成する際にメッセージ・セレクトを設定し、条件に合うメッセージのみを照会できます。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
HMS_BHND * hms_create_browser (HMS_SHND *sess, char *queue, char *msgselector,

                                int flags)

```

- パラメータ

パラメータ	説明
sess	ブラウザーを使用するセッション・ポインタです
queue	照会するキューの名前です
msgselector	メッセージ・セレクトの構文です
flags	以下のいずれかを設定します <ul style="list-style-type: none"> – HMS_QB_FIRST: デスティネーションにある受信可能なメッセージを送信された順番どおりに照会します – 0: デスティネーションにあるメッセージのうち、最後に送信されたメッセージから照会します(デフォルト値)

- 戻り値

戻り値	説明
ブラウザのハンドル・ポインタ	関数の呼び出しに成功しました
NULL	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_create_browser()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。sessやqueueがNULLの場合、queueやmsgselectorが指定された長さを超過した場合に発生します
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPENOENT]	メッセージを受信するデスティネーションが存在しない場合や活性化していないデスティネーションの場合に発生します
[TPEITYPE]	queueに渡したデスティネーションがキュータイプではありません
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリ割り当てに失敗した場合です
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_BHND *browser;
    hms_msg_t *msg;
    char buf[100];
    long len = 100;
    ...
    session = hms_create_session("svghms", 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }
```

```

/* キュー・ブラウザーの作成 */
browser = hms_create_browser(session, "queue1",
                             "HMSType = 'cat' AND weight < 3"), 0);
if (brow == NULL) { error processing }
...
msg = hms_alloc(session, 1024);
if (msg == NULL) { error processing }
buf = tpalloc("STRING", NULL, 0);
if (buf == NULL) { error processing }

/* キュー・ブラウザーを使用してメッセージ・データを取得する */
hms_browser_nextmsg(browser, &msg, 0);
if (ret == -1) { error processing }
ret = hms_get_body(msg, buf, strlen(buf));
if (ret == -1)
    { error processing }
else
    printf("message : %s\n", buf);
...

/* キュー・ブラウザーの終了 */
ret = hms_close_browser(browser, 0);
if (ret == -1) { error processing }
ret = hms_close_session(session, 0);
if (ret == -1) { error processing }
...
}

```

- 関連関数

hms_close_browser()

3.6.2. hms_close_browser

キュー・ブラウザーを閉じ、割り当てられていたリソースを返す関数です。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
int hms_close_browser (HMS_BHND *browser, int flags)

```

- パラメータ

パラメータ	説明
browser	ブラウザー・ハンドルのポインタです

パラメータ	説明
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	現在の呼び出しに成功しました
-1	現在の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_close_browser()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。ブラウザーがNULLです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで作成して再度行います
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_BHND *browser;
    hms_msg_t *msg;
    char buf[100];
    long len = 100;
    ...

    session = hms_create_session("svghms", 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }
    browser = hms_create_browser(session, "queue1",
                                "HMSType = 'cat' AND weight < 3"), 0);
    if (brow == NULL) { error processing }
    ...
    hms_browser_nextmsg(browser, &msg, 0);
}
```



```

    if (ret == -1) { error processing }
    ...

    /* キュー・ブラウザの終了 */
    ret = hms_close_browser(browser, 0);
    if (ret == -1) { error processing }
    ret = hms_close_session(session, 0);
    if (ret == -1) { error processing }
    ...
}

```

- 関連関数

hms_create_browser()

3.6.3. hms_browser_nextmsg

最後に届いたメッセージから順番にメッセージを照会するための関数です。内部的な動作はhms_recv()と似ていますが、照会のみ可能で、メッセージはキューで消費されないという点が異なります。

hms_create_browser()で作成する際に設定したflags値に従い、HMS_QB_FIRSTの場合はデスティネーションに送信された順に照会され、デフォルト値の場合は最後に届いたメッセージ(キューの最後)から順番にメッセージを照会します。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
int hms_browser_nextmsg (HMS_BHND *browser, hms_msg_t **msg, int flags)

```

- パラメータ

パラメータ	説明
browser	ブラウザ・ハンドルのポインタです
msg	メッセージ・バッファのポインタです
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_browser_nextmsg()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。browserあるいはmsgがNULLです
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます
[TPECLOSE]	HMSとのセッション接続が終了しました。hms_close_session()を使用して現在のセッションを終了し、新しいセッションで再度行います
[TPEOS]	HMSまたは運用システムにエラーが発生しました。たとえば、内部的に使用するバッファのメモリ割り当てに失敗した場合です
[TPEMATCH]	該当デスティネーションに照会可能なメッセージが存在しません
[TPENOREADY]	HMSがNOT READY状態です

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_BHND *browser;
    hms_msg_t *msg;
    char buf[100];
    long len = 100;
    ...
    session = hms_create_session("svghms", 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }
    browser = hms_create_browser(session, "queue1",
                                "HMSType = 'cat' AND weight < 3"), 0);
    if (brow == NULL) { error processing }
    ...
    msg = hms_alloc(session, 1024);
    if (msg == NULL) { error processing }
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL) { error processing }

    /* キュー・ブラウザーを通じてメッセージ・データを取得する */
    hms_browser_nextmsg(browser, &msg, 0);
    if (ret == -1) { error processing }

    ret = hms_get_body(msg, buf, strlen(buf));
    if (ret == -1)
```

```

        { error processing }
    else
        printf("message : %s\n", buf);
    ...
    ret = hms_close_browser(browser, 0);
    if (ret == -1) { error processing }
    ret = hms_close_session(session, 0);
    if (ret == -1) { error processing }
    ...
}

```

- 関連関数

hms_create_browser(), hms_browser_get_queue(), hms_browser_get_msgselector()

