HITACHI
Inspire the Next

OpenTP1 Version 7
分散トランザクション処理機能

TP1/Extension for .NET Framework 使用の手引

解説・手引・文法・操作書

3000-3-D67-10

前書き

■ 対象製品

R-15452-27 uCosminexus TP1/Extension for .NET Framework 07-50（適用 OS：Windows 7，Windows 8，Windows 8.1，Windows 10，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016）

このプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

■ 発行

2018 年 10 月 3000-3-D67-10

■ 著作権

変更内容

変更内容（3000-3-D67-10） uCosminexus TP1/Extension for .NET Framework 07-50

追加・変更内容	変更個所
以下の適用 OS を追加した。 • Windows 7 • Windows 8 • Windows 8.1 • Windows 10 • Windows Server 2008 R2 • Windows Server 2012 • Windows Server 2012 R2 • Windows Server 2016	―
以下の適用 OS を削除した。 • Windows Server 2003 • Windows XP	―
Visual Studio 2013 以降に対応した。 また，Visual Studio 2008 以前の記述を削除した。	―
開発支援機能に関する記述を削除した。	1.1.2(2)，1.2.1，2.1.3(2)(a)，2.11.3，4.2，4.10.3，5.2.4
Extension .NET を開発および使用する場合の前提ソフトウェアを変更した。	1.2.1，4.11.2(1)
アプリケーションを開発する言語から J#および COBOL に関する記述を削除した。	1.2.1，6.
バージョンアップ時の API，定義，およびコマンドの変更点を追加した。	付録 C

単なる誤字・脱字などはお断りなく訂正しました。

今版（3000-3-D67-10）では，前版（3000-3-D67）の目次構成を変更しました。前版との対応は次のようになっています。

旧（3000-3-D67）	新（3000-3-D67-10）
1. 概要	1. 概要
2. 機能	2. 機能
3. システム定義【TP1/Server Base】	3. システム定義【TP1/Server Base】
4. UAP の作成と実行 ： 4.2.1 SPP.NET の作成手順の概要	4. UAP の作成と実行 ： 4.2.1 SPP.NET の作成手順の概要

旧（3000-3-D67）	新（3000-3-D67-10）
4.2.2　SPP.NET ウィザードを利用した SPP.NET の作成 4.2.3　SUP.NET の作成手順の概要 4.2.4　SUP.NET ウィザードを利用した SUP.NET の作成 4.2.5　SPP.NET および SUP.NET 作成時の注意事項 ：	4.2.2　SUP.NET の作成手順の概要 4.2.3　SPP.NET および SUP.NET 作成時の注意事項 ：
5.　TP1/LiNK でのユーザサーバの実行環境設定【TP1/LiNK】	5.　TP1/LiNK でのユーザサーバの実行環境設定【TP1/LiNK】
6.　運用コマンド	6.　運用コマンド
7.　クラスリファレンス	7.　クラスリファレンス
8.　障害運用	8.　障害運用
9.　開発支援機能	－
付録 A　クラス名およびメソッド名と C 言語の関数名の対応 付録 B　DCCM3 と接続する場合の注意事項 付録 C　用語解説	付録 A　クラス名およびメソッド名と C 言語の関数名の対応 付録 B　DCCM3 と接続する場合の注意事項 付録 C　バージョンアップ時の変更点 付録 D　用語解説

はじめに

このマニュアルは，プログラムプロダクト R-15452-27 uCosminexus TP1/Extension for .NET Framework の機能と使い方について説明したものです。

本文中に記載されている製品のうち，このマニュアルの対象製品ではない製品については，OpenTP1 Version 7 対応製品の発行時期をご確認ください。

■ 対象読者

システム管理者，システム設計者，プログラマ，オペレータの方を対象としています。また，.NET Framework を利用したプログラミングの知識を持っていることを前提としています。

■ マニュアルの構成

このマニュアルは，次に示す章と付録から構成されています。

第 1 章　概要

OpenTP1 for .NET Framework の概要について説明しています。

第 2 章　機能

TP1/Extension for .NET Framework の機能について説明しています。

第 3 章　システム定義【TP1/Server Base】

TP1/Extension for .NET Framework で使用するシステム定義について説明しています。この定義は，TP1/Server Base の場合だけ指定できます。

第 4 章　UAP の作成と実行

TP1/Extension for .NET Framework で使用する UAP の作成方法と実行手順について説明しています。

第 5 章　TP1/LiNK でのユーザサーバの実行環境設定【TP1/LiNK】

SPP.NET および SUP.NET をユーザサーバとして使用する場合に TP1/LiNK で設定する内容について説明しています。TP1/Server Base の場合，この章で説明する設定は不要です。

第 6 章　運用コマンド

TP1/Extension for .NET Framework で使用する運用コマンドについて説明しています。

第 7 章　クラスリファレンス

TP1/Extension for .NET Framework で使用するクラスについて説明しています。

第 8 章　障害運用

TP1/Extension for .NET Framework で障害が発生した場合の対処方法について説明しています。

付録 A　クラス名およびメソッド名と C 言語の関数名の対応

SPP.NET および SUP.NET で利用できるクラスおよびメソッドと C 言語の関数名の対応について説明しています。

付録 B　DCCM3 と接続する場合の注意事項

RPC やメッセージ送受信機能を使用して DCCM3 と接続する場合の注意事項について説明しています。

付録 C　バージョンアップ時の変更点

各バージョンでの API，定義，およびコマンドの変更点について説明しています。

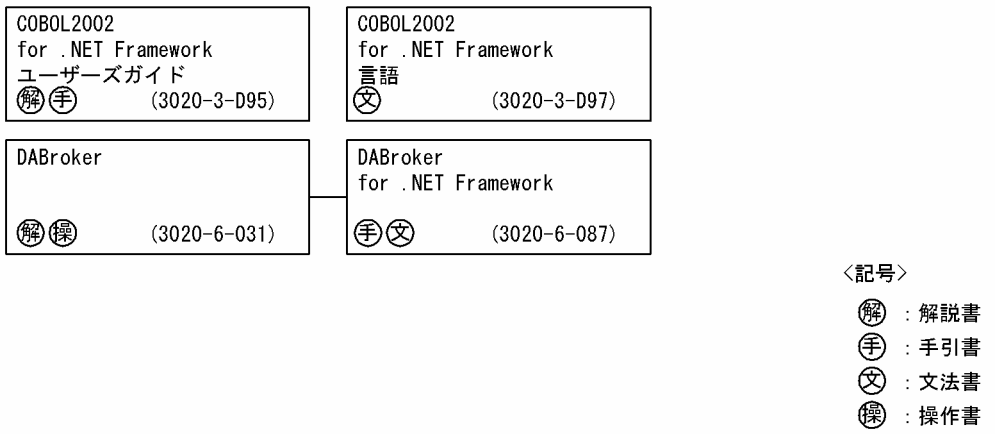
付録 D　用語解説

TP1/Extension for .NET Framework に関する用語について説明しています。

■ 関連マニュアル

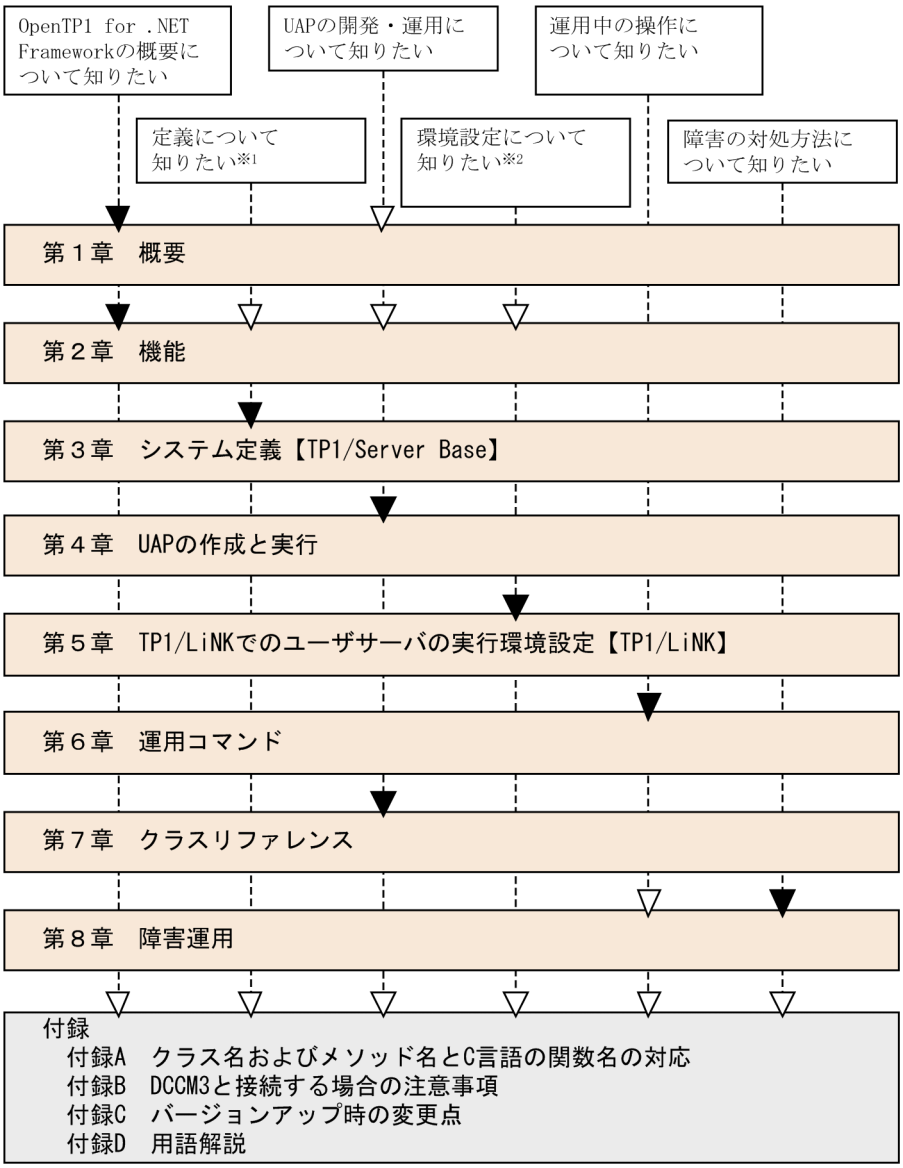
注※　このマニュアルおよびこのマニュアルが対象とする製品の発行時期についてはご確認ください。

●関連製品



■ 読書手順

このマニュアルは，利用目的に合わせて，章を選択して読むことができます。次の案内に従ってお読みいただくことをお勧めします。

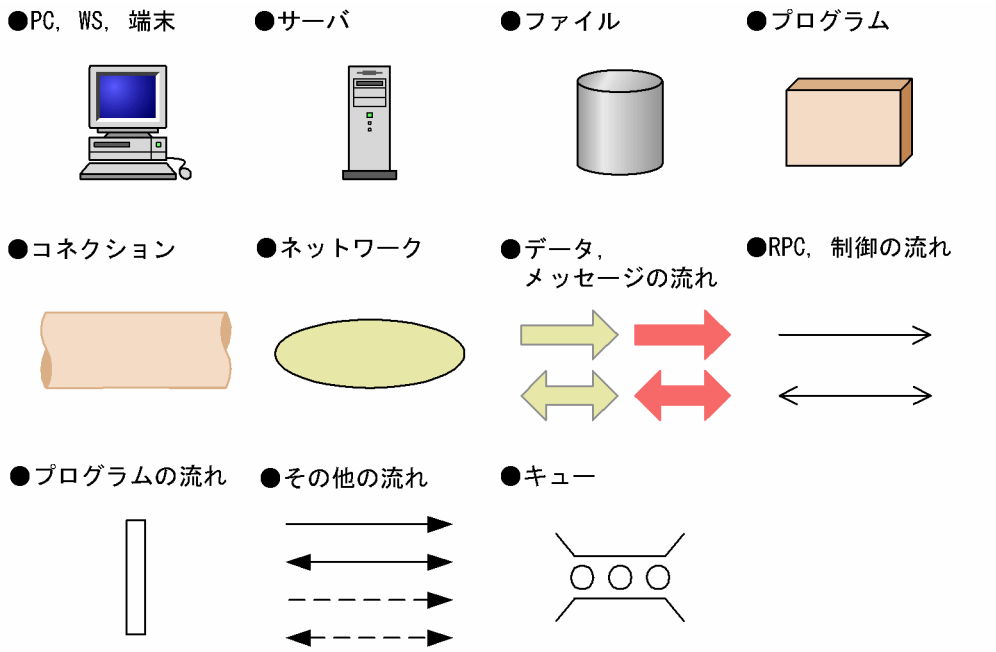


（凡例）　：　必ず読む項目　　：　必要に応じて読む項目

注※1　TP1/Server Baseの場合にお読みください。
注※2　TP1/LiNKの場合にお読みください。

■ 図中で使用する記号

このマニュアルの図中で使用する記号を，次のように定義します。



■ 文法の記号

(1) 文法記述記号

文法の記述形式について説明する記号です。

文法記述記号	意味
〔　　〕	この記号で囲まれている項目は省略できることを示します。 (例) spp2tsp 〔-n 名前空間名称〕 これは，「spp2tsp」と指定するか，または「spp2tsp -n 名前空間名称」と指定することを示します。
…	この記号で示す直前の項目を繰り返し指定できることを示します。 (例) -R アセンブリ名称〔,アセンブリ名称〕… これは，-R オプションのアセンブリ名称を繰り返し指定できることを示します。
|	この記号で区切られた項目は選択できることを示します。 (例) watchTimeNotification="true | false" これは，watchTimeNotification 属性に true か false のどちらかを指定できることを示します。
{　　}	この記号で囲まれている複数の項目のうちから一つを選択できることを示します。 (例) -l {cs | vjs | vb} これは，cs，vjs，vb の三つのオプションのうち，どれか一つを指定することを示します。
____(下線)	この記号で示す項目は，該当オプション，要素，属性，またはコマンド引数を省略した場合の仮定値を示します。 (例) -E {big | little} これは，オプションの指定を省略した場合，big オプションを仮定することを示します。

(2) 属性表示記号

ユーザ指定値の範囲などを説明する記号です。

属性表示記号	意味
～	この記号のあとにユーザ指定値の属性を示します。
《　》	ユーザ指定値を省略した場合の仮定値を示します。
〈　〉	ユーザ指定値の構文要素記号を示します。
((　))	ユーザ指定値の指定範囲を示します。

(3) 構文要素記号

ユーザ指定値の内容を説明する記号です。

構文要素記号	意味
〈英字記号〉	アルファベット（A～Z，a～z）と#，@，¥
〈符号なし整数〉	数字（0～9）
〈識別子〉	先頭がアルファベット（A～Z，a～z）で始まる英数字列
〈記号名称〉	先頭が英字記号で始まる英数字記号列
〈文字列〉	任意の文字の並び
〈パス名〉	記号名称，円記号（¥），およびピリオド（.）の並び （ただし，パス名は使用している OS に依存します）
〈ファイル名〉	任意の文字の並び （ただし，ファイル名は使用している OS，およびアプリケーションに依存します）

■ このマニュアルでの表記

このマニュアルでは，製品の名称を省略して表記しています。製品の名称と，このマニュアルでの表記を次に示します。

製品名称	このマニュアルでの表記	
Microsoft(R) .NET Framework	.NET Framework	
Microsoft(R) Visual C#	C#	
Microsoft(R) Visual Basic	Visual Basic	
Microsoft(R) Visual Basic .NET	VB.NET	
uCosminexus TP1/Client for .NET Framework	TP1/Client for .NET Framework	Client .NET

製品名称	このマニュアルでの表記	
COBOL2002 for .NET Framework Developer	COBOL2002 for .NET Framework	
COBOL2002 for .NET Framework Server Runtime		
COBOL2002 for .NET Framework Server Suite		
uCosminexus TP1/Connector for .NET Framework	TP1/Connector for .NET Framework	Connector .NET
VOS1 DCCM3	DCCM3	
VOS3 XDM/DCCM3		
uCosminexus TP1/Extension for .NET Framework	TP1/Extension for .NET Framework	Extension .NET
Visual J#	J#	
MSDN(R)	MSDN	
uCosminexus TP1/Client/J	TP1/Client/J	
uCosminexus TP1/Client/P	TP1/Client/P	
uCosminexus TP1/Client/P(64)		
uCosminexus TP1/Client/W	TP1/Client/W	
uCosminexus TP1/Client/W(64)		
uCosminexus TP1/Messaging	TP1/Messaging	
uCosminexus TP1/Multi	TP1/Multi	
uCosminexus TP1/NET/TCP/IP	TP1/NET/TCP/IP	
uCosminexus TP1/NET/TCP/IP(64)		
uCosminexus TP1/LiNK	TP1/LiNK	TP1/Server
uCosminexus TP1/Server Base	TP1/Server Base	
uCosminexus TP1/Server Base(64)		
Microsoft(R) Visual Studio(R) Community 2013	Visual Studio 2013	Visual Studio
Microsoft(R) Visual Studio(R) Professional 2013		
Microsoft(R) Visual Studio(R) Premium 2013		
Microsoft(R) Visual Studio(R) Ultimate 2013		
Microsoft(R) Visual Studio(R) Express 2013 for Windows Desktop	Visual Studio Express 2013 for Windows Desktop	
Microsoft(R) Visual Studio(R) Community 2015	Visual Studio 2015	

製品名称	このマニュアルでの表記		
Microsoft(R) Visual Studio(R) Professional 2015			
Microsoft(R) Visual Studio(R) Enterprise 2015			
Microsoft(R) Visual Studio(R) Express 2015 for Windows Desktop	Visual Studio Express 2015 for Windows Desktop		
Microsoft(R) Visual Studio(R) Community 2017	Visual Studio 2017		
Microsoft(R) Visual Studio(R) Professional 2017			
Microsoft(R) Visual Studio(R) Enterprise 2017			
Microsoft(R) Windows(R) Software Development Kit (v7.0) for Windows(R) 7 and .NET Framework 3.5 Service Pack 1	Windows SDK 7.0		Windows SDK
Microsoft(R) Windows(R) Software Development Kit (v7.1) for Windows(R) 7 and .NET Framework 4	Windows SDK 7.1		
Microsoft(R) Windows(R) 7 Enterprise	Windows 7	Windows 7	Windows
Microsoft(R) Windows(R) 7 Professional			
Microsoft(R) Windows(R) 7 Ultimate			
Microsoft(R) Windows(R) 7 Enterprise(x64)	Windows 7 x64 Edition		
Microsoft(R) Windows(R) 7 Professional(x64)			
Microsoft(R) Windows(R) 7 Ultimate(x64)			
Windows(R) 8 Enterprise	Windows 8	Windows 8	
Windows(R) 8 Pro			
Windows(R) 8 Enterprise(x64)	Windows 8 x64 Edition		
Windows(R) 8 Pro(x64)			
Windows(R) 8.1 Enterprise	Windows 8.1	Windows 8.1	
Windows(R) 8.1 Pro			
Windows(R) 8.1 Enterprise(x64)	Windows 8.1 x64 Edition		
Windows(R) 8.1 Pro (x64)			
Windows(R) 10 Pro	Windows 10	Windows 10	
Windows(R) 10 Enterprise			
Windows(R) 10 Pro (x64)	Windows 10 x64 Edition		

製品名称	このマニュアルでの表記		
Windows(R) 10 Enterprise(x64)			
Microsoft(R) Windows Server(R) 2008 R2 Datacenter Edition	Windows Server 2008 R2		
Microsoft(R) Windows Server(R) 2008 R2 Enterprise Edition			
Microsoft(R) Windows Server(R) 2008 R2 Standard Edition			
Microsoft(R) Windows Server(R) 2012 Datacenter	Windows Server 2012		
Microsoft(R) Windows Server(R) 2012 Standard			
Microsoft(R) Windows Server(R) 2012 R2 Datacenter	Windows Server 2012 R2		
Microsoft(R) Windows Server(R) 2012 R2 Standard			
Microsoft(R) Windows Server(R) 2016 Datacenter	Windows Server 2016		
Microsoft(R) Windows Server(R) 2016 Standard			

- TP1/Extension for .NET Framework，TP1/Client for .NET Framework，および TP1/Connector for .NET Framework を総称する場合は，OpenTP1 for .NET Framework と表記しています。

■ 前提製品の違いによる機能相違点の表記

TP1/Extension for .NET Framework は，TP1/Server Base または TP1/LiNK を前提としています。前提製品の違いによって記述を書き分ける場合，次に示す表記を使用しています。

表記	意味
【TP1/Server Base】	TP1/Server Base の場合
【TP1/LiNK】	TP1/LiNK の場合

■ 略語一覧

このマニュアルで使用する英略語の一覧を次に示します。

英略語	英字での表記
ACL	Access Control List
ADO	ActiveX Data Object
AP	Application Program
API	Application Programming Interface

英略語	英字での表記
ASP	Active Server Pages
CLR	Common Language Runtime
CLS	Common Language Specification
CTS	Common Type System
CUP	Client User Program
CUP.NET	Client User Program for .NET Framework
DB	Data Base
DBMS	Database Management System
DLL	Dynamic Linking Library
GAC	Global Assembly Cache
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IIS	Internet Information Services
MHP	Message Handling Program
OLTP	Online Transaction Processing
OS	Operating System
PC	Personal Computer
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SPP	Service Providing Program
SPP.NET	Service Providing Program for .NET Framework
SUP	Service Using Program
SUP.NET	Service Using Program for .NET Framework
TCP/IP	Transmission Control Protocol/Internet Protocol
TSDL	TP1 Service Description Language
TSP	TP1 Service Proxy
UAP	User Application Program
WS	Workstation
WWW	World Wide Web

英略語	英字での表記
XA	Extended Architecture
XML	eXtensible Markup Language

■ このマニュアルで使用している記号

注意事項

　間違いやすい点や注意しなければならない点などについて説明しています。

ポイント

　その説明の要点などについて説明しています。

参考

　補足的な情報などについて説明しています。

■ KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

目次

1 概要

TP1/Extension for .NET Framework（Extension .NET）は，OpenTP1 for .NET Framework を構成する三つの製品のうちの一つです。この章では Extension .NET の前提知識として，OpenTP1 for .NET Framework の概要について説明します。

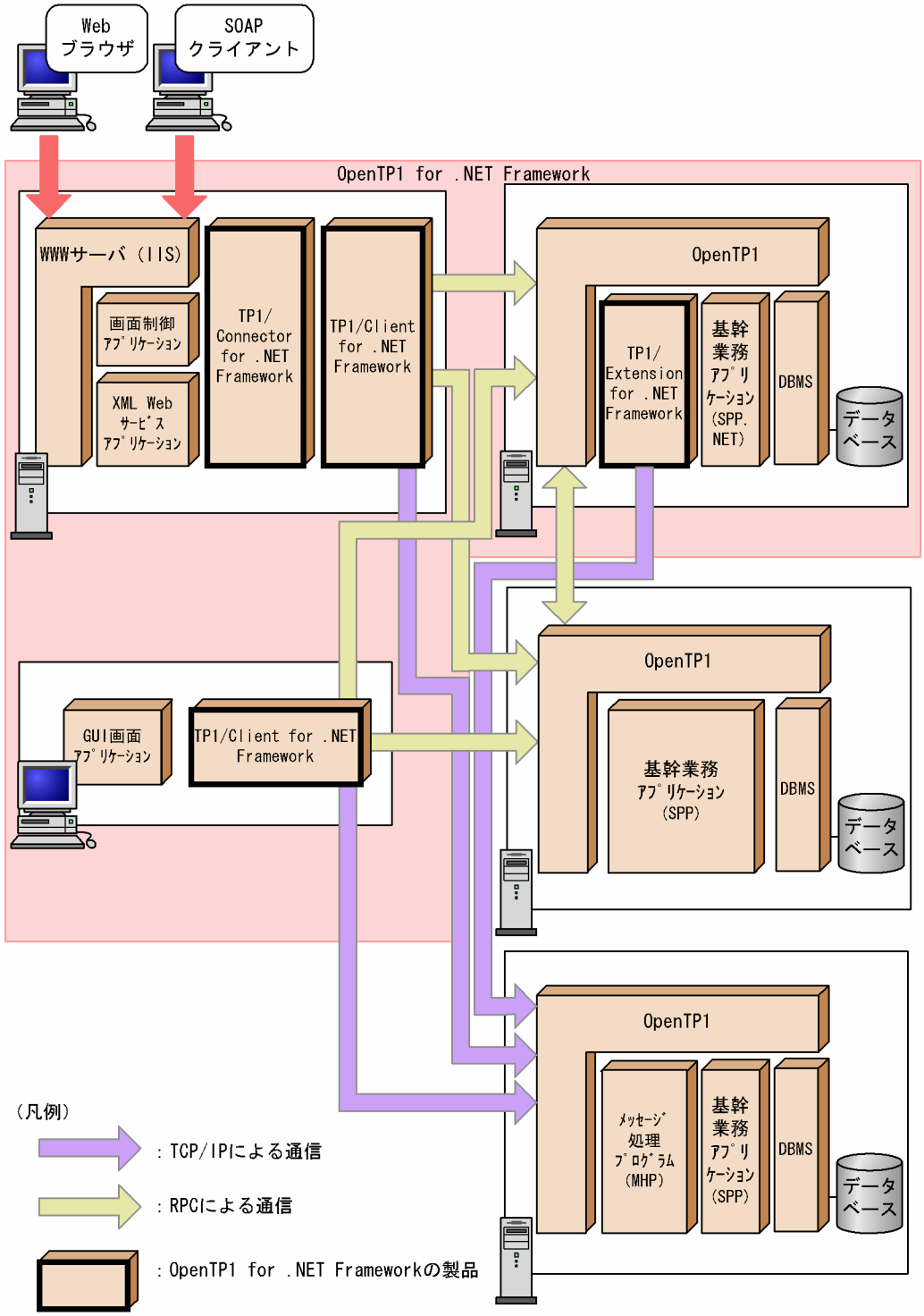
1.1 OpenTP1 for .NET Framework とは

近年，クライアント/サーバシステムに Windows プラットフォームを選択するオンラインシステムが増えています。小さな社内業務システムから社会インフラを担う基幹システムまで，あらゆるオンラインシステムが Windows プラットフォーム上で構築されるようになりました。

OpenTP1 for .NET Framework は，OpenTP1 を Microsoft の.NET Framework に対応させることによって，.NET Framework の持つシステム開発の柔軟性，高生産性と OpenTP1 の持つ分散トランザクションシステムとしての信頼性を融合し，Windows プラットフォーム上で高いシステム生産性，運用性を持つ次世代基幹システムの基盤を提供します。

OpenTP1 for .NET Framework の概要を次の図に示します。

図 1-1　OpenTP1 for .NET Framework の概要



1.1.1　OpenTP1 for .NET Framework の構成

OpenTP1 for .NET Framework は次の表に示す三つの製品から構成されます。

これらの製品は，それぞれ CLS 準拠のクラスライブラリを提供します。

表 1-1　OpenTP1 for .NET Framework の構成

製品名称	機能概要
TP1/Extension for .NET Framework	.NET Framework 環境で OpenTP1 のサーバアプリケーションを開発し，動作させるための製品です。TP1/Extension for .NET Framework には，前提製品として TP1/Server Base または TP1/LiNK が必要です。
TP1/Client for .NET Framework	.NET Framework 環境で OpenTP1 のクライアントアプリケーションを開発するための機能を提供します。
TP1/Connector for .NET Framework	TP1/Client for .NET Framework と連携して，WWW サーバ（IIS の ASP.NET 環境）などの.NET Framework 環境から，OpenTP1 のサーバに対してサービスを要求する場合に使用します。 マルチスレッド環境でコネクションプーリング，バッファプーリングなどの機能を提供します。 TP1/Connector for .NET Framework には，前提製品として TP1/Client for .NET Framework が必要です。

なお，このマニュアルでは，これらの製品を次の表に示すように表記します。

表 1-2　OpenTP1 for .NET Framework の製品名称とこのマニュアルでの表記

製品名称	このマニュアルでの表記
TP1/Extension for .NET Framework	Extension .NET
TP1/Client for .NET Framework	Client .NET
TP1/Connector for .NET Framework	Connector .NET

1.1.2　OpenTP1 for .NET Framework の利点

.NET Framework に対応した OpenTP1 for .NET Framework を使用すると，次のような利点があります。

(1)　高信頼/高性能

OpenTP1 が持つトランザクション制御機能，スケジューリング制御機能，プロセス管理機能，障害回復機能などを.NET Framework 環境で利用できます。

(2)　アプリケーション開発環境の拡充

- .NET Framework の活用によって，OpenTP1 のアプリケーションを開発する言語を C#，J#，Visual Basic，および COBOL 言語から柔軟に選択できます。

  また，サーバアプリケーションからクライアントアプリケーションまでを，一貫して同一のプログラム言語で開発することもできます。

- .NET Framework が持つ優れた開発環境やライブラリなどを活用することによって，アプリケーション開発を迅速に行うことができます。これによって，システム構築に掛かる時間を短縮し，システム開発の生産性の向上が図れます。

(3) ASP.NET XML Web サービスの利用

- ASP.NET XML Web サービスや ASP.NET などに代表される Microsoft のアプリケーション開発技術を，OpenTP1 環境で利用できます。

  これによって，OpenTP1 のサービスを，ASP.NET XML Web サービスとして公開できます。
- 定義されたインタフェース情報から ASP.NET XML Web サービスを自動生成する機能が利用できます。

(4) OpenTP1 システムとの親和性

サービス関数のインタフェースをサービス定義として定義することによって，C 言語，C++言語，および COBOL 言語で作成した OpenTP1 の基幹業務アプリケーションを.NET Framework 環境から呼び出すことができます。

1.2　OpenTP1 for .NET Framework を利用したシステム構成

OpenTP1 for .NET Framework を利用したシステム構成について説明します。

1.2.1　TP1/Extension for .NET Framework 使用時の前提条件

TP1/Extension for .NET Framework をインストールする前に，次に示すソフトウェアがインストールされている必要があります。

- uCosminexus TP1/Server Base 07-50 以降または uCosminexus TP1/LiNK 07-51 以降
- .NET Framework 3.5 Service Pack 1
- Visual Studio（同梱する Windows SDK を使用），または Windows SDK

  注

  Visual Studio，および Windows SDK のバージョン詳細については，リリースノートを参照してください。

(1)　UAP を開発する場合

開発言語として C#または Visual Basic を使用するときは，次のどれかのソフトウェアがインストールされている必要があります。

- Visual Studio
- .NET Framework 3.5 Service Pack 1
- .NET Framework 4.5.2 以降

注

Visual Studio，および.NET Framework のバージョン詳細については，リリースノートを参照してください。

(2)　使用する機能と必要なソフトウェア

使用する機能によって，次に示すソフトウェアがインストールされている必要があります。

表 1-3　使用する機能と必要なソフトウェアの対応

使用する機能	必要なソフトウェア
性能検証用トレースの API	TP1/Extension 1 07-50 以降
TAM ファイルサービス	【TP1/Server Base】 TP1/FS/Table Access 07-50 以降
メッセージ送受信機能	【TP1/Server Base】

使用する機能	必要なソフトウェア
	• TP1/Message Control 07-50 以降 • TP1/NET/Library 07-50 以降 • TP1/NET/TCP/IP 07-50 以降
DBMS とのトランザクション連携	• DABroker for .NET Framework 01-01 以降 • DABroker 03-23-02 以降

(3) 注意事項

前提ソフトウェアのサポート終了に伴い，次の言語のサポートを終了しました。マニュアルには次の言語の記述がありますが，サポートしておりません。

- J#
- COBOL 言語

1.2.2 クライアント/サーバシステムの形態

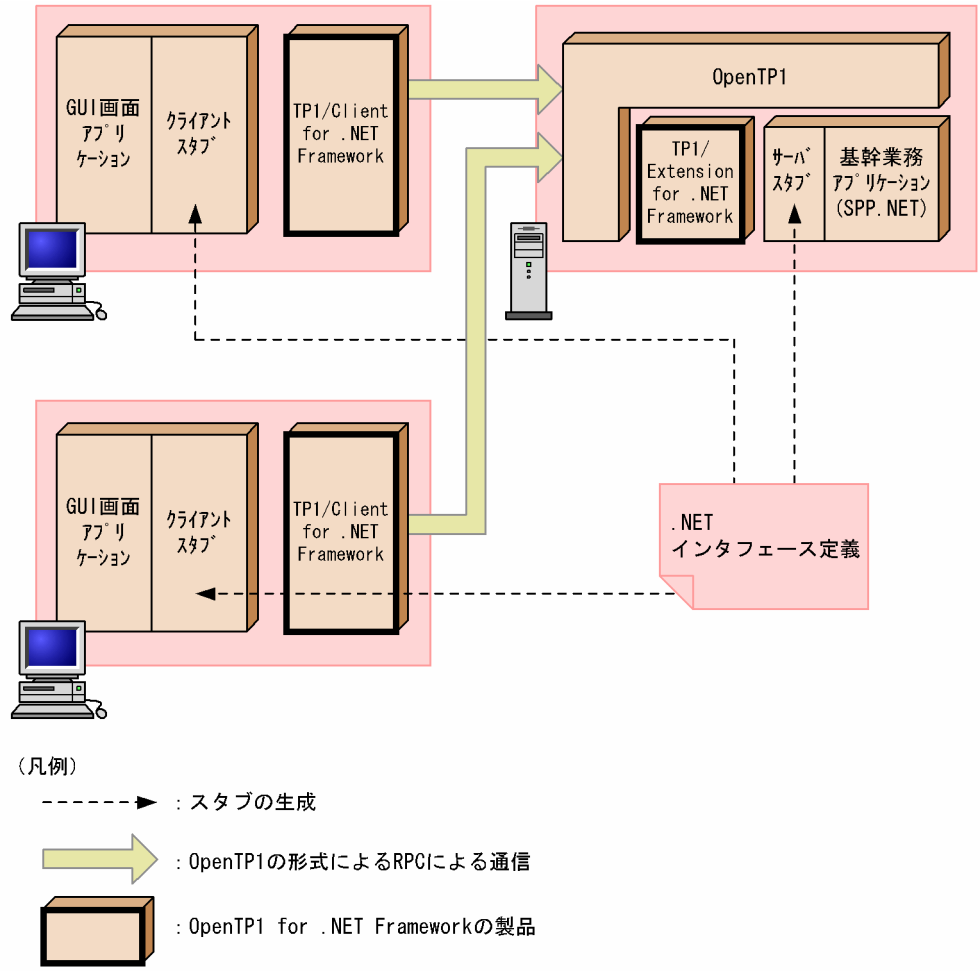
TP1/Client for .NET Framework を使用することで，.NET Framework 環境で動作する OpenTP1 のクライアントアプリケーション（GUI 画面アプリケーションなど）を作成できます。

また，TP1/Extension for .NET Framework を使用することで，.NET Framework 環境で動作する OpenTP1 のサーバアプリケーション（基幹業務アプリケーションなど）を作成できます。

このように，クライアントアプリケーションからサーバアプリケーションまで，一貫して.NET Framework 環境で開発できます。

OpenTP1 for .NET Framework を利用したクライアント/サーバシステムの形態のシステム構成例を次の図に示します。

図 1-2　OpenTP1 for .NET Framework を利用したクライアント/サーバシステムのシステム構成例



.NET Framework 環境では，リモートプロシジャコール（RPC）のインタフェース情報を.NET インタフェース定義に定義します。この.NET インタフェース定義から運用コマンドを使用して，クライアントスタブ，サーバスタブを生成します。

RPC メッセージのデータ変換（文字コード変換，エンディアン変換など）は，これらクライアントスタブ，サーバスタブが自動的に行います。

1.2.3　WWW サーバ経由での接続

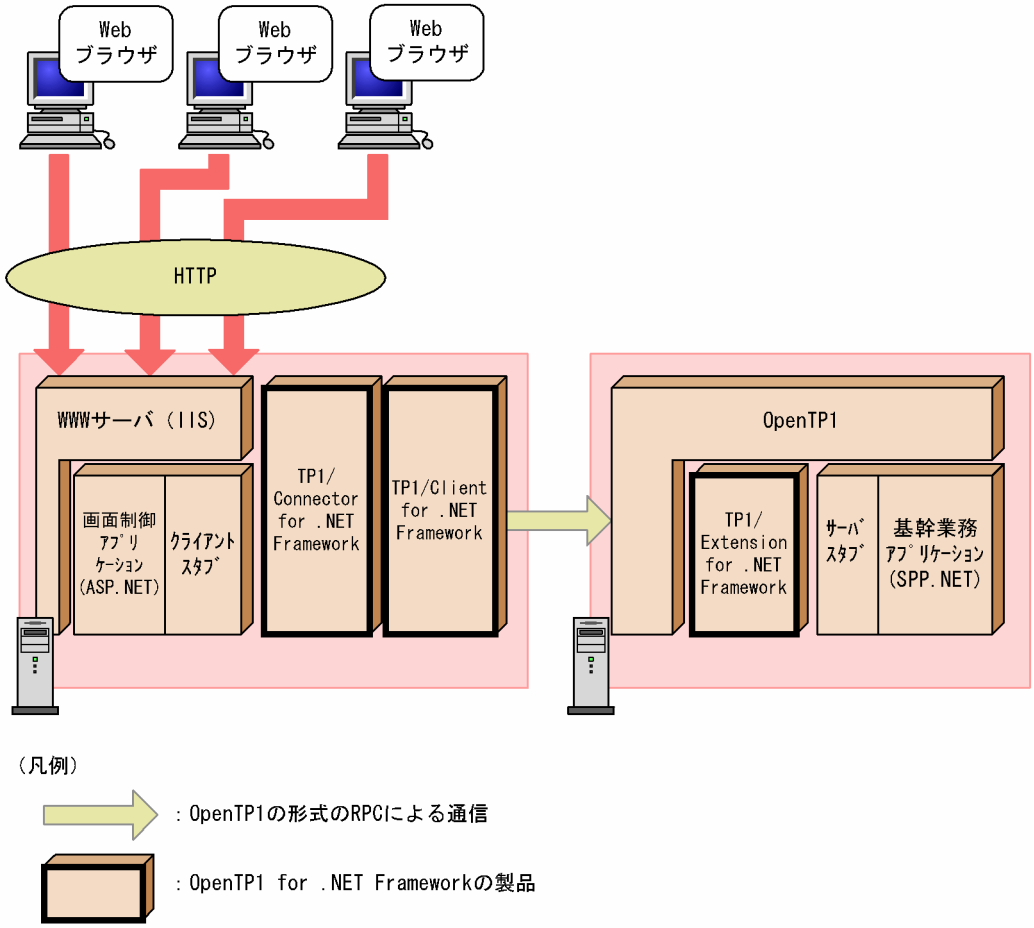
クライアントとなる Web ブラウザから，WWW サーバ（IIS）を経由して OpenTP1 サーバ上の基幹業務アプリケーションにアクセスできます。

WWW サーバ上で TP1/Client for .NET Framework と TP1/Connector for .NET Framework を使用することによって，ASP.NET Web アプリケーションである画面制御アプリケーションから，OpenTP1 を経由してバックエンドの基幹業務アプリケーションを呼び出すことができます。

TP1/Connector for .NET Framework は，マルチスレッド環境でコネクションプーリング機能やバッファプーリング機能などを提供します。

WWW サーバ経由で OpenTP1 のサーバに接続する場合のシステム構成例を次の図に示します。

図 1-3　WWW サーバ経由で OpenTP1 サーバに接続する場合のシステム構成例



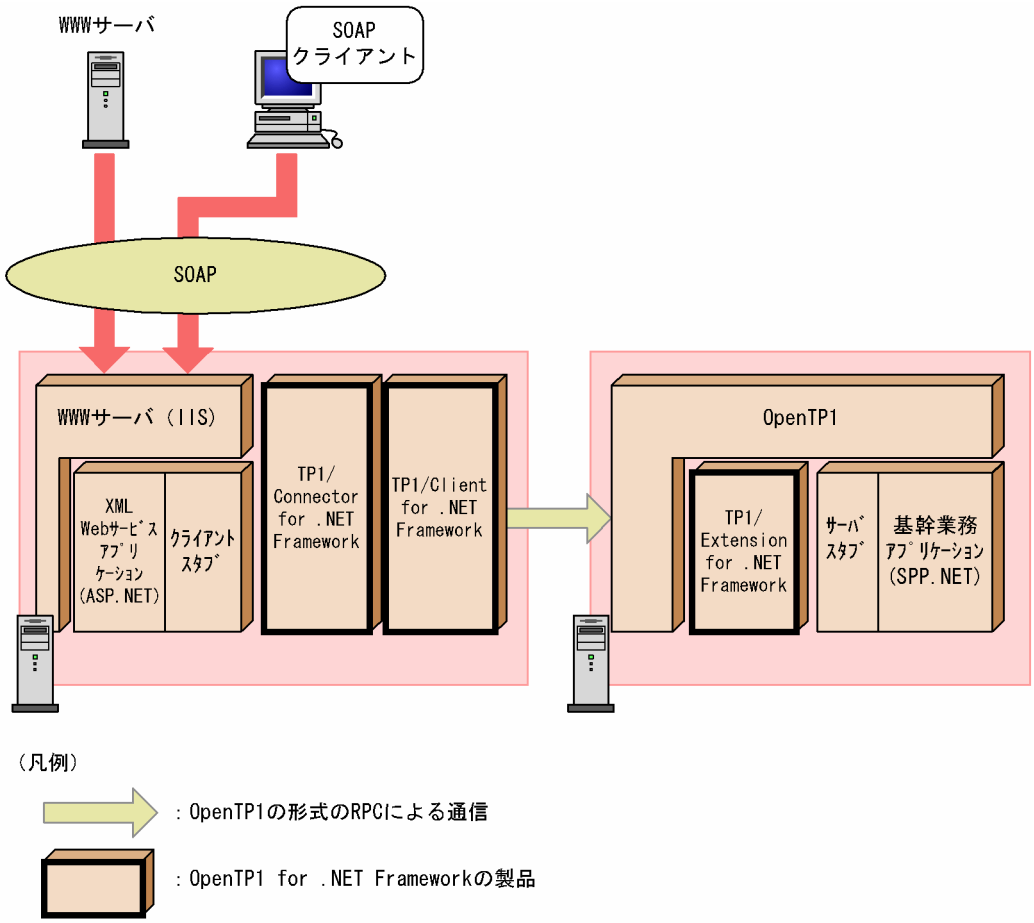
1.2.4　ASP.NET XML Web サービスとの接続

OpenTP1 for .NET Framework を使用することで，OpenTP1 のサービスを ASP.NET XML Web サービスとして公開できます。

WWW サーバ上で TP1/Client for .NET Framework と TP1/Connector for .NET Framework を使用することによって，ASP.NET 環境で動作する XML Web サービスアプリケーションから，バックエンドの基幹業務アプリケーションを呼び出すことができます。

ASP.NET XML Web サービスから OpenTP1 サーバに接続する場合のシステム構成例を次の図に示します。

図 1-4　ASP.NET XML Web サービスから OpenTP1 サーバに接続する場合のシステム構成例



WWW サーバ（IIS）上で動作するアプリケーションが，HTTP を受けて動作するか，SOAP を受けて動作するかという点が，WWW サーバ経由で OpenTP1 サーバに接続する場合と異なります。