3.6.4. hms_browser_get_msgselector

ブラウザーを作成する際に設定していたメッセージ・セクタの構文を取得するための関数です。

ブラウザーを作成する際に設定していたメッセージ・セクタの構文をバッファにコピーし、その長さをlenに設定します。バッファはメッセージ・セクタの構文の長さを受用できるだけの十分なサイズである必要があり、APIを呼び出す前にバッファのサイズをlenに設定する必要があります。コピーするバッファのサイズがメッセージ・セクタの構文の長さより小さい場合や、メッセージ・セクタが設定されていない場合は-1を返します。

- プロトタイプ

```

# include <usrinc/hmsapi.h>
int hms_browser_get_msgselector(HMS_BHND *browser, char *buf, long *len,
                                int flags)

```

- パラメータ

パラメータ	説明
browser	ブラウザー・ハンドルのポインタです
buf	メッセージ・セクタの構文をコピーするバッファです
len	バッファのサイズの入力により、メッセージ・セクタの構文の長さを返します
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました

戻り値	説明
-1	関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

● エラー

以下の状況下で、hms_browser_get_msgselector()は失敗し、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。browserやbufがNULLである場合やlen(バッファのサイズ)がメッセージ・セクタの構文の長さより小さい場合に発生します
[TPENOENT]	メッセージ・セクタが設定されていません

● 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_BHND *browser;
    hms_msg_t *msg;
    char buf[100];
    long len = 100;
    ...

    session = hms_create_session("svghms", 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }

    browser = hms_create_browser(session, "queue1",
                                "HMSType = 'cat' AND weight < 3"), 0);
    if (brow == NULL) { error processing }
    ...

    /* メッセージ・セクタの構文の取得 */
    ret = hms_browser_get_msgselector(browser, buf, &len);
    if (ret == -1)
        { error processing }
    else
        printf("Message Selector : %s\n", buf);
    ...
}
```

```

    ret = hms_close_browser(browser, 0);
    if (ret == -1) { error processing }
    ret = hms_close_session(session, 0);
    if (ret == -1) { error processing }
    ...
}

```

- 関連関数

hms_create_browser(), hms_browser_nextmsg(), hms_browser_get_queue()

3.6.5. hms_browser_get_queue

ブラウザを作成する際に設定していたデスティネーションの名前をバッファにコピーし、その長さをlenに設定する関数です。バッファはデスティネーションの名前を受用できるだけの十分なサイズである必要があり、APIを呼び出す前にバッファのサイズをlenに設定する必要があります。

- プロトタイプ

```

#include <usrinc/hmsapi.h>
int hms_browser_get_queue (HMS_BHND * browser, char *buf, long *len, int flags)

```

- パラメータ

パラメータ	説明
browser	ブラウザ・ハンドルのポインタです
buf	デスティネーションの名前をコピーするバッファです
len	バッファのサイズの入力により、デスティネーションの名前の長さを返します
flags	現在は使用されていません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功しました
-1	コピーするバッファのサイズがデスティネーションの名前より小さい場合であり、関数の呼び出しに失敗しました。tperrnoにエラーコードが設定されます

- エラー

以下の状況下で、hms_browser_get_queue()は失敗し、tperrnoに以下のいずれかの値が設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではありません。 browserやbufがNULLである場合やlen(バッファのサイズ)がデスティネーションの名前より小さい場合に発生します
[TPESYSTEM]	HMSシステム・エラーが発生しました。詳細情報はログファイルに記録されます

- 例

```

...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

main(int argc, char* argv[])
{
    HMS_SHND *session;
    HMS_BHND *browser;
    hms_msg_t *msg;
    char buf[100];
    long len = 100;
    ...
    session = hms_create_session("svghms", 0, HMS_AUTO_ACK, 0);
    if (session == NULL) { error processing }
    browser = hms_create_browser(session, "queue1",
                                "HMSType = 'cat' AND weight < 3"), 0);
    if (brow == NULL) { error processing }
    ...
    /* デスティネーション名の取得 */
    ret = hms_browser_get_queue(browser, buf, &len, 0);
    if (ret == -1) { error processing }
    else
        printf("Destination : %s\n", buf);
    ...
    ret = hms_close_browser(browser, 0);
    if (ret == -1) { error processing }
    ret = hms_close_session(session, 0);
    if (ret == -1) { error processing }
    ...
}

```

- 関連関数

hms_create_browser(), hms_browser_nextmsg(), hms_browser_get_msgselector()

第4章 HMS管理

本章では、HMSを使用するためにビルドする手順と、起動と終了、および状態照会について説明します。

4.1. HMSのビルド

HMSは信頼性を提供することを目的として、ストレージを使用して一部の重要な情報を管理します。このような機能をサポートするために、HMSはストレージ機能が実装されたライブラリーを使用します。

基本的にサポートされるストレージ以外に、ユーザーの運用環境に設置された外部ストレージも拡張して使用できるようにサポートされます。この場合は、使用するストレージの種類によってサポートされるライブラリーが異なるため、該当ストレージのライブラリーを使用してHMSを再度ビルドする必要があります。

4.1.1. 事前準備

基本ストレージではない外部ストレージを使用する場合、HMSを該当ストレージ・ライブラリーに合わせて新しくビルドする必要があります。HMSをビルドするためには、運用システムにストレージが設定されており、該当ストレージをサポートするライブラリーが必要です。また、運用システムに設置されたストレージを正しく使用するための環境変数の設定を確認します。

たとえば、UNIXシステムで外部ストレージをOracleにして使用する場合、以下の環境変数値を設定します。あるいは、正しく設定されているかを確認します。

```
ORACLE_HOME=/data/home/oracle
PATH=$PATH:$ORACLE_HOME/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
```

HMSが使用するストレージのライブラリー以外に、Tmaxに各ストレージの種類に合わせて提供するHMSライブラリーも準備されている必要があります。HMSライブラリーは、基本ストレージを使用する場合はlibhms.aの名前で提供され、外部ストレージを使用する場合はlibhms_<STORAGE>.aの形式の名前で提供されます。

たとえば、外部ストレージでOracleを使用する場合、HMSライブラリーは、libhms_<storage>.aのような形式の名前で提供されます。

参考

使用する外部ストレージのHMSライブラリーが存在しない場合、Tmax製品の担当者にお問い合わせください。

4.1.2. Makefile

外部ストレージを使用してHMSを新しくビルドする場合、Makefileを使用します。Makefileはユーザーのプラットフォームおよび開発環境に合わせて修正して使用します。

以下は、Linux環境で外部ストレージとしてOracleを使用するためのHMS Makefileの例です。

```
# HMS Makefile for Oracle
# Linux
TARGET    = hms_ora
APOBJ     = dummy.o
APPDIR    = $(TMAXDIR)/appbin
TMAXLIBD= $(TMAXDIR)/lib
TMAXLIBS= -lhms_ora -lpthread

CFLAGS    =

all: $(TARGET)

include $(ORACLE_HOME)/precomp/lib/env_precomp.mk
ORALIBDIR = $(LIBHOME)
ORALIB    = $(PRODLIBS)

$(TARGET): $(APOBJ)
    $(CC) $(CFLAGS) -o $(TARGET) -L$(TMAXLIBD) -L$(ORALIBDIR) $(ORALIB) $(APOBJ)

    $(TMAXLIBS)
    mv $(TARGET) $(APPDIR)/.

$(APOBJ):
    $(CC) $(CFLAGS) -c dummy.c

clean:
    -rm -f core $(TARGET) $(APPDIR)/$(TARGET)

~

~
```

4.1.3. HMSビルドおよび確認

HMS Makefileを使用する場合、ビルドされたHMSは\$TMAXDIRの下のappbinディレクトリーに移動されます。appbinディレクトリーに移動されなかった場合、ビルドされたHMSをappbinにコピーします。

HMSのファイル名はユーザーが指定できます。指定したHMSの名前はTmax環境設定ファイルのSVRGROUPセクションのHMSNAME項目についての値となります。

4.2. HMSの起動および終了

HMSは、Tmaxが運用中の状況で動作します。Tmaxが終了している場合、Tmaxをまず起動させます。オプションなしでtmbootコマンドを通じて起動させた場合、HMSも一緒に起動されますが、Tmaxシステム・プロセスのみ起動している場合、HMSは別途起動させる必要があります。

以下は、指定したHMSを起動および終了するコマンドです。

- 起動

```
tmboot -q <SVGNAME>
```

- 終了

```
tmdown -q <SVGNAME>
```

起動と終了コマンドに使われる<SVGNAME>は、Tmax環境設定ファイルに定義したSVRGROUPセクションの名前です。詳しい内容については、「[2.1.3. SVRGROUPセッション](#)」を参照してください。

4.3. HMSの状態照会

HMSが運用中の状況で現在の環境設定についての情報を確認し、システムの状態、HMSシステムに接続されているクライアントの情報、送信されたメッセージまたは処理されたメッセージ、失敗したメッセージなどの現状情報を管理する必要があります。ローカルおよびグローバルHMSの情報を管理するために、Tmaxで提供するtmadminプログラムを使用します。

参考

tmadminについての詳細情報は、『Tmax 運用ガイド』を参照してください。

4.3.1. デスティネーション情報の照会

tmadminのstコマンドを使用して環境設定ファイルで設定したデスティネーションの一覧と各デスティネーションで処理中のメッセージとクライアントの情報を確認できます。

```
tmadmin st -q
```

以下は、コマンドを実行した際に照会されるローカルHMS環境についての情報です。

```
$$2 Locke2 (tmadm): st -q
-----
G  dest          cqcount   type      apqcnt    acqcnt    f_dscrd   t_dscrd   cons_cnt  prod_cnt
```

-	queue01	58	QUEUE	169	111	0	0	30
5								
-	topic01	32	TOPIC	42	283	0	0	28
3								

以下は、照会される情報の各フィールドについての説明です。

フィールド	説明
G	該当デスティネーションがGLOBALに設定されている場合は「O」と表示され、GLOBALではない場合はハイフン(-)で表示されます
dest	デスティネーションの名前で、環境設定ファイルでHMSセクションに設定された名前が表示されます
cqcount	該当デスティネーションでまだ処理されていないメッセージ件数です
type	該当デスティネーションのタイプを表示します。QUEUEとTOPICで表示されます
apqcnt	現在までに送信されて累積された総メッセージ件数です
acqcnt	コンシューマによって処理されたメッセージの総件数で、トピック・タイプは1つのメッセージに対してすべてのコンシューマが受信しなければ処理が完了しないため、「 apqcnt 」の値より「 acqcnt 」の値が大きくなる場合があります
f_dscrd	受信したメッセージに対してfailed処理を行うことで受信が失敗したメッセージの件数です
t_dscrd	メッセージの送信にTTLが設定された場合、有効時間の経過によって失敗したメッセージの件数です
cons_cnt	該当デスティネーションに接続したコンシューマの数です
prod_cnt	該当デスティネーションに接続したプロデューサーの数です

4.3.2. クライアント情報の照会

tmadminのstコマンドを使用して、特定デスティネーションに対する各クライアントの詳細情報を照会できます。