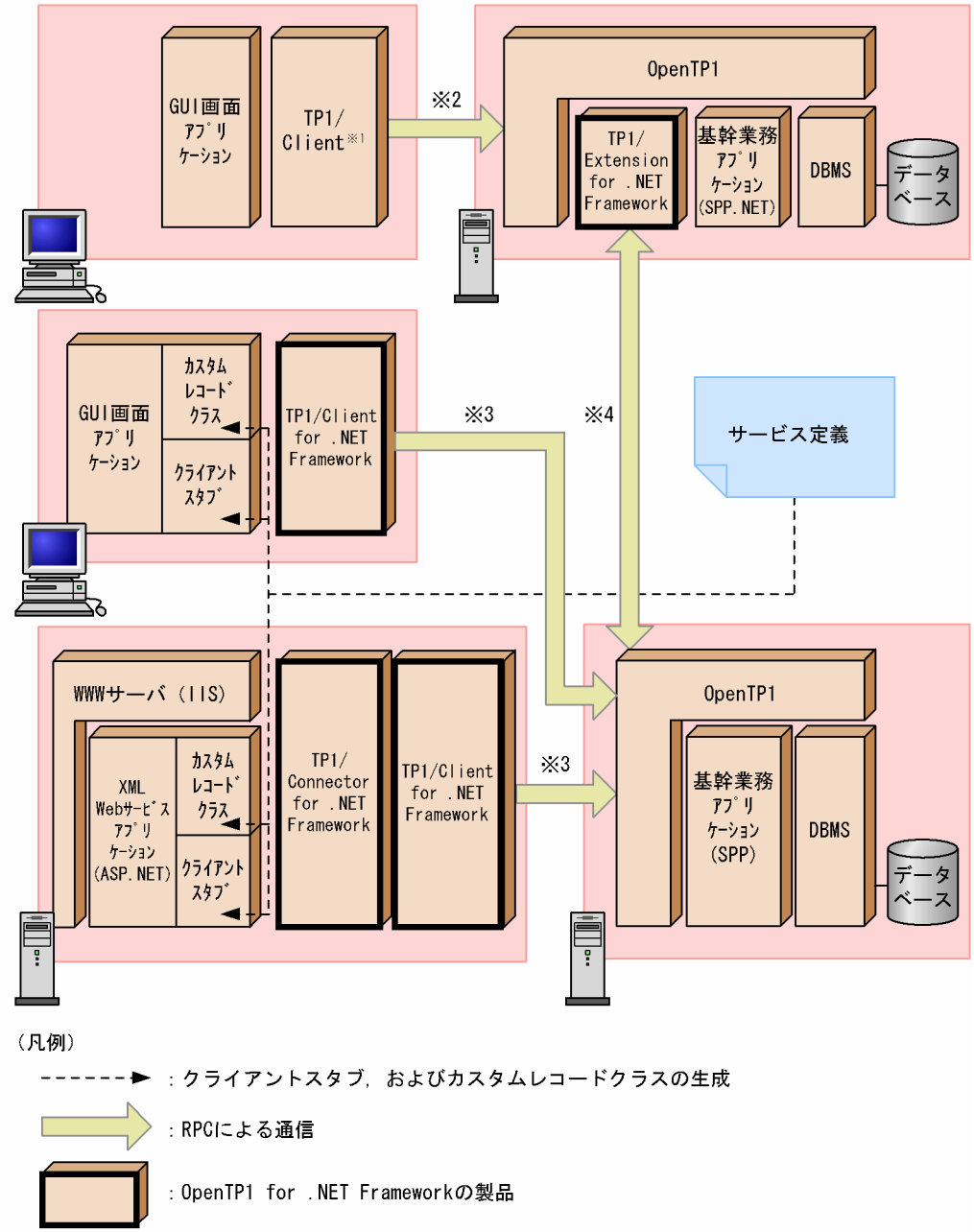
1.2.5　OpenTP1 システムとの接続

OpenTP1 for .NET Framework は，OpenTP1 システム（TP1/Server Base および TP1/LiNK）と接続することによって，次のことができます。

- OpenTP1 システムのクライアントアプリケーション（CUP）からの OpenTP1 for .NET Framework 環境の基幹業務アプリケーション（SPP.NET）の呼び出し
- OpenTP1 for .NET Framework 環境のクライアントアプリケーション（CUP.NET）からの基幹業務アプリケーション（SPP）の呼び出し

OpenTP1 システムと接続する場合のシステム構成例を次に示します。

図 1-5　OpenTP1 システムと接続する場合のシステム構成例



注※1

TP1/Client/P，TP1/Client/W，または TP1/Client/J

注※2

OpenTP1 の形式の RPC

注※3

カスタムレコードを使用した RPC

注※4

バイナリデータを使用した RPC

基幹業務アプリケーション（SPP）の RPC インタフェースが固定の場合，サービス定義にインタフェース情報を定義します。このサービス定義から，クライアントスタブ，およびカスタムレコードクラスを生成します。

.NET Framework 環境のクライアントアプリケーション（CUP.NET）は，生成されたクライアントスタブ，カスタムレコードクラスを使用して基幹業務アプリケーション（SPP）を呼び出すことができます。

なお，バイナリデータを使用する RPC インタフェースを利用して，基幹業務アプリケーション（SPP）を呼び出すこともできます。

1.3 OpenTP1 for .NET Framework のアプリケーション

1.3.1 ユーザアプリケーションプログラム（UAP）

OpenTP1 for .NET Framework では，次の UAP を利用できます。

また，OpenTP1 for .NET Framework の UAP と OpenTP1 の SPP，SUP，CUP の間で RPC を実行できます。

(1) SPP.NET

SPP.NET は，TP1/Extension for .NET Framework 上で動作する，マネージコードで記述されたサーバ UAP です。SPP.NET は，クライアント UAP から要求されたサービスを実行します。

SPP.NET は，TP1/Extension for .NET Framework が提供するランタイムホストで実行されます。

SPP.NET の詳細については，「2.1.1 SPP.NET の概要」を参照してください。

(2) SUP.NET

SUP.NET は，TP1/Extension for .NET Framework 上で動作する，マネージコードで記述されたクライアント UAP です。SUP.NET は，SPP.NET または SPP に対してサービスを要求します。

SUP.NET は，.NET Framework が提供するランタイムホストで実行されます。

SUP.NET の詳細については，「2.1.2 SUP.NET の概要」を参照してください。

(3) CUP.NET

CUP.NET は，TP1/Client for .NET Framework または TP1/Connector for .NET Framework を利用する，マネージコードで記述されたクライアント UAP です。CUP.NET は，SPP.NET または SPP に対してサービスを要求します。

CUP.NET は，Windows フォームアプリケーション，ASP.NET Web アプリケーション，ASP.NET XML Web サービスなど，さまざまなアプリケーション形態が可能であり，形態に応じたランタイムホストで実行されます。

1.3.2 RPC インタフェース

OpenTP1 for .NET Framework の UAP から，サービス要求をする場合の RPC インタフェースには次の 3 種類があります。

- .NET インタフェース定義を使用した RPC

- サービス定義（カスタムレコード）を使用した RPC
- バイナリデータ（インデクスドレコード）を使用した RPC

(1)　RPC インタフェースの種類

RPC インタフェースの種類を次に示します。

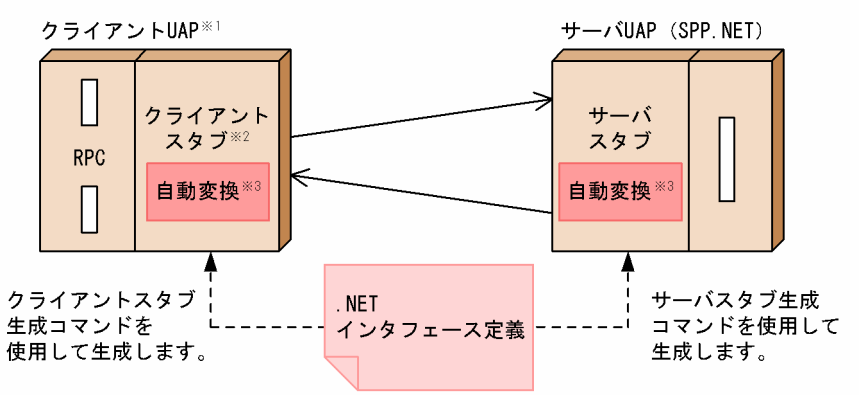
(a)　.NET インタフェース定義を使用した RPC

.NET インタフェース定義から運用コマンドでサーバスタブ，およびクライアントスタブを生成して使用します。これによって，.NET Framework のクラスのメソッド呼び出しと同様の形式で RPC が行えます。また，.NET Framework のデータ型をパラメタや戻り値として送受信できます。

OpenTP1 for .NET Framework を使用したクライアント/サーバシステムの場合に適しています。

.NET インタフェース定義を使用した RPC の概要を次の図に示します。

図 1-6　.NET インタフェース定義を使用した RPC の概要



注※1

クライアント UAP には次の UAP が該当します。

- SPP.NET
- SUP.NET
- CUP.NET（TP1/Connector for .NET Framework を使用するアプリケーションを含みます）

注※2

サービス定義から生成されるクライアントスタブとは異なるものです。

注※3

スタブが RPC データの文字コードの自動変換，およびエンディアンの自動識別をします。

(b)　サービス定義（カスタムレコード）を使用した RPC

SPP.NET または SPP の入出力データ形式とサービス名をサービス定義として定義します。

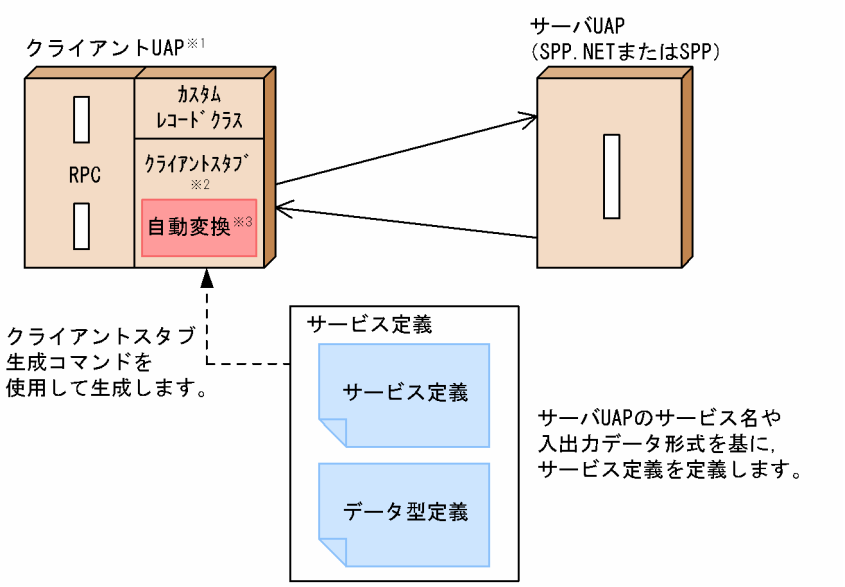
このサービス定義から運用コマンドでカスタムレコードとクライアントスタブを生成して使用します。

クライアント側では.NET Framework のデータ型で入出力データの設定や参照ができます。また，サーバとのプラットフォームやプログラム言語，文字コードなどの違いをユーザが意識することなくアプリケーションを作成できます（運用コマンドでスタブを生成する際に，文字コードやエンディアンなどを指定します）。

OpenTP1 for .NET Framework 環境のクライアントから，SPP にアクセスする場合などに適しています。

サービス定義（カスタムレコード）を使用した RPC の概要を次の図に示します。

図 1-7　サービス定義（カスタムレコード）を使用した RPC の概要



注※1

クライアント UAP には次の UAP が該当します。

- SPP.NET
- SUP.NET
- CUP.NET（TP1/Connector for .NET Framework を使用するアプリケーションを含みます）

注※2

.NET インタフェース定義から生成されるクライアントスタブとは異なるものです。

注※3

スタブが RPC データの文字コードの自動変換，およびエンディアンの自動識別をします。

(c)　バイナリデータ（インデクスドレコード）を使用した RPC

バイナリデータ（バイト配列型）の集合を入力用レコード，および出力用レコードに設定して使用します。

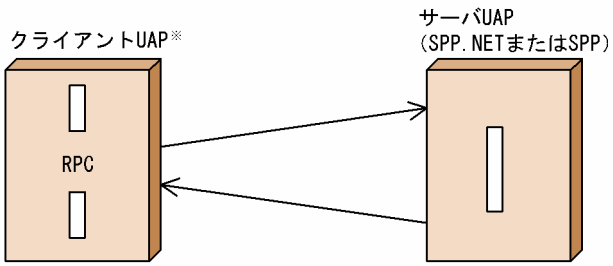
TP1/Connector for .NET Framework で，バイナリデータを使用した RPC を実行する場合は，インデクスドレコードを使用します。

バイナリデータ（インデクスドレコード）を使用した RPC の場合，クライアントとサーバのプラットフォームやプログラム言語，文字コードなどの違いをユーザが意識する必要があります。

OpenTP1 for .NET Framework 環境以外のクライアントから SPP.NET にアクセスする場合などに適しています。

バイナリデータ（インデクスドレコード）を使用した RPC の概要を次の図に示します。

図 1-8　バイナリデータ（インデクスドレコード）を使用した RPC の概要



注※

クライアント UAP には次の UAP が該当します。

- SPP.NET
- SPP
- SUP.NET
- SUP
- CUP.NET（TP1/Connector for .NET Framework を使用するアプリケーションを含みます）
- CUP

1.3.3　アプリケーション開発の流れ

新規にクライアント/サーバシステムを開発する場合，およびクライアントだけを開発して既存のサービスを利用する場合のアプリケーション開発の流れを，RPC インタフェースごとに図 1-9，および図 1-10 に示します。

図 1-9　アプリケーション開発の流れ（RPC インタフェースに.NET インタフェース定義，またはサービス定義を使用する場合）

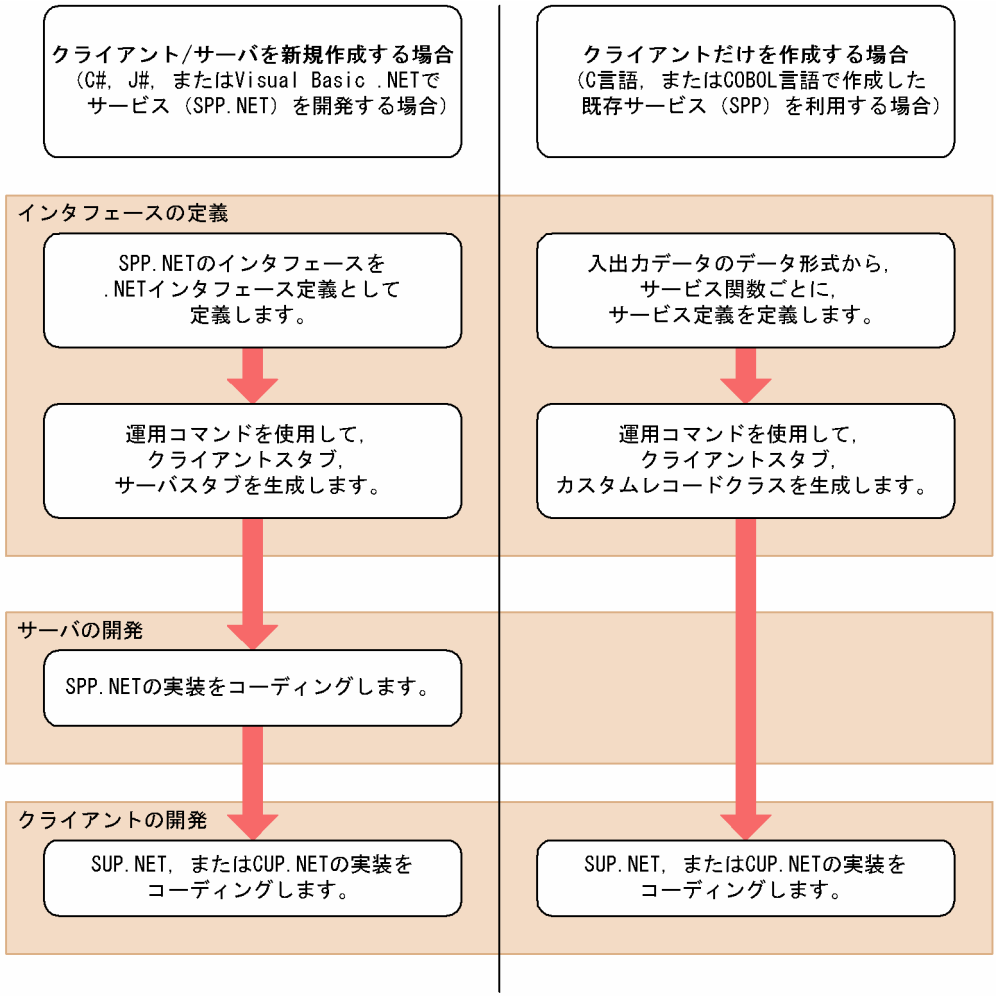
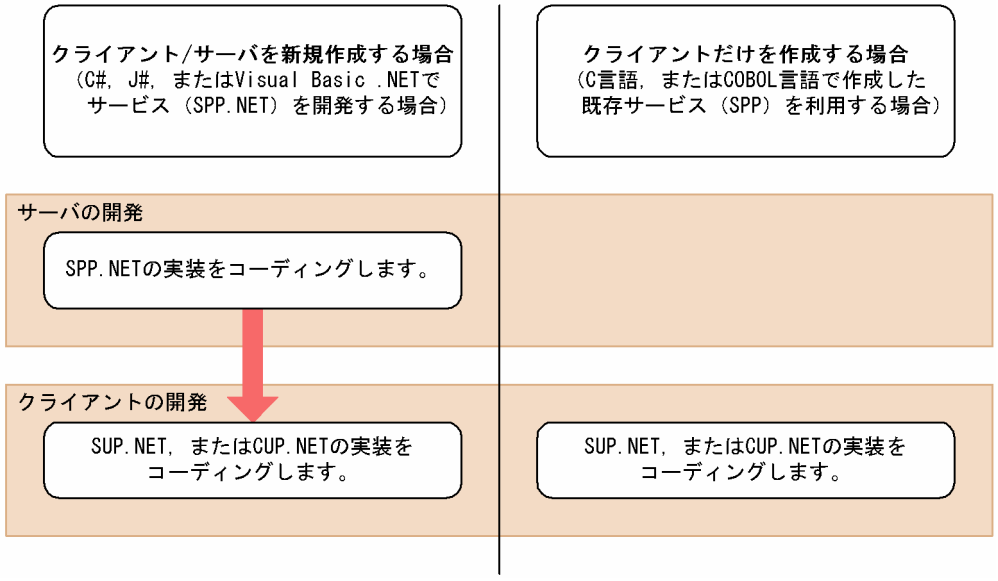


図 1-10　アプリケーション開発の流れ（RPC インタフェースにバイナリデータを使用する場合）



2 機能

この章では，TP1/Extension for .NET Framework（Extension .NET）の機能について説明します。なお，必要に応じて次のマニュアルも参照してください。

・「OpenTP1　解説」

・「TP1/LiNK　使用の手引」

2.1　Extension .NET で使用できる UAP

Extension .NET を利用すると，TP1/Server 上で.NET Framework を使用した UAP の開発，および運用ができます。

Extension .NET で使用できる UAP は，SPP.NET，および SUP.NET です。これらの UAP は，次に示すプログラム言語で記述できます。

- C#
- J#
- Visual Basic
- COBOL 言語※

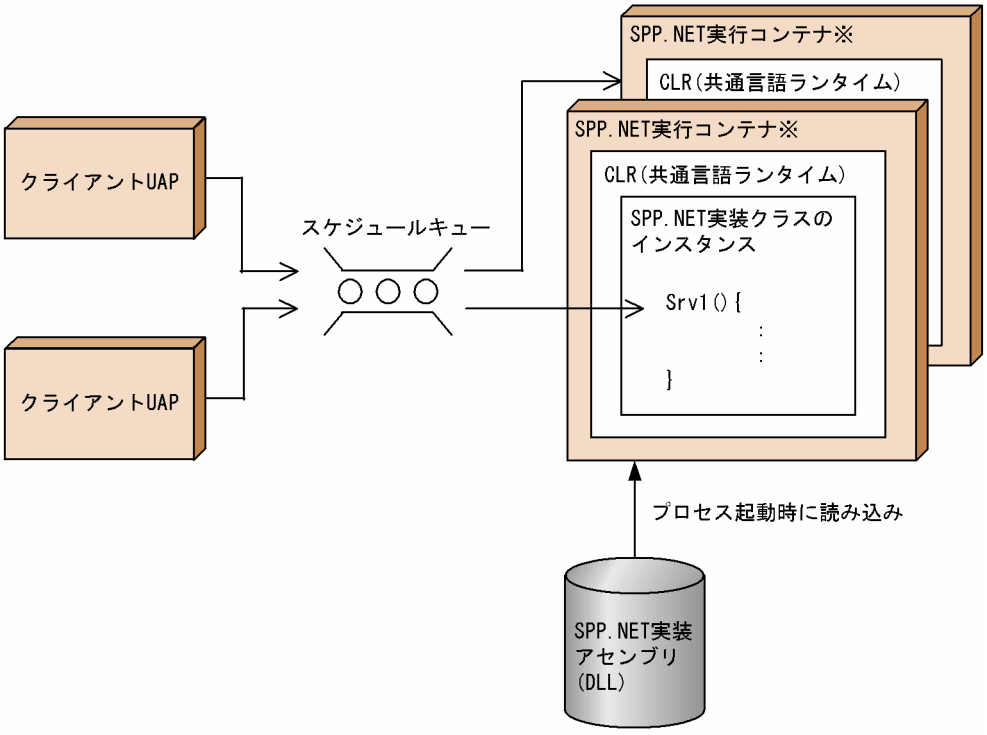
  注※

  使用できる機能には制限があります。詳細については，「2.1.3　COBOL2002 for .NET Framework を使用した UAP の開発および実行」を参照してください。

2.1.1　SPP.NET の概要

SPP.NET は，DLL 形式のアセンブリとして実装します。SPP.NET は，Extension .NET が提供する SPP.NET 実行コンテナによって実行されます。このコンテナは SPP と同様にユーザサーバとして実行できます。また，SPP などのユーザサーバと同様にシングルスレッドで動作します。SPP.NET の実装クラスはこのユーザサーバのプロセス起動時にインスタンス化されます。一つのユーザサーバでは一つのインスタンスが生成されます。SPP.NET では，Main メソッドの実装や，Rpc クラスの Open メソッドのなどの手続きは不要です。

図 2-1　SPP.NET の実行



注※

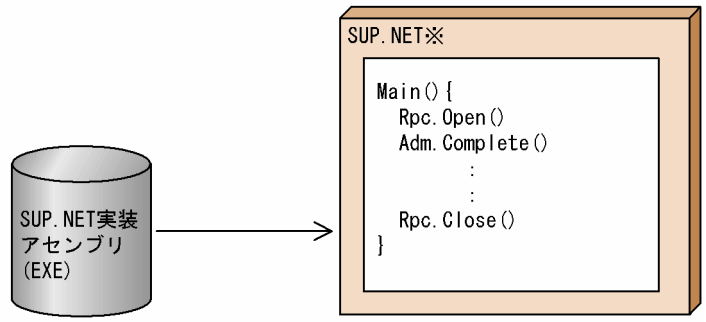
ユーザサーバとして実行されます。

SPP.NET の実装方法の詳細については，「4.　UAP の作成と実行」を参照してください。

2.1.2　SUP.NET の概要

SUP.NET は EXE 形式のアセンブリとして実装し，SUP と同様にユーザサーバとして直接実行できます。SUP.NET では Main メソッドを実行し，Rpc クラスの Open メソッド，Adm クラスの Complete メソッド，Rpc クラスの Close メソッドなどを呼び出す必要があります。

図 2-2　SUP.NET の実行



注※

ユーザサーバとして実行されます。

SUP.NET の実装方法の詳細については，「4.　UAP の作成と実行」を参照してください。

2.1.3　COBOL2002 for .NET Framework を使用した UAP の開発および実行

COBOL2002 for .NET Framework を使用することで，COBOL 言語で SPP.NET および SUP.NET の開発・実行ができます。COBOL2002 for .NET Framework を使用すると，次のような利点があります。

- COBOL 言語で開発された既存の SPP および SUP と連携できる
- COBOL 言語で開発された既存の SPP および SUP を容易に SPP.NET および SUP.NET に移行できる
- COBOL 言語で開発された既存の COBOL プログラム資産を部品として再利用できる

UAP の開発手順の詳細については，「4.8　COBOL 言語での UAP の作成方法」を参照してください。COBOL 言語の仕様については，マニュアル「COBOL2002 for .NET Framework　ユーザーズガイド」を参照してください。また，COBOL 言語で SPP.NET および SUP.NET を開発する場合の OpenTP1 の API については，マニュアル「OpenTP1　プログラム作成リファレンス　COBOL 言語編」を参照してください。

なお，COBOL 言語で開発した SPP.NET および SUP.NET の実行手順は，ほかの言語で開発した場合と同じです。

(1)　Extension .NET が提供する P/Invoke 指示ファイルのサンプル

Extension .NET は，COBOL2002 for .NET Framework で UAP 開発時に必要となる，P/Invoke 指示ファイル（.piv）のサンプルを提供しています。TP1/Server Base または TP1/LiNK と併用する場合のサンプルは，それぞれ次のディレクトリに格納されています。

【TP1/Server Base】

```
〈インストールディレクトリ〉¥examples¥ExtNET¥COBOL.NET¥PIV¥TP1Base.piv
```

【TP1/LiNK】

```
〈インストールディレクトリ〉¥sample¥ExtNET¥COBOL.NET¥PIV¥TP1LiNK.piv
```

P/Invoke 指示ファイルの使用方法については，「4.8.3　TP1/Server の COBOL 言語のサービスルーチンの利用」を参照してください。

(2)　注意事項

(a)　UAP の開発時の注意事項

COBOL2002 for .NET Framework を使用する場合，次の機能は利用できません。

- .NET インタフェース定義を使用した SPP.NET の開発
- .NET インタフェース定義を使用した RPC
- サービス定義（カスタムレコード）を使用した RPC
- DBMS とのトランザクション連携
- Extension .NET が提供するクラスライブラリ※

注※

Extension .NET が提供するクラスライブラリを使用できない代わりに，TP1/Server が提供する COBOL 言語のサービスルーチンを使用できます。COBOL 言語のサービスルーチンについては，マニュアル「OpenTP1　プログラム作成リファレンス　COBOL 言語編」を参照してください。

(b)　UAP の実行時の注意事項

- COBOL2002 for .NET Framework のランタイムが必要とする環境変数を設定する必要があります。設定方法を次に示します。

  【TP1/Server Base】

  ユーザサービス定義などのシステム定義で環境変数を設定します。

  詳細については，「3.　システム定義【TP1/Server Base】」を参照してください。

  【TP1/LiNK】

  TP1/LiNK のダイアログボックスで環境変数を設定します。

  詳細については，「5.2　ユーザサーバの環境設定（SPP.NET）」を参照してください。

- PROPAGATE 翻訳指令に OFF を指定した SPP.NET で，CobolException.CobolRuntimeError（実行時エラー）が発生した場合，ユーザサーバが異常終了します。

  CobolException.CobolRuntimeError については，マニュアル「COBOL2002 for .NET Framework　ユーザーズガイド」を参照してください。

- 環境変数 NETCBL_SYSOUT および NETCBL_SYSERR を設定して，COBOL2002 for .NET Framework のランタイムが出力する標準出力および標準エラー出力を任意のファイルにリダイレクトさせてください。この設定を省略すると，標準出力および標準エラー出力がファイルにリダイレクトされないため，実行時エラーの内容がわかりません。

  TP1/Server Base を使用する場合は，TP1/Server Base の標準出力リダイレクト機能を利用して，標準出力および標準エラー出力をファイルにリダイレクトできます。標準出力リダイレクト機能については，TP1/Server Base の「Windows 版ご使用上の注意事項」を参照してください。TP1/LiNK には，標準出力および標準エラー出力をリダイレクトするための機能がありません。

2.2 アプリケーションプログラミングインタフェース（API）の種類

SPP.NET および SUP.NET 上で利用できるアプリケーションプログラミングインタフェース（API）を次に示します。

- システム運用の管理
- 資源の排他制御※1
- ユーザジャーナルの取得※1
- メッセージログの出力
- メッセージ送受信※2
- 性能検証用トレース※3
- リモート API 機能
- リモートプロシジャコール
- リアルタイム統計情報の取得
- TAM ファイルサービス※1，※4
- トランザクション制御※5

注※1

TP1/Server Base の場合だけ使用できます。TP1/LiNK の場合は使用できません。

注※2

SUP.NET からは使用できません。

TP1/Server Base の場合は TP1/Message Control，TP1/NET/Library，および TP1/NET/TCP/IP が必要です。TP1/LiNK の場合は TP1/Messaging が必要です。

注※3

TP1/Extension 1 が必要です。

注※4

TP1/FS/Table Access が必要です。

注※5

OpenTP1 独自のインタフェースの関数だけ使用できます。

各 API の機能については，マニュアル「OpenTP1　プログラム作成の手引」を参照してください。

2.3 リモートプロシジャコール（RPC）

SPP.NET または SUP.NET は，ほかの UAP とリモートプロシジャコール（RPC）を使用して通信できます。

RPC とは，UAP プロセスからほかの UAP プロセスにサービスを要求し，サービスを要求された UAP プロセスが要求元の UAP に処理結果を返す通信です。

サービスを要求する UAP とサービスを提供する UAP は，クライアントとサーバの関係になります。サービスを要求する側の UAP をクライアント UAP，サービスを提供する側の UAP をサーバ UAP といいます。

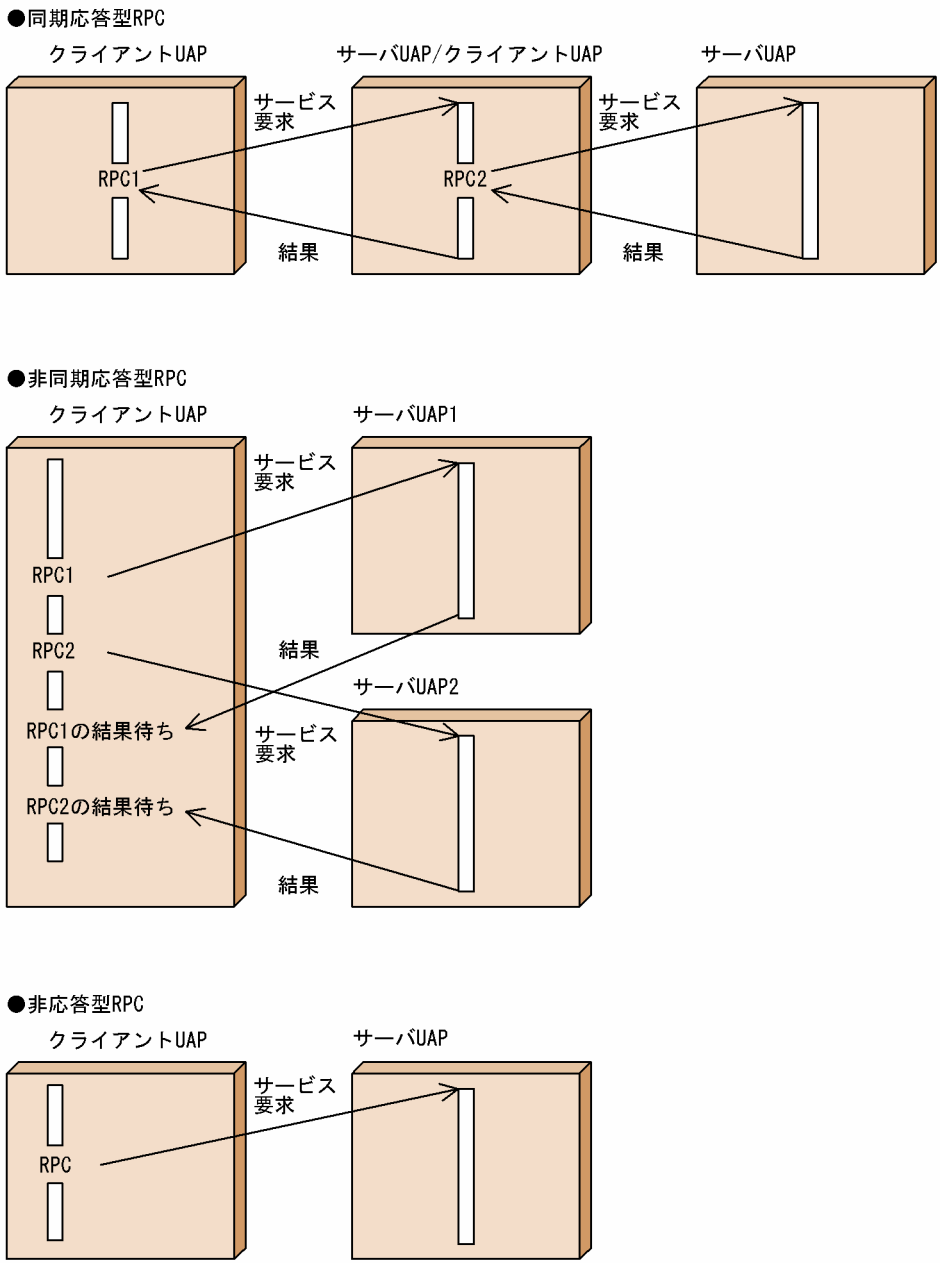
2.3.1 RPC の形態

Extension .NET の RPC には，次に示す 3 種類の形態があります。

- 同期応答型 RPC
- 非同期応答型 RPC
- 非応答型 RPC

RPC の形態別の処理概要を次の図に示します。

図 2-3　RPC の形態別の処理概要



(1)　同期応答型 RPC

クライアント UAP からサーバ UAP に問い合わせメッセージを送信し，問い合わせた応答メッセージを受け取る形態です。クライアント側ではサーバからの応答を受け取るまでメソッドがリターンしません。同一のサービスを複数回呼び出した場合，サーバプロセスはそのつどスケジュールされます。

(2)　非同期応答型 RPC

クライアント UAP からサーバ UAP に問い合わせメッセージを送信し，問い合わせた応答メッセージを待たないで処理を続ける形態です。

(3) 非応答型 RPC

クライアント UAP からサーバ UAP にメッセージを送信し，応答は受け取らない形態です。クライアント側ではサーバへのメッセージの送信が完了するとメソッドがリターンします。ただし，何らかの通信障害や呼び出したサービスが誤っていてもエラーを受け取ることができないため，注意が必要です。

2.3.2 RPC の連鎖（連鎖 RPC）

サーバ UAP の実行プロセスは，マルチサーバ（同じサーバ UAP を複数のプロセスで同時に起動する機能）の場合，サービスが要求されるたびに起動されます。一つのクライアント UAP から同じサービスグループを 2 回以上呼び出したとき，そのサービスグループのサーバ UAP が以前と同じプロセスで実行されるとは限りません。

ただし，同期応答型 RPC で，かつ同じサービスグループに属するサービスを 2 回以上要求する場合に限り，そのサービスを以前と同じプロセスで実行させることができます。これを**連鎖 RPC** といいます。

連鎖 RPC でサービスを要求すると，マルチサーバのサーバ UAP でも前回の RPC と同じプロセスで実行されるため，トランザクション処理に必要なプロセスを最小限にできます。UAP のプロセスはサービスグループごとに確保されるため，同じサービスグループに属していれば，異なるサービスに対しても一つのプロセスでサービスを実行できます。

なお，トランザクションとして連鎖 RPC を使用する場合は，一つのグローバルトランザクションで動作します。

2.3.3 RPC の形態による RPC インタフェースの使用可否

SPP.NET または SUP.NET からサービス要求をする場合の，RPC の形態による RPC インタフェースの使用可否を次の表に示します。

なお，RPC の形態については「2.3.1 RPC の形態」および「2.3.2 RPC の連鎖（連鎖 RPC）」を参照してください。RPC インタフェースの詳細については「1.3.2 RPC インタフェース」を参照してください。

表 2-1 RPC の形態による RPC インタフェースの使用可否

RPC の形態	RPC インタフェース		
	.NET インタフェース定義を使用した RPC	サービス定義（カスタムレコード）を使用した RPC	バイナリデータを使用した RPC
同期応答型 RPC	○	○	○
非同期応答型 RPC	×	×	○
非応答型 RPC	○※	○	○
連鎖 RPC	○	○	○

（凡例）

○：使用できます。 ×：使用できません。

注※

呼び出すメソッドの戻り値のデータ型が System.Void で，かつ引数が値渡しだけの場合に使用できます。それ以外の場合は RPC 要求時に例外が発生します。

2.3.4 RPC の種類

Extension .NET では，次の RPC を使用できます。

- リモート API 機能を使用した RPC
- ネームサービスを使用した RPC
- スケジューラダイレクト機能を使用した RPC
- 通信先を指定した RPC

(1) リモート API 機能を使用した RPC

リモート API 機能とは，クライアント側のノードにある UAP が発行した API を，OpenTP1 がサーバ側に転送してサーバ側のプロセスで代理実行する機能です。

リモート API 機能を要求するクライアント側のノードにある UAP を **rap クライアント**といいます。rap クライアントが発行した API を，OpenTP1 の **rap リスナー**が受け付け，**rap サーバ**がサーバ側のノードで実行します。rap リスナー，rap サーバは OpenTP1 のユーザサービスとして動作します。

Extension .NET では，リモート API 機能を使用した RPC を使用できます。

なお，リモート API 機能を使用する場合は，rap リスナー，rap サーバと常設コネクションを確立する必要があります。常設コネクションについては，「2.3.5 常設コネクション」を参照してください。

(2) ネームサービスを使用した RPC

OpenTP1 のネームサービスを使用した RPC を使用できます。

(3) スケジューラダイレクト機能を使用した RPC

スケジューラダイレクト機能とは，RPC の実行時に OpenTP1 のネームサービスを使用しないで，直接スケジューラサービスに問い合わせる機能です。

Extension .NET では，スケジューラダイレクト機能を使用した RPC を使用できます。

(4) 通信先を指定した RPC

サービス要求を実行するサーバ（通信先の OpenTP1 ノード）を明示的に指定して RPC を使用できます。

2.3.5 常設コネクション

OpenTP1 は，リモート API を要求した UAP（rap クライアント）と rap サーバとの間に，論理的な通信路を設定します。このような論理的な通信路を**常設コネクション**といいます。

これによって，rap クライアントと TP1/Server の rap サーバの間のコネクションを確立したままでメッセージを送受信できます。

2.3.6 RPC 送受信メッセージの最大長拡張機能

RPC 送受信メッセージの最大長拡張機能を使用すると，送受信できるメッセージの最大長を 1～8 メガバイトで指定できます。ただし，RPC 送受信メッセージの最大長拡張機能を使用しない場合または 1 メガバイトを指定した場合，RPC 送受信メッセージの最大長は，TP1ServerLimits クラスの DCRPC_MAX_MESSAGE_SIZE で定義した値になります。

(1) RPC 送受信メッセージの最大長拡張機能の使用方法

RPC 送受信メッセージの最大長拡張機能を使用する場合，次の指定をしてください。

(a) TP1/Server Base の場合

SPP.NET がサーバとして動作するとき，SPP.NET がクライアントとして動作するとき，または SUP.NET のときで指定方法が異なります。

SPP.NET がサーバとして動作するとき

次の二つの指定をします。

指定 1

TP1/Server Base のシステム共通定義の rpc_max_message_size オペランドで，送受信できるメッセージの最大長を指定します。

rpc_max_message_size オペランドについては，マニュアル「OpenTP1　システム定義」を参照してください。

指定 2

1. ユーザサービスデフォルト定義の次のオペランドで，送受信できるメッセージの最大長を指定します。
   - njs_input_max_message_size オペランド
   - njs_output_max_message_size オペランド

ユーザサービスデフォルト定義については，3 章の「ユーザサービスデフォルト定義」を参照してください。

2. ユーザサービス定義の次のオペランドで，送受信できるメッセージの最大長を指定します。
   - njs_input_max_message_size オペランド
   - njs_output_max_message_size オペランド

   ユーザサービス定義については，3 章の「ユーザサービス定義」を参照してください。

SPP.NET がクライアントとして動作するとき，または SUP.NET のとき

.NET インタフェース定義を使用するときは指定 1 および指定 2 を，.NET インタフェース定義を使用しないときは指定 1 だけを実施します。

指定 1

TP1/Server Base のシステム共通定義の rpc_max_message_size オペランドで，送受信できるメッセージの最大長を指定します。

rpc_max_message_size オペランドについては，マニュアル「OpenTP1　システム定義」を参照してください。

指定 2

if2cstub コマンドの-m オプションで，送受信できるメッセージの最大長を指定します。

if2cstub コマンドについては，6 章の「if2cstub（クライアントスタブ生成コマンド（.NET インタフェース定義用））」を参照してください。

(b)　TP1/LiNK の場合

SPP.NET がサーバとして動作するとき，SPP.NET がクライアントとして動作するとき，または SUP.NET のときで指定方法が異なります。

SPP.NET がサーバとして動作するとき

次の二つの指定をします。

指定 1

1. [システム環境設定] ウィンドウの［詳細設定(Q)...］ボタンをクリックします。

   [RPC 詳細設定] ダイアログが表示されます。

   [システム環境設定] ウィンドウの表示方法については，マニュアル「TP1/LiNK　使用の手引」を参照してください。

2. [その他] タブを選択します。
3. [RPC 送受信電文の最大長(M)] で，送受信できるメッセージの最大長を指定します。

指定 2

1. [SPP.NET 環境設定] ダイアログの［詳細設定(T)...］ボタンをクリックします。

   [SPP.NET 詳細設定] ダイアログが表示されます。

   [SPP.NET 環境設定] ダイアログの表示方法については，「5.2　ユーザサーバの環境設定(SPP.NET)」を参照してください。

2. [RPC] タブを選択します

3. [RPC 要求電文の最大長(R)] および「RPC 応答電文の最大長(P)」で，送受信できるメッセージの最大長を指定します。

SPP.NET がクライアントとして動作するとき，または SUP.NET のとき

.NET インタフェース定義を使用するときは指定 1 および指定 2 を，.NET インタフェース定義を使用しないときは指定 1 だけを実施します。

指定 1

1. [システム環境設定] ウィンドウの［詳細設定(Q)...］ボタンをクリックします。

   [RPC 詳細設定] ダイアログが表示されます。

   [システム環境設定] ウィンドウの表示方法については，マニュアル「TP1/LiNK　使用の手引」を参照してください。

2. [その他] タブを選択します。

3. [RPC 送受信電文の最大長(M)] で，送受信できるメッセージの最大長を指定します。

指定 2

if2cstub コマンドの-m オプションで，送受信できるメッセージの最大長を指定します。

if2cstub コマンドについては，6 章の「if2cstub（クライアントスタブ生成コマンド（.NET インタフェース定義用））」を参照してください。

(2)　注意事項

RPC 送受信メッセージの最大長拡張機能を使用する場合の注意事項を次に示します。

- RPC を使用したサービス要求時に RPC 送受信メッセージが最大長を超えた場合，例外（エラーコード TP1Error.DCRPCER_MESSAGE_TOO_BIG または TP1Error.DCRPCER_REPLY_TOO_BIG）が返ります。
- 次の場合，動作は保証できません。
  - 通信先を指定した遠隔サービス（Rpc クラスの CallTo メソッド）のスケジューラポート指定機能を使用した場合
  - ネームサービスを使わない通信（ユーザサービスネットワーク定義の dcsvdef コマンド）の場合

    dcsvdef コマンドについては，マニュアル「OpenTP1　システム定義」を参照してください。
- TP1/Server Base のシステム共通定義の rpc_max_message_size オペランドを指定する場合の前提条件および注意事項については，マニュアル「OpenTP1　システム定義」を参照してください。

2.4　トランザクション制御機能

Extension .NET から OpenTP1 のトランザクションを制御することができます。

2.4.1　トランザクション制御機能の概要

トランザクション制御機能を使用すると，アプリケーションでトランザクションを管理できます。そのため，処理が失敗した場合でも，データが失われたり，不整合が発生したりすることを避けられます。

(1)　トランザクションのコミットとロールバック

トランザクションとは，関連する複数の処理を一つのまとまった処理として扱うための論理的な単位です。一つのトランザクション内で実行した処理は，すべてが有効になるか，すべてが無効になるかのどちらかです。

トランザクションを有効にすることを，トランザクションの**コミット**といいます。トランザクションがコミットして初めてトランザクション処理の結果が有効になります。

トランザクションをコミットするかどうか最終的に決定するトランザクション処理の区切りを，**同期点**といいます。

トランザクションを無効にして，処理対象としていた資源をトランザクション開始直前の状態に戻すことを，トランザクションの**ロールバック**といいます。トランザクションをコミットできなかった場合や，処理の不整合を検出した場合などは，これまでの処理をロールバックで取り消して，データの整合性を保ちます。

(2)　グローバルトランザクション

UAP で RPC を実行すると，トランザクションは複数の UAP プロセスにわたります。

複数のプロセスで構成されるトランザクションを，**グローバルトランザクション**といいます。グローバルトランザクションを構成する各プロセスを，**トランザクションブランチ**といいます。特に，トランザクションの開始を宣言したプロセスを**ルートトランザクションブランチ**といいます。

(3)　連鎖モードと非連鎖モード

トランザクション処理の同期点取得には，一つのトランザクションの終了後，同期点を取得して次のトランザクションを続けて起動する**連鎖モードのコミット**と，トランザクションの終了で同期点を取得したあと，新たなトランザクションを起動しない**非連鎖モードのコミット**があります。

また同様に，**連鎖モードのロールバック**と，**非連鎖モードのロールバック**があります。連鎖モードのロールバックでは，ロールバック処理後も UAP プロセスはグローバルトランザクションの範囲内にありますが，非連鎖モードのロールバックでは，UAP プロセスはグローバルトランザクションの範囲外となります。

(4) RPC の形態と同期点の関係

RPC の形態と同期点の関係を次に示します。

(a) 同期応答型 RPC と同期点の関係

同期応答型 RPC で要求を行うトランザクション処理の場合，クライアント UAP に処理結果が戻って，同期点処理を終えた時点で，トランザクションの終了となります。

(b) 非応答型 RPC と同期点の関係

非応答型 RPC で要求を行うトランザクション処理の場合，クライアント UAP と，サーバ UAP の処理終了で同期を取ります。

2.4.2 トランザクションの開始と同期点取得

SPP.NET または SUP.NET から Trn クラスの Begin メソッドを呼び出して，トランザクションを開始します。

Begin メソッドを呼び出してから，同期点取得（コミット）までが，グローバルトランザクションの範囲となります。Begin メソッドを呼び出したあと，そのグローバルトランザクションの中で新たな Begin メソッドは呼び出せません。