```
tmadmin st -q <DestinationName> -c
```

以下は、コマンドを実行する際に照会されるクライアント情報です。

```
$3 Locke2 (tmadm): st -q queue01 -c
```

```
-----
      sesi      clid      cname      clitype  qcnt   cnt   f_dscrd  t_dscrd   listener
```

0	0x39d	prodasync	ARCV	49	0	0	0	ASYNCSVC
0x1	0	prod41	SND	0	32	0	0	
0x1	0	prod42	SND	0	11	0	0	
0x1	0	cons51	RCV	3	0	0	0	
0x1	0	cons52	RCV	9	0	0	0	
<pre> \$\$\$ Locke2 (tmadm): st -q topic01 -c </pre>								
sesi	clid	cname	clitype	qcnt	cnt	f_dscrd	t_dscrd	listener
0x1	0	prod11	PUB	0	2	0	0	
0x1	0	prod12	PUB	0	1	0	0	
0x1	0	cons11	SUB	30	0	0	0	
0x2	0x2	prod21	PUB	0	16	0	0	
0x2	0x2	prod22	PUB	0	8	0	0	
0x2	0x2	cons21	SUB	23	0	0	0	
0x3	0	prod31	PUB	0	3	0	0	
0x4	0x4	cons41	ADSUB	32	0	0	0	ASYNCSVC2

以下は、照会される情報の各フィールドについての説明です。

フィールド	説明
sesi	該当クライアントを作成したセッションの番号です
clid	クライアントIDです
cname	クライアントを作成する際に指定したクライアント名として、以下の関数のパラメータとして使用されます – hms_create_consumer, hms_create_sender, hms_create_subscriber, hms_create_producer, hms_create_receiver, hms_create_publisher
clitype	該当するデスティネーションに接続されたクライアントのタイプです – SND :キュータイプのセNDER(hms_create_sender()で作成) – PUB :トピックタイプのパブリッシャー(hms_create_publisher()で作成) – RCV :キュータイプのレシーバー(hms_create_receiver()で作成) – SUB :トピックタイプのサブスクライバー(hms_create_subscriber()で作成)

フィールド	説明
	<ul style="list-style-type: none"> – DSUB : トピックタイプの恒久サブスクライバー(hms_create_durable_subscriber())で作成) – ARCV : 非同期セッションで作成されたキュータイプのセnder – ASUB : 非同期セッションで作成されたトピックタイプのサブスクライバー – ADSUB : 非同期セッションで作成されたトピックタイプの恒久サブスクライバー
qcnt	各サブスクライバーが実際に持っているメッセージの件数です
cnt	各クライアントが送信あるいは受信したメッセージの件数です
f_dscrd	受信後、メッセージをfailedと処理することで失敗したメッセージの件数です
t_dscrd	送信する際に設定したTTL(有効時間)が経過して失敗処理されたメッセージの件数です
listener	非同期セッションで作成したコンシューマの場合、メッセージを受信するサービス名です

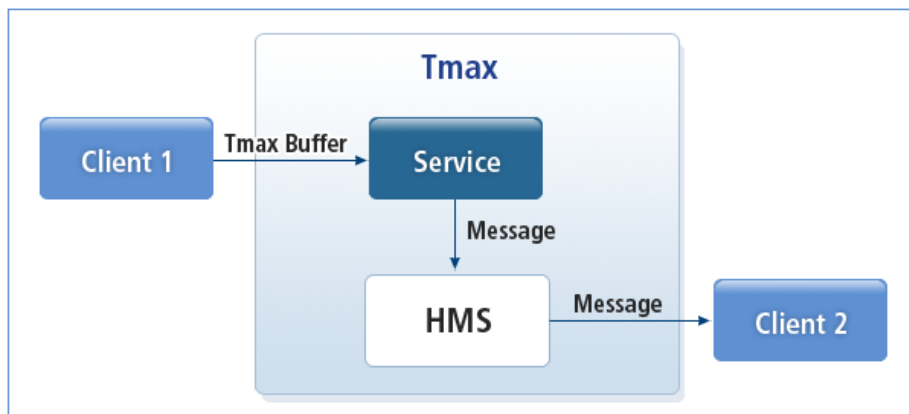
第5章 例

本章では、アプリケーションの例を通じてAPIの基本的な使用方法とフローについて説明します。

5.1. メッセージ送信プログラム

以下は、クライアントとサーバーでHMS APIを使用してメッセージを送受信する簡単なプログラムのフローです。

[図 5.1] メッセージ送信プログラムのフロー



1. クライアント(Client1)は、stringタイプのバッファーに入力された文字列をコピーしてサービス呼び出します。
2. サーバーのサービス・ルーチンではこの文字列を受け取って小文字を大文字に変更した後、HMSにメッセージを送信します。
3. クライアント(Client2)は、一定の時間が過ぎた後にHMSからメッセージを受信します。

5.1.1. HMSの環境設定

以下は、HMSの環境設定ファイルの例です。

```
*DOMAIN
hms      SHMKEY = 74347,
         TPORTNO = 8808

*NODE
```

```

Locke2  TMAXDIR = "/home/tmax5/tmax",
        APPDIR = "/home/tmax5/tmax/appbin/",
        MAXSESSION = 100

*SVRGROUP
hms01   NODENAME = "Locke2",  CPC = 1, SVGTYPE = "HMS", RESTART = Y,
        OPENINFO = "ORACLE_XA+Acc=P/scott/tiger+SesTm=60",
        HMSINDEX = 2, HMSMSGLIVE = 1, HMSMAXTHR = 2, HMSMAXDBTHR = 5,
        HMSNAME = hms_ora
svg1    NODENAME = "Locke2"

*HMS
queue01 SVGNAME = hms01, BOOT = "WARM", TYPE = "QUEUE"
topic01 SVGNAME = hms01, BOOT = "WARM", TYPE = "TOPIC"

*SERVER
svr      SVGNAME = svg1

*SERVICE
SVC      SVRNAME = svr

```

5.1.2. クライアント・プログラム

以下は、クライアント・プログラムの例です。

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

int main(int argc, char **argv)
{
    char *sndbuf, *rcvbuf;
    long rcvlen, sndlen;
    HMS_SHND *sess;
    HMS_CHND *cons;
    hms_msg_t *msg;

    if (argc != 2) {
        printf("Usage : %s <message>\n\n", argv[0]);
        exit(1);
    }
}

```

```

if (tmaxreadenv("tmax.env", "TMAX") == -1) {
    printf("error: tmaxreadenv() failed - %d\n", tperrno);
    exit(1);
}

if (tpstart((TPSTART_T *) NULL) == -1) {
    printf("error: tpstart() fail - %d\n", tperrno);
    exit(1);
}

if ((sndbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
    printf("error: sndbuf alloc failed !\n");
    tpend();
    exit(1);
}

if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
    printf("error: rcvbuf alloc failed !\n");
    tpfree((char *)sndbuf);
    tpend();
    exit(1);
}

strcpy(sndbuf, argv[1]);

if(tpcall("SVC", sndbuf, 0, &rcvbuf, &rcvlen, 0)==-1){
    printf("error: Can't send request to service SVC\n");
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
    exit(1);
}

sleep(5);

/* RECV MESSAGE FROM HMS */
if ((sess = hms_create_session("hms01", 0, HMS_AUTO_ACK, 0)) == NULL) {
    printf("error: hms_create_session() failed tperrno = %d\n", tperrno);
    tpend();
    exit(1);
}

if ((cons = hms_create_receiver(sess, "queue01", "cons01", NULL, NULL, 0))
    == NULL) {
    printf("error: hms_create_receiver() failed tperrno = %d\n", tperrno);
    tpend();
}

```

```

        exit(1);
    }

    /* ALLOCATION */
    if ((msg = hms_alloc(sess, 1024)) == NULL) {
        printf("error: hms_alloc() failed tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    /* RECV MESSAGE */
    if (hms_recvex(cons, &msg, 5, 0) == -1) {
        printf("error: hms_recvex() failed tperrno = %d\n", tperrno);
        hms_free(msg);
        tpend();
        exit(1);
    }

    /* GET BODY */
    rcvlen = 1024;
    if (hms_get_body(msg, rcvbuf, &rcvlen) == -1) {
        printf("error: hms_get_body() failed tperrno = %d\n", tperrno);
        hms_free(msg);
        tpend();
        exit(1);
    }

    printf("HMS MESSAGE : %s\n", rcvbuf);

    /* CLOSE RECEIVER */
    if (hms_close_receiver(cons, 0) == -1) {
        printf("error: hms_close_receiver() failed tperrno = %d\n", tperrno);
        hms_free(msg);
        tpend();
        exit(1);
    }

    /* CLOSE SESSION */
    if (hms_close_session(sess, 0) == -1) {
        printf("error: hms_close_session() failed tperrno = %d\n", tperrno);
        hms_free(msg);
        tpend();
        exit(1);
    }

    hms_free(msg);
    tpend();

```



```
    return 0;
}
```

5.1.3. サーバー・プログラム

以下は、サーバー・プログラムの例です。

<SVR.C>

```
#include <stdio.h>
#include <stdlib.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

HMS_SHND *sess = NULL;
HMS_CHND *prod = NULL;

int tpsvrinit(int argc, char **argv)
{
    while(1) {
        sess = hms_create_session("hms01", 0, HMS_AUTO_ACK, 0);
        if (sess != NULL) {
            break;
        }
        if (tperrno != TPENOREADY) {
            printf("hms_create_session(hms01) : FAIL tperrno = %d\n", tperrno);
            return -1;
        }
    }
    prod = hms_create_sender(sess, "queue01", "prod_svc", 0);
    if (prod == NULL) {
        printf("hms_create_sender() : FAIL tperrno = %d\n", tperrno);
        return -1;
    }
    return 1;
}

int tpsvrdone()
{
    hms_close_sender(prod, 0);
    hms_close_session(sess, 0);
    return 1;
}

SVC(TPSVCINFO *msg)
```

```

{
    int n, i;
    hms_msg_t *hmsmsg = NULL;
    char *data = msg->data;
    int len = msg->len, asize;
    printf("SVC STARTED!\n");

    /* TOUPPER */
    for (i = 0; i < len; i++)
        data[i] = toupper(data[i]);

    /* ALLOCATION */
    asize = len + 1024;
    hmsmsg = hms_alloc(sess, asize);
    if (hmsmsg == NULL) {
        printf("hms_alloc : fail tperrno = %d\n", tperrno);
        tpreturn(TPFAIL, 0, NULL, 0, 0);
    }

    /* SET BODY */
    n = hms_set_body(hmsmsg, data, len);
    if (n < 0) {
        hms_free(hmsmsg);
        printf("hms_set_body : fail tperrno = %d\n", tperrno);
        tpreturn(TPFAIL, 0, NULL, 0, 0);
    }

    /* SEND : hms01, persistent */
    n = hms_sendex(prod, hmsmsg, HMS_DLV_PERSISTENT, 0, 0, 0);
    if (n < 0) {
        hms_free(hmsmsg);
        printf("hms_sendex(prod) : fail tperrno = %d\n", tperrno);
        tpreturn(TPFAIL, 0, NULL, 0, 0);
    }

    /* FREE */
    hms_free(hmsmsg);

    printf("SVC SUCCESS!\n");
    tpreturn(TPSUCCESS, 0, NULL, 0, 0);
}