なお，グローバルトランザクション中に Rpc クラスの Call メソッドで呼び出した SPP.NET および SPP のサービスは，ユーザサービス定義の設定によって，自動的にグローバルトランザクションの範囲にできます。

2.4.3 同期点取得

(1) コミット

トランザクションが正常終了したときの同期点取得（コミット）は，SPP.NET または SUP.NET から Trn クラスの Commit メソッドを呼び出して行います。

グローバルトランザクションは，すべてのトランザクションブランチが正常に終了したことで正常終了となります。

(a) 連鎖，非連鎖モードでのコミット

トランザクション処理の同期点取得には，次の 2 種類があります。

- 連鎖モードのコミット
  連鎖モードのコミットは，Trn クラスの CommitChained メソッドを呼び出して要求します。

- 非連鎖モードのコミット
  非連鎖モードのコミットは，Trn クラスの Commit メソッドを呼び出して要求します。

(b) コミット要求メソッドを呼び出さない場合の処理

次の場合，トランザクションはロールバックされます。

- Trn クラスの Commit メソッドを呼び出さないで SPP.NET または SUP.NET が終了したとき
- Trn クラスの Commit メソッドを呼び出す前に SPP.NET または SUP.NET が異常終了したとき

(2) ロールバック

(a) TP1/Server の処理でのエラーの場合

トランザクションでエラーが発生すると，Trn クラスの Commit メソッドで例外が発生します。そのトランザクションは部分回復対象としてロールバックされます。グローバルトランザクション内のどれか一つのトランザクションブランチでエラーが発生した場合でも，グローバルトランザクション全体がロールバックの対象となります。

このとき TP1/Server は，トランザクションブランチをロールバック対象とみなして，部分回復処理をします。

(b) ロールバック要求メソッドを呼び出す場合

トランザクションを SPP.NET または SUP.NET の判断でロールバックしたいときは，SPP.NET または SUP.NET からロールバック要求のメソッドを呼び出します。

トランザクション処理のロールバックには，次の 2 種類があります。

- 連鎖モードのロールバック
  連鎖モードのロールバックは，Trn クラスの RollbackChained メソッドを呼び出して要求します。RollbackChained メソッドを呼び出してロールバックすると，このメソッドを呼び出した SPP.NET および SUP.NET のプロセスは，ロールバック処理後も，グローバルトランザクションの範囲内にあります。
- 非連鎖モードのロールバック
  非連鎖モードのロールバックは，Trn クラスの Rollback メソッドを呼び出して要求します。Rollback メソッドを呼び出してロールバックすると，このメソッドを呼び出した SPP.NET および SUP.NET のプロセスは，ロールバック処理後，グローバルトランザクションの範囲外となります。

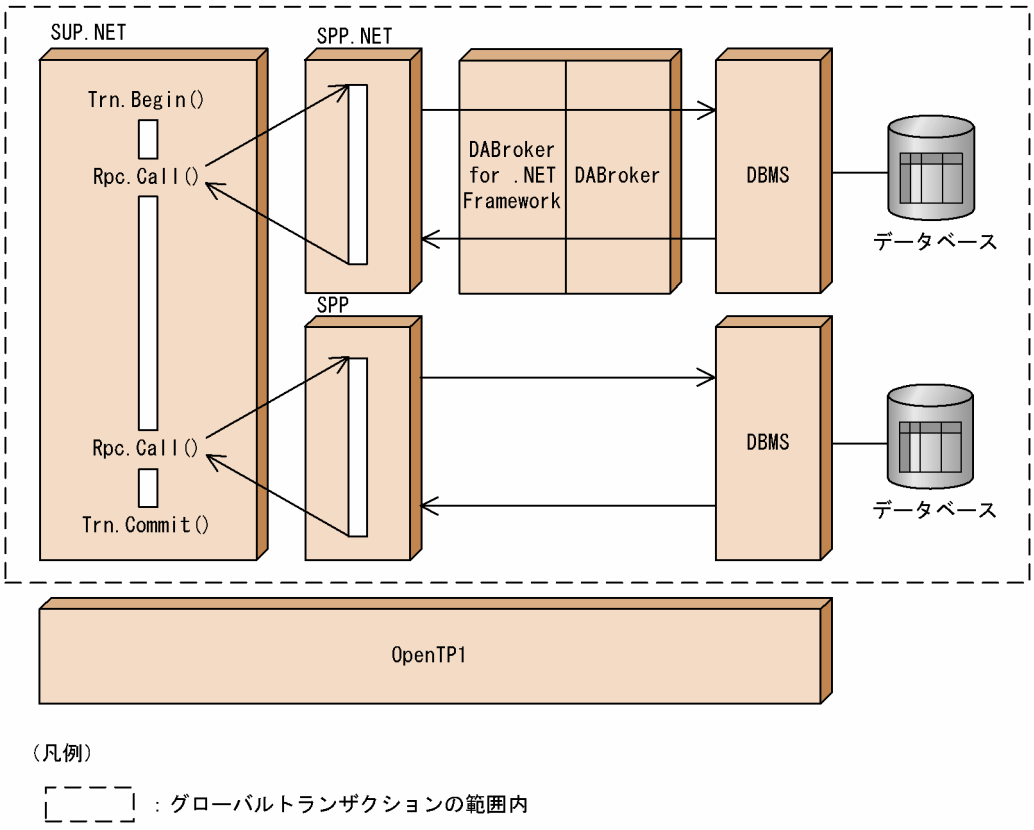
2.4.4　DBMS との連携

DABroker for .NET Framework は，.NET Framework のデータベースアクセス技術である ADO.NET に準拠した.NET Framework データプロバイダです。DABroker for .NET Framework を使用することで，X/Open に準拠した XA インタフェースによって，DBMS とのトランザクション連携ができます。

ADO.NET の.NET Framework データプロバイダ，または DBMS が提供する.NET Framework 用のデータプロバイダを使用する場合は，OpenTP1 のグローバルトランザクションに参加できません。この場合，DBMS とのローカルトランザクションだけが使用できます。そのため，グローバルトランザクションの開始および終了（コミット，ロールバック）はできますが，ローカルトランザクションとは別のトランザクションとして動作します。なお，OpenTP1 for .NET Framework の UAP 以外のサーバアプリケーション（SPP など）を呼び出す場合は，グローバルトランザクションと連携してリソースマネジャの更新処理を実行できます。

DBMS との連携を次の図に示します。

図 2-4　DBMS との連携



連携できる DBMS とそのバージョンについては，「Readme ファイル」，およびマニュアル「DABroker for .NET Framework」を参照してください。

2.4.5　トランザクションシーケンス

OpenTP1 for .NET Framework と DBMS との連携で使用するメソッドは，次に示す順番で発行します。

1. DBMS とのコネクションの生成，オープン
2. データベースアクセス処理
3. コネクションの解放

XA インタフェースによるトランザクション連携は，Extension .NET が提供する Rpc クラスおよび Trn クラスと，DABroker for .NET Framework データプロバイダのクラスを組み合わせることで実現します。このときのメソッドの正しい発行順序を，**トランザクションシーケンス**と呼びます。UAP を作成するユーザは，トランザクションシーケンスを意識する必要があります。トランザクションシーケンスが不正な場合は，次のような障害が発生する可能性があるため，注意してください。

- DABroker for .NET Framework が提供するクラスの呼び出しに失敗する
- データベースが正しく更新できない

(1)　SPP.NET のトランザクションシーケンス

SPP.NET には，次の 3 種類のメソッドが実装されています。

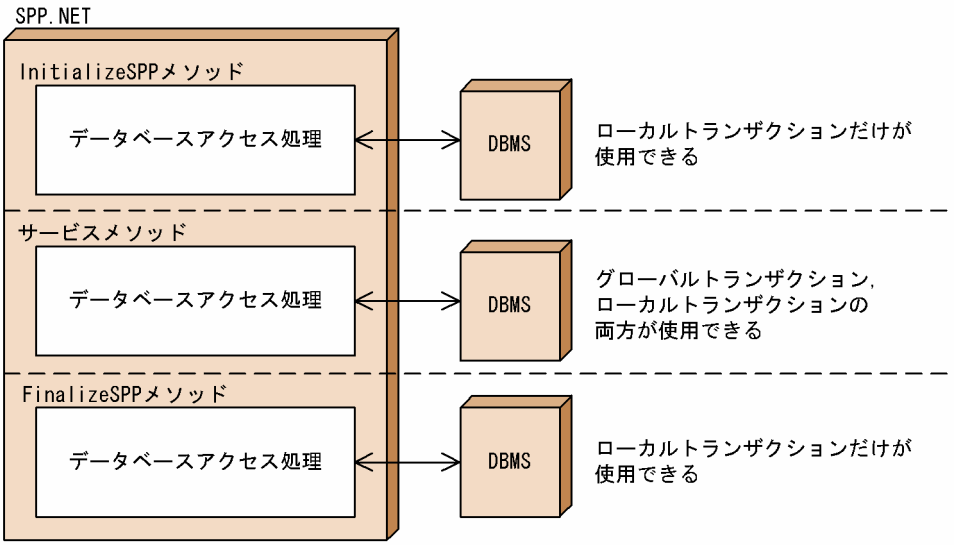
- InitializeSPP メソッド
- FinalizeSPP メソッド
- サービスメソッド

これらのメソッドの中で，OpenTP1 のグローバルトランザクションに参加できるメソッドはサービスメソッドだけです。そのため，XA インタフェースによるトランザクション連携をする DBMS に対してのデータベースアクセス処理は，サービスメソッド内で実行してください。InitializeSPP メソッドおよび FinalizeSPP メソッド内では，XA インタフェースによるトランザクション連携をする DBMS に対してのデータベースアクセス処理はできません。また，コネクションの生成およびオープンは，グローバルトランザクションの状態に関係なく実行できます。

なお，XA インタフェースによるトランザクション連携をしない DBMS に対するデータベースアクセス処理は，3 種類のメソッド中のどのメソッドでも実行できます。

SPP.NET のトランザクション連携種別を次の図に示します。

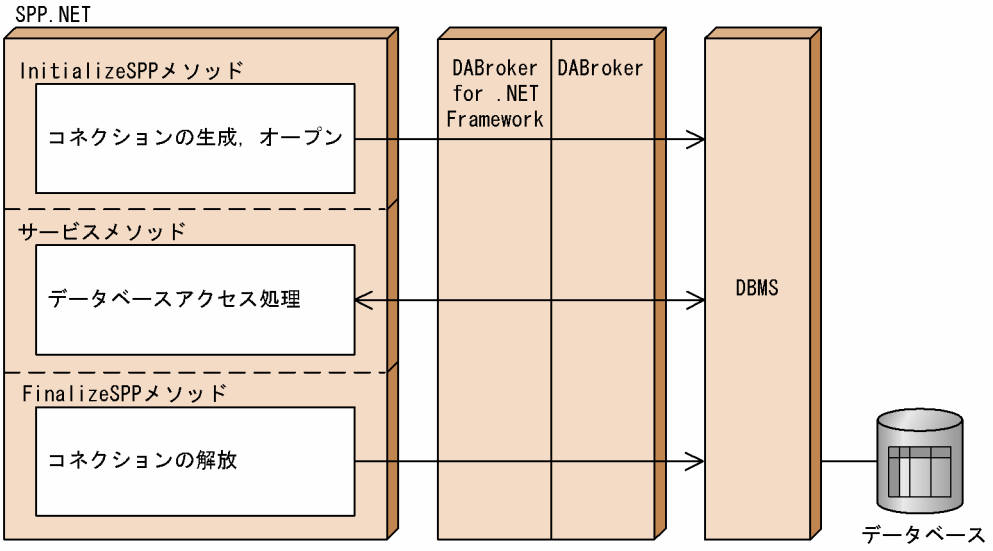
図 2-5　SPP.NET のトランザクション連携種別



(a)　InitializeSPP メソッド内で DBMS と接続する場合

InitializeSPP メソッド内で DBMS と接続する場合のメソッドの発行位置を，次の図に示します。

図 2-6　InitializeSPP メソッド内で DBMS と接続する場合のメソッドの発行位置



1. InitializeSPP メソッド内でコネクションを生成，オープンします。
2. サービスメソッド内でデータベースアクセス処理をします。
3. FinalizeSPP メソッド内でコネクションを解放します。

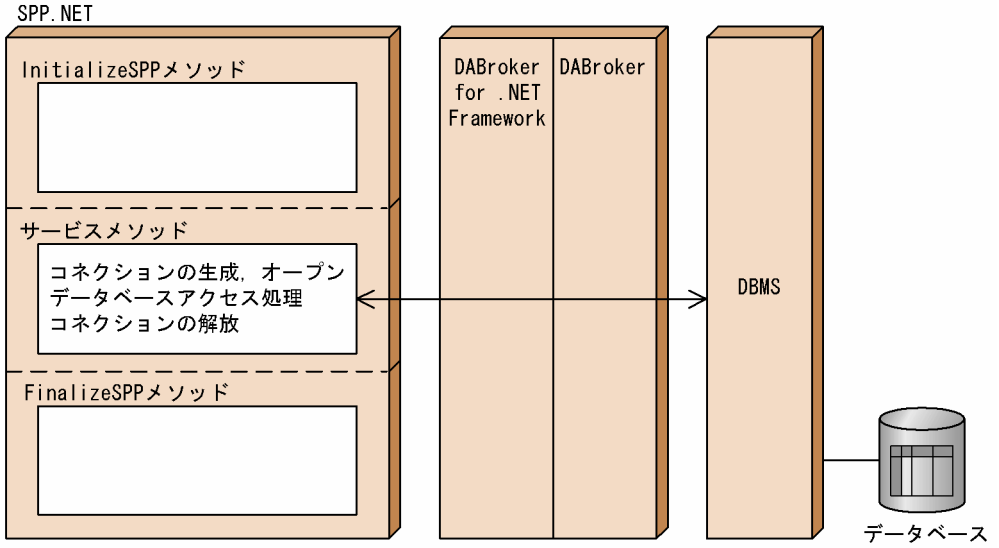
InitializeSPP メソッド内で作成したコネクションは，FinalizeSPP メソッドまで使用できます。

DABroker for .NET Framework を使用して DBMS とトランザクション連携をする場合は，この方式での接続を推奨します。

(b)　サービスメソッド内で DBMS と接続する場合

サービスメソッド内で DBMS と接続する場合は，次の図に示すとおり，コネクションの生成およびオープンだけでなく，データベースアクセス処理，およびコネクションの解放もサービスメソッド内で実行します。

図 2-7　サービスメソッド内で DBMS と接続する場合のメソッドの発行位置



この場合，サービスメソッドを実行する場合にだけ DBMS と接続するので，リソースの消費を最小限に抑えられます。ただし，コネクションの生成，オープン，および解放をサービスメソッドの呼び出しごとに実行するため，処理に伴うオーバヘッドを考慮する必要があります。

(2)　SUP.NET のトランザクションシーケンス

SUP.NET は，次に示す流れに沿って処理をします。

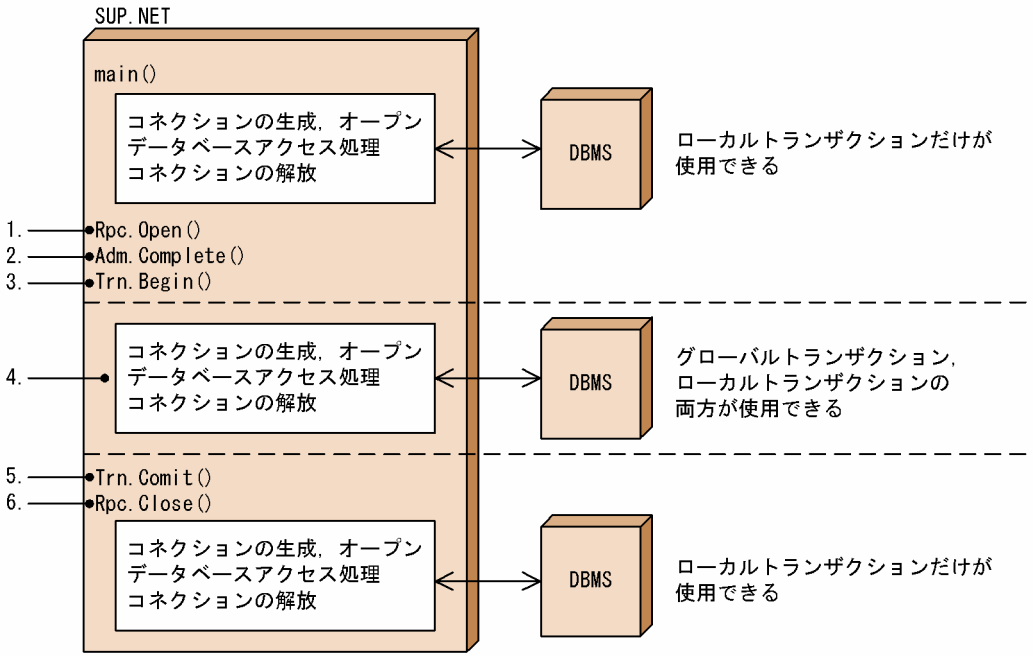
1. RPC のオープン
2. SUP.NET の開始処理の完了報告
3. トランザクションの開始
4. サービスの実行
5. トランザクションの終了
6. RPC のクローズ

XA インタフェースによるトランザクション連携をする DBMS に対してのデータベースアクセス処理は，4.の部分でだけ実行できます。この場合，コネクションの生成およびオープンも，4.の部分で実行します。

XA インタフェースによるトランザクション連携をしない DBMS に対してのデータベースアクセス処理は，SUP.NET のメイン関数内であればどこでも実行できます。コネクションの生成およびオープンも同様です。

SUP.NET のトランザクション連携種別を次の図に示します。

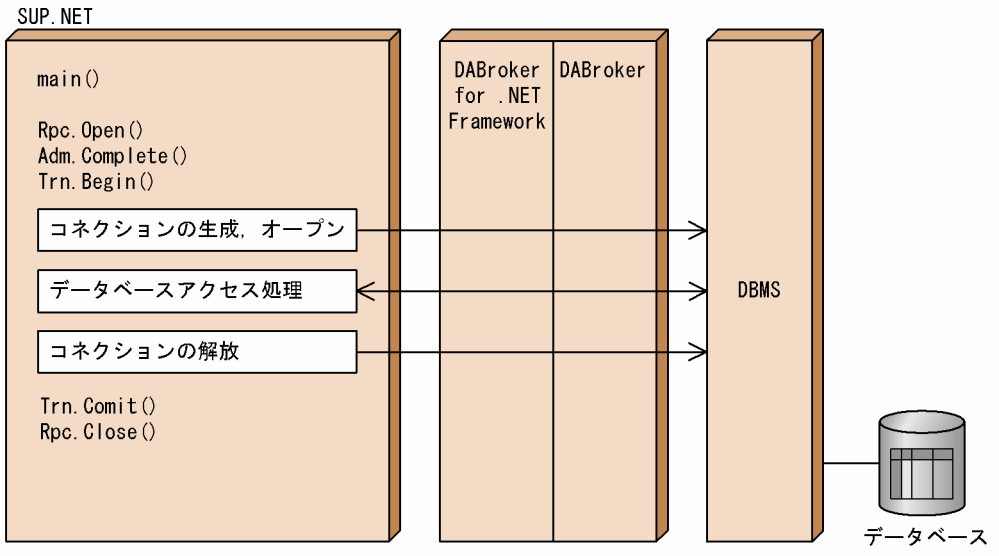
図 2-8　SUP.NET のトランザクション連携種別



1. RPC のオープン
2. SUP.NET の開始処理の完了報告
3. トランザクションの開始
4. サービスの実行
5. トランザクションの終了
6. RPC のクローズ

DBMS と XA インタフェースによるトランザクション連携をする場合，SUP.NET のトランザクションシーケンスは，次の図のとおりになります。

図 2-9　SUP.NET のトランザクションシーケンス



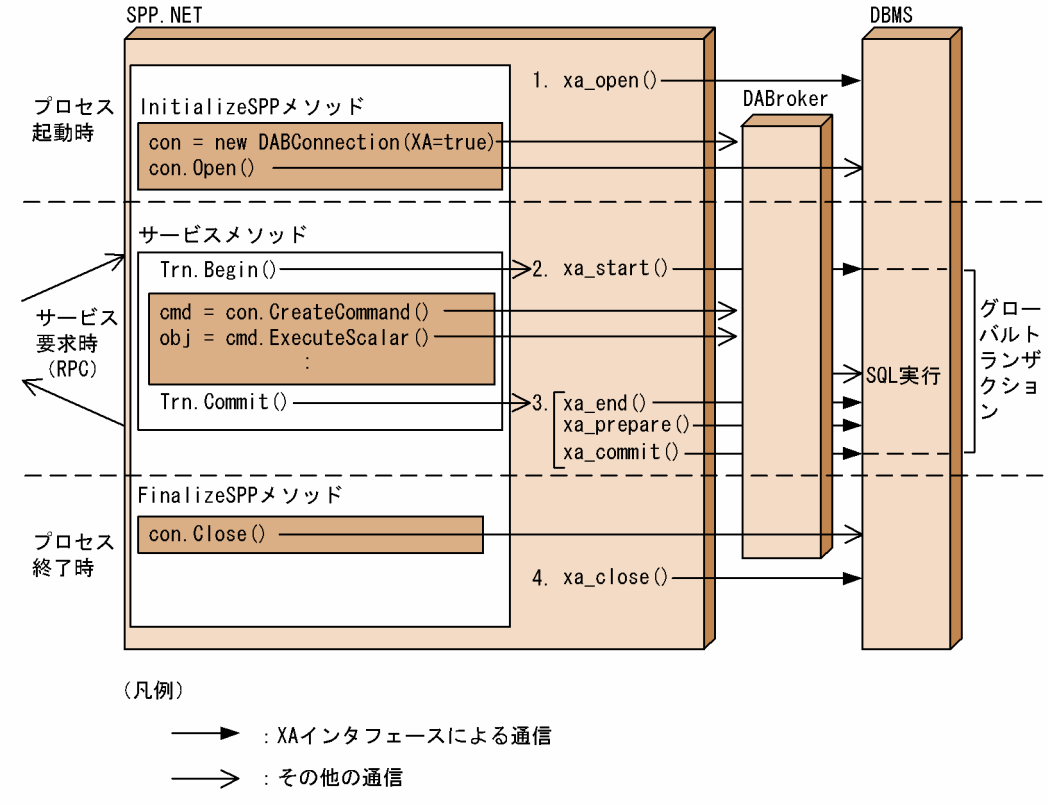
2.4.6　DBMS とのトランザクション連携の仕組み

DABroker for .NET Framework を使用して DBMS とトランザクション連携をする場合，OpenTP1 は DBMS に対するトランザクションの開始，終了などの通知に，XA インタフェースを使用します。

(1)　SPP.NET とのトランザクション連携

SPP.NET とトランザクション連携をするときに通知される XA インタフェースの発行例を次の図に示します。

図 2-10　SPP.NET とトランザクション連携をするときに通知される XA インタフェースの発行例

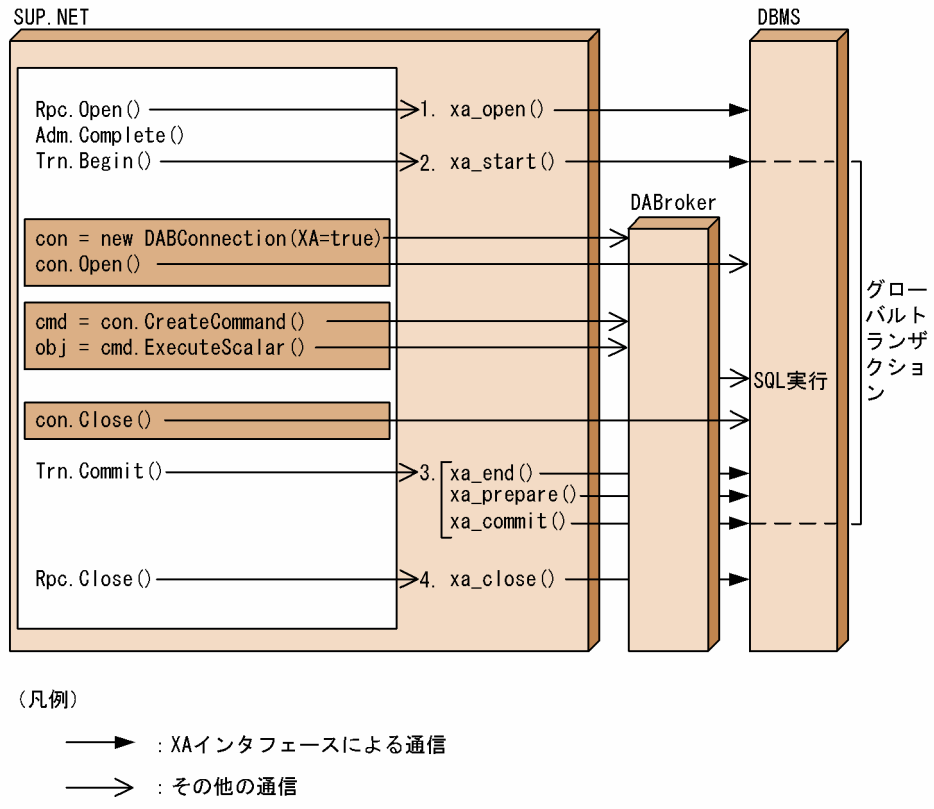


1. SPP.NET 実行コンテナから発行された dc_rpc_open 関数の延長で，トランザクション制御用のライブラリに指定した DBMS に xa_open 関数を発行します。
2. Trn.Begin メソッドを発行すると，DBMS に xa_start 関数を発行します。
3. Trn.Commit メソッドを発行すると，DBMS に xa_end 関数，xa_prepare 関数，xa_commit 関数の順で関数を発行します。なお，この例ではトランザクションの最適化は考慮しません。
4. SPP.NET 実行コンテナから発行された dc_rpc_close 関数の延長で，トランザクション制御用のライブラリに指定した DBMS に xa_close 関数を発行します。

(2)　SUP.NET とのトランザクション連携

SUP.NET とトランザクション連携をするときに通知される XA インタフェースの発行例を次の図に示します。

図 2-11　SUP.NET とトランザクション連携をするときに通知される XA インタフェースの発行例



1. Rpc.Open メソッドを発行すると，トランザクション制御用のライブラリに指定した DBMS に xa_open 関数を発行します。
2. Trn.Begin メソッドを発行すると，DBMS に xa_start 関数を発行します。
3. Trn.Commit メソッドを発行すると，DBMS に xa_end 関数，xa_prepare 関数，xa_commit 関数の順で関数を発行します。なお，この例ではトランザクションの最適化は考慮しません。
4. Rpc.Close メソッドを発行すると，トランザクション制御用のライブラリに指定した DBMS に xa_close 関数を発行します。

2.4.7　トランザクション連携の利用手順

OpenTP1 のアプリケーションでトランザクション連携をするときの手順を次に示します。利用するリソースマネジャが OpenTP1 のリソースマネジャだけの場合と，DBMS を利用する場合とで手順が異なります。なお，DBMS を利用する場合は，DABroker および DABroker for .NET Framework が必要です。

(1)　OpenTP1 のリソースマネジャだけを利用する場合

(a)　TP1/Server Base のとき

1. TP1/Server Base にリソースマネジャを登録します。

詳細については，マニュアル「OpenTP1　プログラム作成の手引」を参照してください。

2. njsmkdll コマンドを実行して，トランザクション制御用ライブラリを作成します。
njsmkdll コマンドについては，TP1/Server Base の「Windows 版ご使用上の注意事項」を参照してください。

3. トランザクション連携をする SPP.NET または SUP.NET の環境設定をします。
ユーザサービス定義で njs_xa_connect オペランドおよび njs_xa_dllname オペランドを指定します。ユーザサービス定義の定義内容については，3 章の「ユーザサービス定義」を参照してください。

4. TP1/Server Base を起動します。

5. ユーザサーバを起動します。

(b)　TP1/LiNK のとき

1. TP1/LiNK にリソースマネジャを登録します。
TP1/LiNK が提供する［リソースマネジャ］ダイアログボックスを使用して登録します。［リソースマネジャ］ダイアログボックスについては，「5.4　リソースマネジャの接続」を参照してください。

2. トランザクション連携をする SPP.NET または SUP.NET の環境設定をします。
［SPP.NET 環境設定］ダイアログボックスまたは［SUP 環境設定］ダイアログボックスから XA 接続の設定をします。XA 接続の設定については，「5.2.4　XA 接続の設定」を参照してください。

3. TP1/LiNK を起動します。

4. ユーザサーバを起動します。

(2)　DBMS を利用する場合

(a)　TP1/Server Base のとき

1. TP1/Server Base に DBMS を登録します。
詳細については，マニュアル「OpenTP1　プログラム作成の手引」を参照してください。

2. njsmkdll コマンドを実行して，トランザクション制御用ライブラリを作成します。
njsmkdll コマンドについては，TP1/Server Base の「Windows 版ご使用上の注意事項」を参照してください。

3. DABroker の環境設定，および接続先データベースの定義を設定します。
設定方法については，マニュアル「DABroker」を参照してください。

4. トランザクション連携をする SPP.NET または SUP.NET の環境設定をします。
ユーザサービス定義で njs_xa_connect オペランドおよび njs_xa_dllname オペランドを指定します。ユーザサービス定義の定義内容については，3 章の「ユーザサービス定義」を参照してください。

5. TP1/Server Base を起動します。

6. ユーザサーバを起動します。

(b) TP1/LiNK のとき

1. TP1/LiNK に DBMS を登録します。

   TP1/LiNK が提供する［リソースマネジャ］ダイアログボックスを使用して登録します。［リソースマネジャ］ダイアログボックスについては，「5.4 リソースマネジャの接続」を参照してください。

2. DABroker の環境設定，および接続先データベースの定義を設定します。

   設定方法については，マニュアル「DABroker」を参照してください。

3. トランザクション連携をする SPP.NET または SUP.NET の環境設定をします。

   ［SPP.NET 環境設定］ダイアログボックスまたは［SUP 環境設定］ダイアログボックスから XA 接続の設定をします。XA 接続の設定については，「5.2.4 XA 接続の設定」を参照してください。

4. TP1/LiNK を起動します。

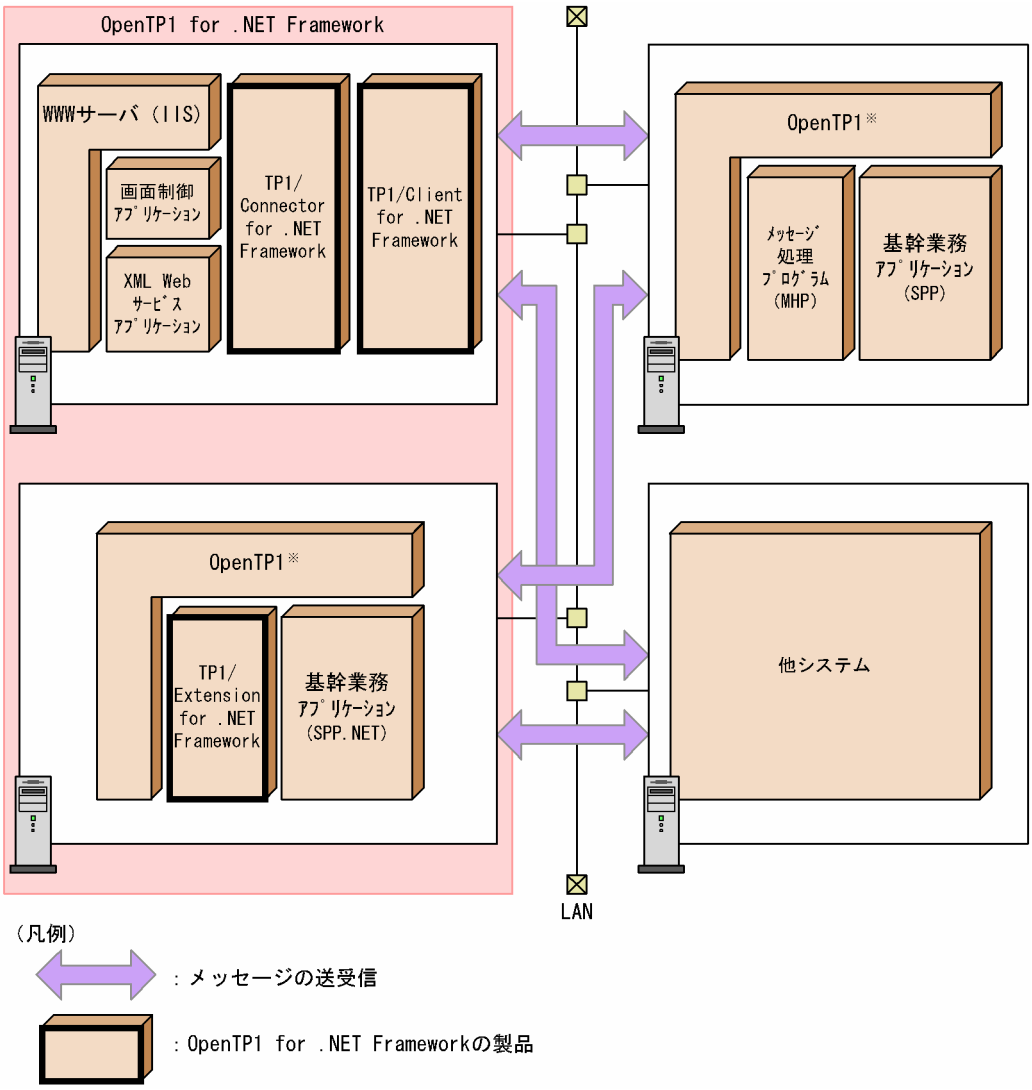
5. ユーザサーバを起動します。

2.5　メッセージ送受信機能

Extension .NET では，**メッセージ送受信機能**を使用することで，TCP/IP プロトコルによって OpenTP1 の MHP や OpenTP1 以外のシステムと通信できます。

メッセージ送受信機能による通信の例を次に示します。

図 2-12　メッセージ送受信機能による通信の例



注※

TP1/NET/TCP/IP，TP1/Messaging など，メッセージ制御機能を提供する OpenTP1 システムのプログラムを組み込む必要があります。

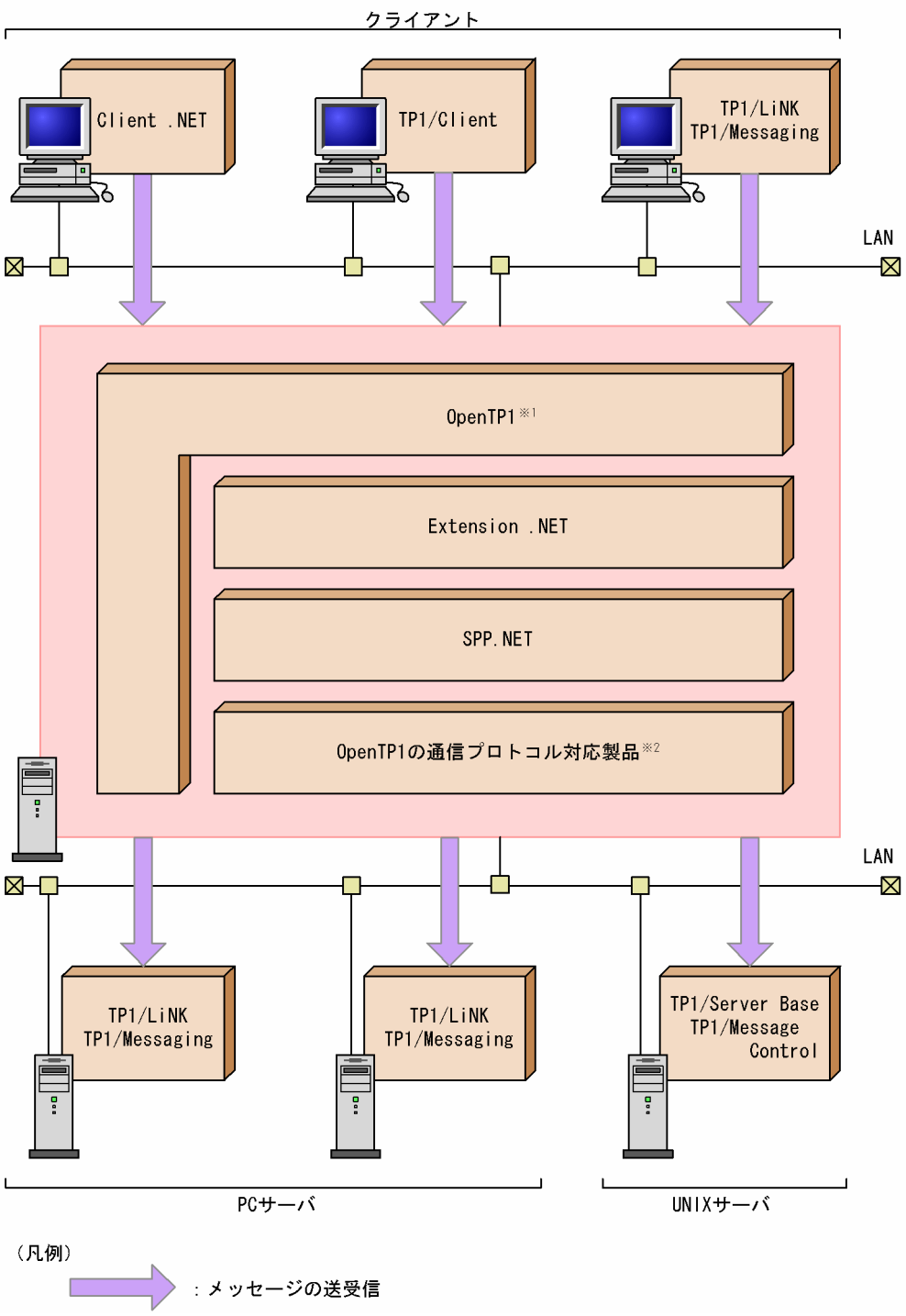
Extension .NET でメッセージ送受信機能を使用すると，SPP.NET は次のサーバおよびクライアントと通信できます。

- OpenTP1 のメッセージ制御機能（MCF）を使用した，PC および UNIX のサーバ
- TP1/Client を使用した，PC および UNIX のクライアント

メッセージ送受信機能を使用するためには，Extension .NET のほかに，メッセージ制御機能を提供する OpenTP1 システムのプログラム（TP1/NET/TCP/IP，TP1/Messaging など）を組み込む必要があります。

メッセージ送受信機能を使用する場合のシステム構成例を次に示します。

図 2-13　メッセージ送受信機能を使用する場合のシステム構成例



注※1

TP1/Server Base または TP1/LiNK が使用できます。

注※2

【TP1/Server Base】

TP1/NET/TCP/IP が使用できます。

【TP1/LiNK】

TP1/Messaging が使用できます。

メッセージ送受信機能の通信プロトコルには TCP/IP が使用されます。SPP.NET の通信相手には，MHP または任意のソケットアプリケーションが使用できます。

なお，Extension .NET を使用したメッセージ送受信機能は，SPP.NET からだけ使用できます。SUP.NET からこの機能を使用することはできません。

2.5.1 メッセージ送受信機能の使用方法

メッセージ送受信機能を使用する場合の手順を次に示します。

1. あらかじめ，メッセージ制御機能を提供する OpenTP1 システムのプログラム（TP1/NET/TCP/IP，TP1/Messaging など）をシステムに組み込んでおきます。また，必要に応じて，リソースマネジャを登録しておいてください。

2. サービスメソッドに，Mcf クラスが提供するメッセージ送受信の処理を記述します。

3. TP1/Server Base または TP1/LiNK を起動します。

   【TP1/Server Base】

   SPP.NET のユーザサービス定義の njs_use_mcf オペランドに Y を指定した状態で起動します。

   【TP1/LiNK】

   ［SPP.NET 詳細設定］ダイアログボックスの［その他］タブにある，［メッセージ送受信機能を使用する(M)］のチェックボックスをオンにした状態で起動します。［SPP.NET 詳細設定］ダイアログボックスの［その他］タブの詳細については，「5.2.1(5) ［SPP.NET 詳細設定］ダイアログボックス（［その他］タブ）」を参照してください。

4. サービス要求によって SPP.NET のサービスメソッドを呼び出します。

2.5.2 通信形態

メッセージ送受信機能を使用すると，SPP.NET と MHP などの相手システムとの間で，次の形態によるメッセージの送受信ができます。

- SPP.NET から相手システムへのメッセージの一方送信
- SPP.NET と相手システムとのメッセージの送受信

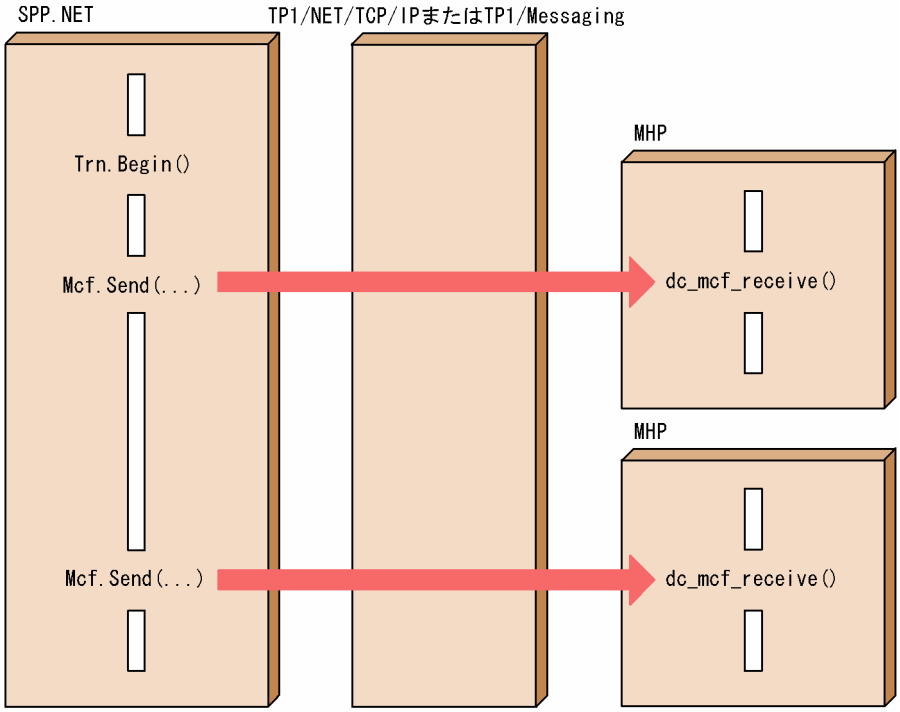
(1)　メッセージの一方送信

SPP.NET から相手システムに対して一方的にメッセージを送信できます。これを**メッセージの一方送信**といいます。

SPP.NET で Mcf クラスの Send メソッドを呼び出して，相手システムへメッセージを送信します。なお，メッセージの一方送信はトランザクション内で使用できます。

通信プロトコルに TCP/IP を使用して，MHP にメッセージを一方送信する形態の例を次の図に示します。

図 2-14　メッセージの一方送信（SPP.NET）



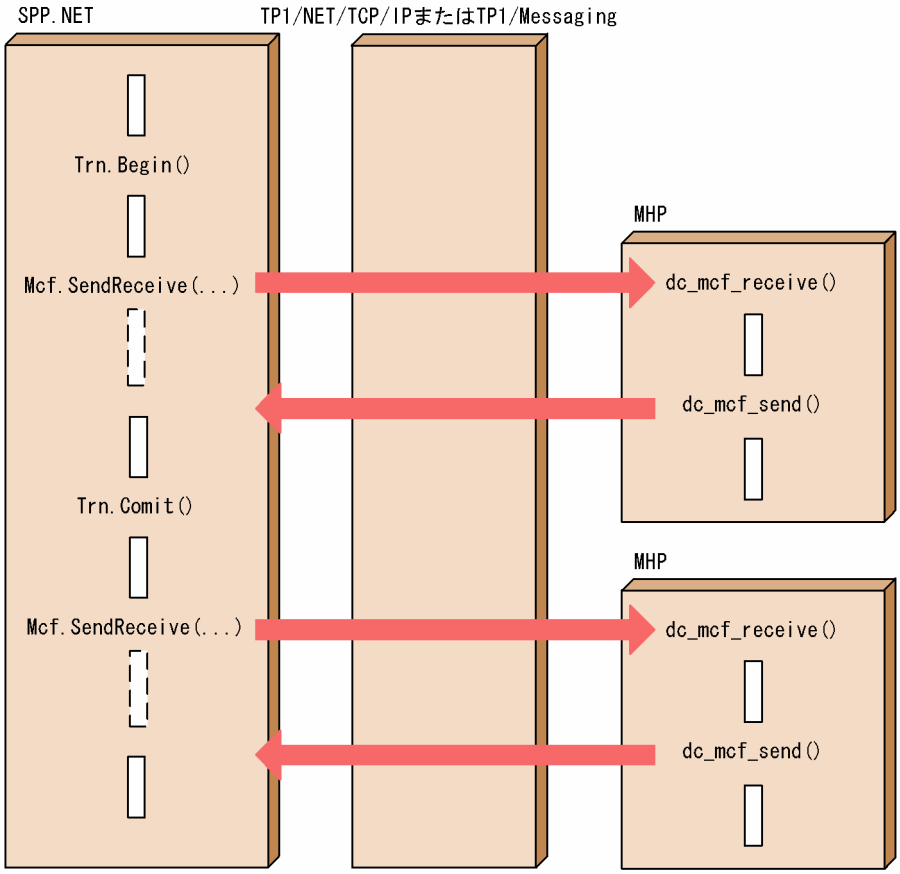
(2)　メッセージの送受信

SPP.NET と相手システムとの間で，メッセージを送受信できます。

SPP.NET で Mcf クラスの SendReceive メソッドを呼び出して相手システムへメッセージを送信し，相手システムからのメッセージを受信します。メッセージの一方送信時は，それぞれ別のコネクションを使用してメッセージを送信しますが，メッセージ送受信時は，同じコネクションを使用してメッセージを送受信します。なお，メッセージの送受信は，トランザクション内でもトランザクション外でも使用できます。

通信プロトコルに TCP/IP を使用して，MHP とメッセージを送受信する形態の例を次の図に示します。

図 2-15　メッセージの送受信（SPP.NET）



2.5.3　注意事項

(1)　メッセージ受信時の注意事項

(a)　メッセージ長

通信プロトコルに TCP/IP を使用する場合，一つのメッセージを複数のパケットに分割したり，複数のメッセージを一つのパケットに詰め込んだりします。そのため，ユーザが指定するメッセージ長以外に，受信したメッセージの切れ目を判断できません。ユーザはメッセージ長を含めた固定長のヘッダを最初に受信し，ヘッダに含まれているメッセージ長を指定して，実際のメッセージを受信してください。

指定したメッセージ長より短いメッセージを受信した場合，SPP.NET は，メッセージが分割されているものとみなします。そのため，指定した長さ分のメッセージを受信するまで，SPP.NET に制御を戻しません。

タイムアウトやエラーの発生によって，指定した長さ分のメッセージを受信していない場合でも，その時点までのメッセージを受信できます。

(2)　メッセージ送信時の注意事項

(a)　障害発生時のメッセージの消失

OS が管理する TCP/IP の送信バッファへのメッセージ書き込みが正常終了したあとで，OS 内部で実際の送信が完了するまでの間に通信障害が発生した場合，SPP.NET および相手システムではメッセージ送信の障害を検出できません。メッセージ送受信の場合は受信時に障害検知やタイムアウトによって検知できますが，一方送信の場合は正常終了します。したがって，ユーザはあらかじめメッセージ中に通番を付けるなどして，障害に備えてください。

(3)　その他の注意事項

通信プロトコルに TCP/IP を使用する場合，TP1/NET/TCP/IP または TP1/Messaging の受信メッセージの組み立て機能を使用すると，相手システムが送信したデータの先頭に 4 バイトのメッセージ長が付与されます。また，相手システムにデータを送信する場合は，先頭 4 バイトにメッセージ長を設定する必要があります。メッセージ長は，ネットワークバイトオーダーにしてください。なお，このメッセージ長は，相手システムが受信するときには削除されます。SPP.NET では，送信時および受信時に，メッセージ長を意識する必要があるので注意してください。TP1/NET/TCP/IP または TP1/Messaging の受信メッセージの組み立て機能を使用する場合は，TP1/NET/TCP/IP または TP1/Messaging のプロトコル固有定義で次のように指定します。

```
mcftalccn -u masm=yes
```

TP1/NET/TCP/IP または TP1/Messaging のプロトコル固有定義の詳細については，マニュアル「OpenTP1　プロトコル　TP1/NET/TCP/IP 編」を参照してください。

2.6 ノード間負荷バランス機能

ノード間負荷バランス機能は，RPC による要求が特定のノードに集中しないようにノード間で負荷を分散する機能です。ノード間負荷バランス機能を使用するためには，負荷分散の前提として次の条件を満たしている必要があります。

- 複数のノードに同一のサービスを提供するユーザサーバが起動されていること。
- 各 OpenTP1 ノードはシステム共通定義の all_node オペランドに自分以外のノードを定義して，お互いの OpenTP1 ノードで起動されているユーザサーバの情報（ネーム情報）をやり取りしていること。

2.7　リアルタイム統計情報の取得

UAP 内の任意区間での処理の実行時間および実行回数を，**リアルタイム統計情報**として取得できます。オフラインの業務をする UAP では，任意区間でのリアルタイム統計情報は取得できません。リアルタイム統計情報の取得方法については，マニュアル「OpenTP1　プログラム作成の手引」を参照してください。

Extension .NET では，この機能を使用するために Rts クラスを提供しています。

2.8　資源の排他制御【TP1/Server Base】

OpenTP1 では，複数のユーザが資源を共用できるようにしています。資源を管理して整合性を保つことができる機能を**排他制御**といいます。資源の排他制御については，マニュアル「OpenTP1　解説」を参照してください。

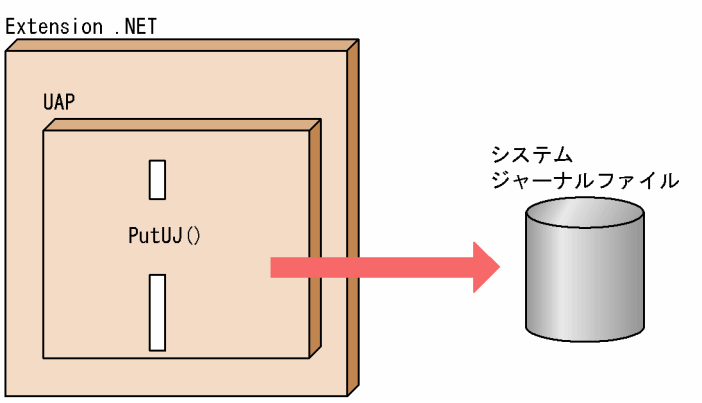
Extension .NET では，この機能を使用するために Lck クラスを提供しています。このクラスは，TP1/Server Base の UAP でだけ使用できます。TP1/LiNK の UAP では使用できません。

2.9　ユーザジャーナルの取得【TP1/Server Base】

UAP の中で UAP 履歴情報取得機能を使用し，任意の情報を**ユーザジャーナル（UJ）**としてシステムジャーナルファイルに取得できます。ユーザジャーナルの取得方法を次の図に示します。ユーザジャーナルの取得方法については，マニュアル「OpenTP1 プログラム作成の手引」を参照してください。

Extension .NET では，この機能を使用するために Jnl クラスを提供しています。このクラスは，TP1/Server Base の UAP でだけ使用できます。TP1/LiNK の UAP では使用できません。

図 2-16　ユーザジャーナルの取得方法



2.10 TAM ファイルサービス（TP1/FS/Table Access）【TP1/Server Base】

TAM ファイルとは，OpenTP1 専用のユーザファイルのことです。TAM ファイルを使うと，メモリ上のテーブルから OpenTP1 ファイル（直接編成ファイル）へ高速にアクセスできます。TAM ファイルを使う場合は，あらかじめ，システムに TP1/FS/Table Access を組み込んでおきます。TAM ファイルサービスについては，マニュアル「OpenTP1 プログラム作成の手引」を参照してください。

Extension .NET では，この機能を使用するために Tam クラス，TamKeyTable 構造体，および TamStatusTable 構造体を提供しています。このクラスは，TP1/Server Base の UAP でだけ使用できます。TP1/LiNK の UAP では使用できません。

2.11　環境設定

Extension .NET のインストール後に必要な，環境設定やセキュリティポリシーの設定などについて説明します。

2.11.1　インストール後の環境設定

Visual Studio で UAP を開発する場合は，必要に応じて次に示すアセンブリを UAP のプロジェクトの参照設定に追加してください。

%DCDIR%¥bin¥Hitachi.OpenTP1.Server.dll

「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

なお，このアセンブリはインストール時にグローバルアセンブリキャッシュ（GAC）に登録されます。

2.11.2　セキュリティポリシーの設定

Extension .NET をインストールおよび実行することで，インストールおよび実行した OS 上の.NET Framework のコードアクセスセキュリティポリシーを変更してしまうことはありません。

Extension .NET の機能を使用するためには，次に示すファイルに必要なアクセス許可が与えられていなければなりません。必要なアクセス許可が与えられていない場合は，コマンドやアプリケーションを実行できません。

(1)　.NET Framework のコードアクセスセキュリティポリシーの影響を受けるファイル（アセンブリ）

- %DCDIR%¥bin¥if2sstub.exe
- %DCDIR%¥bin¥if2cstub.exe
- %DCDIR%¥bin¥if2tsdl.exe
- %DCDIR%¥bin¥spp2cstub.exe
- %DCDIR%¥bin¥njstlsup.exe
- %DCDIR%¥bin¥Hitachi.OpenTP1.Server.dll
- %DCDIR%¥tools¥ExtNET¥DeveloperAssist¥Hitachi.OpenTP1.DevAssist4VSNet.dll
- %DCDIR%¥tools¥ExtNET¥DeveloperAssist¥Hitachi.OpenTP1.DevTools.dll
- 各 SPP.NET のアプリケーションベースディレクトリ，SUP.NET の配置ディレクトリ下の SPP.NET，SUP.NET 実装アセンブリ，およびそれらが参照するすべてのアセンブリ

「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

(2) 必要なアクセス許可

Extension .NET が提供するすべてのアセンブリに，完全信頼を指定する必要があります。

(3) 注意事項

.NET Framework が提供する各アセンブリに対してはアクセス許可の制限をしないでください。制限をした場合，インストールやアンインストールが失敗したり，運用コマンドやユーザサーバの実行ができなかったりすることがあります。

2.11.3 マルチ OpenTP1 環境での Extension .NET の環境設定

Extension .NET は，同一マシン上で複数の OpenTP1 を運用するマルチ OpenTP1 機能に対応しています。マルチ OpenTP1 の詳細については，マニュアル「OpenTP1　解説」を参照してください。

マルチ OpenTP1 環境で Extension .NET を利用する場合，Extension .NET はすべて同一のバージョンにしてください。なお，マルチ OpenTP1 環境で Extension .NET を利用する場合，TP1/Server が提供する dcsetupml.exe を使用してセットアップします。dcsetupml.exe の詳細については，TP1/Server Base の場合は「Windows 版ご使用上の注意事項」を，TP1/LiNK の場合はマニュアル「TP1/LiNK 使用の手引」を参照してください。

マルチ OpenTP1 環境の構築方法については，TP1/Server Base の場合は「Windows 版ご使用上の注意事項」を，TP1/LiNK の場合はマニュアル「TP1/LiNK　使用の手引」を参照してください。

2.11.4 メッセージの出力先

Extension .NET が提供する運用コマンドが出力するメッセージは，標準出力または標準エラー出力に出力されます。それ以外（クラスライブラリ，SPP.NET 実行コンテナなど）が出力するメッセージは，メッセージログファイルまたは.NET エラーログファイルに出力されます。

メッセージログファイルについては，マニュアル「TP1/LiNK　使用の手引」またはマニュアル「OpenTP1　運用と操作」を，.NET エラーログファイルについては「8.3　.NET エラーログファイル」を参照してください。

3 システム定義【TP1/Server Base】

この章では，TP1/Extension for .NET Framework（Extension .NET）で使用するシステム定義について説明します。この定義は，TP1/Server Base の場合だけ指定できます。TP1/LiNK で指定した場合，動作の保証はしません。

システム共通定義

形式

set 形式

```
set njs_appbase_directory=”%DCDIR%¥aplibディレクトリ”
```

機能

OpenTP1 システム共通の実行環境を定義します。

説明

set 形式のオペランド

●njs_appbase_directory="%DCDIR%¥aplib ディレクトリ"　～〈1～128 文字のパス名〉

「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

OpenTP1 インストールディレクトリ下の aplib ディレクトリを完全パスで指定します。Extension .NET を使用する場合，必ずこのオペランドを指定してください。

このオペランドは，ユーザサービス定義およびユーザサービスデフォルト定義でも指定できます。

指定値の優先順位は次のとおりです（1.> 2.> 3.)。

1. ユーザサービス定義
2. ユーザサービスデフォルト定義
3. システム共通定義

ユーザサービスデフォルト定義

形式

set 形式

```
〔set njs_appbase_directory=
 ”デフォルトアプリケーションベースディレクトリ”〕
〔set njs_input_max_message_size=”RPC要求メッセージの最大長”〕
〔set njs_output_max_message_size=”RPC応答メッセージの最大長”〕
```

機能

ユーザサービス定義の省略時解釈値を定義します。

説明

set 形式のオペランド

●njs_appbase_directory="デフォルトアプリケーションベースディレクトリ"　～〈1～128 文字のパス名〉

実装アセンブリを参照するデフォルトアプリケーションベースディレクトリを完全パスで指定します。

このオペランドは，ユーザサービス定義およびシステム共通定義でも指定できます。

指定値の優先順位は次のとおりです（1.> 2.> 3.)。

1. ユーザサービス定義
2. ユーザサービスデフォルト定義
3. システム共通定義

●njs_input_max_message_size="RPC 要求メッセージの最大長"　～〈符号なし整数〉((1～8))《1》(単位：メガバイト)

SPP.NET がクライアント UAP から受け付ける RPC 要求メッセージの最大長を指定します。

ここで指定する値は，TP1/Server Base のシステム共通定義の rpc_max_message_size オペランドで指定した値以下にしてください。rpc_max_message_size オペランドについては，マニュアル「OpenTP1 システム定義」を参照してください。

このオペランドは，ユーザサービス定義でも指定できます。ユーザサービス定義で指定した場合，ユーザサービス定義の値が優先されます。

●njs_output_max_message_size="RPC 応答メッセージの最大長"　～〈符号なし整数〉((1～8))《1》（単位：メガバイト）

SPP.NET がクライアント UAP にサービスの応答をする場合，SPP.NET がクライアント UAP に返す RPC 応答メッセージの最大長を指定します。

ここで指定する値は，TP1/Server Base のシステム共通定義の rpc_max_message_size オペランドで指定した値以下にしてください。rpc_max_message_size オペランドについては，マニュアル「OpenTP1 システム定義」を参照してください。

このオペランドは，ユーザサービス定義でも指定できます。ユーザサービス定義で指定した場合，ユーザサービス定義の値が優先されます。

ユーザサービス定義

形式

set 形式

```
 set module=”実行形式プログラム名”
〔set service=”サービス名=エントリポイント名”
          〔,”サービス名=エントリポイント名”〕…〕
〔set njs_server_assembly=”実装アセンブリ名称”〕
〔set njs_server_implement_class=”実装クラス名称”〕
〔set njs_server_stub_class=”サーバスタブクラス名称”〕
〔set njs_use_interface=Y|N〕
〔set njs_appbase_directory=”アプリケーションベースディレクトリ”〕
〔set njs_xa_connect=Y|N〕
〔set njs_xa_dllname=”トランザクション制御用ライブラリ”〕
〔set njs_use_mcf=Y|N〕
〔set njs_input_max_message_size=”RPC要求メッセージの最大長”〕
〔set njs_output_max_message_size=”RPC応答メッセージの最大長”〕
```

機能

ユーザサーバの実行環境をユーザサーバごとに定義します。

説明

set 形式のオペランド

●module="実行形式プログラム名"　～〈1～14 文字の識別子〉

SPP.NET の場合は，njsnetsv を指定します。SUP.NET の場合は，このサービスグループを実行する実行形式プログラム名を指定します。

●service="サービス名=エントリポイント名"〔,"サービス名=エントリポイント名"〕…　～〈1～31 文字の識別子〉

SPP.NET の場合は，サービス名とエントリポイント名にサービスメソッド名を指定します。SUP.NET の場合は，指定する必要はありません。

●njs_server_assembly="実装アセンブリ名称"　～〈1～128 文字の識別子とピリオド〉

SPP.NET の実装クラス，サーバスタブなどが含まれるアセンブリの名称を指定します。ファイルの拡張子（dll）は不要です。SPP.NET の場合，必ずこのオペランドを指定してください。

●njs_server_implement_class="実装クラス名称"　～〈1～128 文字の識別子とピリオド〉

SPP.NET の実装クラス名称を指定します。名前空間を含む完全限定名で指定してください。SPP.NET の場合，必ずこのオペランドを指定してください。

●njs_server_stub_class="サーバスタブクラス名称"　～〈1～128 文字の識別子とピリオド〉

サーバスタブのクラス名称を指定します。名前空間を含む完全限定名で指定してください。.NET インタフェース定義を使用した SPP.NET の場合，必ずこのオペランドを指定してください。

●njs_use_interface=Y|N　～《Y》

SPP.NET が.NET インタフェース定義を使用するかどうかを指定します。

- Y：.NET インタフェース定義を使用します。
- N：.NET インタフェース定義を使用しません。

●njs_appbase_directory="アプリケーションベースディレクトリ"　～〈1～128 文字のパス名〉

実装アセンブリを配置したアプリケーションベースディレクトリを完全パスで指定します。

ここで指定を省略した場合，ユーザサービスデフォルト定義の値を仮定します。ユーザサービスデフォルト定義でも指定を省略した場合，システム共通定義の値を仮定します。

●njs_xa_connect=Y|N　～《N》

XA 接続をするかどうかを指定します。

- Y：XA 接続をします。
- N：XA 接続をしません。

●njs_xa_dllname="トランザクション制御用ライブラリ"　～〈1～128 文字のパス名〉

ユーザサーバで参照するトランザクション制御用ライブラリを完全パスで指定します。

●njs_use_mcf=Y|N　～《N》

メッセージ送受信機能を使用するかどうかを指定します。このオペランドは，SPP.NET の場合にユーザサービス定義でだけ指定できます。

- Y：メッセージ送受信機能を使用します。
- N：メッセージ送受信機能を使用しません。

●njs_input_max_message_size="RPC 要求メッセージの最大長"　～〈符号なし整数〉((1～8))《1》（単位：メガバイト）

SPP.NET がクライアント UAP から受け付ける RPC 要求メッセージの最大長を指定します。

ここで指定する値は，TP1/Server Base のシステム共通定義の rpc_max_message_size オペランドで指定した値以下にしてください。rpc_max_message_size オペランドについては，マニュアル「OpenTP1 システム定義」を参照してください。

なお，このオペランドは SPP.NET のユーザサービス定義の場合だけ指定できます。

ここで指定を省略した場合，ユーザサービスデフォルト定義の値を仮定します。

●njs_output_max_message_size="RPC 応答メッセージの最大長"　～〈符号なし整数〉((1～8))《1》(単位：メガバイト)

SPP.NET がクライアント UAP にサービスの応答をする場合，SPP.NET がクライアント UAP に返す RPC 応答メッセージの最大長を指定します。

ここで指定する値は，TP1/Server Base のシステム共通定義の rpc_max_message_size オペランドで指定した値以下にしてください。rpc_max_message_size オペランドについては，マニュアル「OpenTP1 システム定義」を参照してください。

なお，このオペランドは SPP.NET のユーザサービス定義の場合だけ指定できます。

ここで指定を省略した場合，ユーザサービスデフォルト定義の値を仮定します。

4 UAP の作成と実行

この章では，TP1/Extension for .NET Framework（Extension .NET）で使用する UAP の作成方法と実行手順について説明します。

4.1　OpenTP1 for .NET Framework 環境での UAP 開発時に必要な定義

OpenTP1 for .NET Framework 環境での UAP 開発時に必要な定義について説明します。

4.1.1　.NET インタフェース定義

.NET インタフェース定義は，次の UAP から SPP.NET に対して.NET インタフェース定義を使用して RPC を実行する場合に定義します。

- SPP.NET
- SUP.NET
- CUP.NET（TP1/Connector for .NET Framework を利用して SPP.NET にサービスを要求するアプリケーションを含みます）

ここでは，.NET インタフェース定義の定義方法の詳細について説明します。.NET インタフェース定義を使用した RPC については，「1.3.2(1)(a)　.NET インタフェース定義を使用した RPC」を参照してください。

(1)　.NET インタフェース定義の定義方法

(a)　.NET インタフェース定義に使用する言語

SPP.NET の.NET インタフェース定義は，SPP.NET を開発するプログラム言語で定義します。.NET インタフェース定義を定義する場合，次のプログラム言語が使用できます。

- C#
- J#
- Visual Basic

各プログラム言語でのインタフェースの定義文法は，Visual Studio や.NET Framework SDK のドキュメントを参照してください。

なお，.NET インタフェース定義は，Visual Studio や.NET Framework SDK が提供する各プログラム言語のコンパイラで，.NET Framework および OpenTP1 for .NET Framework が提供するクラスライブラリだけを参照してコンパイルできる必要があります。

(b)　.NET インタフェース定義の定義規則

SPP.NET のインタフェースを定義する場合，次の規則があります。これらの規則に従っていない場合，SPP.NET の.NET インタフェース定義として使用できません。

- 名前空間名，インタフェース名，メソッド名には，半角英数字および半角アンダスコア（_）が使用できます。なお，名前空間の区切り文字として半角ピリオド（.）が使用できます。

- メソッドの引数および戻り値に使用できるデータ型を次の表に示します。

表 4-1　メソッドの引数および戻り値で使用できるデータ型

共通型システム	プログラム言語		
	C#	J#	Visual Basic
System.Void※	void	void	−
System.Byte	byte	ubyte	Byte
System.Int16	short	short	Short
System.Int32	int	int	Integer
System.Int64	long	long	Long
System.String	string	String	String
System.Byte[]	byte[]	ubyte[]	Byte()
System.Int16[]	short[]	short[]	Short()
System.Int32[]	int[]	int[]	Integer()
System.Int64[]	long[]	long[]	Long()
System.String[]	string[]	String[]	String()
TP1 ユーザ構造体	−	−	−
TP1 ユーザ構造体配列	−	−	−

（凡例）

−：プログラム言語による差異はありません。

注※

System.Void はメソッドの戻り値にだけ指定できます。

- メソッドの引数に使用できるパラメタ属性を次の表に示します。

表 4-2　メソッドの引数に使用できるパラメタ属性

意味	プログラム言語		
	C#	J#※2	Visual Basic
値渡し	var（省略可）	なし	ByVal（省略可）
出力渡し	out	指定不可※1	指定不可※1
参照渡し	ref	Holder クラス※2	ByRef

注※1

out 属性は C#固有の属性です。C#以外では指定できません。C#以外のプログラム言語で使用する場合は，代わりに参照渡しを使用してください。

注※2

J#ではパラメタ属性の代わりにデータ型で表現します。J#では値型の参照渡しができないため，OpenTP1 が提供する Holder クラスを使用してください。

OpenTP1 が提供する Holder クラスを次の表に示します。

Holder クラス名	保持する値の型	
	J#	共通型システム
Hitachi.OpenTP1.UByteHolder	ubyte	System.Byte
Hitachi.OpenTP1.ShortHolder	short	System.Int16
Hitachi.OpenTP1.IntHolder	int	System.Int32
Hitachi.OpenTP1.LongHolder	long	System.Int64
Hitachi.OpenTP1.StringHolder	String	System.String
Hitachi.OpenTP1.UByteArrayHolder	ubyte[]	System.Byte[]
Hitachi.OpenTP1.ShortArrayHolder	short[]	System.Int16[]
Hitachi.OpenTP1.IntArrayHolder	int[]	System.Int32[]
Hitachi.OpenTP1.LongArrayHolder	long[]	System.Int64[]
Hitachi.OpenTP1.StringArrayHolder	String[]	System.String[]

注 1

これらの Holder クラスは J#でだけ使用できます。C#および Visual Basic では ref，ByRef などのパラメタ属性を使用してください。

注 2

これらの Holder クラスは，メソッドの引数としてだけ使用できます。メソッドの戻り値には使用できません。

- メソッド名の長さは 31 文字以内でなければなりません。なお，メソッド名は SPP.NET のサービス名になります。
- メソッド名は英字で始まる必要があります。
- メソッドのオーバロードはできません。
- 大文字，小文字だけが異なる同じ名称のメソッドを定義することはできません。
- J#で参照渡しをする場合，/** @ref */属性は使用できません。参照渡しをする場合は，必ず Holder クラスを使用してください。

(2) TP1 ユーザ構造体

(a) TP1 ユーザ構造体とは

TP1 ユーザ構造体は，複数の値をまとめて保持できるクラスのことです。

(b)　TP1 ユーザ構造体の定義規則

TP1 ユーザ構造体を定義する場合の規則を次に示します。

- TP1 ユーザ構造体として使用するクラスは，Hitachi.OpenTP1.TP1UserStruct クラスを継承します。
- デフォルトコンストラクタ（アクセス修飾子が public で，引数のないコンストラクタ）を持つ必要があります。
- set，get 両方のアクセサを持つ public プロパティが一つ以上必要です。
- 大文字，小文字だけが異なる同一名称のプロパティを定義することはできません。

注意事項

public プロパティ以外のすべてのメンバは RPC で送受信されるデータには含まれません。

(c)　TP1 ユーザ構造体のプロパティとして利用できるデータ型

TP1 ユーザ構造体のプロパティとして利用できるデータ型，およびメソッドの引数として使用できるパラメタ属性を次の表に示します。

表 4-3　TP1 ユーザ構造体のプロパティとして利用できるデータ型

共通型システム	プログラム言語		
	C#	J#	Visual Basic
System.Byte	byte	ubyte	Byte
System.Int16	short	short	Short
System.Int32	int	int	Integer
System.Int64	long	long	Long
System.String	string	String	String
System.Byte[]	byte[]	ubyte[]	Byte()
System.Int16[]	short[]	short[]	Short()
System.Int32[]	int[]	int[]	Integer()
System.Int64[]	long[]	long[]	Long()
System.String[]	string[]	String[]	String()
TP1 ユーザ構造体	－	－	－
TP1 ユーザ構造体配列	－	－	－

（凡例）

－：プログラム言語による差異はありません。

(d)　TP1 ユーザ構造体の参照渡しの方法

TP1 ユーザ構造体の参照渡しをするには，パラメタ属性を使用します。ただし，J#の場合は参照渡しができないため，Hitachi.OpenTP1.ITP1UserStructHolder を実装する必要があります。実装上の規則を次に示します。

- プロパティとして利用できるデータ型は TP1 ユーザ構造体，または TP1 ユーザ構造体の配列に限られます。
- プロパティは set，get 両方のアクセサを持ち，アクセス修飾子が public である必要があります。
- プロパティは一つしか定義してはいけません。
- 実装クラスの名称規則を，次の表に示します。

プロパティのデータ型	実装クラスの名称
TP1 ユーザ構造体	TP1 ユーザ構造体クラス名称＋"Holder"
TP1 ユーザ構造体配列	TP1 ユーザ構造体クラス名称＋"ArrayHolder"

- 実装クラスの名前空間は，プロパティのデータ型と同じ名前空間である必要があります。

なお，プロパティ以外のすべてのメンバは RPC で送受信されるデータには含まれません。TP1 ユーザ構造体の参照渡しの使用例については，「4.1.1(3)(b)　J#での定義例」を参照してください。

(3)　.NET インタフェース定義の定義例

(a)　C#での定義例

```
using System;
using Hitachi.OpenTP1;
namespace MyCompany
{

  public interface IGyoumuA
  {
    void Service1(string dataId, byte[] data);
    string[] Service2(string key);
    int Service3(int inCount, ref string[] ids);
    short Service4(MyStruct inStruct);
  }
  public class MyStruct : TP1UserStruct
  {
      private int id;
      private string name;

      public MyStruct(){}

      public int Id
      {
        set{
           id = value;
        }
```

```
        get{
           return id;
        }
      }

      public string Name
      {
         set{
            name = value;
         }
         get{
            return name;
         }
      }
  }
}
```

(b)　J#での定義例

```
package MyCompany;
import System.*;
import Hitachi.OpenTP1.*;

public interface IGyoumuA
{
    void Service1(String dataId, ubyte[] data);
    String[] Service2(String key);
    int Service3(int inCount, StringArrayHolder ids);
    short Service4(MyStruct inStruct, MyStructHolder outStruct);
}

//TP1ユーザ構造体の参照渡しをするためのクラス
public class MyStructHolder implements ITP1UserStructHolder
{
   private MyStruct val;
   /** @property */
   public void set_Value(MyStruct value)
   {
          val = value;
   }
   /** @property */
   public MyStruct get_Value()
   {
          return val;
   }
}

//TP1ユーザ構造体
public class MyStruct extends TP1UserStruct
{
   private int id;
   private String name;

   public MyStruct()
   {
   }
```

```
    /** @property */
    public void set_Id(int value)
    {
            id = value;
    }
    /** @property */
    public int get_Id()
    {
            return id;
    }
    /** @property */
    public void set_Name(String value)
    {
            name = value;
    }
    /** @property */
    public String get_Name()
    {
            return name;
    }
}
```

(c) Visual Basic での定義例

```
Imports System
Imports Hitachi.OpenTP1
Namespace MyCompany
  Public Interface IGyoumuA
    Sub Service1(ByVal dataId As String, ByVal data() As Byte)
    Function Service2(ByVal key As String) As String()
    Function Service3(ByVal inCount As Integer, _
                      ByRef ids() As String) As Integer
    Function Service4(ByVal inStruct As MyStruct) As Short
  End Interface

  Public Class MyStruct
      Inherits TP1UserStruct
      Private idValue As Integer
      Private nameValue As String

      Public Sub New()
      End Sub

      Public Property Id() As Integer
          Set(ByVal value As Integer)
               idValue = value
          End Set
          Get
               Return idValue
          End Get
      End Property

      Public Property Name() As String
          Set(ByVal value As String)
               nameValue = value
          End Set
          Get
```

```
            Return nameValue
        End Get
    End Property
  End Class
End Namespace
```

4.1.2　サービス定義

サービス定義は，次の UAP から SPP.NET，または SPP に対してサービス定義を使用して RPC を実行する場合に定義します。

- SPP.NET
- SUP.NET
- CUP.NET（TP1/Connector for .NET Framework を利用して SPP.NET，または SPP にサービスを要求するアプリケーションを含みます）

サービス定義は，サービス定義のファイルとサービス定義が参照するデータ型定義のファイルから構成されます。この節では，サービス定義の定義方法の詳細について説明します。サービス定義を使用した RPC については，「1.3.2(1)(b)　サービス定義（カスタムレコード）を使用した RPC」を参照してください。

(1)　データ型定義ファイル

データ型定義とは，RPC でやり取りされる入力データや出力データの形式をメッセージ単位で定義するものです。

サービス定義から参照するデータ型定義ファイルは，次に示す形式で独立したファイルとして定義します。

(a)　形式

```
struct データ型定義名称 {
        データ型 メンバ名称[配列指定];
    〔〔データ型 メンバ名称[配列指定];〕…〕
};
 〔〔struct データ型定義名称 {
        データ型 メンバ名称[配列指定];
    〔〔データ型 メンバ名称[配列指定];〕…〕
};〕…〕
```

(b)　説明

●データ型定義名称　　～〈31 文字以内の識別子〉

データ型定義に付ける名称を指定します。

指定された名称がカスタムレコードクラスのクラス名となります。

●データ型

メンバ名称で示される変数に対するデータ型を指定します。

クライアントスタブ生成コマンド（spp2cstub）は，定義された各メンバのデータ型を次の表に示すように，.NET Framework のデータ型に対応づけてカスタムレコードクラスを生成します。

また，データの内容を次の表に示すように変換します。

表 4-4　データ型の変換規則と変換内容

データ型定義ファイルで定義されたデータ型	データ型の説明	カスタムレコードクラスで定義される.NET Framework のデータ型	データ変換の内容
char	文字列	System.String	char は.NET Framework の System.String に対応づけられます。カスタムレコードを入力データとして使用する場合，スタブ生成コマンドに指定したエンコード方式に従ってバイト配列に変換しデータを扱います。 バイト配列に変換した結果が 32 バイトで，データ型定義が char a[30]の場合，2 バイトは破棄されます。
int	32 ビット符号付き整数	System.Int32	int は.NET Framework の System.Int32 に対応づけられます。 カスタムレコードを入力データとして使用する場合，スタブ生成コマンドに指定されたエンディアンでバイト配列に変換して，RPC の要求メッセージに設定します。 カスタムレコードを出力データとして使用する場合，応答メッセージから 4 バイトをスタブ生成コマンドに指定されたエンディアンで読み込み，データにセットします。
short	16 ビット符号付き整数	System.Int16	short は.NET Framework の System.Int16 に対応づけられます。カスタムレコードを入力データとして使用する場合，指定されたエンディアンでバイト配列に変換して，RPC の要求メッセージに設定します。 カスタムレコードを出力データとして使用する場合，RPC の応答メッセージから 2 バイトをスタブ生成コマンドに指定されたエンディアンで読み込み，データにセットします。
long	32 ビット符号付き整数	System.Int32	long は.NET Framework の System.Int32 に対応づけられます。 カスタムレコードを入力データとして使用する場合，スタブ生成コマンドに指定されたエンディアンでバイト配列に変換して，RPC の要求メッセージに設定します。 カスタムレコードを出力データとして使用する場合，応答メッセージから 4 バイトをスタブ生成コマンドに指定されたエンディアンで読み込み，データにセットします。
byte	バイナリデータ	System.Byte	byte は.NET Framework の System.Byte に対応づけられます。 一次元配列指定にだけ使用できます。
struct	構造体	class（インナークラス）	データ型定義で struct として定義されるデータ型のことを構造体と呼びます。

データ型定義ファイルで定義されたデータ型	データ型の説明	カスタムレコードクラスで定義される.NET Framework のデータ型	データ変換の内容
			struct はインナークラスに対応づけられます。

データ型が int，long，または struct の場合は，先頭からのオフセットが 4 の整数倍でなければなりません。また，short の場合は，先頭からのオフセットが 2 の整数倍でなければなりません。

なお，スタブ生成コマンドでは，自動的にバウンダリ調整をしないため，先頭からのオフセットが正しい整数倍でない場合，エラーとなるので注意が必要です。

バウンダリ調整については，「(d)　バウンダリ調整」を参照してください。

データ型が struct の場合は，次に示すように定義します。

```
struct 構造体名称 {
        データ型 メンバ名称[配列指定];
    〔〔データ型 メンバ名称[配列指定];〕…〕
} メンバ名称[配列指定];
```

データ型定義で struct として定義されるメンバは，その構造体名称がそのままインナークラスのクラス名称になり，メンバ名称がインナークラスの型を持つプロパティ名称になります。

ここで指定するデータ型，メンバ名称，配列要素数はデータ型定義に従います。また，この構造体のデータ型定義先頭からのオフセットは，4 の整数倍でなければなりません。また，構造体自身のサイズも 4 の整数倍でなければなりません。

ただし，構造体を構成するメンバ（構造体の中に構造体がある場合は，そのメンバも含む）がすべて char または byte の場合は，任意のオフセットおよび任意のサイズを定義できます。

データ型定義で struct として定義される配列型は，可変長構造体配列として扱えます。可変長構造体配列については，「(e)　可変長構造体配列」を参照してください。

●構造体名称　　～〈31 文字以内の識別子〉

構造体に付ける名称を指定します。

構造体名称の先頭文字は半角英字とします。2 文字目以降は半角英数字および半角アンダスコア（_）が使用できます。

●メンバ名称　　～〈31 文字以内の識別子〉

メンバ名称を指定します。

メンバ名称の先頭文字は半角英字で指定してください。

●配列指定

配列要素数　　～〈符号なし整数〉((1～8388608))

メンバがデータ型の配列の場合，配列要素数を指定します。

データ型が int，short，long，または struct のときは，一次元配列が指定できます（配列指定なしも指定できます）。

データ型が byte のときは，一次元配列だけ指定できます。

データ型が char のときは，二次元配列まで指定できます。

一次元配列

```
[配列要素数]
```

二次元配列

```
[配列要素数][配列要素数]
```

(c) データ型定義の定義例

```
struct in_data {
 long I_basho[3];
 long I_kakaku;
 long I_tokuchou;
};

struct out_data {
 char o_name[20];
 char o_basho[16];
 char o_tokuchou[20];
 long o_kakaku;
 char o_inf[80];
};

struct out_data2 {
 char o_name[20];
 char o_basho[16];
 char o_tokuchou[20];
 long o_kakaku;
 char o_inf[80][20];
};

struct put_data {
 int  o_num;
 struct data {
  char o_name[20];
  char o_basho[16];
  char o_tokuchou[20];
  long o_kakaku;
  char o_inf[80];
 } data_t[100];
};
```

(d) バウンダリ調整

バウンダリ調整とは，データ型定義の各変数を決められたバイト境界に配置することをいいます。バウンダリ調整は，コンパイラが自動的に行います。

例えば，次に示すような構造体を使用している場合，先頭からのオフセットを正しくするため，実際は OS，およびコンパイラによって，data と num の間に 1 バイトの補正が入ります。

構造体の定義例

```
struct s_data {
    char data[3];
    long num;
} s_data_t
```

バウンダリ調整後の定義例

```
struct s_data {
  char data[3];
    〈1バイト〉
    long num;
} s_data_t
```

この例の場合には，次のように修正して使用してください。

修正前

```
struct s_data {
    char data[3];
    long num;
} s_data_t;
```

修正後

```
struct s_data {
    char data[3];
    char wk;
    long num;
} s_data_t;
```

ポイント

OpenTP1 for .NET Framework 以外の OpenTP1 システムで使用していた構造体をデータ型定義として使用する場合などは，バウンダリ調整に注意してください。

(e) 可変長構造体配列

データ型が struct で，配列型の場合は，可変長構造体配列として扱えます。可変長構造体配列は次に示す形式で定義します。

可変長構造体配列の形式

```
int 配列要素数を示すメンバ名称
struct 構造体名称 {
                    データ型 メンバ名称[配列指定];
                    〔〔データ型 メンバ名称[配列指定];〕…〕
} メンバ名称[配列要素数を示すメンバ名称:配列の最大要素数];
```

可変長構造体配列を指定する場合は，可変長構造体配列の配列要素数を「データ型が int 型の配列要素数を示すメンバ名称:配列の最大要素数」という形式で記述します。構造体名称，データ型，メンバ名称および配列指定については，「(b)　説明」を参照してください。

可変長構造体配列の定義例

```
struct put_data {
                int o_num;
                struct data{
                            char o_name[20];
                            char o_basho[16];
                            char o_tokuchou[20];
                            long o_kakaku;
                            char o_inf[80];
                            } data_t[o_num:100];
                };
```

可変長構造体配列を使用する場合は，次の点に注意してください。

- 配列要素数を示すメンバ名称は可変長構造体配列より前に定義してください。
- 配列要素数を示すメンバ名称の値は配列の最大要素数に指定した値以下にしてください。配列要素数を示すメンバ名称の値が配列の最大要素数に指定した値を超える場合はエラーが発生します。
- 可変長構造体配列を使用する場合は，構造体の長さは配列の最大要素数に指定した値で計算されるので，配列の最大要素数を指定するときはデータ型定義の長さが 1 から 8388608 バイトの範囲になるように指定してください。ただし，RPC 送受信メッセージの最大長拡張機能を使用しない場合は，データ型定義の長さは 1 から 1048576 バイトの範囲になるように指定してください。
- 生成されたカスタムレコードを使用する場合，可変長構造体配列に対応するプロパティには，配列の最大要素数以内の構造体配列を指定してください。null または配列の最大要素数を超える構造体配列を指定した場合は，TP1MarshalException 例外が発生します。

(f)　注意事項

- データ型定義では任意の位置にコメントを記述できます。コメントは「/*」で始め，「*/」で終了します。なお，コメントのネストはできません。
- データ型定義全体の長さは 1 から 8388608 バイトの範囲になるように記述してください。ただし，RPC 送受信メッセージの最大長拡張機能を使用しない場合は，1 から 1048576 バイトの範囲になるように記述してください。
- データ型定義ファイルでは，1 文を複数行にわたって指定しないでください。
- データ型定義名称として dc および DC で始まる名称は使用しないでください。
- データ型定義名称，メンバ名称としてクライアントスタブおよびカスタムレコードクラスを生成するプログラム言語のキーワードは使用できません。
- 生成されたカスタムレコードを使用する場合，配列および固定長構造体配列に対応するプロパティには要素数分の配列数を指定してください。null または要素数分以外の配列数を指定した場合は，TP1MarshalException 例外が発生します。

- 一つのデータ型定義に同じ名前のメンバ名称を指定しないでください。

(2) サービス定義ファイル

サービス定義とは，サービスの入出力データに対応するデータ型定義を定義するものです。

サービス定義ファイルは，次に示す形式で独立したファイルとして定義します。

(a) 形式

```
#include ”データ型定義ファイル名”
 〔〔#include ”データ型定義ファイル名”〕
…〕

interface サービス定義名称 {
      サービス名称(入力データ型定義名，出力データ型定義名);
   〔〔サービス名称(入力データ型定義名，出力データ型定義名);〕…〕
}
```

(b) 説明

●データ型定義ファイル名　　～〈ファイル名〉

このサービス定義で参照するデータ型定義が定義されているファイル名を指定します。ファイル名は次のどちらかの形式で指定します。

- 拡張子を含んだファイル名だけを指定する。
- 相対パスまたは絶対パスを含んだファイル名で指定する。

なお，パス指定時の区切り文字には「/」または「¥」が使用できます。

●サービス定義名称　　～〈31 文字以内の識別子〉

このサービス定義に付けるサービス定義名称を指定します。

任意に名称を付けることができます。デフォルトではこの名称からクライアントスタブなどのクラス名が生成されます。

●サービス名称　　～〈31 文字以内の識別子〉

このサービス定義に含めるサービス名称を指定します。

対象となるユーザサーバが持つサービス名称を指定してください。

●入力データ型定義名　　～〈31 文字以内の識別子〉

各サービスの入力データに対応するデータ型定義名称を指定します。

入力データ型定義名は，このサービス定義ファイルの#include ディレクティブで指定したデータ型定義ファイルで定義されたデータ型定義名称でなければなりません。

●出力データ型定義名　　～〈31 文字以内の識別子〉

各サービスの出力データに対応するデータ型定義名称を指定します。

出力データ型定義名は，このサービス定義ファイルの#include ディレクティブで指定したデータ型定義ファイルで定義されたデータ型定義名称でなければなりません。

ただし，出力データがない場合は，出力データ型定義名には DC_NODATA を指定してください（非応答型 RPC の場合だけ使用できます）。

(c) サービス定義の定義例

サービス定義の定義例 1（業務 1 のサービス定義）

```
#include "mydata.h"
/* 業務1のサービス定義 */
interface GYOUMU1 {
    GETDATA1(in_data, out_data);
    GETDATA2(in_data, out_data2);
}
```

サービス定義の定義例 2（業務 2 のサービス定義）

```
#include "datas/mydata.h"
/* 業務2のサービス定義 */
interface GYOUMU2 {
    GET_DATA1(in_data, out_data);
    PUT_DATA1(put_data, DC_NODATA);  /* 非応答型 */
}
```

データ型定義の定義例（mydata.h）

```
struct in_data {
 long I_basho[3];
 long I_kakaku;
};

struct out_data {
 char o_name[20];
 char o_basho[16];
 long o_kakaku;
 char o_inf[80];
};

struct out_data2 {
 char o_name[20];
 char o_basho[16];
 long o_kakaku;
 char o_inf[80][20];
};

struct put_data {
 int  o_num;
 struct data {
  char o_name[20];
  char o_basho[16];
  long o_kakaku;
  char o_inf[80];
 } data_t[100];
};
```

(d) 注意事項

- サービス定義では任意の位置にコメントを記述できます。コメントは「/*」で始め，「*/」で終了します。なお，コメントのネストはできません。
- コメント文中に「//」は使用できません。

4.2 UAP の作成手順の概要

Extension .NET で使用する次の UAP の作成手順の概要について説明します。

- SPP.NET
- SUP.NET

4.2.1 SPP.NET の作成手順の概要

SPP.NET の作成手順の概要を次に示します。

(1) .NET インタフェース定義を使用する場合

1. .NET インタフェース定義を定義します。
2. サーバスタブを生成します。
3. SPP.NET の実装クラスをコーディングします。
4. .NET インタフェース定義，サーバスタブ，SPP.NET 実装クラスを一つのアセンブリにまとめて SPP.NET をビルドします。

   OpenTP1 が提供するアセンブリ（DLL）への参照設定が必要です。また，必要に応じて SPP.NET が利用するアセンブリ（DLL）への参照設定も必要です。

(2) .NET インタフェース定義を使用しない場合

1. SPP.NET の実装クラスをコーディングします。
2. SPP.NET 実装クラスをビルドしてアセンブリを生成します。

   OpenTP1 が提供するアセンブリ（DLL）への参照設定が必要です。また，必要に応じて SPP.NET が利用するアセンブリ（DLL）への参照設定も必要です。

4.2.2 SUP.NET の作成手順の概要

SUP.NET の作成手順の概要を次に示します。

1. SUP.NET の実装クラスをコーディングします。
2. クライアントスタブ，SUP.NET 実装クラスを一つのアセンブリ（EXE）にまとめて SUP.NET をビルドします。

   OpenTP1 が提供するアセンブリ（DLL）への参照設定が必要です。また，必要に応じて SUP.NET が利用するアセンブリ（DLL）への参照設定も必要です。

4.2.3　SPP.NET および SUP.NET 作成時の注意事項

SPP.NET および SUP.NET の実装クラスをコーディングする場合，SUP.NET がクライアントスタブを使用してほかの SPP.NET を呼び出すときは，あらかじめそのクライアントスタブを生成しておく必要があります。