```

5.1.4. プログラムのコンパイル

クライアントとサーバー・プログラムは、Tmaxアプリケーションをコンパイルする手順と同様にコンパイルします。

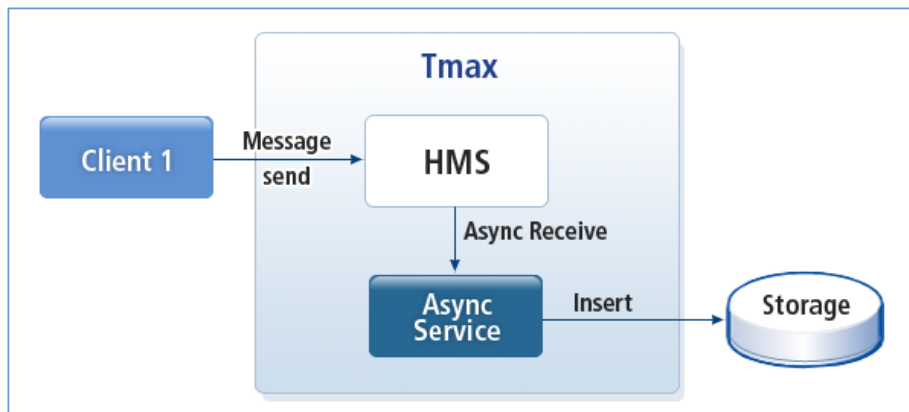
参考

プログラムのコンパイルについての詳細は、『Tmax アプリケーション開発ガイド』を参照してください。

5.2. メッセージ保存プログラム

以下は、サーバーとクライアント・プログラムでHMSから転送されたメッセージをデータベースに格納するプログラムのフローです。

[図 5.2] メッセージ保存プログラムのフロー



1. クライアントは、ユーザーの入力を受け、HMSへキュータイプのメッセージを送信します。
2. サーバー側では非同期セッションを通じてコンシューマを作成し、メッセージが転送された場合はASYNCサービスがこのメッセージを受信します。
3. ASYNCサービスは、受信したメッセージをデータベースに挿入します。

5.2.1. HMSの環境設定

以下は、HMSの環境設定についての例です。

```
*DOMAIN
hms      SHMKEY = 74347,
         TPORTNO = 8808

*NODE
Locke2   TMAXDIR = "/home/tmax5/tmax",
         APPDIR  = "/home/tmax5/tmax/appbin/",
         MAXSESSION = 100
```

```

*SVRGROUP
hms01  NODENAME = "Locke2",  CPC = 1, SVGTYPE = "HMS", RESTART = Y,
      OPENINFO = "ORACLE_XA+Acc=P/scott/tiger+SesTm=60",
      HMSINDEX = 2, HMSMSGLIVE = 1, HMSMAXTHR = 2, HMSMAXDBTHR = 5,
      HMSNAME = hms_ora
svg1   NODENAME = "Locke2", RESTART = N,
      OPENINFO = "ORACLE_XA+Acc=P/scott/tiger+SesTm=60",
      DBNAME = "ORACLE", TMSNAME = tms_ora, MINTMS = 1

*HMS
queue01 SVGNAME = hms01, BOOT = "WARM", TYPE = "QUEUE"
topic01 SVGNAME = hms01, BOOT = "WARM", TYPE = "TOPIC"

*SERVER
async  SVGNAME = svg1, CLOPT = "-- -i"

*SERVICE
ASYNCSVC  SVRNAME = async

```

5.2.2. クライアント・プログラム

以下は、クライアント・プログラムの例です。

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

int main(int argc, char **argv)
{
    long len;
    HMS_SHND *sess;
    HMS_PHND *prod;
    hms_msg_t *msg;
    char *data;
    int no;

    if (argc != 3) {
        printf("Usage : %s <no> <message>\n\n", argv[0]);
        exit(1);
    }

    if (tmaxreadenv("tmax.env", "TMAX") == -1) {

```

```

        printf("error: tmaxreadenv() failed - %d\n", tperrno);
        exit(1);
    }

    if (tpstart((TPSTART_T *) NULL) == -1) {
        printf("error: tpstart() fail - %d\n", tperrno);
        exit(1);
    }

    len = strlen(argv[2]);
    data = argv[2];
    no = atoi(argv[1]);

    /* SEND MESSAGE TO HMS */
    if ((sess = hms_create_session("hms01", 0, HMS_AUTO_ACK, 0)) == NULL) {
        printf("error: hms_create_session() failed tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    if ((prod = hms_create_sender(sess, "queue01", "prod01", 0)) == NULL) {
        printf("error: hms_create_sender() failed tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    /* ALLOCATION */
    if ((msg = hms_alloc(sess, len + 1024)) == NULL) {
        printf("error: hms_alloc() failed tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    /* SET BODY */
    if (hms_set_body(msg, data, len) == -1) {
        printf("error: hms_set_body() failed tperrno = %d\n", tperrno);
        hms_free(msg);
        tpend();
        exit(1);
    }

    /* SET PROPERTY */
    if (hms_set_property(msg, "NO", HMS_INT, (char *)&no, sizeof(int)) == -1) {
        printf("error: hms_set_property() failed tperrno = %d\n", tperrno);
        hms_free(msg);
        tpend();
        exit(1);
    }

```

```

    }

    /* SEND MESSAGE */
    if (hms_sendex(prod, msg, HMS_DLV_PERSISTENT, 0, 0, 0) == -1) {
        printf("error: hms_sendex() failed tperrno = %d\n", tperrno);
        hms_free(msg);
        tpend();
        exit(1);
    }

    if (hms_close_sender(prod, 0) == -1) {
        printf("error: hms_close_sender() failed tperrno = %d\n", tperrno);
        hms_free(msg);
        tpend();
        exit(1);
    }

    if (hms_close_session(sess, 0) == -1) {
        printf("error: hms_close_session() failed tperrno = %d\n", tperrno);
        hms_free(msg);
        tpend();
        exit(1);
    }

    hms_free(msg);
    tpend();

    return 0;
}

```

5.2.3. サーバー・プログラム

以下は、サーバー・プログラムの例です。

<async.pc>

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/hmsapi.h>

HMS_SHND *sess;
HMS_CHND *cons;
int svrinit_start = 0;