そのため，利用するサーバの.NET インタフェース定義を事前に入手するか，または必要に応じてサービス定義を作成しておく必要があります。

4.3　SPP.NET の作成方法

.NET インタフェース定義を使用する場合，および.NET インタフェース定義を使用しない場合の SPP.NET の作成方法について説明します。

4.3.1　.NET インタフェース定義を使用した SPP.NET の実装

SPP.NET が OpenTP1 for .NET Framework の UAP からだけ利用されることが明らかな場合，.NET インタフェース定義を使用した SPP.NET を実装しておくと便利です。

.NET インタフェース定義を使用して SPP.NET を実装する場合，.NET インタフェース定義および SPP.NET の実装には次のプログラム言語が使用できます。

- C#
- J#
- Visual Basic

注意事項

.NET インタフェース定義を使用した SPP.NET は，C 言語および COBOL 言語で作成されたアプリケーションから呼び出すことはできません。

(1)　SPP.NET を実装する場合の規則

SPP.NET を実装する場合は，次の規則に従って各プログラム言語でクラスを実装します。

- インタフェースを定義したプログラム言語と同じ言語で実装します。
- Hitachi.OpenTP1.Server.SPPBase クラスを継承します。
- public クラス（Public クラス）として宣言します。
- SPP.NET の.NET インタフェース定義で宣言したインタフェースを一つだけ実装します。
- SPP.NET の.NET インタフェース定義で宣言した各メソッドを public メソッドとして実装します。各メソッドには必ず TP1RpcMethod 属性を宣言します。なお，メソッド名は SPP.NET のサービス名になります。
- SPP.NET の初期化処理は InitializeSPP メソッド，終了時処理は FinalizeSPP メソッドで実装します。それぞれの処理が不要な場合は InitializeSPP メソッド，または FinalizeSPP メソッドを実装する必要はありません。

(2) .NET インタフェース定義を使用して SPP.NET を実装する場合の注意事項

.NET インタフェース定義を変更した場合，SPP.NET の実装側ではサーバスタブ，利用側ではクライアントスタブを必ず再生成する必要があります。.NET インタフェース定義と，サーバスタブおよびクライアントスタブが一致していない場合，RPC が正しく実行されません。

4.3.2 .NET インタフェース定義を使用した SPP.NET のコーディング例

(1) C#の場合のコーディング例

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;

namespace MyCompany
{
  public class GyoumuAImpl : SPPBase, IGyoumuA
  {
    public GyoumuAImpl()
    {
      // コンストラクタの処理
      // ** コンストラクタでOpenTP1のクラスは使用できません **
      // ** 必要な初期化処理はInitializeSPPで行ってください **
    }
    ~GyoumuAImpl()
    {
      // デストラクタの処理
      // ** デストラクタでOpenTP1のクラスは使用できません **
      // ** 必要な終了処理はFinalizeSPPで行ってください **
    }

    // SPP初期化および終了処理メソッドの実装
    public override void InitializeSPP() {
      // SPPの初期化処理
    }
    public override void FinalizeSPP() {
      // SPPの終了処理
    }

    // サービスメソッドの実装
    [TP1RpcMethod]
    public void Service1(string dataId, byte[] data)
    {
      // Service1()の処理
    }

    [TP1RpcMethod]
    public string[] Service2(string key)
    {
      // Service2()の処理
```

```
    }

    [TP1RpcMethod]
    public int Service3(int inCount, ref string[] ids)
    {
      // Service3()の処理
    }

    [TP1RpcMethod]
    public short Service4(MyStruct inStruct)
    {
      // Service4()の処理
    }

    // SPP内部メソッドの実装（自由に実装できます）
    // RPCで呼び出すことはできません
    public string GetUserInfo()
    {
      // 処理
    }
    private void PutErrorLog(string errorInfo, int errorCode)
    {
      // 処理
    }
  }
}
```

(2) J#の場合のコーディング例

```
package MyCompany;
import System.*;
import Hitachi.OpenTP1.*;
import Hitachi.OpenTP1.Server.*;

public class GyoumuAImpl extends SPPBase implements IGyoumuA
{
  public GyoumuAImpl()
  {
    // コンストラクタの処理
    // ** コンストラクタでOpenTP1のクラスは使用できません **
    // ** 必要な初期化処理はInitializeSPPで行ってください **
  }

  protected void Finalize()
  {
    // Finalizeメソッドの処理
    // ** FinalizeメソッドでOpenTP1のクラスは使用できません **
    // ** 必要な終了処理はFinalizeSPPで行ってください **
  }
  // SPP初期化および終了処理メソッドの実装
  public void InitializeSPP() {
    // SPPの初期化処理
  }
  public void FinalizeSPP() {
    // SPPの終了処理
  }
```

```
  // サービスメソッドの実装
  /** @attribute TP1RpcMethod() */
  public void Service1(String dataId, ubyte[] data)
  {
    // Service1()の処理
  }

  /** @attribute TP1RpcMethod() */
  public String[] Service2(String key)
  {
    // Service2()の処理
  }

  /** @attribute TP1RpcMethod() */
  public int Service3(int inCount,
                      StringArrayHolder ids)
  {
    // Service3()の処理
  }

  /** @attribute TP1RpcMethod() */
  public short Service4(MyStruct inStruct,
                        MyStructHolder outStruct)
  {
    // Service4()の処理
  }

  // SPP内部メソッドの実装（自由に実装できます）
  // RPCで呼び出すことはできません
  public String GetUserInfo()
  {
    // 処理
  }
  private void PutErrorLog(String errorInfo, int errorCode)
  {
    // 処理
  }
}
```

(3) Visual Basic の場合のコーディング例

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Server

Namespace MyCompany
 Public Class GyoumuAImpl
    Inherits SPPBase
    Implements IGyoumuA
    Sub GyoumuAImpl()
      ' コンストラクタの処理
      ' ** コンストラクタでOpenTP1のクラスは使用できません **
      ' ** 必要な初期化処理はInitializeSPPで行ってください **
    End Sub
```

```
    Sub Finalize()
      ’Finalizeメソッドの処理
      ’** Finalizeメソッドで0penTP1のクラスは使用できません **
      ’** 必要な終了処理はFinalizeSPPで行ってください **
    End Sub
    ’SPP初期化および終了処理メソッドの実装
    Public Overrides Sub InitializeSPP()
      ’SPPの初期化処理
    End Sub
    Public Overrides Sub FinalizeSPP()
      ’SPPの終了処理
    End Sub

    ’サービスメソッドの実装
    <TP1RpcMethod()> _
    Public Sub Service1(ByVal dataId As String, _
                        ByVal data() As Byte) _
    Implements IGyoumuA.Service1
      ’Service1()の処理
    End Sub

    <TP1RpcMethod()> _
    Public Function Service2(ByVal key As String) As String() _
    Implements IGyoumuA.Service2
      ’Service2()の処理
    End Function

    <TP1RpcMethod()> _
    Public Function Service3(ByVal inCount As Integer, _
                             ByRef ids() As String) _
    As Integer Implements IGyoumuA.Service3
      ’Service3()の処理
    End Function

    <TP1RpcMethod()> _
    Public Function Service4(ByVal inStruct As MyStruct) _
                             As Short _
    Implements IGyoumuA.Service4
      ’Service4()の処理
    End Function

    ’SPP内部メソッドの実装（自由に実装できます）
    ’RPCで呼び出すことはできません
    Public Function GetUserInfo() As String
      ’処理
    End Function
    Private Sub PutErrorLog(errorInfo As String, _
                            errorCode As Integer)
      ’処理
    End Sub
  End Class
End Namespace
```

4.3.3　.NET インタフェース定義を使用しない SPP.NET の実装

.NET インタフェース定義を使用しない SPP.NET は，SUP.NET や CUP.NET などの OpenTP1 for .NET Framework の UAP 以外にも，C 言語および COBOL 言語で作成したアプリケーション（SUP，CUP など）から呼び出すことができます。

SPP.NET を OpenTP1 for .NET Framework の UAP 以外のクライアントアプリケーションから利用する可能性がある場合は，必ず.NET インタフェース定義を使用しない SPP.NET を実装しておきます。

.NET インタフェース定義を使用しない SPP.NET を実装する場合，次のプログラム言語が使用できます。

- C#
- J#
- Visual Basic
- COBOL 言語

COBOL 言語で実装する方法については，「4.8　COBOL 言語での UAP の作成方法」を参照してください。

(1)　SPP.NET を実装する場合の規則

SPP.NET を実装する場合は，次の規則に従って各プログラム言語でクラスを実装します。

- Hitachi.OpenTP1.Server.SPPBase クラスを継承します。
- public クラス（Public クラス）として宣言します。
- SPP.NET のサービスメソッドを public メソッドとして実装します。各メソッドには必ず TP1RpcMethod 属性を宣言します（COBOL 言語の場合は TP1RPCMETHOD 翻訳指令を追加します）。なお，メソッド名は SPP.NET のサービス名になります。
- SPP.NET の初期化処理は InitializeSPP メソッド，終了時処理は FinalizeSPP メソッドで実装します。それぞれの処理が不要な場合は InitializeSPP メソッド，または FinalizeSPP メソッドを実装する必要はありません。

4.3.4　.NET インタフェース定義を使用しない SPP.NET のコーディング例

(1)　C#の場合のコーディング例

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;

namespace MyApplication
{
  public class MySPP : SPPBase
  {
```

```
    public MySPP()
    {
      // コンストラクタの処理
      // ** コンストラクタでOpenTP1のクラスは使用できません **
      // ** 必要な初期化処理はInitializeSPPで行ってください **
    }
    ~MySPP()
    {
      // デストラクタの処理
      // ** デストラクタでOpenTP1のクラスは使用できません **
      // ** 必要な終了処理はFinalizeSPPで行ってください **
    }
    // SPP初期化および終了処理メソッドの実装
    public override void InitializeSPP() {
      // SPPの初期化処理
    }
    public override void FinalizeSPP() {
      // SPPの終了処理
    }
    // サービスメソッドの実装（引数の並びは固定です）
    // クライアントからRPCで呼び出されます
    [TP1RpcMethod]
    public void Service1(byte[] inData, int inLen,
                         byte[] outData, ref int outLen)
    {
      // Service1()の処理
    }
    [TP1RpcMethod]
    public void Service2(byte[] inData, int inLen,
                         byte[] outData, ref int outLen)
    {
      // Service2()の処理
    }
    // SPP内部メソッドの実装（自由に実装できます）
    // RPCで呼び出すことはできません
    public string GetUserInfo()
    {
      // 処理
    }
    private void PutErrorLog(string errorInfo, int errorCode)
    {
      // 処理
    }
  }
}
```

(2) J#の場合のコーディング例

```
package MyApplication;
import System.*;
import Hitachi.OpenTP1.*;
import Hitachi.OpenTP1.Server.*;

public class MySPP extends SPPBase
{
  public MySPP()
```

```
  {
    // コンストラクタの処理
    // ** コンストラクタでOpenTP1のクラスは使用できません **
    // ** 必要な初期化処理はInitializeSPPで行ってください **
  }
  protected void Finalize()
  {
    // Finalizeメソッドの処理
    // ** FinalizeメソッドでOpenTP1のクラスは使用できません **
    // ** 必要な終了処理はFinalizeSPPで行ってください **
  }
  // SPP初期化および終了処理メソッドの実装
  public void InitializeSPP() {
    // SPPの初期化処理
  }
  public void FinalizeSPP() {
    // SPPの終了処理
  }

  // サービスメソッドの実装（引数の並びは固定です）
  // クライアントからRPCで呼び出されます
  /** @attribute TP1RpcMethod() */
  void Service1(ubyte[] inData, int inLen,
                ubyte[] outData, IntHolder outLen)
  {
    // Service1()の処理
  }

  /** @attribute TP1RpcMethod () */
  void Service2(ubyte[] inData, int inLen,
                ubyte[] outData, IntHolder outLen)
  {
    // Service2()の処理
  }

  // SPP内部メソッドの実装（自由に実装できます）
  // RPCで呼び出すことはできません
  public String GetUserInfo()
  {
    // 処理
  }
  private void PutErrorLog(String errorInfo, int errorCode)
  {
    // 処理
  }
}
```

(3) Visual Basic の場合のコーディング例

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Server

Namespace MyApplication
  Public Class MySPP
    Inherits SPPBase
```

```
    Sub MySPP()
      ' コンストラクタの処理
      ' ** コンストラクタでOpenTP1のクラスは使用できません **
      ' ** 必要な初期化処理はInitializeSPPで行ってください **
    End Sub
    Sub Finalize()
      'Finalizeメソッドの処理
      ' ** FinalizeメソッドでOpenTP1のクラスは使用できません **
      ' ** 必要な終了処理はFinalizeSPPで行ってください **
    End Sub
    ' SPP初期化および終了処理メソッドの実装
    Public Overrides Sub InitializeSPP()
      ' SPPの初期化処理
    End Sub
    Public Overrides Sub FinalizeSPP()
      ' SPPの終了処理
    End Sub

    ' サービスメソッドの実装（引数の並びは固定です）
    ' クライアントからRPCで呼び出されます
    <TP1RpcMethod()> _
    Public Sub Service1(ByVal inData() As Byte, _
                        ByVal inLen As Integer, _
                        ByVal outData() As Byte, _
                        ByRef outLen As Integer)
      ' Service1()の処理
    End Sub

    <TP1RpcMethod()> _
    Public Sub Service2(ByVal inData() As Byte, _
     ByVal inLen As Integer, _
     ByVal outData() As Byte, ByRef outLen As Integer)
      ' Service2()の処理
    End Sub

    ' SPP内部メソッドの実装（自由に実装できます）
    ' RPCで呼び出すことはできません
    Public Function GetUserInfo() As String
      ' 処理
    End Function
    Private Sub PutErrorLog(errorInfo As String, _
                            errorCode As Integer)
      ' 処理
    End Sub
  End Class
End Namespace
```

4.4　.NET インタフェース定義を使用した SPP.NET の呼び出し方法

SPP.NET または SUP.NET から.NET インタフェース定義を使用して SPP.NET を呼び出す方法について説明します。

Client .NET，Connector .NET の各 UAP からの呼び出し方法については，それぞれマニュアル「TP1/Client for .NET Framework　使用の手引」，マニュアル「TP1/Connector for .NET Framework　使用の手引」の.NET インタフェース定義を使用した SPP.NET の呼び出し方法を参照してください。

4.4.1　クライアントスタブの生成

クライアントスタブ生成コマンド（if2cstub）を使用して，.NET インタフェース定義からクライアントスタブを生成します。

クライアントスタブは，クライアントスタブを利用するクライアントアプリケーション（SPP.NET または SUP.NET）を記述するプログラム言語で生成してください。

.NET インタフェース定義を記述したプログラム言語と生成するクライアントスタブのプログラム言語が異なる場合，クライアントスタブのメソッドの引数のパラメタ属性は次のように対応づけられます。

表 4-5　クライアントスタブでのパラメタ属性の対応づけ

プログラム言語とメソッドの引数のパラメタ属性（.NET インタフェース定義）		メソッドの引数のパラメタ属性（クライアントスタブ）		
		C#	J#	Visual Basic
C#	なし	なし	なし	ByVal
	out	out	Holder クラス※1	ByRef※1
	ref	ref	Holder クラス	ByRef
J#	なし	なし	なし	ByVal
	Holder クラス	ref※2	Holder クラス	ByRef※2
Visual Basic	なし	なし	なし	ByVal
	ByVal	なし	なし	ByVal
	ByRef	ref	Holder クラス	ByRef

注※1

呼び出し元で値を設定しても，値はサーバに渡されません。

注※2

Holder クラスは，各 Holder クラスが保持する型の参照渡しに対応づけられます。

4.4.2　クライアントスタブの使用方法

クライアントスタブを使用してサービス要求をする手順を次に示します。

1. クライアントスタブをサービスグループごとにインスタンス化します。
   SPP.NET で使用する場合は，InitializeSPP メソッドでインスタンス化するか，または生成したインスタンスを static フィールドなどに保持するようにしてください。
2. 入力パラメタにデータを設定します。
3. 必要に応じてプロパティ（Flags など）を設定します。
4. サービスメソッドを呼び出します。

以降，必要に応じて 2.～4.を繰り返し実行できます。

4.4.3　.NET インタフェース定義から生成したクライアントスタブの使用例

クライアントスタブの使用例を次に示します。

この例で呼び出す SPP.NET のサービスメソッドの情報は次のとおりです。

- サービスグループ名：GRP1
- インタフェース名：MyCompany.IGyoumuA
- 呼び出すサービスメソッド名（サービス名）：Service3

なお，コメント中の(1)，(2)などは「4.4.2　クライアントスタブの使用方法」の説明の番号に対応しています。

(1)　.NET インタフェース定義の定義例（C#の場合）

```
namespace MyCompany
{
  using System;

  public interface IGyoumuA
  {
    void Service1(string dataId, byte[] data);
    string[] Service2(string key);
    int Service3(int inCount, ref string[] ids);
  }
}
```

(2) クライアントスタブの使用例（SPP.NET，C#の場合）

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;

namespace MyCompany
{
  public class CallerSampleImpl : SPPBase, ICallerSample
  {
    private IGyoumuAStub server = null;
    …
    public override void InitializeSPP()
    {
      // (1) クライアントスタブの生成
      server = new IGyoumuAStub(”GRP1”);
    }

    public override void FinalizeSPP()
    {
      server = null;
    }

    [TP1RpcMethod]
    public int CallService3()
    {
      try {
        string[] ids =
          // (2) 入力データの設定
          new string[3]{”data1”, ”data2”, ”data3”};
        // (3) 同期応答型RPCに設定
        server.Flags = TP1ServerFlags.DCNOFLAGS;
        // (4) service3を呼び出す
        int ret = server.Service3(3, ref ids);
        return ret;
      } catch (TP1UserException exp) {
        // Service3()からユーザ例外がスローされた
      } catch (TP1RemoteException exp) {
        // Service3()で予期しない例外発生
      } catch (TP1ServerException exp) {
        // OpenTP1(クラスライブラリ)が検知したエラー
      } catch (TP1Exception exp) {
        // その他OpenTP1が検知したエラー
      } catch (Exception exp) {
        // 予期しない例外
    }
  }
}
```

(3) クライアントスタブの使用例（SPP.NET，J#の場合）

```
package MyCompany;
import System.*;
import Hitachi.OpenTP1.*;
import Hitachi.OpenTP1.Server.*;
```

```
public class CallerSampleImpl
       extends SPPBase implements ICallerSample
{
    private IGyoumuAStub server = null;
    …
    public void InitializeSPP() throws TP1ServerException
    {
      // (1) クライアントスタブの生成
      server = new IGyoumuAStub("GRP1");
    }
    public void FinalizeSPP()
    {
      server = null;
    }
    /** @attribute TP1RpcMethod() */
    public int CallService3()
    {
      try {
        String[] ids =
          // (2) 入力データの設定
          new String[]{"data1", "data2", "data3"};
        // (2) 入力データの設定
        StringArrayHolder idsHolder = new StringArrayHolder();
        // (2) 入力データの設定
        idsHolder.set_Value(ids);
        // (3) 同期応答型RPCに設定
        server.set_Flags(TP1ServerFlags.DCNOFLAGS);
        // (4) service3を呼び出す
        int ret = server.Service3(3, idsHolder);
        return ret;
      } catch (TP1UserException exp) {
        // Service3()からユーザ例外がスローされた
      } catch (TP1RemoteException exp) {
        // Service3()で予期しない例外発生
      } catch (TP1ServerException exp) {
        // OpenTP1(クラスライブラリ)が検知したエラー
      } catch (TP1Exception exp) {
        // その他OpenTP1が検知したエラー
      } catch (System.Exception exp) {
        // 予期しない例外
      }
    }
}
```

(4) クライアントスタブの使用例（SPP.NET，Visual Basic の場合）

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Server

Namespace MyCompany
  Public Class CallerSampleImpl
    Inherits SPPBase
    Implements ICallerSample
    Private server As IGyoumuAStub
```

```
    ...
    Public Overrides Sub InitializeSPP()
      '(1) クライアントスタブの生成
      server = New IGyoumuStub("GRP1")
    End Sub

    Public Overrides Sub FinalizeSPP()
      server = Nothing
    End Sub

    <TP1RpcMethod()> _
    Public Function CallService3() As Integer _
         Implements ICallerSample.CallerService3
      Try
        Dim ret As Integer
        Dim ids() As String = _
          '(2) 入力データの設定
          New String(){"data1","data2","data3"}
        '(3) 同期応答型RPCに設定
        server.Flags = TP1ServerFlags.DCNOFLAGS
        '(4) service3を呼び出す
        ret = server.Service3(3, ids)
      Catch exp As TP1UserException
      'Service3()からユーザ例外がスローされた
      Catch exp As TP1RemoteException
      'Service3()で予期しない例外発生
      Catch exp As TP1ServerException
      'OpenTP1(クラスライブラリ)が検知したエラー
      Catch exp As TP1Exception
      'その他OpenTP1が検知したエラー
      Catch exp As Exception
      '予期しない例外
      End Try
    End Function
  End Class
End Namespace
```

(5) クライアントスタブの使用例（SUP.NET，C#の場合）

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;

namespace MyCompany
{
  public class CallerSample
  {
  public static void Main(string[] args)
    {
      try {
        IGyoumuAStub server =
          // (1) クライアントスタブの生成
          new IGyoumuAStub("GRP1");
        Rpc.Open();          // RPCオープン
        Adm.Complete();      // SUP.NET開始処理完了通知
```

```
        // (2) 入力データの設定
        string[] ids = new string[]{"data1","data2","data3"};
        // (3) 同期応答型RPCに設定
        server.Flags = TP1ServerFlags.DCNOFLAGS;
        // (4) service3を呼び出す
        int ret = server.Service3(3, ref ids);

        Rpc.Close();        // RPCクローズ
      } catch (TP1UserException exp) {
        // Service3()からユーザ例外がスローされた
      } catch (TP1RemoteException e) {
        // Service3()で予期しない例外発生
      } catch (TP1ServerException exp) {
        // OpenTP1(クラスライブラリ)が検知したエラー
      } catch (TP1Exception exp) {
        // その他OpenTP1が検知したエラー
      } catch (Exception exp) {
        // 予期しない例外
      }
    }
  }
}
```

(6) クライアントスタブの使用例（SUP.NET，J#の場合）

```
package MyCompany;
import System.*;
import Hitachi.OpenTP1.*;
import Hitachi.OpenTP1.Server.*;

public class CallerSample
{
  public void main(String args[])
    {
     try {
      int ret;
        // (1) クライアントスタブの生成
        IGyoumuAStub server = new IGyoumuAStub("GRP1");
        Rpc.Open();                // RPCオープン
        Adm.Complete();            // SUP.NET開始処理完了通知

        // (2) 入力データの設定
        StringArrayHolder idsHolder = new StringArrayHolder();
        // (2) 入力データの設定
        String[] ids = new String[]{"data1","data2","data3"};
        // (2) 入力データの設定
        idsHolder.set_Value(ids);
        // (3) 同期応答型RPCに設定
        server.set_Flags(TP1ServerFlags.DCNOFLAGS);
        // (4) service3を呼び出す
        ret = server.Service3(3, idsHolder);

        Rpc.Close();               // RPCクローズ
      } catch (TP1UserException exp) {
        // Service3()からユーザ例外がスローされた
      } catch (TP1RemoteException exp) {
```

```
        // Service3()で予期しない例外発生
      } catch (TP1ServerException exp) {
        // OpenTP1(クラスライブラリ)が検知したエラー
      } catch (TP1Exception exp) {
        // その他OpenTP1が検知したエラー
      } catch (System.Exception exp) {
        // 予期しない例外
      }
    }
}
```

(7) クライアントスタブの使用例（SUP.NET，Visual Basic の場合）

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Server

Namespace MyCompany
Public Class CallerSample

    Public Shared Sub Main(ByVal args() As String)
      Dim ret As Integer
      Try
        ’(1) クライアントスタブの生成
        Dim server As IGyoumuAStub = New IGyoumuAStub(”GRP1”)
        Rpc.Open()                  ’RPCオープン
        Adm.Complete()              ’SUP.NET開始処理完了通知

        Dim ids() As String = _
          ’(2) 入力データの設定
          New String(){”data1”,”data2”,”data3”}
        ’(3) 同期応答型RPCに設定
        server.Flags = TP1ServerFlags.DCNOFLAGS
        ’(4) service3を呼び出す
        ret = server.Service3(3, ids)

        Rpc.Close()                 ’RPCクローズ
      Catch exp As TP1UserException
      ’Service3()からユーザ例外がスローされた
      Catch exp As TP1RemoteException
      ’Service3()で予期しない例外発生
      Catch exp As TP1ServerException
      ’OpenTP1(クラスライブラリ)が検知したエラー
      Catch exp As TP1Exception
      ’その他OpenTP1が検知したエラー
      Catch exp As Exception
      ’予期しない例外
      End Try
    End Sub
  End Class
End Namespace
```

4.4.4　.NET インタフェース定義を使用する場合のデータ長の見積もり

RPC 要求インタフェースに.NET インタフェース定義を使用する場合，Extension .NET が入出力パラメタや戻り値と，送受信メッセージの間の変換をします。ただし，送受信メッセージが RPC メッセージの最大長を超えないように，要求メッセージ長および応答メッセージ長の値を見積もっておく必要があります。

要求メッセージ長および応答メッセージ長の見積もり式を次に示します

要求メッセージ長=（メソッドの入力パラメタおよび参照パラメタの最大データ長の合計値）+512

応答メッセージ長=（メソッドの出力パラメタ，参照パラメタ，および戻り値の最大データ長の合計値）+512

最大データ長の合計値には，データ型ごとの最大データ長を合計した値を代入してください。各データ型の最大データ長を次の表に示します。

表 4-6　各データ型の最大データ長

データ型	最大データ長（バイト）※
System.Byte	1
System.Int16	3
System.Int32	7
System.Int64	15
System.String	格納された文字数×2+13
System.Byte[]	a+11
System.Int16[]	2×a+11
System.Int32[]	4×a+11
System.Int64[]	8×a+11
System.String[]	∑（Stringに格納された文字数×2+13）+11
TP1 ユーザ構造体	各メンバの最大長の合計+4
TP1 ユーザ構造体配列	∑（各メンバの最大長の合計+4）+11

（凡例）

a：配列の要素数

注※

スタブによって自動的にアライメント調整されるサイズを含んでいます。

4.4.5　.NET インタフェース定義を使用する場合の留意事項

.NET インタフェース定義を使用する場合の留意事項を次に示します。

- .NET インタフェース定義を使用してクライアントスタブを生成する場合，クライアントスタブの各インスタンスは，要求メッセージのうち最大の要求メッセージ長分のメモリを使用します。このメモリはクライアントスタブの解放時に解放されます。したがって，大きな要求メッセージを使用した場合で，そのクライアントスタブを頻繁に使用しないときまたはそれ以上大きな要求メッセージを使用しないときは，使用するたびにクライアントスタブを解放することでメモリ使用量を軽減できます。

  逆に，同じクライアントスタブを頻繁に使用する場合は，クライアントスタブを毎回生成しないで，同じクライアントスタブを使い続けると性能が向上します。
- クライアントスタブを使用するプロセスでは，応答メッセージ用として，1 メガバイトのメモリを使用します。ただし，if2cstub コマンドの-m オプションを指定した場合は，「指定値×1 メガバイト」のメモリを使用します。したがって，必要以上に大きな値を設定しないようにしてください。
- 複数のクライアントスタブを使用するプロセスでは，複数あるクライアントスタブの中で最もメモリを使用する値を応答メッセージ用としてプロセスで使用します。

4.5　サービス定義を使用した SPP.NET または SPP の呼び出し方法

SPP.NET または SUP.NET からサービス定義を使用して SPP.NET を呼び出す方法について説明します。

Client .NET，Connector .NET の各 UAP からの呼び出し方法については，それぞれマニュアル「TP1/Client for .NET Framework　使用の手引」，マニュアル「TP1/Connector for .NET Framework　使用の手引」のサービス定義を使用した SPP.NET または SPP の呼び出し方法を参照してください。

4.5.1　クライアントスタブの生成

クライアントスタブ生成コマンド（spp2cstub）を使用して，サービス定義からクライアントスタブおよびカスタムレコードクラスを生成します。

クライアントスタブおよびカスタムレコードクラスは，クライアントスタブを利用するクライアントアプリケーション（SPP.NET または SUP.NET）を記述するプログラム言語で生成してください。

データ型定義で指定された各メンバは，カスタムレコードクラスの public プロパティとなります。

4.5.2　クライアントスタブの使用方法

クライアントスタブを使用してサービス要求をする手順を次に示します。

1. クライアントスタブをサービスグループごとにインスタンス化します。
   SPP.NET で使用する場合は，InitializeSPP メソッドでインスタンス化するか，または生成したインスタンスを static フィールドなどに保持するようにしてください。
2. 入力用，および出力用カスタムレコードクラスをインスタンス化します。
3. 入力用カスタムレコードのプロパティに入力データを設定します。
4. 必要に応じてプロパティ（Flags など）を設定します。
5. サービスメソッドを呼び出します。
6. 出力用カスタムレコードのプロパティから応答データを参照します。

以降，必要に応じて 2.～6.，または 3.～6.を繰り返し実行できます。

4.5.3　サービス定義から生成したクライアントスタブの使用例

クライアントスタブの使用例を次に示します。

この例で呼び出す SPP.NET のサービスメソッドの情報は次のとおりです。

- サービスグループ名：SVGRP1
- サービス定義名称：GYOUMU1
- 呼び出すサービスメソッド名（サービス名）：GETDATA1
- 入力用カスタムレコードクラス名：in_data
- 出力用カスタムレコードクラス名：out_data

なお，コメント中の(1)，(2)などは「4.5.2　クライアントスタブの使用方法」の説明の番号に対応しています。

(1)　サービス定義の定義例（C＃の場合）

(a)　サービス定義の定義例（業務 1 のサービス定義）

```
#include ”mydata.h”
/* 業務1のサービス定義 */
interface GYOUMU1 {
    GETDATA1(in_data, out_data);
    GETDATA2(in_data, out_data2);
}
```

(b)　データ型定義の定義例（mydata.h）

```
struct in_data {
        long i_basho[3];
        long i_kakaku;
};
struct out_data {
        char o_name[20];
        char o_basho[16];
        long o_kakaku;
        char o_inf[80];
};
struct out_data2 {
        int o_count;
        struct data {
                char o_name[20];
                char o_basho[16];
                long o_kakaku;
                char o_inf[80];
        } data_t[100];
};
```

(2)　カスタムレコードクラスの出力例（C#の場合，in_data.cs）

```
using Hitachi.OpenTP1.Common;

namespace MyCompany
{
    public class in_data : RecordImpl
```

```
    {
        public in_data() : base("default")
        {
            …
        }
        public in_data(string recordName) : base(recordName)
        {
            …
        }
        …
        private int[] _i_basho;
        public int[] i_basho
        {
            get
            {
                return _i_basho;
            }
            set
            {
                _i_basho = value;
            }
        }

        private int _i_kakaku = 0;
        public int i_kakaku
        {
            get
            {
                return _i_kakaku;
            }
            set
            {
                _i_kakaku = value;
            }
        }
        …
    }
}
```

(3) クライアントスタブの使用例（SPP.NET，C#の場合）

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;

namespace MyCompany
{
public class CallerSample2Impl : SPPBase, ICallerSample2
  {
    private GYOUMU1Stub server = null;
    …
    public override void InitializeSPP()
    {
      // (1) クライアントスタブの生成
      server = new GYOUMU1Stub("SVGRP1");
    }
```

```
    public override void FinalizeSPP()
    {
      server = null;
    }
    [TP1RpcMethod]
    public string CallGETDATA1()
    {
      string ret;
      try {
        // (2) 入力用カスタムレコードの生成
        in_data inRecord = new in_data();
        // (2) 出力用カスタムレコードの生成
        out_data outRecord = new out_data();
        inRecord.i_basho[0] = 56;  // (3) 入力データの設定
        inRecord.i_basho[1] = 43;  // (3) 入力データの設定
        inRecord.i_basho[2] = 18;  // (3) 入力データの設定
        // (4) 同期応答型RPCに設定
        server.Flags = TP1ServerFlags.DCNOFLAGS;
        // (5) GETDATA1を呼び出す
        server.GETDATA1(inRecord, outRecord);
        // (6) 応答データの取り出し
        ret = outRecord.o_name.Trim();
        return ret;
      } catch (TP1UserException exp) {
        // GETDATA1()からユーザ例外がスローされた
      } catch (TP1RemoteException exp) {
        // GETDATA1()で予期しない例外発生
      } catch (TP1ServerException exp) {
        // OpenTP1(クラスライブラリ)が検知したエラー
      } catch (TP1Exception exp) {
        // その他OpenTP1が検知したエラー
      } catch (Exception exp) {
        // 予期しない例外
      }
    }
  }
}
```

(4) クライアントスタブの使用例（SPP.NET，J#の場合）

```
package MyCompany;
import System.*;
import Hitachi.OpenTP1.*;
import Hitachi.OpenTP1.Server.*;

public class CallerSample2Impl
       extends SPPBase implements ICallerSample2
{
    private GYOUMU1Stub server = null;
    …
    public void InitializeSPP() throws TP1ServerException
    {
      // (1) クライアントスタブの生成
      server = new GYOUMU1Stub("SVGRP1");
    }
    public void FinalizeSPP()
```

```
    {
      server = null;
    }

    /** @attribute TP1RpcMethod() */
    public String CallGETDATA1()
    {
      String ret;
      try {
        // (2) 入力用カスタムレコードの生成
        in_data inRecord = new in_data();
        // (2) 出力用カスタムレコードの生成
        out_data outRecord = new out_data();
        inRecord.get_i_basho()[0] = 56;  // (3) 入力データの設定
        inRecord.get_i_basho()[1] = 43;  // (3) 入力データの設定
        inRecord.get_i_basho()[2] = 18;  // (3) 入力データの設定
        // (4) 同期応答型RPCに設定
        server.set_Flags(TP1ServerFlags.DCNOFLAGS);
        // (5) GETDATA1を呼び出す
        server.GETDATA1(inRecord, outRecord);
        // (6) 応答データの取り出し
        ret = outRecord.get_o_name().trim();
        return ret;
      } catch (TP1UserException exp) {
        // GETDATA1()からユーザ例外がスローされた
      } catch (TP1RemoteException exp) {
        // GETDATA1()で予期しない例外発生
      } catch (TP1ServerException exp) {
        // OpenTP1(クラスライブラリ)が検知したエラー
      } catch (TP1Exception exp) {
        // その他OpenTP1が検知したエラー
      } catch (System.Exception exp) {
        // 予期しない例外
      }
    }
}
```

(5) クライアントスタブの使用例（SPP.NET，Visual Basic の場合）

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Server

Namespace MyCompany
  Public Class CallerSample2Impl
    Inherits SPPBase
    Implements ICallerSample2
    Private server As GYOUMU1Stub
    …
    Public Overrides Sub InitializeSPP()
      ’(1) クライアントスタブの生成
      server = New GYOUMU1Stub(”SVGRP1”)
    End Sub
    Public Overrides Sub FinalizeSPP()
      server = Nothing
    End Sub
```

```
    <TP1RpcMethod()> _
    Public Function CallGETDATA1 () As String _
      Implements ICallerSample2.CallGETDATA1
      Try
        Dim ret As String
        '(2) 入力用カスタムレコードの生成
        Dim inRecord As in_data = New in_data()
        '(2) 出力用カスタムレコードの生成
        Dim outRecord As out_data = New out_data()
        inRecord.i_basho(0) = 56  '(3) 入力データの設定
        inRecord.i_basho(1) = 43  '(3) 入力データの設定
        inRecord.i_basho(2) = 18  '(3) 入力データの設定
        '(4) 同期応答型RPCに設定
        server.Flags = TP1ServerFlags.DCNOFLAGS
        '(5) GETDATA1を呼び出す
        server.GETDATA1(inRecord, outRecord)
        '(6) 応答データの取り出し
        ret = outRecord.o_name.Trim()
      Catch exp As TP1UserException
        'GETDATA1()からユーザ例外がスローされた
      Catch exp As TP1RemoteException
        'GETDATA1()で予期しない例外発生
      Catch exp As TP1ServerException
        'OpenTP1(クラスライブラリ)が検知したエラー
      Catch exp As TP1Exception
        'その他OpenTP1が検知したエラー
      Catch exp As Exception
        '予期しない例外
      End Try
    End Function
  End Class
End Namespace
```

(6) クライアントスタブの使用例（SUP.NET，C#の場合）

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;

namespace MyCompany
{
  public class Caller2Sample
  {
    ...
    public static void Main(string[] args)
    {
      try {
        // (1) クライアントスタブの生成
        GYOUMU1Stub server = new GYOUMU1Stub("SVGRP1");
        Rpc.Open();          // RPCオープン
        Adm.Complete();      // SUP.NET開始処理完了通知

        // (2) 入力用カスタムレコードの生成
        in_data inRecord = new in_data();
        // (2) 出力用カスタムレコードの生成
        out_data outRecord = new out_data();
```

```
        inRecord.i_basho[0] = 56;  // (3) 入力データの設定
        inRecord.i_basho[1] = 43;  // (3) 入力データの設定
        inRecord.i_basho[2] = 18;  // (3) 入力データの設定
        // (4) 同期応答型RPCに設定
        server.Flags = TP1ServerFlags.DCNOFLAGS;
        // (5)GETDATA1を呼び出す
        server.GETDATA1(inRecord, outRecord);
        // (6) 応答データの取り出し
        string name = outRecord.o_name.Trim();

        Rpc.Close();               // RPCクローズ
      } catch (TP1UserException exp) {
        // GETDATA1()からユーザ例外がスローされた
      } catch (TP1RemoteException exp) {
        // GETDATA1()で予期しない例外発生
      } catch (TP1ServerException exp) {
        // OpenTP1(クラスライブラリ)が検知したエラー
      } catch (TP1Exception exp) {
        // その他OpenTP1が検知したエラー
      } catch (Exception exp) {
        // 予期しない例外
      }
    }
  }
}
```

(7) クライアントスタブの使用例（SUP.NET，J#の場合）

```
package MyCompany;
import System.*;
import Hitachi.OpenTP1.*;
import Hitachi.OpenTP1.Server.*;

public class Caller2Sample
{
  ...
    public static void main(String[] args)
    {
      try {
        String name;
        // (1) クライアントスタブの生成
        GYOUMU1Stub server = new GYOUMU1Stub("SVGRP1");
        Rpc.Open();          // RPCオープン
        Adm.Complete();      // SUP.NET開始処理完了通知

        // (2) 入力用カスタムレコードの生成
        in_data inRecord = new in_data();
        // (2) 出力用カスタムレコードの生成
        out_data outRecord = new out_data();
        inRecord.get_i_basho()[0] = 56;  // (3) 入力データの設定
        inRecord.get_i_basho()[1] = 43;  // (3) 入力データの設定
        inRecord.get_i_basho()[2] = 18;  // (3) 入力データの設定
        // (4) 同期応答型RPCに設定
        server.set_Flags(TP1ServerFlags.DCNOFLAGS);
        // (5) GETDATA1を呼び出す
        server.GETDATA1(inRecord, outRecord);
```

```
        // (6) 応答データの取り出し
        name = outRecord.get_o_name().trim();

        Rpc.Close();                    // RPCクローズ
      } catch (TP1UserException exp) {
        // GETDATA1()からユーザ例外がスローされた
      } catch (TP1RemoteException exp) {
        // GETDATA1()で予期しない例外発生
      } catch (TP1ServerException exp) {
        // OpenTP1(クラスライブラリ)が検知したエラー
      } catch (TP1Exception exp) {
        // その他OpenTP1が検知したエラー
      } catch (Exception exp) {
        // 予期しない例外
      }
    }
}
```

(8) クライアントスタブの使用例（SUP.NET，Visual Basic の場合）

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Server

Namespace MyCompany
  Public Class Caller2Sample
  ...
    Public Shared Sub Main(ByVal args() As String)
      Dim name As String
      Dim inRecord As in_data
      Dim outRecord As out_data
      Try
        ' (1) クライアントスタブの生成
        Dim server As GYOUMU1Stub = _
                      New GYOUMU1Stub("SVGRP1")
        Rpc.Open()          ' RPCオープン
        Adm.Complete()      ' SUP.NET開始処理完了通知

        ' (2) 入力用カスタムレコードの生成
        inRecord = New in_data()
        ' (2) 出力用カスタムレコードの生成
        outRecord = New out_data()
        inRecord.i_basho(0) = 56  ' (3) 入力データの設定
        inRecord.i_basho(1) = 43  ' (3) 入力データの設定
        inRecord.i_basho(2) = 18  ' (3) 入力データの設定
        ' (4) 同期応答型RPCに設定
        server.Flags = TP1ServerFlags.DCNOFLAGS
        ' (5) GETDATA1を呼び出す
        server.GETDATA1(inRecord, outRecord)
        ' (6) 応答データの取り出し
        name = outRecord.o_name.Trim()

        Rpc.Close()               ' RPCクローズ
      Catch exp As TP1UserException
      ' GETDATA1()からユーザ例外がスローされた
      Catch exp As TP1RemoteException
```

```
      ' GETDATA1()で予期しない例外発生
      Catch exp As TP1ServerException
      ' OpenTP1(クラスライブラリ)が検知したエラー
      Catch exp As TP1Exception
      ' その他OpenTP1が検知したエラー
      Catch exp As Exception
      ' 予期しない例外
      End Try
    End Sub
  End Class
End Namespace
```

4.5.4 サービス定義から生成したクライアントスタブを使用する場合のデータ長の見積もり

サービス定義から生成したクライアントスタブを使用する場合，Extension .NET がカスタムレコードと，送受信メッセージの間の変換をします。ただし，送受信メッセージが RPC メッセージの最大長を超えないように，要求メッセージ長および応答メッセージ長の値を見積もっておく必要があります。

要求メッセージ長および応答メッセージ長の見積もり式を次に示します

要求メッセージ長=データ型定義の各メンバのデータ長の合計値

応答メッセージ長=データ型定義の各メンバのデータ長の合計値

データ長の合計値には，データ型ごとのデータ長を合計した値を代入してください。各データ型のデータ長を次の表に示します。

表 4-7　各データ型のデータ長

データ型	データ長（バイト）
char	1
short	2
int	4
long	4
char[a]	1×a
byte[a]	1×a
short[a]	2×a
int[a]	4×a
long[a]	4×a
char[a][b	a×b
struct	構造体の各メンバのサイズの合計

データ型	データ長（バイト）
struct[a]	構造体の各メンバのサイズの合計×a

（凡例）

a，b：配列の要素数

なお，可変長構造体配列の場合，配列の最大要素数で見積もるようにしてください。

要求メッセージ長および応答メッセージ長の見積もりが完了したら，カスタムレコードクラスの private const フィールドの_length を参照して，見積もり後のメッセージ長を確認します。ただし，可変長構造体配列を使用している場合，この方法では確認できません。可変長構造体配列を使用している場合は，spp2cstub コマンドに-b オプションを指定してコマンドを実行し，標準出力に出力されたバッファサイズを参照して，メッセージ長を確認してください。

4.6　バイナリデータを使用した SPP.NET または SPP の呼び出し方法

SPP.NET または SUP.NET からバイナリデータを使用して SPP.NET または SPP を呼び出す方法について説明します。

Client .NET，Connector .NET の各 UAP からの呼び出し方法については，それぞれマニュアル「TP1/Client for .NET Framework　使用の手引」のバイナリデータを使用した SPP.NET または SPP の呼び出し方法，マニュアル「TP1/Connector for .NET Framework　使用の手引」のインデクスドレコードを使用した SPP.NET または SPP の呼び出し方法を参照してください。

4.6.1　バイナリデータを使用する場合の留意事項

バイナリデータを使用して RPC 要求をする場合は，.NET インタフェース定義，サービス定義，およびクライアントスタブは使用しません。

.NET インタフェース定義を使用していない SPP.NET を呼び出すことができます。.NET インタフェース定義を使用した SPP.NET に対してはこの方法での RPC 要求はできません。

> **参考**
>
> バイナリデータを使用した RPC 要求は，通常，C 言語や COBOL 言語で記述された SPP の呼び出しに使用します。

バイナリデータを使用して RPC 要求をする場合は，Rpc クラスの Call メソッドを使用します。Call メソッドの引数には入力データ用のバイト配列と出力データ用のバイト配列があり，これらの領域を SPP.NET，または SUP.NET で用意しておく必要があります。

RPC のメッセージの形式は呼び出し元の SPP.NET または SUP.NET と呼び出し先の SPP.NET または SPP との間であらかじめ決めておきます。この形式に従って，SPP.NET または SUP.NET でバイナリデータとしての編集処理をする必要があります。

4.6.2　バイナリデータを使用する場合の Call メソッドの使用例

バイナリデータを使用する場合の Call メソッドの使用例を次に示します。

この例で呼び出す SPP のサービスの情報は次のとおりです。

- サービスグループ名：SVGRP1
- 呼び出すサービス名：SERV1

(1) Call メソッドの使用例（SUP.NET，C#の場合）

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;

namespace MyCompany
{
  public class MyApplication1
  {
    ...
    public static void Main(string[] args)
    {
      try {
        Rpc.Open();                  // RPCオープン
        Adm.Complete();              // SUP.NET開始処理完了通知

        int    maxInLen = 512;       // 入力データ格納領域長
        int    maxOutLen = 512;      // 応答データ格納領域長
        int    inLen = 0;            // 入力データ長
        int    outLen = maxOutLen;   // 応答データ最大長
        byte[] inData;               // 入力データ格納領域
        byte[] outData;              // 応答データ格納領域
        String inStr = "Say Hello to OpenTP1!";  // 入力データ
        String outStr = null;                    // 応答データ
        byte[] inDataTemp;

        inData = new byte[maxInLen];
        System.Text.Encoding enc = System.Text.Encoding.Default;
        inDataTemp = enc.GetBytes(inStr);
        System.Array.Copy(           // 入力データの設定
            inDataTemp, 0, inData, 0, inDataTemp.Length);
        inLen += inDataTemp.Length;
        outData = new byte[maxOutLen];
        // RPC実行
        Rpc.Call("SVGRP1", "SERV1", inData, inLen,
            outData, ref outLen, TP1ServerFlags.DCNOFLAGS);
        outStr = enc.GetString(outData, 0, outLen);

        Rpc.Close();                 // RPCクローズ
      } catch (TP1ServerException exp) {
        // OpenTP1(クラスライブラリ)が検知したエラー
      } catch (TP1Exception exp) {
        // その他OpenTP1が検知したエラー
      } catch (Exception exp) {
        // 予期しない例外
      }
    }
  }
}
```

(2) Call メソッドの使用例（SUP.NET，J#の場合）

```
package MyCompany;
import System.*;
```

```
import Hitachi.OpenTP1.*;
import Hitachi.OpenTP1.Server.*;

public class MyApplication1
{
  ...
  public static void main(String[] args)
  {
    try {
      Rpc.Open();                    // RPCオープン
      Adm.Complete();                // SUP.NET開始処理完了通知

      int     maxInLen = 512;        // 入力データ格納領域長
      int     maxOutLen = 512;       // 応答データ格納領域長
      int     inLen  = 0;            // 入力データ長
      int     outLen = maxOutLen;    // 応答データ最大長
      ubyte[] inData;                // 入力データ格納領域
      ubyte[] outData;               // 応答データ格納領域
      String  inStr = "Say Hello to OpenTP1!";  // 入力データ
      String  outStr = null;                    // 応答データ
      ubyte[] inDataTemp;

      inData = new ubyte[maxInLen];
      System.Text.Encoding enc =
                  System.Text.Encoding.get_Default();
      inDataTemp = enc.GetBytes(inStr);
      System.Array.Copy(             // 入力データの設定
          inDataTemp, 0, inData, 0, inDataTemp.length);
      inLen += inDataTemp.length;
      outData = new ubyte[maxOutLen];
      // RPC実行
      Rpc.Call("SVGRP1", "SERV1", inData, inLen,
          outData, outLen, TP1ServerFlags.DCNOFLAGS);
      outStr = enc.GetString(outData, 0, outLen);

      Rpc.Close();                   // RPCクローズ
    } catch (TP1ServerException exp) {
      // OpenTP1(クラスライブラリ)が検知したエラー
    } catch (TP1Exception exp) {
      // OpenTP1が検知したエラー
    } catch (System.Exception exp) {
      // 予期しない例外
    }
  }
}
```

(3) Call メソッドの使用例（SUP.NET，Visual Basic の場合）

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Server

Namespace MyCompany
  Public Class MyApplication1
    ...
    Public Shared Sub Main(ByVal args() As String)
```

```
      Dim maxInLen As Integer = 512       ' 入力データ格納領域長
      Dim maxOutLen As Integer = 512      ' 応答データ格納領域長
      Dim inLen As Integer                ' 入力データ長
      Dim outLen As Integer               ' 応答データ最大長
      Dim inData(maxInLen - 1) As Byte    ' 入力データ格納領域
      Dim outData(maxOutLen - 1) As Byte  ' 応答データ格納領域
      Dim inStr As String                 ' 入力データ
      Dim outStr As String                ' 応答データ
      Dim inDataTemp() As Byte
      Dim enc As System.Text.Encoding
      Try
        Rpc.Open()           ' RPCオープン
        Adm.Complete()       ' SUP.NET開始処理完了通知

        inLen = 0
        outLen = maxOutLen
        inStr = "Say Hello to OpenTP1!"
        enc = System.Text.Encoding.Default
        inDataTemp = enc.GetBytes(inStr)
        System.Array.Copy(inDataTemp, 0, inData,
            0, inDataTemp.Length)         ' 入力データの設定
        inLen += inDataTemp.Length
        ' RPC実行
        Rpc.Call("SVGRP1", "SERV1", inData, inLen,
            outData, outLen, TP1ServerFlags.DCNOFLAGS)
        outStr = enc.GetString(outData, 0, outLen)

        Rpc.CloseRpc()       ' RPCクローズ
      Catch exp As TP1ServerException
      ' OpenTP1(クラスライブラリ)が検知したエラー
      Catch exp As TP1Exception
      ' その他OpenTP1が検知したエラー
      Catch exp As Exception
      ' 予期しない例外
      End Try
    End Sub
  End Class
End Namespace
```

4.7　DABroker for .NET Framework を使用した UAP の作成方法

DABroker for .NET Framework のデータプロバイダを使用して，XA インタフェースによる DBMS とのトランザクション連携をする SPP.NET，および SUP.NET を作成する方法について説明します。

なお，.NET インタフェース定義およびサービス定義の使用有無に関係なく，すべての SPP.NET および SUP.NET は DABroker for .NET Framework と連携できます。

4.7.1　DABroker for .NET Framework を使用した UAP 作成の概要

XA インタフェースによる DBMS とのトランザクション連携をする SPP.NET および SUP.NET では，次の操作で DABroker for .NET Framework のクラスを使用します。

- DBMS とのコネクションの作成，オープン，および解放
- データベースアクセス処理

DABroker for .NET Framework のクラス，接続文字列などの詳細については，マニュアル「DABroker for .NET Framework」を参照してください。

また，UAP を作成する際は，トランザクションシーケンスを意識する必要があります。詳細については，「2.4.5　トランザクションシーケンス」を参照してください。

4.7.2　DABroker for .NET Framework を使用した SPP.NET および SUP.NET のコーディング例

DABroker for .NET Framework と連携した SPP.NET および SUP.NET のコーディング例を示します。

(1)　SPP.NET，C#の場合のコーディング例（InitializeSPP メソッドで DBMS とのコネクションを生成する）

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;
using Hitachi.DABroker;   // DABroker for .NET Frameworkのクラス

namespace MyCompany
{
  public class CallerSampleImpl : SPPBaseICallerSample
  {
    private IGyoumuAStub server = null;
    private DABConnection dabCon = null;          // DBMSとの接続
    …
    public override void InitializeSPP()
    {
```

```
      server = new IGyoumuAStub("GRP1");
      // クライアントスタブの生成
      dabCon = new DABConnection("接続文字列");
      // コネクションの生成
      try {
        dabCon.Open();                // コネクションのオープン
      } catch (DABException exp) {
        // DABrokerから例外がスローされた
      } catch (Exception exp) {
        // 予期しない例外
      }
    }

    public override void FinalizeSPP()
    {
      server = null;
      if (dabCon != null) {           // コネクションの解放
        dabCon.Close();
        dabCon = null;
      }
    }

    [TP1RpcMethod]
    public void DBService()
    {
      try {
        Trn.Begin();
        …                             // データベースアクセス処理
        Trn.Commit();
      } catch (DABException exp) {
        // DABrokerから例外がスローされた
        Trn.Rollback();               // ロールバック
      } catch (Exception exp) {
        // 予期しない例外
      }
    }
  }
}
```

(2) SPP.NET，C#の場合のコーディング例（InitializeSPP メソッドで DBMS とのコネクションを生成しない）

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;
using Hitachi.DABroker;   // DABroker for .NET Frameworkのクラス

namespace MyCompany
{
  public class CallerSampleImpl : SPPBaseICallerSample
  {
    private IGyoumuAStub server = null;
    …
    public override void InitializeSPP()
    {
```

```
      server = new IGyoumuAStub("GRP1");
      // クライアントスタブの生成
    }

    public override void FinalizeSPP()
    {
      server = null;
    }

    [TP1RpcMethod]
    public void DBService()
    {
      private DABConnection dabCon = null;
      // DBMSとの接続
      try {
        Trn.Begin();
        dabCon = new DABConnection("接続文字列");
        // コネクションの生成
        dabCon.Open();              // コネクションのオープン
        …                           // データベースアクセス処理
        Trn.Commit();
      } catch (DABException exp) {
        // DABrokerから例外がスローされた
        Trn.Rollback();             // ロールバック
      } catch (Exception exp) {
        // 予期しない例外
      } finally {
        if (dabCon != null) {       // コネクションの解放
          dabCon.Close();
          dabCon = null;
        }
      }
    }
  }
}
```

(3) SUP.NET，C#の場合のコーディング例

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Server;
using Hitachi.DABroker;   // DABroker for .NET Frameworkのクラス

namespace MyCompany
{
  public class CallerSample
  {
    …
    public static void Main(string[] args)
    {
      private DABConnection dabCon = null;
      // DBMSとの接続
      try {
        …
        Rpc.Open();                    // RPCオープン
        Adm.Complete();                // SUP.NET開始処理完了通知
```

```
        …
        Trn.Begin();
        dabCon = new DABConnection(”接続文字列”);
        // コネクションの生成
        dabCon.Open();                  // コネクションのオープン
        // データベースアクセス処理
        Trn.Commit();
        …
      Rpc.Close();                      // RPCクローズ
    } catch (DABException exp) {
      // DABrokerから例外がスローされた
      Trn.Rollback();                   // ロールバック
    } catch (TP1UserException exp) {
      // サービスを要求したメソッドからユーザ例外がスローされた
    } catch (TP1RemoteException exp) {
      // サービスを要求したメソッドで予期しない例外が発生
    } catch (TP1ServerException exp) {
      // OpenTP1（クラスライブラリ）が検知したエラー
    } catch (TP1Exception exp) {
      // OpenTP1（クラスライブラリ以外）が検知したエラー
    } catch (Exception exp) {
      // 予期しない例外
    } finally {
      if (dabCon != null) {             // コネクションの解放
        dabCon.Close();
        dabCon = null;
      }
    }
  }
}
```

4.7.3 DABroker for .NET Framework を使用した UAP を作成するときの注意事項

DABroker for .NET Framework を使用した UAP を作成するときの注意事項を示します。

- DABroker for .NET Framework がサポートするプログラム言語については，マニュアル「DABroker for .NET Framework」を参照してください。
- 連携できる DBMS とそのバージョンについては，TP1/Server の「Readme ファイル」，およびマニュアル「DABroker for .NET Framework」を参照してください。また，XA インタフェースによってグローバルトランザクションと連携できる DBMS とそのバージョンについては，TP1/Server の「Readme ファイル」を参照してください。
- 接続文字列に「XA="false"」を指定したコネクションは，グローバルトランザクションと連携できません。
- 連携する DBMS に HiRDB を使用する場合は，接続文字列の DataSource 属性に指定する環境変数グループ名は指定しないでください。

- 連携する DBMS に HiRDB を使用する場合で，接続文字列の XA 属性に true を指定したときは，DataSource 属性に HiRDB 環境変数グループ名を指定しないでください。XA 属性に false を指定したときは，DataSource 属性に HiRDB 環境変数グループ名を指定してください。
- 連携する DBMS に ORACLE を使用する場合は，TP1/Server のユーザサーバ用 xa_open 関数用文字列にはデータベースフィールドを指定してください。このデータベースフィールドで指定したデータベース名称と，接続文字列の DataSource 属性に指定するデータベース名称を一致させる必要があります。

4.8 COBOL 言語での UAP の作成方法

COBOL 言語で UAP を作成する方法について説明します。なお，必要に応じて次のマニュアルも参照してください。

- 「COBOL2002 for .NET Framework　ユーザーズガイド」
- 「COBOL2002 for .NET Framework　言語」

4.8.1 COBOL 言語での SPP.NET の実装

COBOL 言語で SPP.NET を実装する方法として次の二つの方法があります。

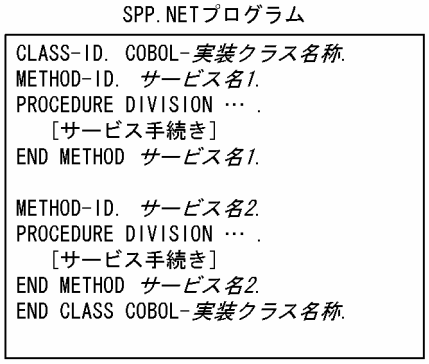
- オブジェクト指向で SPP.NET を作成する（オブジェクト指向型 SPP.NET）
- 手続き型のプログラム定義で SPP.NET を作成する（手続き型 SPP.NET）

ここでは，オブジェクト指向型 SPP.NET および手続き型 SPP.NET を作成する方法について説明します。

(1) オブジェクト指向型 SPP.NET

オブジェクト指向型 SPP.NET は，SPP.NET の標準のアプリケーション開発形態である「オブジェクト指向」で作成します。サービスをクラスのメソッドとして記述し，複数のサービスを統括するクラス定義を記述します。

図 4-1　オブジェクト指向型 SPP.NET のクラス定義



オブジェクト指向で SPP.NET を作成する場合，サービスをクラスのメソッドとして記述します。メソッド名はサービス名と同じにします。

(a) オブジェクト指向型 SPP.NET を実装する場合の規則

オブジェクト指向型 SPP.NET を実装する場合は，次の規則に従ってクラスを実装します。

- PROPAGATE 翻訳指令を使って例外自動伝播（propagate）を OFF にしてください。PROPAGATE 翻訳指令の詳細については，マニュアル「COBOL2002 for .NET Framework　言語」を参照してください。
- Hitachi.OpenTP1.Server.SPPBase クラスを継承します。
- リポジトリ段落で，継承する Hitachi.OpenTP1.Server.SPPBase クラスとサービスメソッドの仮引数で使用する System.Byte 配列を参照するように指定します。
- PUBLIC クラスとして宣言します。
- SPP.NET のサービスメソッドを PUBLIC メソッドとして実装します。なお，メソッド名は SPP.NET のサービス名になります。
- SPP.NET の初期化処理は InitializeSPP メソッド，終了時処理は FinalizeSPP メソッドで実装します。それぞれの処理が不要な場合は InitializeSPP メソッド，または FinalizeSPP メソッドを実装する必要はありません。
- SPP.NET のサービスメソッドとするために，TP1RPCMETHOD 指令を付加する必要があります。TP1RPCMETHOD 指令については，マニュアル「COBOL2002 for .NET Framework　言語」を参照してください。
- サービスメソッドの仮引数は，常に次の四つの引数を指定します。

仮引数	渡し方	型	説明
第 1 引数	値渡し	バイト配列オブジェクト	入力パラメタのバイト配列
第 2 引数	値渡し	PIC S9(9) COMP-5	入力パラメタ長
第 3 引数	値渡し	バイト配列オブジェクト	出力パラメタのバイト配列
第 4 引数	参照渡し	PIC S9(9) COMP-5	出力パラメタ長

引数の定義例を次に示します。

```
REPOSITORY.
 CLASS BYTE-ARRAY AS 'System.Byte' IS ARRAY.
...
LINKAGE SECTION.
 01 INDATA              USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 INDATA-LEN          PIC S9(9) COMP-5.
 01 OUTDATA             USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 OUTDATA-LEN         PIC S9(9) COMP-5.
PROCEDURE DIVISION USING BY VALUE      INDATA INDATA-LEN OUTDATA
                         BY REFERENCE  OUTDATA-LEN.
```

(b)　オブジェクト指向型 SPP.NET のコーディング例

オブジェクト指向型 SPP.NET のコーディング例を次に示します。

```
 >> PROPAGATE OFF
IDENTIFICATION DIVISION.
CLASS-ID. MYSPP AS "MYAPPLICATION.MYSPP" IS PUBLIC
          INHERITS SPPBASE.
```

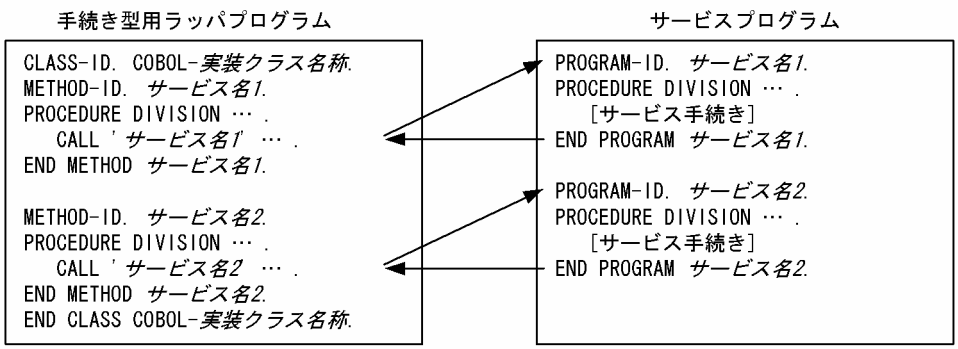
```
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
   CLASS SPPBASE    AS 'Hitachi.OpenTP1.Server.SPPBase'.
   CLASS BYTE-ARRAY AS 'System.Byte' IS ARRAY.

 IDENTIFICATION DIVISION.
 OBJECT.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 PROCEDURE DIVISION.

* SPP初期化および終了処理メソッドの実装
 IDENTIFICATION DIVISION.
 METHOD-ID. InitializeSPP IS PUBLIC OVERRIDE.
 PROCEDURE DIVISION.
* SPPの初期化処理
 END METHOD InitializeSPP.

 IDENTIFICATION DIVISION.
 METHOD-ID. FinalizeSPP IS PUBLIC OVERRIDE.
 PROCEDURE DIVISION.
* SPPの終了処理
 END METHOD FinalizeSPP.

* サービスメソッド定義（引数の並びは固定です）
* クライアントからRPCで呼び出されます
 >> TP1RPCMETHOD
 IDENTIFICATION DIVISION.
 METHOD-ID. SERVICE1 IS PUBLIC.
 DATA DIVISION.
 LINKAGE SECTION.
 01 INDATA       USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 INDATA-LEN   PIC S9(9) COMP-5.
 01 OUTDATA      USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 OUTDATA-LEN  PIC S9(9) COMP-5.
 PROCEDURE DIVISION USING BY VALUE     INDATA INDATA-LEN OUTDATA
                          BY REFERENCE OUTDATA-LEN.
* SERVICE1の処理
 END METHOD SERVICE1.
 >> TP1RPCMETHOD
 IDENTIFICATION DIVISION.
 METHOD-ID. SERVICE2 IS PUBLIC.
 DATA DIVISION.
 LINKAGE SECTION.
 01 INDATA       USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 INDATA-LEN   PIC S9(9) COMP-5.
 01 OUTDATA      USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 OUTDATA-LEN  PIC S9(9) COMP-5.
 PROCEDURE DIVISION USING BY VALUE     INDATA INDATA-LEN OUTDATA
                          BY REFERENCE OUTDATA-LEN.
* SERVICE2の処理
 END METHOD SERVICE2.

* SPP内部メソッドの実装（自由に実装できます）
* RPCで呼び出すことはできません
 IDENTIFICATION DIVISION.
```

```
 METHOD-ID. GETUSERINFO IS PUBLIC.
 DATA DIVISION.
 LINKAGE SECTION.
 PROCEDURE DIVISION.
* 処理
 END METHOD GETUSERINFO.

 IDENTIFICATION DIVISION.
 METHOD-ID. PUTERRORLOG IS PRIVATE.
 DATA DIVISION.
 LINKAGE SECTION.
 PROCEDURE DIVISION.
* 処理
 END METHOD PUTERRORLOG.

 END OBJECT.
 END CLASS MYSPP.
```

(2) 手続き型 SPP.NET

手続き型 SPP.NET は，サービスプログラムを手続き型のプログラム定義を使って記述します。サーバプログラムは，オブジェクトサーバの形態を取る必要があるため，COBOL 言語の手続き型プログラムであるサービスプログラムを呼び出すための，COBOL 言語のクラス定義が必要になります。このクラス定義で構成される COBOL 言語のプログラムを，手続き型用ラッパプログラムと呼びます。この手続き型用ラッパプログラムのクラス定義では，サービスプログラムと同じ名前のメソッドを定義します。そして，そのメソッドの中身は，メソッドの仮引数のバイト配列を英数字項目に変換する処理と，サービスプログラムを CALL 文で呼び出すだけにします。既存の SPP を SPP.NET にする場合は，この方法が適しています。

図 4-2 手続き型 SPP.NET の定義と制御



(a) 手続き型 SPP.NET を実装する場合の規則

手続き型 SPP.NET を実装する場合は，次の規則に従ってクラスを実装します。

- Hitachi.OpenTP1.Server.SPPBase クラスを継承します。
- リポジトリ段落で，継承する Hitachi.OpenTP1.Server.SPPBase クラスとサービスメソッドの仮引数で使用する System.Byte 配列を参照するように指定します。
- PUBLIC クラスとして宣言します。

- SPP.NET のサービスメソッドを PUBLIC メソッドとして実装します。なお，メソッド名は SPP.NET のサービス名になります。
- SPP.NET の初期化処理は InitializeSPP メソッド，終了時処理は FinalizeSPP メソッドで実装します。それぞれの処理が不要な場合は InitializeSPP メソッド，または FinalizeSPP メソッドを実装する必要はありません。
- SPP.NET のサービスメソッドとするために，TP1RPCMETHOD 指令を付加する必要があります。TP1RPCMETHOD 指令については，マニュアル「COBOL2002 for .NET Framework　言語」を参照してください。
- サービスメソッドの仮引数は，常に次の四つの引数を指定します。

仮引数	渡し方	型	説明
第 1 引数	値渡し	バイト配列オブジェクト	入力パラメタのバイト配列
第 2 引数	値渡し	PIC S9(9) COMP-5	入力パラメタ長
第 3 引数	値渡し	バイト配列オブジェクト	出力パラメタのバイト配列
第 4 引数	参照渡し	PIC S9(9) COMP-5	出力パラメタ長

引数の定義例を次に示します。

```
REPOSITORY.
 CLASS BYTE-ARRAY AS 'System.Byte' IS ARRAY.
...
LINKAGE SECTION.
 01 INDATA              USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 INDATA-LEN          PIC S9(9) COMP-5.
 01 OUTDATA             USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 OUTDATA-LEN         PIC S9(9) COMP-5.
PROCEDURE DIVISION USING BY VALUE      INDATA INDATA-LEN OUTDATA
                         BY REFERENCE  OUTDATA-LEN.
```

- サービスをプログラム定義で記述します。プログラム名はサービス名と同じにします。また，引数は次のようにします。

仮引数	渡し方	型	説明
第 1 引数	参照渡し	PIC X(n)または集団項目	入力パラメタのバイト列
第 2 引数	参照渡し	PIC S9(9) COMP-5	入力パラメタ長
第 3 引数	参照渡し	PIC X(m)または集団項目	出力パラメタのバイト列
第 4 引数	参照渡し	PIC S9(9) COMP-5	出力パラメタ長

引数の定義例を次に示します。

```
LINKAGE SECTION.
 01 INDATA              PIC X(n).
 01 INDATA-LEN          PIC S9(9) COMP-5.
 01 OUTDATA             PIC X(m).
 01 OUTDATA-LEN         PIC S9(9) COMP-5.
PROCEDURE DIVISION USING INDATA INDATA-LEN OUTDATA OUTDATA-LEN.
```

(b) 手続き型 SPP.NET のコーディング例

■ 手続き型用ラッパプログラム

手続き型用ラッパプログラムは，SPP.NET の実装クラスを定義し，各サービスプログラムを呼び出すためのプログラムです。手続き型用ラッパプログラムのコーディング例を次に示します。

```
 IDENTIFICATION DIVISION.
 CLASS-ID. MYSPP AS "MYAPPLICATION.MYSPP" IS PUBLIC
           INHERITS SPPBASE.

 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
   CLASS SPPBASE    AS 'Hitachi.OpenTP1.Server.SPPBase'.
   CLASS BYTE-ARRAY AS 'System.Byte' IS ARRAY.

 IDENTIFICATION DIVISION.
 OBJECT.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 PROCEDURE DIVISION.

* SPP初期化および終了処理メソッドの実装
 IDENTIFICATION DIVISION.
 METHOD-ID. InitializeSPP IS PUBLIC OVERRIDE.
 PROCEDURE DIVISION.
* SPPの初期化処理
 END METHOD InitializeSPP.

 IDENTIFICATION DIVISION.
 METHOD-ID. FinalizeSPP IS PUBLIC OVERRIDE.
 PROCEDURE DIVISION.
* SPPの終了処理
 END METHOD FinalizeSPP.

* サービスメソッド定義（引数の並びは固定です）
* クライアントからRPCで呼び出されます
 >> TP1RPCMETHOD
 IDENTIFICATION DIVISION.
 METHOD-ID. SERVICE1 IS PUBLIC.
 DATA DIVISION.
 LOCAL-STORAGE SECTION.
 01 INDATA-X     PIC X(n).
 01 OUTDATA-X    PIC X(m).
 LINKAGE SECTION.
 01 INDATA       USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 INDATA-LEN   PIC S9(9) COMP-5.
 01 OUTDATA      USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 OUTDATA-LEN  PIC S9(9) COMP-5.
 PROCEDURE DIVISION USING BY VALUE     INDATA INDATA-LEN OUTDATA
                          BY REFERENCE OUTDATA-LEN.
* バイト配列から英数字項目への変換
     CALL 'CBLBYTEARRAYTOX'  USING INDATA INDATA-LEN INDATA-X.
* サービスプログラムの呼び出し
     CALL 'SERVICE1' USING  INDATA-X INDATA-LEN
```

```
                              OUTDATA-X OUTDATA-LEN.
* 英数字項目からバイト配列への変換
     CALL 'CBLXTOBYTEARRAY'  USING OUTDATA-X OUTDATA OUTDATA-LEN.
 END METHOD SERVICE1.

 >> TP1RPCMETHOD
 IDENTIFICATION DIVISION.
 METHOD-ID. SERVICE2 IS PUBLIC.
 DATA DIVISION.
 LOCAL-STORAGE SECTION.
 01 INDATA-X     PIC X(n).
 01 OUTDATA-X    PIC X(m).
 LINKAGE SECTION.
 01 INDATA       USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 INDATA-LEN   PIC S9(9) COMP-5.
 01 OUTDATA      USAGE OBJECT REFERENCE BYTE-ARRAY.
 01 OUTDATA-LEN  PIC S9(9) COMP-5.
 PROCEDURE DIVISION USING BY VALUE     INDATA INDATA-LEN OUTDATA
                          BY REFERENCE OUTDATA-LEN.
* バイト配列から英数字項目への変換
     CALL 'CBLBYTEARRAYTOX'  USING INDATA INDATA-LEN INDATA-X.
* サービスプログラムの呼び出し
     CALL 'SERVICE2' USING  INDATA-X INDATA-LEN
                              OUTDATA-X OUTDATA-LEN.
* 英数字項目からバイト配列への変換
     CALL 'CBLXTOBYTEARRAY'  USING OUTDATA-X OUTDATA OUTDATA-LEN.
 END METHOD SERVICE2.

 END OBJECT.
 END CLASS MYSPP.
```

■ サービスプログラム

サービスプログラムは手続き型のプログラムとして作成します。既存の SPP から SPP.NET に移行する場合，SPP のサービスプログラムはそのまま SPP.NET のサービスプログラムとして使用できます。サービスプログラムの詳細については，マニュアル「OpenTP1　プログラム作成リファレンス　COBOL 言語編」を参照してください。

引数の定義例を次に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SERVICE1.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 LINKAGE SECTION.
 77  INDATA       PIC X(n).
 77  INDATA-LEN   PIC S9(9) COMP-5.
 77  OUTDATA      PIC X(m).
 77  OUTDATA-LEN  PIC S9(9) COMP-5.
 PROCEDURE DIVISION USING INDATA INDATA-LEN OUTDATA OUTDATA-LEN.
* SERVICE1の処理
 END PROGRAM SERVICE1.
```

(3)　SPP.NET のコンパイル

SPP.NET のコンパイルには，コマンドライン上でコンパイルする方法と Visual Studio 上でコンパイルする方法があります。

(a)　コマンドラインでのコンパイル方法

次のようにコンパイルします。

```
netcbl -Target.Library [コンパイラオプション]･･･
    [<手続き型SPP向けラッパプログラム>.cbl]                    ･･･　※1
     <サービスプログラム>.cbl

     -Reference Hitachi.OpenTP1.Server.dll
     -NetLib "%DCDIR%¥bin"

     -PInvoke  P/Invoke指示ファイル名                          ･･･　※2

     -OutputFile <SPP.NETプログラム>.dll
```

注※1

手続き型 SPP.NET プログラムで作成する場合に指定します。

注※2

COBOL 言語のプログラムの中で従来の SPP で使用していた TP1/Server の COBOL 言語のサービスルーチンを利用する場合に指示ファイルを指定します。詳細については，「4.8.3　TP1/Server の COBOL 言語のサービスルーチンの利用」を参照してください。

(b)　Visual Studio 上でコンパイルする場合

Visual Studio 上でコンパイルする場合は，通常の COBOL2002 for .NET Framework のアプリケーションの作成と同様の方法でコンパイルしてください。

なお，次の点に注意してください。

- 必ず［プロジェクト］－［参照の追加］で，コンポーネント名「TP1/Extension for .NET Framework Class Library」を選択してください。
- TP1/Server の COBOL 言語のサービスルーチンを利用する場合には，コンパイラオプションの設定で，P/Invoke 指示ファイルを引数とした-PInvoke オプションを設定してください。

4.8.2　COBOL 言語での SUP.NET の実装

COBOL 言語で SUP.NET を作成する方法について説明します。

(1)　SUP.NET

SUP.NET は，TP1/Server の COBOL 言語のサービスルーチンをプラットフォーム呼び出し（P/Invoke）する方法を使って作成します。従来の SUP を作成する方法と同じです。作成方法やサービスルーチンの詳細については，次のマニュアルを参照してください。

- 「OpenTP1　プログラム作成の手引」
- 「OpenTP1　プログラム作成リファレンス　COBOL 言語編」

注意事項

従来の SUP では，同じサービスルーチンに対する複数の呼び出しで，引数の数が異なるものがあっても問題ありませんでした。しかし，SPP.NET では同じサービスルーチンに対する複数の呼び出しで引数の数が異なるものがあると System.InvalidProgramException 例外が発生します。System.InvalidProgramException 例外については，マニュアル「COBOL2002 for .NET Framework　ユーザーズガイド」を参照してください。

(2)　SUP.NET のコンパイル

SUP.NET のコンパイルには，コマンドライン上でコンパイルする方法と Visual Studio 上でコンパイルする方法があります。

(a)　コマンドラインでのコンパイル方法

次のようにコンパイルします。

```
netcbl -Target,Exe [コンパイラオプション]･･･
       <サーバ利用プログラム>.cbl･･･
       -PInvoke P/Invoke指示ファイル名
       -OutputFile <SUP.NETプログラム>.exe
```

(b)　Visual Studio でのコンパイル方法

Visual Studio 上でコンパイルする場合は，通常の COBOL2002 for .NET Framework のアプリケーションの作成と同様の方法でコンパイルしてください。

TP1/Server の COBOL 言語のサービスルーチンを利用する場合には，コンパイラオプションの設定で，P/Invoke 指示ファイルを引数とした-PInvoke オプションを設定してください。

4.8.3　TP1/Server の COBOL 言語のサービスルーチンの利用

従来の SPP で使用していた TP1/Server の COBOL 言語のサービスルーチンを利用する場合は，プラットフォーム呼び出し（P/Invoke）を使います。ソースプログラム上の記述は，通常の呼び出しもプラット

フォーム呼び出しも同じです。しかし，プラットフォーム呼び出しの場合は，P/Invoke 指示ファイルを-PInvoke オプションに指定して，呼び出すプログラムがアンマネージプログラムであることをコンパイラに指示する必要があります。

P/Invoke 指示ファイルのサンプルの格納場所については，「2.1.3(1)　Extension .NET が提供する P/Invoke 指示ファイルのサンプル」を参照してください。

(1)　P/Invoke 指示ファイルの作成例

TP1/Server Base および TP1/LiNK の場合の P/Invoke 指示ファイルの作成例を次に示します。

TP1/Server Base の場合

```
CBLDCADM(DLL="LIBBETRAN.DLL",CallConv=Cdecl)
CBLDCLOG(DLL="LIBBETRAN.DLL",CallConv=Cdecl)
CBLDCPRF(DLL="LIBBETRAN.DLL",CallConv=Cdecl)
CBLDCRAP(DLL="LIBBETRAN.DLL",CallConv=Cdecl)
CBLDCRPC(DLL="NJSCMLIB.DLL",CallConv=Cdecl)
CBLDCTRN(DLL="LIBBETRAN.DLL",CallConv=Cdecl)
CBLDCMCF(DLL="LIBMCF.DLL",CallConv=Cdecl)
CBLDCTAM(DLL="LIBTAM.DLL",CallConv=Cdecl)
```

TP1/LiNK の場合

```
CBLDCADM(DLL="BETRAN.DLL",CallConv=Cdecl)
CBLDCLOG(DLL="BETRAN.DLL",CallConv=Cdecl)
CBLDCPRF(DLL="BETRAN.DLL",CallConv=Cdecl)
CBLDCRAP(DLL="BETRAN.DLL",CallConv=Cdecl)
CBLDCRPC(DLL="NJSCMLIB.DLL",CallConv=Cdecl)
CBLDCTRN(DLL="BETRAN.DLL",CallConv=Cdecl)
CBLDCMCF(DLL="LIBMCF.DLL",CallConv=Cdecl)
```

(2)　P/Invoke 指示ファイル作成時の注意事項

- P/Invoke 指示ファイルの拡張子は piv にしてください。
- CBLDCRPC サービスルーチンの場合は，NJSCMLIB.DLL を必ず指定してください。その他の TP1/Server の COBOL 言語のサービスルーチンについては，適切なライブラリファイル名を指定してください。
- 従来の Windows 版の SPP では，同じサービスルーチンに対する複数の呼び出しで，引数が異なるものがあっても問題ありませんでした。しかし，SPP.NET では同じサービスルーチンに対する複数の呼び出しで引数の数が異なるものがあると System.InvalidProgramException 例外が発生します。System.InvalidProgramException 例外については，マニュアル「COBOL2002 for .NET Framework　ユーザーズガイド」を参照してください。

4.8.4 プログラム作成時の注意事項

- オブジェクト指向型 SPP.NET の場合，サービスとして使用するメソッドは，PROPAGATE 翻訳指令を使って例外自動伝播（propagate）を OFF にしてください。手続き型 SPP.NET の場合は，手続き型用ラッパプログラムの方で PROPAGATE 翻訳指令を使って例外自動伝播（propagate）を OFF にします。
- ThreadAbortException 例外が発生した場合，KFCA00105-E メッセージとともに，アボートコード E00002 が出力されます。終了コードは 6 でプロセスを終了します。ThreadAbortException 例外の発生を伴う次の機能使用時も，終了コード 6 でプロセスが終了するので，できるだけ使用しないでください。
  - CBLABN サービスルーチン
  - STOP RUN 文

  SPP.NET のプログラム構成では，従来の SPP と異なり，メインプログラムを記述しないため，STOP RUN を指定する必要はありません。RETURN-CODE 特殊レジスタの値や，CBLABN サービスルーチンの引数値は終了コードにはなりません。
- FILE STATUS 句や例外処理を使って，できるだけ COBOL 実行時エラーとならないようにコーディングしてください。COBOL 実行時エラーが発生すると例外 ThreadAbortException を引き起こすので，プロセス終了となります。
- SPP.NET または SUP.NET でデータベース操作機能を使用する場合，「COBOL2002 for .NET Framework のインストールディレクトリ¥Bin」のパスを TP1/Server のサーチパスに指定する必要があります。
- AS 指定のないクラス名の場合，名前空間は「COBOL2002.DefaultNamespace」が仮定されます。AS 指定があるクラス名の場合，名前空間は付加されません。

その他，COBOL 言語の注意事項については，マニュアル「COBOL2002 for .NET Framework　ユーザーズガイド」を参照してください。

4.9　アプリケーションの配置と運用

SPP.NET および SUP.NET の配置と運用について説明します。

4.9.1　アプリケーションの配置

アプリケーションの配置ディレクトリと，アプリケーション構成ファイルについて説明します。

なお，以降の説明の「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

(1)　配置ディレクトリ

SPP.NET のアセンブリ（DLL 形式），SUP.NET のアセンブリ（EXE 形式）は，「%DCDIR%¥aplib ディレクトリ」または任意のディレクトリに配置します。

これらのアセンブリが参照しているアセンブリ（グローバルアセンブリキャッシュに登録されているものを除く）も必ず SPP.NET，および SUP.NET のアセンブリと同じディレクトリに配置してください。

また，SPP.NET，および SUP.NET のアセンブリはグローバルアセンブリキャッシュに登録しないでください。

(a)　SPP.NET を任意のディレクトリに配置した場合

【TP1/Server Base】

njs_appbase_directory オペランドでアプリケーションベースディレクトリの設定が必要です。ユーザサービスデフォルト定義，およびユーザサービス定義の njs_appbase_directory オペランドでアプリケーションベースディレクトリの設定をしない場合は，「%DCDIR%¥aplib」がアプリケーションベースディレクトリとなります。ユーザサービスデフォルト定義，およびユーザサービス定義については，「3.　システム定義【TP1/Server Base】」を参照してください。

【TP1/LiNK】

TP1/LiNK の実行環境設定の GUI でアプリケーションベースディレクトリの設定が必要です。アプリケーションベースディレクトリの設定をしない場合は「%DCDIR%¥aplib」がアプリケーションベースディレクトリとなります。TP1/LiNK の実行環境設定の GUI については，「5.　TP1/LiNK でのユーザサーバの実行環境設定【TP1/LiNK】」を参照してください。

(b)　SUP.NET を任意のディレクトリに配置した場合

【TP1/Server Base】

プロセスサービス定義の prcsvpath コマンドの設定が必要です。プロセスサービス定義については，マニュアル「OpenTP1　システム定義」を参照してください。

【TP1/LiNK】

TP1/LiNK の実行環境設定の GUI でプログラムのサーチパスの設定が必要です。TP1/LiNK の実行環境設定の GUI については，「5.　TP1/LiNK でのユーザサーバの実行環境設定【TP1/LiNK】」およびマニュアル「TP1/LiNK　使用の手引」を参照してください。

(2)　アプリケーションが参照するアプリケーション構成ファイル

SPP.NET，SUP.NET の各アプリケーションからアプリケーション構成ファイルにアクセスする場合は，次のディレクトリに次に示す名称でアプリケーション構成ファイルを作成します。

(a)　SPP.NET の場合

SPP.NET の場合，ディレクトリ名，ファイル名は次の表に従って指定します。

項目	指定値
ディレクトリ	アプリケーションベースディレクトリ （%DCDIR%¥aplib，または任意のディレクトリ）
ファイル名	〈ユーザサーバ名〉.config

【指定例】

ユーザサーバ名が SERVER1，アプリケーションベースディレクトリが「%DCDIR%¥aplib」の場合，次のように指定します。

%DCDIR%¥aplib¥SERVER1.config

(b)　SUP.NET の場合

SUP.NET の場合，ディレクトリ名，ファイル名を次の表に従って指定します。

項目	指定値
ディレクトリ	SUP.NET 配置ディレクトリ （%DCDIR%¥aplib，または任意のディレクトリ）
ファイル名	〈SUP.NET アセンブリファイル名〉.config

【指定例】

SUP.NET アセンブリファイル名が「mysup2.exe」，SUP.NET 配置ディレクトリが「D:¥TP1app」の場合，次のように指定します。

D:¥TP1app¥mysup2.exe.config

参考

アプリケーション構成ファイルについては，.NET Framework のドキュメントを参照してください。

4.9.2　ユーザサーバの定義

SPP.NET，SUP.NET は SPP，SUP と同様にユーザサーバとして定義します。定義方法の詳細については，次の項目およびマニュアルを参照してください。

【TP1/Server Base】

- 「3.　システム定義【TP1/Server Base】」
- 「OpenTP1　システム定義」

【TP1/LiNK】

- 「5.　TP1/LiNK でのユーザサーバの実行環境設定【TP1/LiNK】」
- 「TP1/LiNK　使用の手引」

4.9.3　ユーザサーバの運用

SPP.NET，SUP.NET は SPP，SUP と同様にユーザサーバとして運用します。起動／停止，閉塞／閉塞解除，実行状態の確認などができます。

運用方法の詳細については，次のマニュアルを参照してください。

【TP1/Server Base】

「OpenTP1　運用と操作」

【TP1/LiNK】

「TP1/LiNK　使用の手引」

4.10　UAP 作成上の注意事項

4.10.1　実行環境と留意点

(1)　SPP.NET の実行環境と留意点

SPP.NET は DLL 形式のアセンブリとして実装し，Extension .NET が提供する SPP.NET 実行コンテナによって実行されます。

このコンテナは SPP と同様にシングルスレッドで動作するので，複数のサービス要求によって，一つのプロセス上で SPP.NET のサービスメソッドが同時に実行されることはありません。そのため，マルチスレッドを意識したプログラミングをする必要はありません。

(2)　SUP.NET の実行環境と留意点

SUP.NET は EXE 形式のアセンブリとして実装し，そのまま独立したプロセスとして実行されます。この場合，SUP.NET で実装された Main メソッドが起動されます。

4.10.2　例外の捕捉とエラーの判定

OpenTP1 for .NET Framework の UAP では，Extension .NET が提供するクラスライブラリやスタブ，.NET Framework などから例外が発生する場合があります。これらの例外は適切に捕捉してください。

Extension .NET からは，次に示す例外が発生する可能性があります。

表 4-8　発生する可能性がある例外

名前空間	例外名	意味
Hitachi.OpenTP1	TP1Exception	共通の例外基底クラス
	TP1RemoteException	サーバで発生した例外の受信
	TP1UserException	ユーザが任意に使用
Hitachi.OpenTP1.Server	TP1ServerException	クラスライブラリでのエラー検知

(1)　TP1Exception

OpenTP1 のすべての例外の基底クラスです。個別の例外を捕捉しないで，まとめて捕捉したい場合などに，この TP1Exception クラスですべての OpenTP1 の例外を捕捉できます。

(2) TP1RemoteException

クライアントスタブを使用してサービス要求をした場合に，サーバ側で例外が発生したことを示す例外です。呼び出し元のクライアント側で発生した例外とサーバ側で発生した例外を区別できます。

.NET インタフェース定義を使用しないサービスメソッドでユーザ例外が発生した場合，クライアントには DCRPCER_SYSERR_AT_SERVER が返されます。エラーの詳細は.NET エラーログファイルを参照してください。

(3) TP1UserException

クライアント，およびサーバを.NET インタフェース定義を使用した OpenTP1 for .NET Framework の UAP で作成した場合に，サービス要求されたサービスメソッドからクライアントにこの例外を返すことができます。

この例外は UAP で任意に使用できます。また，クライアント側でも TP1RemoteException には変換されません。

なお，.NET インタフェース定義を使用しない場合は，この例外を使用しないでください。使用した場合は，クライアント側にサービス要求のエラーが通知されます。

(4) TP1ServerException

クラスライブラリの各メソッドでエラーを検知した場合に発生します。各メソッドでの発生条件や内容については「7. クラスリファレンス」を参照してください。

.NET インタフェース定義を使用しない SPP.NET では，各サービスメソッドですべての例外を確実に捕捉してください。

サービスメソッドが例外を SPP.NET 実行コンテナに返した場合，クライアント側にサービス要求のエラーが通知されます。

4.10.3 その他の注意事項

(1) クラスライブラリ使用上の注意事項

- Hitachi.OpenTP1 で始まる名前空間は OpenTP1 が使用するため，OpenTP1 以外のアプリケーションでは使用できません。
- Extension .NET が提供するクラスライブラリはスレッドセーフではありません。マルチスレッドで使用した場合の動作は保証できません。

(2) SPP.NET に関する注意事項

- SPP.NET では，サービスメソッドや InitializeSPP メソッド，FinalizeSPP メソッドが呼び出されたスレッドでだけ OpenTP1 が提供するクラスを使用できます。

  それ以外のスレッド（SPP.NET 上で任意に生成したスレッドや.NET Framework が提供するクラスによって生成されたスレッドなど）で OpenTP1 の提供するクラスを使用しないでください。使用した場合の動作は保証できません。