```

```

void abend_callback(HMS_SHND *session)
{
    printf("START ABEND_CALLBACK FUNCTION\n");
    hms_close_session(session, 0);
    printf("hms_close_session success\n");
    while (1) {
        sess = (HMS_SHND *) hms_create_async_session("hms01", abend_callback, 0);

        if (sess == NULL) {
            if (tperrno == TPENOREADY) {
                usleep(500000);
                continue;
            }
            printf("hms_create_session() : FAIL [%d]\n\n", tperrno);
            return;
        }
        break;
    }
    cons = (HMS_CHND *) hms_create_receiver(sess, "queue01", "consasync", NULL,
        "ASYNCSVC", 0);
    if (cons == NULL) {
        printf("hms_create_receiver() : FAIL tperrno = %d\n\n", tperrno);
        return;
    }
    printf("END ABEND_CALLBACK FUNCTION\n");
}

int tpsvrinit(int argc, char **argv)
{
    int c;
    while ((c = getopt(argc, argv, "i")) != EOF) {
        switch (c) {
            case 'i':
                svrinit_start = 1;
                break;
        }
    }
    if (svrinit_start == 1) {
        printf("ASYNC SERVICE svrinit()\n");
        while(1) {
            sess = (HMS_SHND *) hms_create_async_session("hms01", abend_callback,
                0);
            if (sess == NULL) {
                if (tperrno == TPENOREADY) {
                    usleep(500000);

```

```

        continue;
    }
    printf("ASYNC SERVICE hms_create_async_session() failed,
           tperrno[%d]\n", tperrno);
    return;
}
break;
}
cons = (HMS_CHND *) hms_create_consumer(sess, "queue01", HMS_QUEUE,
    "consasync", "", "ASYNCSVC", 0);
if (cons == NULL)
    printf("ASYNC SERVICE hms_create_consumer() failed, tperrno[%d]\n",
        tperrno);
    return -1;
}
printf("ASYNC SERVICE svrinit() success\n");
}
return 1;
}

int tpsvrdone()
{
    if (svrinit_start == 1) {
        printf("ASYNC SERVICE svrdone()\n");
        hms_close_consumer(cons, 0);
        hms_close_session(sess, 0);
    }
    return 1;
}

/* DB INSERT */
EXEC SQL include sqlca.h;

EXEC SQL begin declare section;
int no;
char message[128];
EXEC SQL end declare section;

int DBInsert( int n, char *data )
{
    printf("UPDATE START!!!\n");
    memset( message, 0x00, sizeof(message) );
    no = n;
    strcpy(message, data);

    EXEC SQL insert into hmstest(no, message) values(:no, :message);

```



```

    if ( sqlca.sqlcode != 0 ){
        printf( "insert failed sqlcode = %d\n",sqlca.sqlcode );
        return -1;
    }

    return 1;
}

ASYNCSVC(TPSVCINFO *svc)
{
    hms_msg_t *msg;
    char *data;
    long llen = 4096;
    int prop = 0;
    int type;

    printf("ASYNC SERVICE CALLED\n");
    msg = (hms_msg_t *)svc->data;

    if ((data = (char *)tpalloc("CARRAY", NULL, 4096)) == NULL) {
        printf("ASYNC SERVICE tpalloc return failed. tperrno[%d]\n",
            tperrno);
        tpreturn(TPFAIL, TPFAIL_ACK, svc->data, svc->len, 0);
    }

    if (hms_get_body(msg, data, &llen) < 0) {
        printf("ASYNC SERVICE hms_get_body() return failed. tperrno[%d]\n",
            tperrno);
        tpreturn(TPFAIL, TPFAIL_ACK, svc->data, svc->len, 0);
    }
    data[llen] = '\0';

    llen = sizeof(int);
    if (hms_get_property(msg, "NO", &type, (char *)&prop, &llen) < 0) {
        printf("ASYNC SERVICE hms_get_property() return failed. tperrno[%d]\n",
            tperrno);
    }
    printf("ASYNC SERVICE RECV MESSAGE, BODY[%s], PROPERTY[NO:%d]\n", data, prop);

    if (DBInsert(prop, data) == -1)
        tpreturn(TPFAIL, TPFAIL_ACK, svc->data, svc->len, 0);
    tpreturn(TPSUCCESS, 0, svc->data, svc->len, 0);
}

```

5.2.4. データベース・スクリプト

表作成スクリプト

以下は、Oracleの表作成スクリプトです。

```
sqlplus scott/tiger << EOF
  create table hmstest (
    no number(7),
    message char(128)
  );
EOF
```

表およびデータ出力スクリプト

以下は、Oracleの表およびデータ出力スクリプトです。

```
sqlplus scott/tiger << EOF
desc hmstest;
select * from hmstest;
select count(*) from hmstest;
EOF
```

5.2.5. プログラムのコンパイル

サーバーとクライアント・プログラムは、Tmaxアプリケーションをコンパイルする手順と同様にコンパイルします。

参考

プログラムのコンパイルについての詳細は、『Tmax アプリケーション開発ガイド』を参照してください。

付録 A. メッセージ・セレクトタの使用法

クライアントは、メッセージ・セレクトタを設定することで、一定の条件を満たすメッセージのみを受信できます。メッセージ・セレクトタは、デスティネーションにあるメッセージのうち基準を満たすメッセージを識別して受信できるようにフィルタリングの役割をします。メッセージのプロパティに設定された値とメッセージ・セレクトタで設定した条件を比較します。比較対照はプロパティのみ可能で、主要コンテンツを格納するBodyの値は比較しません。