  また，SPP.NET 上で使用するオブジェクトのデストラクタや Finalize メソッドでも，OpenTP1 が提供するクラスを使用しないでください。
- OpenTP1 が提供する各クラスは，SPP.NET 実行コンテナによって SPP.NET 実装クラスが呼び出されたアプリケーションドメインからだけ使用できます。SPP.NET 内で独自に生成したアプリケーションドメインから OpenTP1 のクラスを使用しないでください。使用した場合の動作は保証できません。
- .NET インタフェース定義と SPP.NET 実装クラスは同じプログラム言語で記述してください。サーバスタブは.NET インタフェース定義と同じプログラム言語を使用して生成されます。
- .NET インタフェース定義，SPP.NET 実装クラス，サーバスタブは一つのアセンブリに入れてください。これらが一つのアセンブリに入っていない場合，SPP.NET の起動ができません。SPP.NET 実装クラスが使用するクラス，または参照するクラス（ほかの SPP.NET を利用する場合は，そのクライアントスタブも含む）については別のアセンブリになっていてもかまいません。
- SPP.NET 実装クラスでは Main メソッドを実装する必要はありません。実装しても実行されません。
- SPP.NET 実装クラスを含むアセンブリは DLL 形式にしてください。
- SPP.NET 内で例外 ThreadAbortException が発生した場合，KFCA00105-E メッセージとともに，アボートコード E00001 が出力され，SPP.NET のプロセスは異常終了します。
- .NET インタフェース定義を使用しないサービスメソッドでユーザ例外が発生した場合，クライアントには DCRPCER_SYSERR_AT_SERVER が返されます。エラーの詳細は.NET エラーログファイルを参照してください。

(3) SUP.NET に関する注意事項

- SUP.NET では，Main メソッドが呼び出されたスレッドでだけ OpenTP1 が提供するクラスを使用することができます。それ以外のスレッド（SUP.NET 上で任意に生成したスレッドや.NET Framework が提供するクラスによって生成されたスレッドなど）で OpenTP1 が提供するクラスを使用しないでください。使用した場合の動作は保証できません。

  また，SUP.NET 上で使用するオブジェクトのデストラクタや Finalize メソッドでも，OpenTP1 が提供するクラスを使用しないでください。
- SUP.NET の実装クラスでは Main メソッドを実装してください。
- SUP.NET の実装を含むアセンブリは EXE 形式にしてください。

(4) Visual Studio 上でコンパイルする場合の注意事項

Visual Studio 上で SPP.NET および SUP.NET をコンパイルする場合，必ず Visual Studio のメニューから［プロジェクト］－［参照の追加］で，コンポーネント名「TP1/Extension for .NET Framework Class Library」を選択してください。

4.11　サンプルプログラムの使用方法

Extension .NET のサンプルプログラムのディレクトリ構成，ビルド方法および実行手順を説明します。なお，以降の説明で，環境変数に DCSMPDIR を設定しない場合は，DCSMPDIR を次のように読み換えてください。

【TP1/Server Base】
　%DCDIR%¥examples

【TP1/LiNK】
　%DCDIR%¥sample

「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

環境変数に DCSMPDIR を設定する方法については，「4.11.2(1) サンプルプログラムをビルドするための準備」の手順 3 を参照してください。

4.11.1　サンプルプログラムのディレクトリ構成

Extension .NET のサンプルプログラムは，次の表に示すディレクトリに格納されています。

表 4-9　サンプルプログラムのディレクトリ構成

ディレクトリ	説明
%DCSMPDIR%¥ExtNET	Extension .NET のサンプルプログラム格納ディレクトリ
%DCSMPDIR%¥ExtNET¥C#	C#のサンプルプログラム格納ディレクトリ
%DCSMPDIR%¥ExtNET¥J#	J#のサンプルプログラム格納ディレクトリ
%DCSMPDIR%¥ExtNET¥VB.NET	Visual Basic のサンプルプログラム格納ディレクトリ
%DCSMPDIR%¥ExtNET¥COBOL.NET	COBOL 言語のサンプルプログラム格納ディレクトリ，および COBOL 言語のサービスルーチン利用のための P/Invoke 指示ファイル
%DCSMPDIR%¥ExtNET¥CONF	サンプルプログラム実行用ユーザサービス定義格納ディレクトリ

各言語のサンプルプログラム格納ディレクトリ下は，次の表に示す名称のディレクトリで構成されます。それぞれのディレクトリには，次の表に示す種類のサンプルプログラムが格納されています。

表 4-10　サンプルプログラムの種類（C#，J#，および Visual Basic の場合）

名称	説明
SPPBIN	RPC の入出力にバイナリデータを使用した SPP.NET です。 このサンプルプログラムでは，.NET インタフェース定義を使用しません。
SPPIF	.NET インタフェース定義を使用した SPP.NET です。

名称	説明
SUPBIN	RPC 要求インタフェースとしてバイナリデータを使用した SUP.NET です。 このサンプルプログラムでは，クライアントスタブを使用しません。
SUPCR	RPC 要求インタフェースとしてサービス定義から生成されたクライアントスタブとカスタムレコードを使用した SUP.NET です。
SUPIF	RPC 要求インタフェースとして.NET インタフェース定義から生成されたクライアントスタブを使用した SUP.NET です。

表 4-11　サンプルプログラムの種類（COBOL 言語の場合）

名称	説明
SPPOBJ	オブジェクト指向型 SPP.NET です。
SPPPRO	手続き型 SPP.NET です。
SUPOBJ	オブジェクト指向型 SPP.NET を呼び出す SUP.NET です。
SUPPRO	手続き型 SPP.NET を呼び出す SUP.NET です。

4.11.2　サンプルプログラムのビルド方法

Extension .NET のサンプルプログラムをビルドする手順を説明します。

(1)　サンプルプログラムをビルドするための準備

1. 次のコマンドプロンプトを起動します。
   - C#，J#，および Visual Basic の場合
     Visual Studio が提供するコマンドプロンプト，または.NET Framework SDK が提供するコマンドプロンプトを起動します。
   - COBOL 言語の場合
     COBOL2002 for .NET Framework の提供するコマンドプロンプトを起動します。
2. Windows Server 2003（64 ビット用）の環境で J#のサンプルプログラムをビルドする場合は，Microsoft .NET Framework v2.0 の 32 ビット版のインストールディレクトリを環境変数 PATH に設定します。
3. 環境変数に DCSMPDIR を設定する場合は，コマンドプロンプトからコマンドを実行します。

   【例】TP1/Server Base の場合

   ```
   set DCSMPDIR=%DCDIR%¥examples
   ```

   【例】TP1/LiNK の場合

   ```
   set DCSMPDIR=%DCDIR%¥sample
   ```

4. 環境変数 Path，LIBPATH に.NET Framework のパスを設定してください。

なお，以下は%SystemRoot%として設定する方法を記載しています。

【例】.NET Framework のバージョンが v3.5 の場合

```
set Path=%SystemRoot%¥Microsoft.NET¥Framework¥v3.5;%Path%
set LIBPATH=%SystemRoot%¥Microsoft.NET¥Framework¥v3.5;%LIBPATH%
```

【例】.NET Framework のバージョンが v4 の場合

```
set Path=%SystemRoot%¥Microsoft.NET¥Framework¥v4.0.30319;%Path%
set LIBPATH=%SystemRoot%¥Microsoft.NET¥Framework¥v4.0.30319;%LIBPATH%
```

(2) サンプルプログラムをまとめてビルドする方法

- COBOL 言語以外のサンプルプログラムをまとめてビルドする場合
  %DCSMPDIR%¥ExtNET¥build_all.bat を実行します。
- COBOL 言語の場合を含むすべてのサンプルプログラムをまとめてビルドする場合
  %DCSMPDIR%¥ExtNET¥build_all2.bat を実行します。

(3) 言語別のサンプルプログラムをまとめてビルドする方法

- C#の場合
  %DCSMPDIR%¥ExtNET¥C#¥build_cs.bat を実行します。
- J#の場合
  %DCSMPDIR%¥ExtNET¥J#¥build_vjs.bat を実行します。
- Visual Basic の場合
  %DCSMPDIR%¥ExtNET¥VB.NET¥build_vb.bat を実行します。
- COBOL 言語の場合
  %DCSMPDIR%¥ExtNET¥COBOL.NET¥build_cbl.bat を実行します。

(4) サンプルプログラムを個別にビルドする方法

各サンプルプログラム格納ディレクトリ下の build.bat を実行します。

【例】%DCSMPDIR%¥ExtNET¥VB.NET¥SPPIF¥build.bat

4.11.3 サンプルプログラムの実行手順

Extension .NET のサンプルプログラムの実行手順を説明します。

(1) ビルドしたアセンブリを OpenTP1 環境にコピー

ビルドによって生成されたアセンブリを，%DCDIR%¥aplib ディレクトリにコピーします。

【例】Visual Basic の SPPIF サンプルの場合

```
copy %DCSMPDIR%¥ExtNET¥VB.NET¥SPPIF¥VBSample.SPPIF.dll %DCDIR%¥aplib
```

【例】Visual Basic の SUPIF サンプルの場合

```
copy %DCSMPDIR%¥ExtNET¥VB.NET¥SUPIF¥VBSample.SUPIF.exe %DCDIR%¥aplib
```

(2) ユーザサービス定義ファイルを OpenTP1 環境にコピー

実行したいサンプルプログラムのユーザサービス定義ファイルを，%DCCONFPATH%にコピーします。

【例】Visual Basic の SPPIF サンプルの場合

```
copy %DCSMPDIR%¥ExtNET¥CONF¥VBSPPIF %DCCONFPATH%
```

(3) OpenTP1 の起動

次の手順で OpenTP1 を起動します。

【TP1/Server Base】

コマンドライン上で，「net start OpenTP1」を実行します。

【TP1/LiNK】

1. ［スタート］－［プログラム］－［TP1_LiNK］－［TP1_LiNK コントロール］メニューを選択します。

   ［TP1_LiNK コントロール］ウィンドウが表示されます。

2. OpenTP1 の状態がオンラインでない場合は，［起動(S)］ボタンをクリックします。

(4) SPP.NET 環境定義の設定

SPP.NET の場合，次の手順で SPP.NET 環境定義を設定します。

【TP1/Server Base】

1. %DCCONFPATH%にコピーしたユーザサービス定義ファイルを notepad.exe から開きます。

   【例】Visual Basic の SPPIF サンプルの場合

   ```
   notepad.exe %DCCONFPATH%¥VBSPPIF
   ```

2. set njs_appbase_directory の値の「%DCDIR%」の部分を OpenTP1 インストールディレクトリの絶対パスに変更します。

   【例】OpenTP1 インストールディレクトリが C:¥OpenTP1 の場合

   ```
   C:¥OpenTP1¥aplib
   ```

【TP1/LiNK】

1. [スタート] − [プログラム] − [TP1_LiNK] − [アプリケーション管理 SPP] メニューを選択します。

   [TP1/LiNK アプリケーション管理 SPP] ウィンドウが表示されます。

2. ユーザサービス定義ファイルと同じ名前のユーザサーバを選択し，[サーバ定義(E)...] ボタンをクリックします。

   [SPP.NET 環境定義] ダイアログボックスが表示されます。

3. [アプリケーションベースディレクトリ(1)] の「%DCDIR%」の部分を OpenTP1 インストールディレクトリの絶対パスに変更します。

   【例】OpenTP1 インストールディレクトリが C:¥OpenTP1 の場合

   C:¥OpenTP1¥aplib

(5) SPP.NET または SUP.NET の実行

次の手順で SPP.NET または SUP.NET を実行します。

【TP1/Server Base】

コマンドライン上で，次のコマンドを実行します。

【例】Visual Basic の SPPIF サンプルの場合

dcsvstart -u VBSPPIF

【TP1/LiNK】

SPP.NET の場合

1. [スタート] − [プログラム] − [TP1_LiNK] − [アプリケーション管理 SPP] メニューを選択します。

   [TP1/LiNK アプリケーション管理 SPP] ウィンドウが表示されます。

2. ユーザサービス定義ファイルと同じ名前のユーザサーバを選択し，[起動(S)] ボタンをクリックします。

SUP.NET の場合

1. [スタート] − [プログラム] − [TP1_LiNK] − [アプリケーション管理 SUP] メニューを選択します。

   [TP1/LiNK アプリケーション管理 SUP] ウィンドウが表示されます。

2. ユーザサービス定義ファイルと同じ名前のユーザサーバを選択し，[起動(S)] ボタンをクリックします。

4.12　Visual Studio での SPP.NET のデバッグ方法

SPP.NET を Visual Studio のデバッガでデバッグする場合の方法と留意事項について説明します。

4.12.1　Visual Studio での SPP.NET のデバッグ手順

SPP.NET を Visual Studio のデバッガでデバッグする場合の方法を次に示します。

1. Visual Studio のメニューから［デバッグ］－［プロセスにアタッチ］を選択します。
   ［プロセスにアタッチ］画面が表示されます。
2. SPP.NET（DLL）の共通実行可能ファイル（EXE）である njsnetsv.exe のプロセスにアタッチします。
3. SPP.NET のプログラム内の適切な場所にブレイクポイントを設定します。
4. リクエストを発生させてデバッグします。

4.12.2　デバッグ時の留意事項

SPP.NET を Visual Studio のデバッガでデバッグする場合の留意事項を次に示します。

- njsnetsv.exe のプロセスは SPP.NET を起動することで生成されます。
- 複数の SPP.NET を起動すると njsnetsv.exe のプロセスが複数起動します。この場合は，プロセス ID などで，どの SPP.NET に対する njsnetsv.exe のプロセスかを判断してください。
- ［プロセスにアタッチ］画面で［すべてのユーザーからのプロセスを表示する］にチェックが付いていない場合，選択できるプロセスに njsnetsv.exe が表示されません。

5 TP1/LiNK でのユーザサーバの実行環境設定【TP1/LiNK】

この章では，SPP.NET および SUP.NET をユーザサーバとして使用する場合に TP1/LiNK で設定する内容について説明します。TP1/Server Base の場合，この章で説明する設定は不要です。

5.1　TP1/LiNK での実行環境設定の概要

Extension .NET を TP1/LiNK にインストールすると，SPP および SUP の環境を設定する TP1/LiNK のウィンドウやダイアログボックスから，SPP.NET および SUP.NET の環境設定ができるようになります。

SPP.NET の環境設定については，SPP の環境設定の GUI で設定できますが，GUI や設定項目などの一部が異なります。

この章では，SPP.NET の環境設定方法のうち，SPP の環境設定方法と異なる点について説明します。また，Extension .NET のインストール後に変更になる点について説明します。

この章で説明していない設定項目については，マニュアル「TP1/LiNK　使用の手引」の SPP の環境設定方法を参照してください。

SUP.NET の環境設定については，SUP の環境設定の GUI で設定できますので，SUP の環境設定手順に従って設定してください。詳細については，「5.3　ユーザサーバの環境設定（SUP.NET）」，およびマニュアル「TP1/LiNK　使用の手引」の SUP の環境設定方法を参照してください。

5.2　ユーザサーバの環境設定（SPP.NET）

［TP1/LiNK アプリケーション管理 SPP］ウィンドウで，SPP.NET の実行環境を設定します。

［スタート］－［プログラム］－［TP1_LiNK］－［アプリケーション管理 SPP］メニューを選択すると，［TP1/LiNK アプリケーション管理 SPP］ウィンドウが表示されます。

図 5-1　［TP1/LiNK アプリケーション管理 SPP］ウィンドウ

［TP1/LiNK アプリケーション管理 SPP］ウィンドウのリストボックスには，SPP および SPP.NET の両方の情報が表示されます。

この節では，このウィンドウから設定する内容のうち，SPP.NET と SPP で設定内容が異なる次の項目について説明します。

- SPP（SPP.NET）の実行環境の設定
- サーチパスの設定
- 自動起動の設定
- XA 接続の設定

その他の設定については，マニュアル「TP1/LiNK　使用の手引」を参照してください。

5.2.1　SPP.NET の実行環境の設定

SPP.NET の実行環境を設定するときは，［TP1/LiNK アプリケーション管理 SPP］ウィンドウの［サーバ定義(E)...］ボタンをクリックします。

リストボックスからユーザサーバ名を選択しないでボタンをクリックすると，［アプリケーション環境 SPP］ダイアログボックスが表示されます。

リストボックスからユーザサーバ名を選択してからボタンをクリックすると，選んだユーザサーバが SPP.NET か SPP かによって，［SPP.NET 環境設定］ダイアログボックスか，または［SPP 環境設定］ダイアログボックスが表示されます。［SPP 環境設定］ダイアログボックスについては，マニュアル「TP1/LiNK　使用の手引」を参照してください。

(1)　［アプリケーション環境 SPP］ダイアログボックス

(a)　ダイアログボックスの説明

リストボックスには，SPP および SPP.NET の両方の情報が表示されます。

ここで説明しない項目については，マニュアル「TP1/LiNK　使用の手引」の［アプリケーション環境 SPP］ダイアログボックスの説明を参照してください。

図 5-2　［アプリケーション環境 SPP］ダイアログボックス

(b)　ボタンの使い方

［新規作成(N)...］ボタン

ユーザサーバの実行環境を新たに設定する場合にクリックします。SPP.NET，SPP のうち，どちらの実行環境を新たに設定するかを選択する［SPP 環境設定選択］ダイアログボックスが表示されます。

［開く(O)...］ボタン

すでに設定されているユーザサーバの実行環境を変更したい場合に，リストボックスのユーザサーバを指定してからクリックします。指定されたユーザサーバが SPP か SPP.NET かによって，［SPP 環境設定］ダイアログボックスか，または［SPP.NET 環境設定］ダイアログボックスが表示されます。

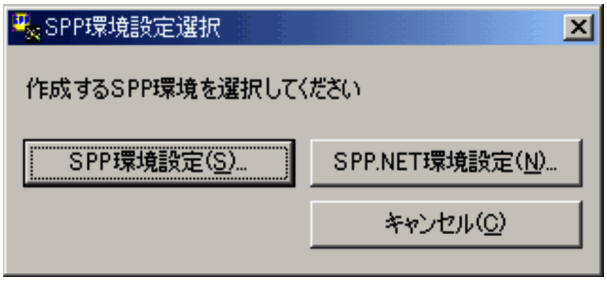
［SPP 環境設定］ダイアログボックスついては，マニュアル「TP1/LiNK　使用の手引」を参照してください。

(2) ［SPP 環境設定選択］ダイアログボックス

(a) ダイアログボックスの説明

SPP または SPP.NET の，どちらの実行環境を新たに設定するかを選択します。

図 5-3 ［SPP 環境設定選択］ダイアログボックス



(b) ボタンの使い方

［SPP 環境設定(S)...］ボタン

SPP の実行環境を設定します。［SPP 環境設定(S)...］ボタンをクリックすると，［SPP 環境設定］ダイアログボックスが表示されます。

［SPP 環境設定］ダイアログボックスについては，マニュアル「TP1/LiNK　使用の手引」を参照してください。

［SPP.NET 環境設定(N)...］ボタン

SPP.NET の環境設定をします。［SPP.NET 環境設定(N)...］ボタンをクリックすると，［SPP.NET 環境設定］ダイアログボックスが表示されます。

［SPP.NET 環境設定］ダイアログボックスについては，「5.2.1(3)　［SPP.NET 環境設定］ダイアログボックス」を参照してください。

［キャンセル(C)］ボタン

［アプリケーション環境 SPP］ダイアログボックスに戻ります。

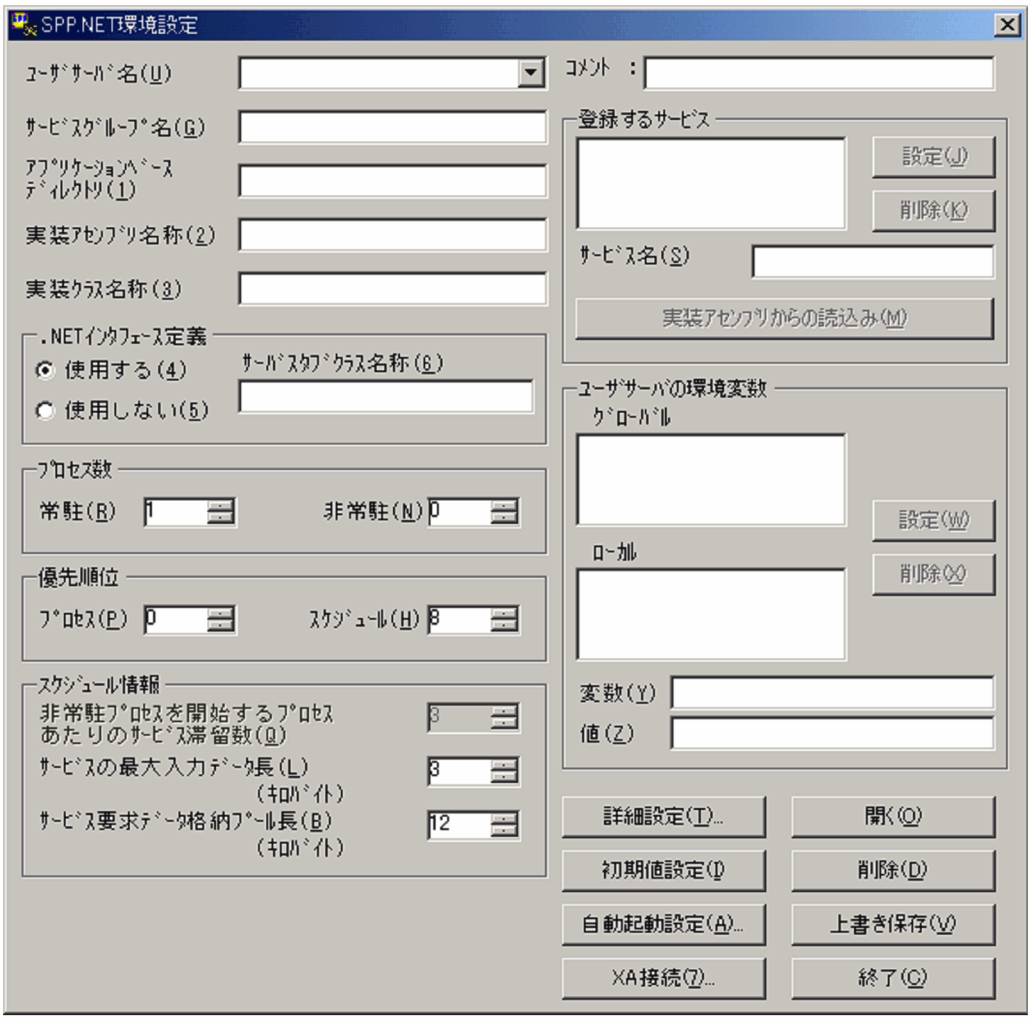
(3) ［SPP.NET 環境設定］ダイアログボックス

(a) ダイアログボックスの説明

SPP.NET の環境を設定します。

ここでは，［SPP 環境設定］ダイアログボックスと設定内容が異なる項目，および［SPP.NET 環境設定］ダイアログボックスだけにある項目について説明します。ここで説明しない項目については，マニュアル「TP1/LiNK　使用の手引」の［SPP 環境設定］ダイアログボックスの説明を参照してください。

図 5-4　[SPP.NET 環境設定] ダイアログボックス



(b)　ダイアログボックスに設定する項目

[ユーザサーバ名(U)]　　～< 1～8 文字の識別子>

SPP.NET の実行環境を新しく設定する場合は，実行環境を設定する SPP.NET のユーザサーバ名を入力します。すでに設定されている SPP.NET の実行環境を変更する場合は，ドロップダウンリストからユーザサーバ名を選びます。なお，ドロップダウンリストには，SPP.NET だけが表示されます。

ユーザサーバ名には，英字の大文字と小文字の区別はありません。小文字で指定しても，すべて大文字で管理されます。

[アプリケーションベースディレクトリ(1)]　　～< 1～128 文字のパス名>

SPP.NET 実行時のアプリケーションベースディレクトリを指定します。

実装アセンブリを任意のディレクトリに配置した場合は，このテキストボックスか，または［アプリケーションベースディレクトリ設定］ダイアログボックスにディレクトリが指定されている必要があります。

［アプリケーションベースディレクトリ設定］ダイアログボックスについては，「5.2.2(2)　［アプリケーションベースディレクトリ設定］ダイアログボックス」を参照してください。

このテキストボックス，および［アプリケーションベースディレクトリ設定］ダイアログボックスの両方とも指定しない場合は，%DCDIR%¥aplib がアプリケーションベースディレクトリとして設定されるので，実装アセンブリは%DCDIR%¥aplib に格納されている必要があります。

なお，「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

［実装アセンブリ名称(2)］　～< 1～128 文字の識別子とピリオド>

SPP.NET 実装クラス，サーバスタブなどが含まれるアセンブリの名称を指定します。なお，ファイルの拡張子（dll）は不要です。

［実装クラス名称(3)］　～< 1～128 文字の識別子とピリオド>

SPP.NET の実装クラス名称を指定します。名前空間を含む完全限定名で指定してください。

［.NET インタフェース定義］欄　～<［使用する(4)］|［使用しない(5)］>《［使用する(4)］》

SPP.NET が.NET インタフェース定義を使用するかどうかを，オプションボタンで指定します。

- **［サーバスタブクラス名称(6)］　～< 1～128 文字の識別子とピリオド>**

  ［.NET インタフェース定義］欄で［使用する(4)］をオンにしたときは，［サーバスタブクラス名称(6)］テキストボックスにサーバスタブのクラス名称を指定します。このとき，名前空間を含む完全限定名で指定します。.NET インタフェース定義を使用した SPP.NET の場合は，必ず入力してください。

  ［.NET インタフェース定義］欄で［使用しない(5)］をオンにしたときは，［サーバスタブクラス名称(6)］テキストボックスには入力できません。

［登録するサービス］欄　～< 1～31 文字の識別子>

この SPP.NET にあるサービス名を入力します。

リストボックスにサービス名を追加する場合は，次に示すどちらかの方法で実行します。

- 欄内下の［サービス名(S)］テキストボックスにサービス名を入力して，［設定(I)］ボタンをクリックします。
- ［実装アセンブリからの読込み(M)］ボタンをクリックして，実装アセンブリからサービス名を自動的に読み込みます。

リストボックスからサービス名を削除する場合は，削除するサービス名をリストボックスから選択し，［削除(K)］ボタンをクリックします。

(c)　ボタンの使い方

［実装アセンブリからの読込み(M)］ボタン

実装アセンブリからサービス名を自動的に読み込みます。このボタンは，次に示すすべての項目が指定されている場合にだけクリックできます。

- ［アプリケーションベースディレクトリ(1)］テキストボックス（省略した場合は［アプリケーションベースディレクトリ設定］ダイアログボックスのテキストボックス）
- ［実装アセンブリ名称(2)］テキストボックス
- ［実装クラス名称(3)］テキストボックス

［詳細設定(T)...］ボタン

［SPP.NET 詳細設定］ダイアログボックスが表示されます。［SPP.NET 詳細設定］ダイアログボックスについては，「5.2.1(4) ［SPP.NET 詳細設定］ダイアログボックス（［RPC］タブ)」および「5.2.1(5) ［SPP.NET 詳細設定］ダイアログボックス（［その他］タブ)」を参照してください。

［XA 接続(Z)...］ボタン

［XA 接続］ダイアログボックスが表示されます。［XA 接続］ダイアログボックスについては，「5.2.4(1) ［XA 接続］ダイアログボックス」を参照してください。

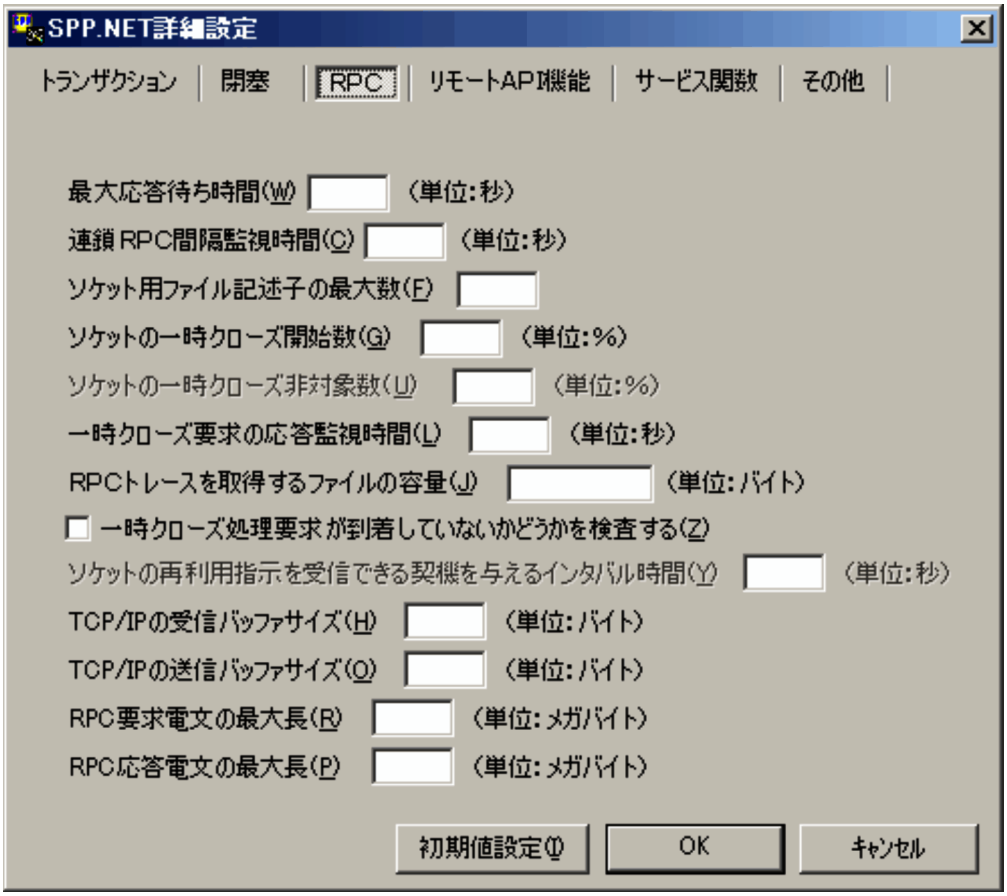
(4) ［SPP.NET 詳細設定］ダイアログボックス（［RPC］タブ）

(a) ダイアログボックスの説明

SPP.NET の環境の詳細を設定します。

ここでは，［SPP 詳細設定］ダイアログボックスと異なる［RPC］タブの設定項目について説明します。ここで説明しない項目については，マニュアル「TP1/LiNK 使用の手引」の［SPP 詳細設定］ダイアログボックスの説明を参照してください。

図 5-5 ［SPP.NET 詳細設定］ダイアログボックス（［RPC］タブ）



(b) ダイアログボックスに設定する項目

[RPC 要求電文の最大長(R)]　～<符号なし整数>((1～8))《1》(単位：メガバイト)

SPP.NET がクライアント UAP から受け付ける RPC 要求メッセージの最大長を指定します。ここで指定する値は，[RPC 詳細設定] ダイアログ（[その他] タブ）で指定する [RPC 送受信電文の最大長(M)] の値以下にしてください。[RPC 詳細設定] ダイアログについては，マニュアル「TP1/LiNK 使用の手引」を参照してください。

[RPC 応答電文の最大長(P)]　～<符号なし整数>((1～8))《1》(単位：メガバイト)

SPP.NET がクライアント UAP にサービスの応答をする場合，SPP.NET がクライアント UAP に返す RPC 応答メッセージの最大長を指定します。ここで指定する値は，[RPC 詳細設定] ダイアログ（[その他] タブ）で指定する [RPC 送受信電文の最大長(M)] の値以下にしてください。[RPC 詳細設定] ダイアログについては，マニュアル「TP1/LiNK　使用の手引」を参照してください。

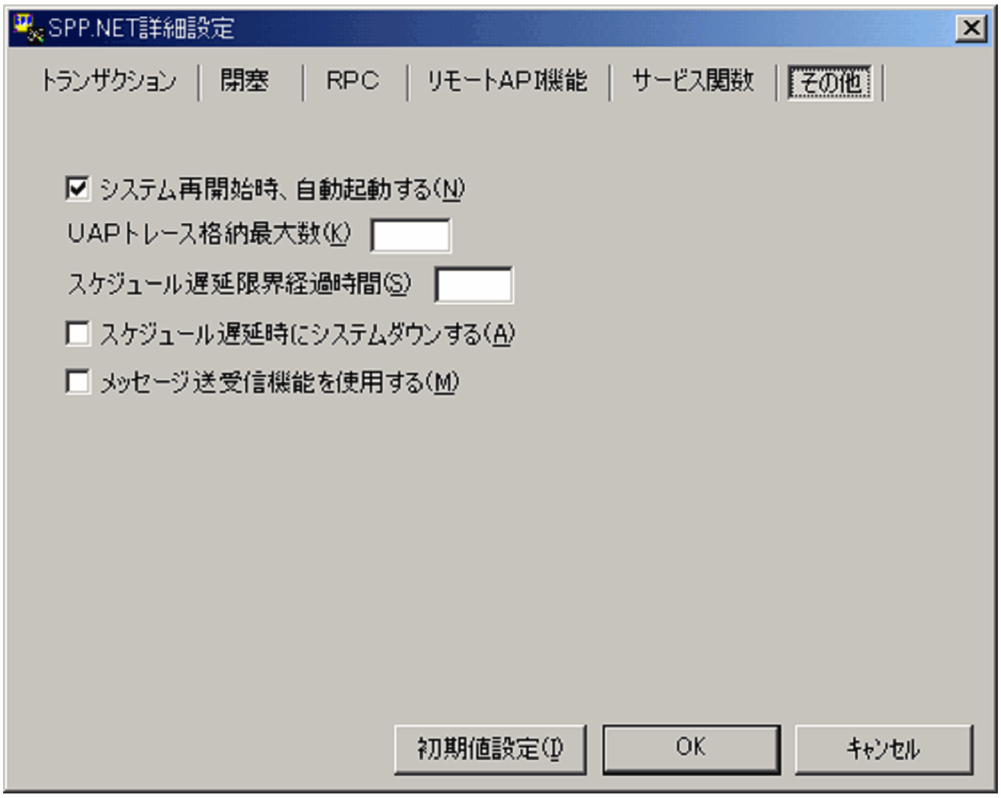
(5) [SPP.NET 詳細設定] ダイアログボックス（[その他] タブ）

(a) ダイアログボックスの説明

SPP.NET の環境の詳細を設定します。

ここでは，[SPP 詳細設定] ダイアログボックスと異なる [その他] タブの設定項目について説明します。ここで説明しない項目については，マニュアル「TP1/LiNK　使用の手引」の [SPP 詳細設定] ダイアログボックスの説明を参照してください。

図 5-6　[SPP.NET 詳細設定] ダイアログボックス（[その他] タブ）



(b)　ダイアログボックスに設定する項目

［メッセージ送受信機能を使用する(M)］

メッセージ送受信機能を使用するかどうかを指定します。チェックボックスをオンにすると，メッセージ送受信機能が使用できます。チェックボックスがオフの場合，メッセージ送受信機能は使用できません。

5.2.2　サーチパスの設定

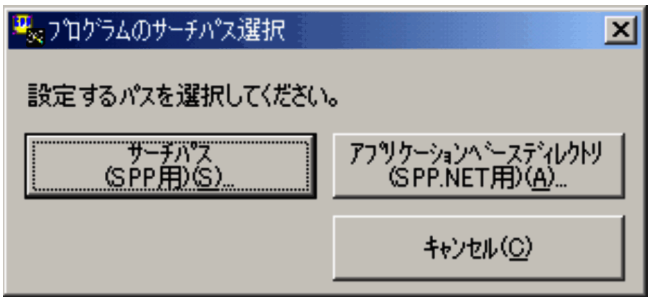
ユーザサーバ固有で有効になるサーチパスを設定するときは，［TP1/LiNK アプリケーション管理 SPP］ウィンドウの［プログラムのサーチパス(P)...］ボタンをクリックします。ボタンをクリックすると，SPP 用のサーチパスか，SPP.NET 用のアプリケーションベースディレクトリのどちらかを設定する，［プログラムのサーチパス選択］ダイアログボックスが表示されます。

(1)　［プログラムのサーチパス選択］ダイアログボックス

(a)　ダイアログボックスの説明

SPP 用のサーチパス，または SPP.NET 用のアプリケーションベースディレクトリ（SPP.NET 用のパス）のどちらを設定するかを選択します。

図 5-7　［プログラムのサーチパス選択］ダイアログボックス



(b)　ボタンの使い方

［サーチパス(SPP 用)(S)...］ボタン

SPP 用のサーチパスを設定する場合は，［サーチパス(SPP 用)(S)...］ボタンをクリックします。ボタンをクリックすると，［TP1/LiNK サーチパス］ダイアログボックスが表示されます。

［TP1/LiNK サーチパス］ダイアログボックスについては，マニュアル「TP1/LiNK　使用の手引」を参照してください。

［アプリケーションベースディレクトリ(SPP.NET 用)(A)...］ボタン

SPP.NET 用のアプリケーションベースディレクトリを設定する場合は，［アプリケーションベースディレクトリ(SPP.NET 用)(A)...］ボタンをクリックします。ボタンをクリックすると，［アプリケーションベースディレクトリ設定］ダイアログボックスが表示されます。

[キャンセル(C)]ボタン

［TP1/LiNK アプリケーション管理 SPP］ウィンドウに戻ります。

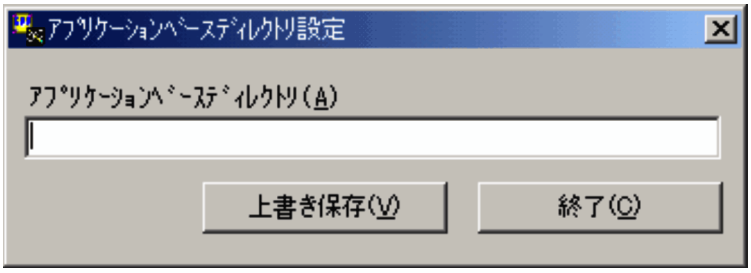
(2) ［アプリケーションベースディレクトリ設定］ダイアログボックス

(a) ダイアログボックスの説明

SPP.NET のアプリケーションベースディレクトリのデフォルト値を指定します。

ここで設定するアプリケーションベースディレクトリは，［SPP.NET 環境設定］ダイアログボックスの［アプリケーションベースディレクトリ(1)］テキストボックスに何も設定されていない場合に使用されます。

図 5-8 ［アプリケーションベースディレクトリ設定］ダイアログボックス



(b) ダイアログボックスに設定する項目

[アプリケーションベースディレクトリ(A)]

アプリケーションベースディレクトリを設定する場合は，［アプリケーションベースディレクトリ(A)］テキストボックスにパスを指定します。

(c) ボタンの使い方

[上書き保存(V)]ボタン

アプリケーションベースディレクトリを設定します。［アプリケーションベースディレクトリ(A)］テキストボックスに何も指定されていない状態で［上書き保存(V)］ボタンをクリックすると，設定が削除されます。

[終了(C)]ボタン

［プログラムのサーチパス選択］ダイアログボックスに戻ります。

5.2.3 自動起動の設定

ユーザサーバを自動起動するかどうかを設定するときは，次のどれかの操作をします。

- ［アプリケーション環境 SPP］ダイアログボックスの［自動起動設定(A)...］ボタンをクリックする。
- ［SPP 環境設定］ダイアログボックスの［自動起動設定(A)...］ボタンをクリックする。

- ［SPP.NET 環境設定］ダイアログボックスの［自動起動設定(A)...］ボタンをクリックする。

ボタンをクリックすると，［自動起動設定］ダイアログボックスが表示されます。このダイアログボックスで，ユーザサーバを自動起動するかどうか，および自動起動の順番を設定します。

図 5-9 ［自動起動設定］ダイアログボックス

ウィンドウ上のリストボックスには，SPP および SPP.NET の両方の情報が表示されます。

ボタンの使い方，およびダイアログボックスに設定する項目については，マニュアル「TP1/LiNK　使用の手引」を参照してください。

参考

次のウィンドウ，およびダイアログボックスから表示される［自動起動設定］ダイアログボックスにも，SPP および SPP .NET の両方の情報が表示されます。

- ［アプリケーション環境 SUP］ダイアログボックス
- ［SUP 環境設定］ダイアログボックス
- ［RAP サービス環境］ウィンドウ
- ［RAP サービス環境設定］ダイアログボックス

5.2.4　XA 接続の設定

ユーザサーバごとに XA 接続するリソースマネジャを設定するときは，次のどれかの操作をします。

- ［SPP.NET 環境設定］ダイアログボックスの［XA 接続(7)...］ボタンをクリックする。
- ［SUP 環境設定］ダイアログボックスの［XA 接続(X)...］ボタンをクリックする。

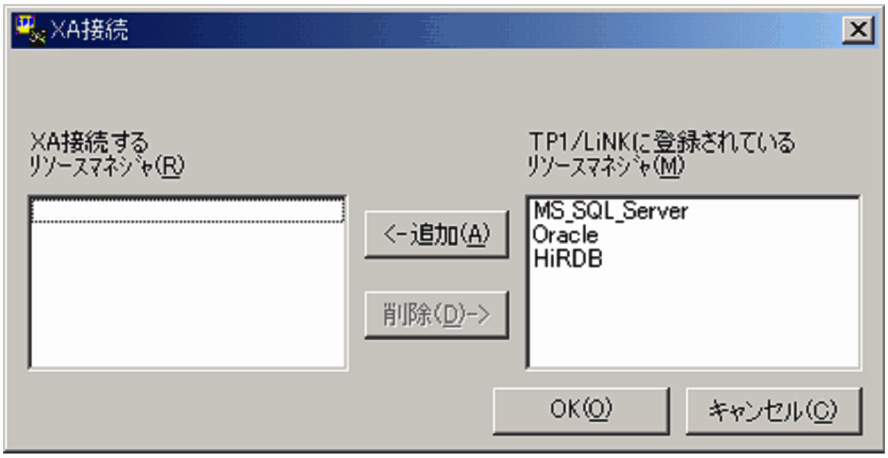
ボタンをクリックすると，［XA 接続］ダイアログボックスが表示されます。

(1) ［XA 接続］ダイアログボックス

(a) ダイアログボックスの説明

ユーザサーバごとに XA 接続するリソースマネジャを設定します。

図 5-10 ［XA 接続］ダイアログボックス



(b) ダイアログボックスに設定する項目

［XA 接続するリソースマネジャ(R)］

ユーザサーバに登録するリソースマネジャの一覧が表示されます。

［TP1/LiNK に登録されているリソースマネジャ(M)］

TP1/LiNK に登録されているリソースマネジャの一覧が表示されます。リソースマネジャを TP1/LiNK に登録する方法については，「5.4 リソースマネジャの接続」を参照してください。

(c) ボタンの使い方

［追加(A)］ボタン

［TP1/LiNK に登録されているリソースマネジャ(M)］リストボックスに表示されているリソースマネジャを［XA 接続するリソースマネジャ(R)］リストボックスに追加します。［XA 接続するリソースマネジャ(R)］リストボックスに追加したリソースマネジャ名は，［TP1/LiNK に登録されているリソースマネジャ(M)］リストボックスから削除されます。

［削除(D)］ボタン

［XA 接続するリソースマネジャ(R)］リストボックスに表示されているリソースマネジャを，［XA 接続するリソースマネジャ(R)］リストボックスから削除します。削除したリソースマネジャ名は，［TP1/LiNK に登録されているリソースマネジャ(M)］リストボックスに表示されます。