メッセージ・セレクトタの文法は、SQL92の表現式の一部を遵守します。メッセージ・セレクトタを評価する順序は左から右へと行われ、括弧を使用して順序を変更できます。メッセージ・セレクトタは、リテラルと記述子、そして表現式で構成されます。

● リテラル

- 単一引用符で囲まれた文字列 (例: 'cat')
- longまたはdoubleの範囲内の数値の文字列 (例: 50, 1.414)

● 記述子

- プロパティのNameに該当します。後に続くリテラルは、プロパティのタイプと一致する必要があります。一致しない場合、該当プロパティの評価はFalseになります。また、メッセージに該当プロパティが存在しない場合、値はNULLと評価されます。
- 記述子は表現式またはリテラルを使用してはなりません。また、大文字と小文字を区別します。
- 記述子に一部の予約済みプロパティは使用できません。予約済みプロパティの種類は「[1.3.4. メッセージ](#)」を参照してください。
- 記述子にメッセージ・セレクトタで使用する文法のキーワードは使用できません。

● 表現式

以下の比較演算子、論理演算子、数値演算子などを使用して条件表現式を使用することができます。

区分	説明
比較演算子	=, >, >=, <, <=, <>
論理演算子	NOT, AND, OR
数値演算子	+, -(単項演算), *, /, +, - (優先順位は +, -(単項演算) > *, / > +, -の順)

– 条件表現式

- 表現式1 [NOT] BETWEEN 表現式2 AND 表現式3

```
"weight BETWEEN 50 AND 70"は"weight >= 50 AND weight <= 70"と同じである。
```

```
"weight NOT BETWEEN 50 AND 70"は"weight < 50 OR weight > 70"と同じである。
```

- 記述子 [NOT] IN (literal1, literal2, ...)

```
"type IN ('cat', 'tiger', 'puma')"は  
"type = 'cat' OR type = 'tiger' OR type = 'puma'"と同じである。
```

```
"type NOT IN (20, 30, 40)"は  
"type <> 20 AND type <> 30 AND type <> 40"と同じである。
```

- 記述子 [NOT] LIKEパターン

パターンは、疑問符(?)、アスタリスク(*)文字を使用できます。疑問符(?)文字は該当の位置にいかなる文字であれ1文字があれば真となり、アスタリスク(*)文字は該当の位置に0個以上のいかなる文字列でも指定できるという意味です。

```
"year LIKE '200?' "は、'2009'は真であるが'200'や'20001'は偽となる。
```

```
"year LIKE '200*' "は'200'で始まるすべての文字列が真となる。
```

以下は、メッセージ・セレクトを使用する例です。

```
"Type = 'cat' AND Weight BETWEEN 5 AND 15"
```

Type = 'cat'が真で、Weight BETWEEN 5 AND 15("weight >= 5そしてweight <= 15")が真の場合、該当メッセージ・セレクト文は真になります。

索引

C

clitype

ADSUB, 96

ARCV, 96

ASUB, 96

DSUB, 96

PUB, 95

RCV, 95

SND, 95

SUB, 95

D

DOMAINセクション, 9

H

HMS, 1

hms_ack(), 47

hms_alloc(), 35

hms_browser_get_msgselector(), 87

hms_browser_get_queue(), 89

hms_browser_nextmsg(), 85

hms_close_browser(), 83

hms_close_consumer(), 72

hms_close_durable_subscriber(), 75

hms_close_producer(), 56

hms_close_publisher(), 58

hms_close_receiver(), 73

hms_close_sender(), 57

hms_close_session(), 27

hms_close_subscriber(), 74

hms_commit(), 31

hms_create_async_session(), 24

hms_create_browser(), 81

hms_create_consumer(), 63

hms_create_durable_subscriber(), 70

hms_create_producer(), 50

hms_create_publisher(), 54

hms_create_receiver(), 66

hms_create_sender(), 52

hms_create_session(), 20

hms_create_subscriber(), 68

hms_create_xa_session(), 22

hms_free(), 37

hms_get_body(), 44

hms_get_property(), 39

hms_recover(), 33

hms_recv(), 79

hms_recvex(), 77

hms_rollback(), 32

hms_send(), 61

hms_sendex(), 59

hms_set_body(), 46

hms_set_property(), 41

hms_unsubscribe_durable_subscriber(), 28

HMSセクション, 13

Hybrid Messaging System, 1

J

JMS, 1

N

NODEセクション, 9

S

SVRGROUPセッション, 10

T

tmadmin, 93

Tmax, xi

X

XAセッション, 3

か

恒久サブスクライバー(Durable Subscriber), 5

キュー(Queue), 4

キュー・ブラウザー(QueueBrowser), 7

クラスタリング(Clustering), 7
グローバル・キュー(Global Queue), 8
グローバル・トピック(Global Topic), 8
コンシューマ(Consumer), 5

さ

サブスクライバー(Subscriber), 5
セッション, 3

た

デスティネーション(Destination), 4
トピック(Topic), 4
トランザクティッド・セッション(Transacted Session), 3

は

非同期セッション(Async Session), 4
プロデューサー(Producer), 4

ま

メイン・スレッド, 2
メッセージBody, 6
メッセージング・システム(Messaging System), 1
メッセージ・セレクタ(Message Selector), 5
メッセージ・セレクタの使用方法, 111
メッセージ・プロパティ, 6

や

予約プロパティ, 6

ら

レシーバー(Receiver), 5

わ

ワーカー・スレッド, 2