［OK(O)］ボタン

設定した情報に変更して，［XA 接続］ダイアログボックスを終了します。

［キャンセル(C)］ボタン

設定した情報を変更しないで，［XA 接続］ダイアログボックスを終了します。

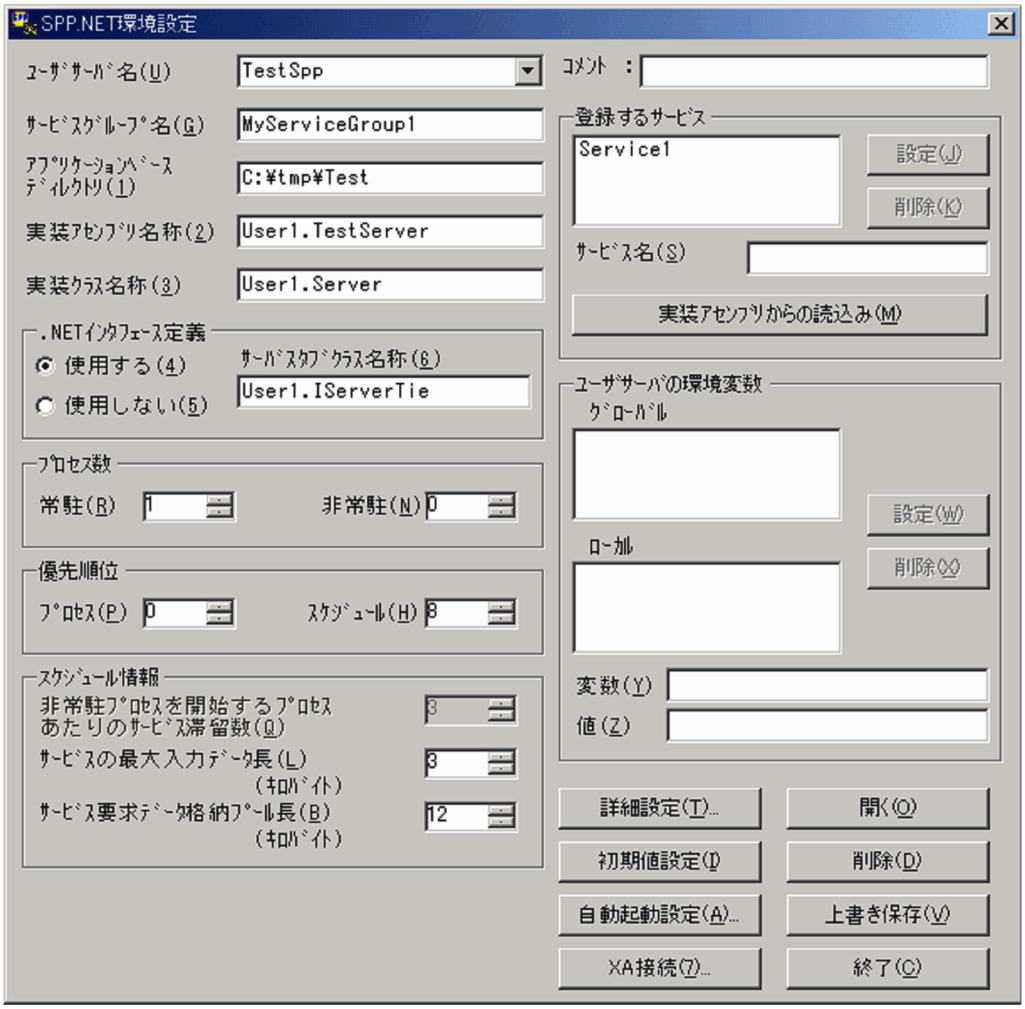
注意事項

［XA 接続するリソースマネジャ(R)］リストボックスに設定したリソースマネジャを TP1/LiNK から削除した場合は，再度［XA 接続］ダイアログボックスを表示させ，［XA 接続するリソースマネジャ(R)］リストボックスの設定をし直してください。

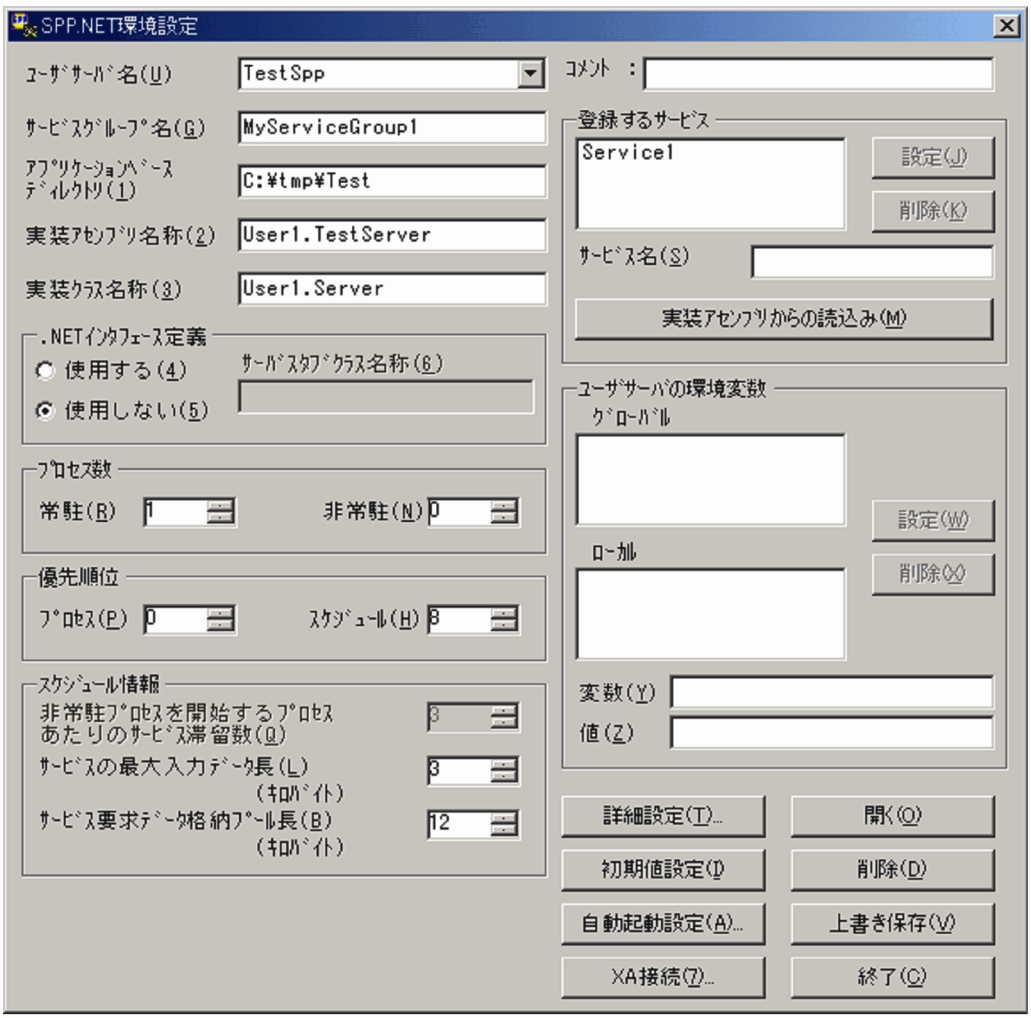
5.2.5　SPP.NET の実行環境設定の指定例

SPP.NET の実行環境設定の指定例を示します。

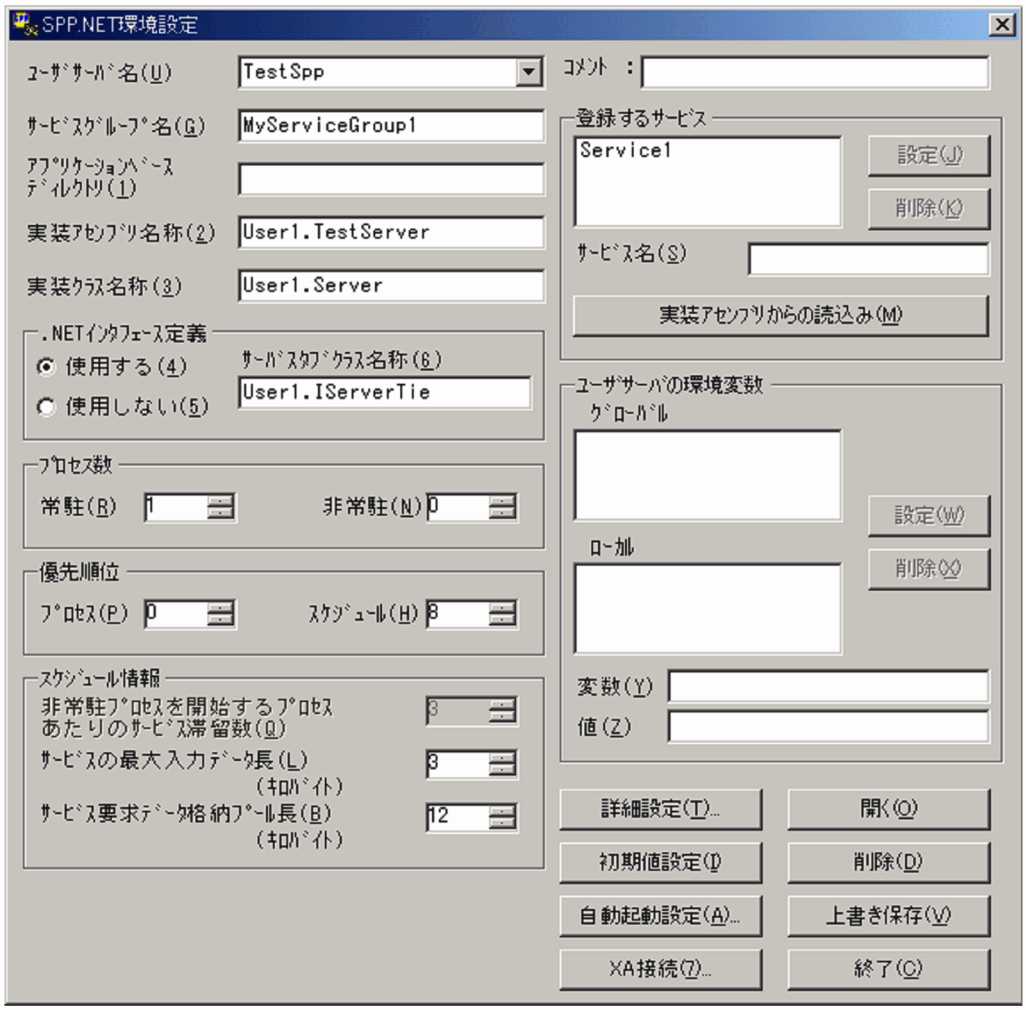
(1)　.NET インタフェース定義を使用する場合



(2)　.NET インタフェース定義を使用しない場合



(3) .NET インタフェース定義を使用し，かつアプリケーションベースディレクトリを指定しない場合

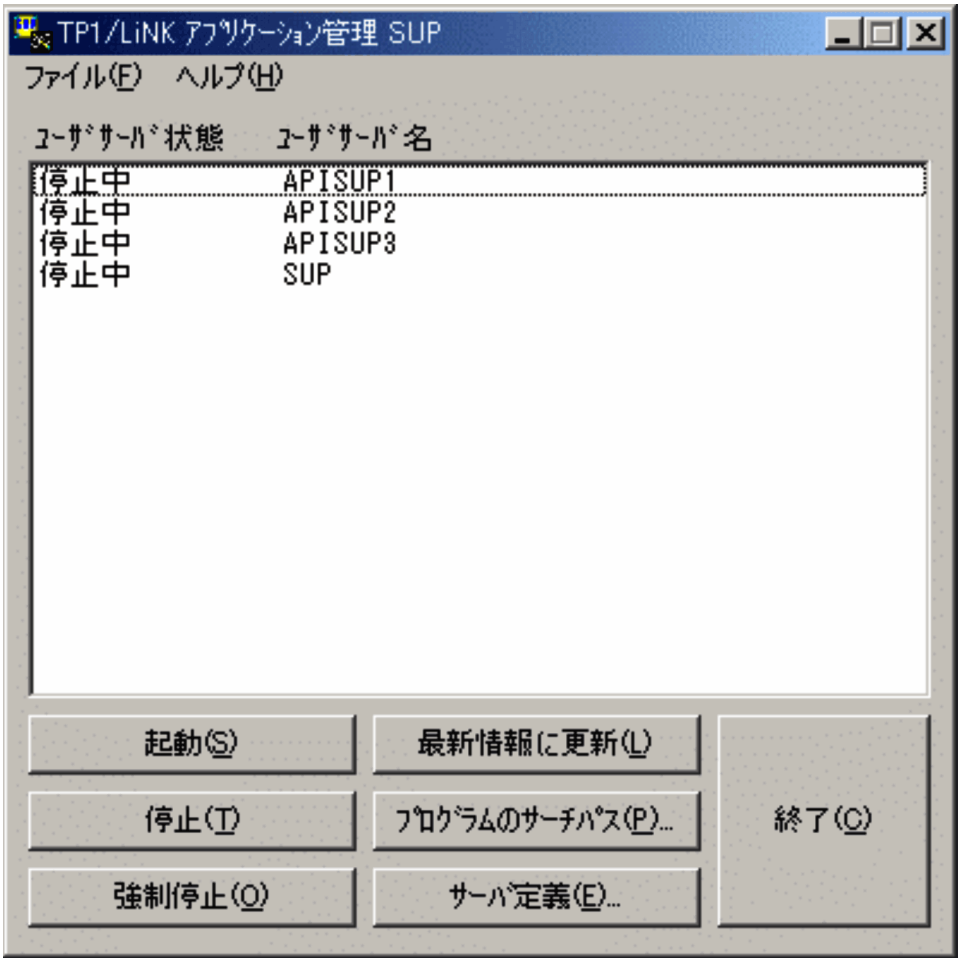


5.3　ユーザサーバの環境設定（SUP.NET）

［TP1/LiNK アプリケーション管理 SUP］ウィンドウで，SUP.NET の実行環境を設定します。

［スタート］－［プログラム］－［TP1_LiNK］－［アプリケーション管理 SUP］メニューを選択すると，［TP1/LiNK アプリケーション管理 SUP］ウィンドウが表示されます。

図 5-11　［TP1/LiNK アプリケーション管理 SUP］ウィンドウ



［TP1/LiNK アプリケーション管理 SUP］ウィンドウのリストボックスには，SUP およびの SUP.NET の両方の情報が表示されます。

この節では，このウィンドウから設定する内容のうち，SUP.NET（SUP）の実行環境の設定について説明します。

SUP.NET（SUP）の詳細設定については，SUP の詳細設定の GUI で設定できます。SUP.NET（SUP）の詳細設定については，マニュアル「TP1/LiNK　使用の手引」を参照してください。

5.3.1　SUP.NET の実行環境の設定

SUP.NET の実行環境を設定するときは，［TP1/LiNK アプリケーション管理 SUP］ウィンドウの［サーバ定義(E)...］ボタンをクリックします。

リストボックスからユーザサーバ名を選択しないでボタンをクリックすると，［アプリケーション環境 SUP］ダイアログボックスが表示されます。

リストボックスからユーザサーバ名を選択してからボタンをクリックすると，選んだユーザサーバを設定する［SUP 環境設定］ダイアログボックスが表示されます。

なお，ユーザサーバが SUP.NET，SUP のどちらの場合でも，［SUP 環境設定］ダイアログボックスで設定できます。

(1)　［アプリケーション環境 SUP］ダイアログボックス

図 5-12　［アプリケーション環境 SUP］ダイアログボックス

リストボックスには，SUP および SUP.NET の両方の情報が表示されます。

ボタンの使い方，およびダイアログボックスに設定する項目については，マニュアル「TP1/LiNK　使用の手引」を参照してください。

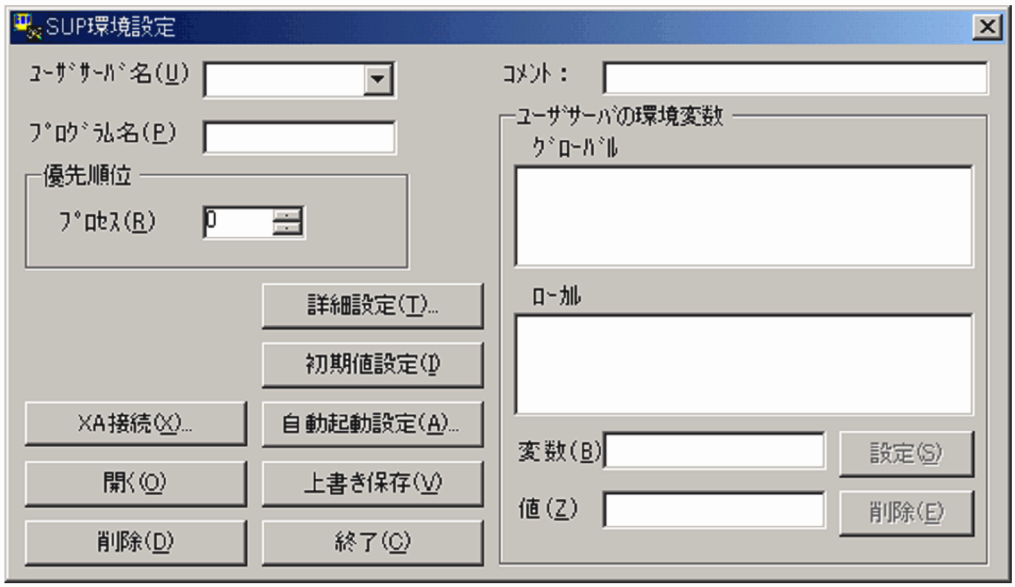
(2)　［SUP 環境設定］ダイアログボックス

(a)　ダイアログボックスの説明

SUP.NET および SUP の環境を設定します。

ここで説明しない項目については，マニュアル「TP1/LiNK　使用の手引」の［SUP 環境設定］ダイアログボックスの説明を参照してください。

図 5-13　[SUP 環境設定] ダイアログボックス



(b)　ボタンの使い方

[XA 接続(X)...] ボタン

[XA 接続] ダイアログボックスが表示されます。SUP.NET の場合だけ使用します。SUP の場合も，このボタンをクリックすると [XA 接続] ダイアログボックスが表示されますが，表示された [XA 接続] ダイアログボックスで設定した内容は無効となります。[XA 接続] ダイアログボックスについては，「5.2.4(1)　[XA 接続] ダイアログボックス」を参照してください。

5.4　リソースマネジャの接続

トランザクション連携をしてリソースマネジャにアクセスする場合，SPP および SUP と，SPP.NET および SUP.NET とでは，次のような違いがあります。

- SPP および SUP でトランザクション連携を行う場合

  トランザクション制御用オブジェクトファイルを作成し，ユーザサーバとリンケージする必要があります。
- SPP.NET および SUP.NET で DBMS とのトランザクション連携を行う場合

  トランザクション制御用オブジェクトを作成する必要はありません。代わりに XA 接続の設定が必要になります。XA 接続の設定については，「5.2.4　XA 接続の設定」を参照してください。

6 運用コマンド

この章では，TP1/Extension for .NET Framework（Extension .NET）で使用する運用コマンドについて説明します。

運用コマンドの種類

Extension .NET で使用できる運用コマンドを，次の表に示します。

表 6-1　運用コマンドの種類（Extension .NET）

運用コマンド	機能
if2cstub	C#，または Visual Basic で定義された SPP.NET 用インタフェース（.NET インタフェース定義）を基に，クライアントスタブクラスのソースファイルを生成します。
if2sstub	C#，または Visual Basic で定義された SPP.NET 用インタフェース（.NET インタフェース定義）を基に，サーバスタブクラスのソースファイルを生成します。
if2tsdl	C#，または Visual Basic で定義された.NET インタフェース定義を基に，TP1 Service Description Language（TSDL）を生成します。
spp2cstub	指定されたサービス定義ファイル，およびそのサービス定義ファイルが参照するデータ型定義ファイルを基に，クライアントスタブクラスおよびカスタムレコードクラスのソースファイルを生成します。

if2cstub（クライアントスタブ生成コマンド（.NET インタフェース定義用））

形式

```
if2cstub {-t {svr|clt|con}
         〔-l {cs|vb}〕
         〔-s 生成ファイル拡張子〕
         〔-n 名前空間名称〕
         〔-o 出力先ディレクトリ〕
         〔-r スタブクラス名称〕
         〔-c {struct|nostruct}〕
         〔-X {normal|dataset}〕
         〔-m RPCメッセージの最大長〕
          -i .NETインタフェース定義ファイル名称
          インタフェース名称
         |-h}
```

機能

C#，または Visual Basic で定義された SPP.NET 用インタフェース（.NET インタフェース定義）を基に，クライアントスタブクラスのソースファイル（以降，クライアントスタブソースファイルと呼びます）を生成します。

オプション

●-t {svr | clt | con}

生成するクライアントスタブの種類を指定します。

オプションの指定と生成されるクライアントスタブを次に示します。

オプションの指定	生成されるクライアントスタブ
svr	SPP.NET または SUP.NET（Extension .NET）用
clt	CUP.NET（Client .NET）用
con	CUP.NET（Connector .NET）用

●-l {cs | vb}

生成するクライアントスタブソースファイルのプログラム言語を指定します。

このオプションを省略した場合，入力元のソースファイルと同じプログラム言語で生成します。入力元のソースファイルのプログラム言語はファイルの拡張子を基に判断されます。

オプションの指定と生成されるクライアントスタブソースファイルのプログラム言語の関係を次に示します。

オプションの指定	生成されるクライアントスタブソースファイルのプログラム言語
cs	C#

オプションの指定	生成されるクライアントスタブソースファイルのプログラム言語
vb	Visual Basic

このオプションを省略した場合に，生成されるクライアントスタブソースファイルのプログラム言語を次に示します。

入力元のソースファイルの拡張子	生成されるクライアントスタブソースファイルのプログラム言語
cs	C#
vb	Visual Basic

●-s　生成ファイル拡張子　　～〈文字列〉

生成するクライアントスタブソースファイルの拡張子を指定します。

このオプションを省略した場合，生成されるクライアントスタブソースファイルのプログラム言語によって，拡張子は次のようになります。

生成されるクライアントスタブソースファイルのプログラム言語	クライアントスタブソースファイルの拡張子
C#	cs
Visual Basic	vb

●-n　名前空間名称　　～〈文字列〉

生成するクライアントスタブクラスの名前空間名称を指定します。

このオプションを省略した場合，入力元のインタフェースが属する名前空間と同じ名前空間でクライアントスタブクラスが生成されます。入力元のインタフェースが名前空間なしの場合，名前空間なしのクライアントスタブクラスが生成されます。

●-o　出力先ディレクトリ　　～〈パス名〉

生成するクライアントスタブソースファイルを出力するディレクトリを指定します。絶対パスまたは相対パスで指定してください。

このオプションを省略した場合，コマンド実行時のディレクトリに出力されます。なお，ファイル名はクライアントスタブごとに「〈名前空間を含まないスタブクラス名称〉.〈拡張子〉」で生成されます。

●-r　スタブクラス名称　　～〈文字列〉

生成するクライアントスタブのクラス名称を指定します。

このオプションを省略した場合，クラス名称は「〈インタフェース名称〉Stub」になります。

●-c {struct | nostruct} ～《struct》

.NET インタフェース定義のメソッドのパラメタまたは戻り値に TP1 ユーザ構造体を使用した場合に，クライアントスタブが利用するための TP1 ユーザ構造体クラスを出力するかどうかを指定します。

このオプションを省略した場合，TP1 ユーザ構造体クラスを出力します。.NET インタフェース定義に TP1 ユーザ構造体を指定していなかった場合，このオプションの指定は無視されます。

struct：TP1 ユーザ構造体クラスを出力します。

nostruct：TP1 ユーザ構造体クラスを出力しません。

●-X {normal | dataset}

生成されるクライアントスタブに，引数および戻り値が XmlDocument クラス（System.Xml.XmlDocument）となるサービスメソッドが追加されます。

サービスメソッドに入力パラメタがある場合，入力データ用 XML スキーマファイルがサービスメソッドごとに出力されます。出力パラメタまたは戻り値がある場合，出力データ用 XML スキーマファイルがサービスメソッドごとに出力されます。

-t オプションに con を指定した場合だけ，このオプションの指定が有効になります。-t オプションに con 以外を指定した場合，このオプションの指定は無視されます。

このオプションを省略した場合，生成されるクライアントスタブに，引数および戻り値が XmlDocument クラスとなるサービスメソッドは追加されません。また，入力データ用 XML スキーマファイルおよび出力データ用 XML スキーマファイルは出力されません。

normal：引数および戻り値の XmlDocument オブジェクト（System.Xml.XmlDocument）を，.NET Framework の DataSet オブジェクト（System.Data.DataSet）と連携させない場合に指定します。normal を指定した場合，出力される入力データ用 XML スキーマファイルおよび出力データ用 XML スキーマファイルは，DataSet オブジェクトで利用できない場合があります。

dataset：引数および戻り値の XmlDocument オブジェクト（System.Xml.XmlDocument）を，.NET Framework の DataSet オブジェクト（System.Data.DataSet）と連携させて利用する場合に指定します。dataset を指定した場合，DataSet オブジェクトで利用できる入力データ用 XML スキーマファイルおよび出力データ用 XML スキーマファイルを出力します。また，その入力データ用 XML スキーマおよび出力データ用 XML スキーマに対応したクライアントスタブを出力します。

●-m RPC メッセージの最大長 ～〈符号なし整数〉((1～8))《1》（単位：メガバイト）

サービスメソッドが指定する RPC 応答メッセージの最大長を指定します。

このオプションに 1～8 以外を指定した場合は，エラーが発生します。なお，このオプションは，-t オプションに svr を指定したときだけ有効です。-t オプションの指定が svr 以外の場合は，-m オプションの指定は無視されます。

このオプションには，必ず次の値以下の値を指定してください。

【TP1/Server Base】

- TP1/Server Base のシステム共通定義の rpc_max_message_size オペランドの値

  rpc_max_message_size オペランドについては，マニュアル「OpenTP1　システム定義」を参照してください。

- ユーザサービスデフォルト定義およびユーザサービス定義の njs_output_max_message_size オペランドの値

  njs_output_max_message_size オペランドについては，3 章の「ユーザサービスデフォルト定義」および「ユーザサービス定義」を参照してください。

【TP1/LiNK】

- ［RPC 詳細設定］ダイアログ（［その他］タブ）で指定する［RPC 送受信電文の最大長(M)］の値

  ［RPC 詳細設定］ダイアログ（［その他］タブ）については，マニュアル「TP1/LiNK　使用の手引」を参照してください。

- ［SPP.NET 詳細設定］ダイアログ（［RPC］タブ）で指定する［RPC 応答電文の最大長(P)］の値

  ［SPP.NET 詳細設定］ダイアログ（［RPC］タブ）については，「5.2.1(4)　［SPP.NET 詳細設定］ダイアログボックス（［RPC］タブ）」を参照してください。

なお，このオプションを指定して生成したクライアントスタブを使用するプロセスでは，応答メッセージ用として，「指定値×1 メガバイト」のメモリを使用します。また，複数のクライアントスタブを使用するプロセスでは，複数あるクライアントスタブの中で最もメモリを使用する値を応答メッセージ用としてプロセスで使用します。したがって，必要以上に大きな値を設定しないようにしてください。

●-i　.NET インタフェース定義ファイル名称　　～〈ファイル名〉

入力元のインタフェースが定義されている.NET インタフェース定義ファイル名称を指定します。絶対パスまたは相対パスで指定してください。

.NET インタフェース定義を作成したプログラム言語は，このオプションで指定したファイルの拡張子によって次のように判断されます。

このオプションで指定した.NET インタフェース定義ファイルの拡張子	.NET インタフェース定義を作成したプログラム言語の判断結果
cs	C#
vb	Visual Basic

●-h

このコマンドの使用方法を標準出力に表示します。

このオプションを指定した場合，ほかのオプションおよびコマンド引数は指定できません。

コマンド引数

●インタフェース名称

生成したいクライアントスタブに対応する SPP.NET のインタフェース名称を完全限定名で指定します。インタフェース名称は一つだけ指定できます。

注意事項

- このコマンドの実行時にエラーが発生した場合は，対応するエラーメッセージが標準エラー出力に出力されます。
- このコマンドで生成したクライアントスタブソースファイルの内容は変更しないでください。
- 入力元となる.NET インタフェース定義ファイルは，使用する Windows のデフォルトコードページ（日本語版 Windows の場合は 932）で保存してください。Unicode など，ほかのコードページで保存したファイルは，このコマンドの入力元として使用できません。

if2sstub（サーバスタブ生成コマンド（.NET インタフェース定義用））

形式

```
if2sstub { 〔-s 生成ファイル拡張子〕
           〔-n 名前空間名称〕
           〔-o 出力先ディレクトリ〕
           〔-r スタブクラス名称〕
            -i .NETインタフェース定義ファイル名称
            インタフェース名称
           |-h}
```

機能

C#，または Visual Basic で定義された SPP.NET 用インタフェース（.NET インタフェース定義）を基に，サーバスタブクラスのソースファイル（以降，サーバスタブソースファイルと呼びます）を生成します。サーバスタブは，入力元のソースファイルと同じプログラム言語で生成します。入力元ソースファイルのプログラム言語はファイルの拡張子を基に判断されます。

入力元ソースファイルの拡張子と生成されるサーバスタブソースファイルのプログラム言語の関係を次に示します。

入力元ソースファイルの拡張子	生成されるサーバスタブソースファイルのプログラム言語
cs	C#
vb	Visual Basic

オプション

●-s　生成ファイル拡張子　　～〈文字列〉

生成するサーバスタブソースファイルの拡張子を指定します。

このオプションを省略した場合，生成されるサーバスタブソースファイルのプログラム言語によって，拡張子は次のようになります。

生成されるサーバスタブソースファイルのプログラム言語	サーバスタブソースファイルの拡張子
C#	cs
Visual Basic	vb

●-n　名前空間名称　　～〈文字列〉

生成するサーバスタブクラスの名前空間名称を指定します。

このオプションを省略した場合，入力元のインタフェースが属する名前空間と同じ名前空間でサーバスタブクラスが生成されます。入力元のインタフェースが名前空間なしの場合，名前空間なしのサーバスタブクラスが生成されます。

●-o　出力先ディレクトリ　　～〈パス名〉

生成するサーバスタブソースファイルを出力するディレクトリを指定します。絶対パスまたは相対パスで指定してください。

このオプションを省略した場合，コマンド実行時のディレクトリに出力されます。なお，ファイル名はサーバスタブごとに「〈名前空間を含まないスタブクラス名称〉.〈拡張子〉」で生成されます。

●-r　スタブクラス名称　　～〈文字列〉

生成するサーバスタブのクラス名称を指定します。

このオプションを省略した場合，クラス名称は「〈インタフェース名称〉Tie」になります。

●-i　.NET インタフェース定義ファイル名称　　～〈ファイル名〉

入力元のインタフェースが定義されている.NET インタフェース定義ファイルを指定します。絶対パスまたは相対パスで指定してください。

.NET インタフェース定義を作成したプログラム言語は，このオプションで指定したファイルの拡張子によって次のように判断されます。

このオプションで指定した.NET インタフェース定義ファイルの拡張子	.NET インタフェース定義を作成したプログラム言語の判断結果
cs	C#
vb	Visual Basic

●-h

このコマンドの使用方法を標準出力に表示します。

このオプションを指定した場合，ほかのオプションおよびコマンド引数は指定できません。

コマンド引数

●インタフェース名称

生成したいサーバスタブに対応する SPP.NET のインタフェース名称を完全限定名で指定します。インタフェース名称は一つだけ指定できます。

注意事項

- このコマンドの実行時にエラーが発生した場合は，対応するエラーメッセージが標準エラー出力に出力されます。
- このコマンドで生成したサーバスタブソースファイルの内容は変更しないでください。
- 生成されるサーバスタブクラスの名前空間を含む完全限定名が 128 文字以内になるよう，名前空間を含むインタフェース名称，名前空間名称，およびサーバスタブクラス名称を指定してください。

- 入力元となる.NET インタフェース定義ファイルは，使用する Windows のデフォルトコードページ（日本語版 Windows の場合は 932）で保存してください。Unicode など，ほかのコードページで保存したファイルは，このコマンドの入力元として使用できません。

if2tsdl（TP1 Service Description Language（TSDL）生成コマンド）

形式

```
if2tsdl { 〔-s 生成ファイル拡張子〕
          〔-o 出力先ディレクトリ〕
           -i .NETインタフェース定義ファイル名称
           インタフェース名称
          |-h}
```

機能

C#，または Visual Basic で定義された.NET インタフェース定義を基に，TP1 Service Description Language（TSDL）を生成します。

TSDL は，OpenTP1 for .NET Framework 環境以外から，TSDL を使用して SPP.NET にサービス要求する場合に使用します。

オプション

●-s　生成ファイル拡張子　　～〈文字列〉

生成する TSDL の拡張子を指定します。

このオプションを省略した場合，拡張子は「tsdl」となります。

●-o　出力先ディレクトリ　　～〈パス名〉

生成する TSDL を出力するディレクトリを指定します。絶対パスまたは相対パスで指定してください。

このオプションを省略した場合，コマンド実行時のディレクトリに出力されます。

●-i　.NET インタフェース定義ファイル名称　　～〈ファイル名〉

入力元のインタフェースが定義されているソースファイルを指定します。絶対パスまたは相対パスで指定してください。

●-h

このコマンドの使用方法を標準出力に表示します。

このオプションを指定した場合，ほかのオプションおよびコマンド引数は指定できません。

コマンド引数

●インタフェース名称

生成したい TSDL に対応する SPP.NET のインタフェース名称を完全限定名で指定します。インタフェース名称は一つだけ指定できます。

注意事項

- このコマンドの実行時にエラーが発生した場合は，対応するエラーメッセージが標準エラー出力に出力されます。
- このコマンドで生成したソースファイルの内容を変更しないでください。
- 入力元となる.NET インタフェース定義ファイルは，使用する Windows のデフォルトコードページ（日本語版 Windows の場合は 932）で保存してください。Unicode など，ほかのコードページで保存したファイルは，このコマンドの入力元として使用できません。

spp2cstub（クライアントスタブ生成コマンド（サービス定義用））

形式

```
spp2cstub {-t {svr|clt|con}
          〔-l {cs|vb}〕
          〔-s 生成ファイル拡張子〕
          〔-n 名前空間名称〕
          〔-o 出力先ディレクトリ〕
          〔-r スタブクラス名称〕
          〔-R データ型定義名称:カスタムレコードクラス名称
               〔,データ型定義名称:カスタムレコードクラス名称〕…〕
          〔-F {space|null}〕
          〔-I エンコーディング名〕
          〔-O エンコーディング名〕
          〔-e {big|little}〕
          〔-E {big|little}〕
          〔-b〕
          〔-X {normal|dataset}〕
           -i サービス定義ファイル名称
          |-h}
```

機能

指定されたサービス定義ファイル，およびそのサービス定義ファイルが参照するデータ型定義ファイルを基に，クライアントスタブクラスおよびカスタムレコードクラスのソースファイルを生成します。

オプション

●-t {svr | clt | con}

生成するクライアントスタブの種類を指定します。

オプションの指定と生成されるクライアントスタブを次に示します。

オプションの指定	生成されるクライアントスタブ
svr	SPP.NET または SUP.NET（Extension .NET）用
clt	CUP.NET（Client .NET）用
con	CUP.NET（Connector .NET）用

●-l {cs | vb} ～《cs》

生成するクライアントスタブクラスおよびカスタムレコードクラスのソースファイルのプログラム言語を指定します。クライアントスタブクラスとカスタムレコードクラスは同じプログラム言語で生成されます。

このオプションを省略した場合，cs が仮定されます。

オプションの指定と生成されるソースファイルのプログラム言語を次に示します。

オプションの指定	生成されるソースファイルのプログラム言語
cs	C#
vb	Visual Basic

●-s　生成ファイル拡張子　　～〈文字列〉

生成するクライアントスタブクラスおよびカスタムレコードクラスのソースファイルの拡張子を指定します。

このオプションを省略した場合，生成されるソースファイルのプログラム言語によって，拡張子は次のようになります。

生成されるソースファイルのプログラム言語	生成されるソースファイルの拡張子
C#	cs
Visual Basic	vb

●-n　名前空間名称　　～〈文字列〉

生成するクライアントスタブクラスおよびカスタムレコードクラスの名前空間名称を指定します。

このオプションを省略した場合，名前空間なしのクライアントスタブクラスおよびカスタムレコードクラスが生成されます。

●-o　出力先ディレクトリ　　～〈パス名〉

生成するクライアントスタブクラスおよびカスタムレコードクラスのソースファイルを出力するディレクトリを指定します。絶対パスまたは相対パスで指定してください。

このオプションを省略した場合，コマンド実行時のディレクトリに出力されます。なお，ファイル名はクライアントスタブクラスごとに「〈名前空間を含まないスタブクラス名称〉.〈拡張子〉」，カスタムレコードクラスごとに「〈名前空間を含まないカスタムレコードクラス名称〉.〈拡張子〉」で生成されます。

●-r　スタブクラス名称　　～〈文字列〉

生成するクライアントスタブのクラス名称を指定します。

このオプションを省略した場合，クラス名称は「〈サービス定義名称〉Stub」になります。なお，一つのサービス定義ファイルには，一つのサービス定義しか指定できません。

●-R　データ型定義名称:カスタムレコードクラス名称　　～〈文字列〉:〈31 文字以内の識別子〉

データ型定義ファイルで定義しているデータ型定義名称と，それを基にして生成するカスタムレコードのクラス名称を指定します。

このオプションを省略した場合，データ型定義名称がカスタムレコードのクラス名称に使用されます。

複数のデータ型定義からカスタムレコードを生成する場合は，データ型定義名称と生成されるカスタムレコードのクラス名称をコロン（:）で区切って指定します。ただし，データ型定義に存在しないデータ型定

義名称を指定した場合，存在しないデータ型定義名称は無視されます。存在するデータ型定義名称に対してだけカスタムレコードクラス名称の指定が有効になります。

●-F {space | null}　～《space》

入力レコードとなるカスタムレコードを引数として渡す場合，データ型定義で指定した文字列領域の余った領域に埋める文字を指定します。

このオプションを省略した場合，余った領域を半角スペースで埋めます。

space：半角スペースで埋めます。

null：ヌル文字で埋めます。

●-I エンコーディング名　～〈文字列〉

RPC で送信する要求データを，エンコードするときに従うエンコード方式のエンコーディング名を指定します。指定できるエンコーディング名については，.NET Framework のドキュメントを参照してください。

このオプションを省略した場合，プラットフォームのデフォルトエンコーディング名になります。

プラットフォームによってサポートされているエンコード方式は異なります。代表的なエンコーディング名は次のとおりです。

エンコーディング名	エンコード（Windows 上の表示名）	備考
euc-jp	日本語（EUC）	－
iso-8859-1	西ヨーロッパ言語（ISO）	－
shift_jis	日本語（シフト JIS）	MS932
unicodeFFFE	Unicode（Big-Endian）	－
us-ascii	US-ASCII	ISO646
utf-8	Unicode（UTF-8）	－
utf-16	Unicode	Little Endian

（凡例）

－：該当しません。

●-O エンコーディング名　～〈文字列〉

RPC で受信した応答データを，デコードするときに従うエンコード方式のエンコーディング名を指定します。指定できるエンコーディング名については，.NET Framework のドキュメントを参照してください。

このオプションを省略した場合，プラットフォームのデフォルトエンコーディング名になります。

プラットフォームによってサポートされているエンコード方式は異なります。代表的なエンコーディング名については，-I オプションの表を参照してください。

●-e {big | little} ～《big》

RPC で送信する要求データを，指定されたエンディアンに変換します。

このオプションを省略した場合，ビッグエンディアンに変換します。

big：ビッグエンディアンに変換します。

little：リトルエンディアンに変換します。

●-E {big | little} ～《big》

RPC で受信した応答データを，指定されたエンディアンであると仮定して変換します。

このオプションを省略した場合，ビッグエンディアンであると仮定して変換します。

big：ビッグエンディアンであると仮定して変換します。

little：リトルエンディアンであると仮定して変換します。

●-b

生成されたカスタムレコードクラスごとに必要なバッファサイズを標準出力に表示します。

【表示例】

```
CustomRecordClassName : BufferSize[Bytes]
MyAppNS.Record1 : 448
MyAppNS.Record2 : 32
```

●-X {normal | dataset}

生成されるクライアントスタブに，引数および戻り値が XmlDocument クラス（System.Xml.XmlDocument）となるサービスメソッドが追加されます。

また，入力データ用 XML スキーマファイルがサービスごとに出力されます。出力データ型定義名称が DC_NODATA 以外の場合，出力データ用 XML スキーマファイルがサービスごとに出力されます。複数のサービスで同じデータ型定義名称を指定した場合，そのデータ型定義に対応する入力データ用 XML スキーマおよび出力データ用 XML スキーマは，一つだけ出力されます。

-t オプションに con を指定した場合だけ，このオプションの指定が有効になります。-t オプションに con 以外を指定した場合，このオプションの指定は無視されます。

このオプションを省略した場合，生成されるクライアントスタブに，引数および戻り値が XmlDocument クラスとなるサービスメソッドは追加されません。また，入力データ用 XML スキーマファイルおよび出力データ用 XML スキーマファイルは出力されません。

normal：引数および戻り値の XmlDocument オブジェクト（System.Xml.XmlDocument）を，.NET Framework の DataSet オブジェクト（System.Data.DataSet）と連携させない場合に指定します。

normal を指定した場合，出力される入力データ用 XML スキーマファイルおよび出力データ用 XML スキーマファイルは，DataSet オブジェクトで利用できない場合があります。

dataset：引数および戻り値の XmlDocument オブジェクト（System.Xml.XmlDocument）を，.NET Framework の DataSet オブジェクト（System.Data.DataSet）と連携させて利用する場合に指定します。dataset を指定した場合，DataSet オブジェクトで利用できる入力データ用 XML スキーマファイルおよび出力データ用 XML スキーマファイルを出力します。また，その入力データ用 XML スキーマおよび出力データ用 XML スキーマに対応したクライアントスタブを出力します。

●-i　サービス定義ファイル名称　　～〈ファイル名〉

サービス定義が指定されているファイル名称を指定します。絶対パスまたは相対パスで指定してください。

●-h

このコマンドの使用方法を標準出力に表示します。

このオプションを指定した場合，ほかのオプションおよびコマンド引数は指定できません。

注意事項

- このコマンドの実行時にエラーが発生した場合は，対応するエラーメッセージが標準エラー出力に出力されます。
- このコマンドで生成したソースファイルの内容を変更しないでください。
- サービス定義ファイルの#include ディレクティブには，サービス定義ファイルの存在するディレクトリからの相対パスを指定します。#include ディレクティブに相対パスを指定していない場合は，データ型定義ファイルはサービス定義ファイルと同じディレクトリになければなりません。
- 入力元となるサービス定義ファイルおよびデータ型定義ファイルは，使用する Windows のデフォルトコードページ（日本語版 Windows の場合は 932）で保存してください。Unicode など，ほかのコードページで保存したファイルは，このコマンドの入力元として使用できません。

7 クラスリファレンス

この章では，TP1/Extension for .NET Framework（Extension .NET）で利用できるクラスについて説明します。

TP1/Extension for .NET Framework で利用できるクラス

TP1/Extension for .NET Framework で利用できるクラスの一覧を次に示します。

表 7-1　TP1/Extension for .NET Framework で利用できるクラスの一覧

名前空間	クラス名	説明
Hitachi.OpenTP1.Server	Adm	システム運用の管理のメソッドを提供します。
	Jnl	ユーザジャーナルを取得するメソッドを提供します。TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では使えません。
	Lck	任意のユーザファイルを排他制御するメソッドを提供します。TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では使えません。
	Log	UAP からメッセージログを出力するメソッドを提供します。
	Mcf	アプリケーション間でメッセージの送受信をするときに使う，メッセージ送受信機能を使用するためのメソッドを提供します。
	Prf	性能検証用トレース機能を提供するメソッドを提供します。
	Rap	リモート API 機能で，コネクションの確立および解放をユーザが管理する場合に使用するメソッドを提供します。
	RpcBindTable 構造体	サービス要求先を特定する検索キーを格納します。
	Rpc	クライアント/サーバ形態の通信をするときに使う，OpenTP1 の RPC のメソッドを提供します。
	Rts	リアルタイム統計情報を取得する場合に使用するメソッドを提供します。
	SPPBase	SPP.NET 実装の抽象クラスです。
	Tam	TAM ファイルサービス機能を使用するメソッドを提供します。TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では使えません。
	TamKeyTable 構造体	レコードのキー値を格納します。TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では使えません。
	TamStatusTable 構造体	TAM テーブルの情報を格納します。TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では使えません。
	TP1ServerException	クラスライブラリの各メソッドでエラーを検知した場合に発生する例外です。

名前空間	クラス名	説明
	TP1ServerFlags	OpenTP1 の各種フラグを提供します。
	TP1ServerLimits	OpenTP1 で使用する長さの制限などを定義します。
	TP1ServerValues	OpenTP1 の各種値を提供します。
	Trn	OpenTP1 独自のトランザクション制御をするメソッドを提供します。
Hitachi.OpenTP1	IRecord インタフェース	カスタムレコードが実装しなければならないレコード名，およびレコードの簡易説明のプロパティを規定します。
	IntArrayHolder	System.Int32 配列を保持するホルダークラスです。
	IntHolder	System.Int32 値を保持するホルダークラスです。
	LongArrayHolder	System.Int64 配列を保持するホルダークラスです。
	LongHolder	System.Int64 値を保持するホルダークラスです。
	ShortArrayHolder	System.Int16 配列を保持するホルダークラスです。
	ShortHolder	System.Int16 値を保持するホルダークラスです。
	StringArrayHolder	System.String 配列を保持するホルダークラスです。
	StringHolder	System.String を保持するホルダークラスです。
	TP1Error	クラスライブラリの各メソッドで返されるエラーや例外に設定されるエラーの値を定義したクラスです。
	TP1Exception	OpenTP1 のすべての例外の基底クラスです。
	TP1MarshalException	読み込みおよび書き込みできないバイト配列のインデクスが参照された場合，またはデータの内容が不正な場合に出力される例外です。
	TP1RemoteException	SPP.NET で例外が発生したことを SPP.NET，SUP.NET，または CUP.NET に知らせる例外です。
	TP1RpcMethod	サービスメソッドカスタム属性です。
	TP1UserException	SPP.NET のサービスメソッド内でユーザがスローする例外です。
	TP1UserStruct	TP1 ユーザ構造体を利用するには，このクラスを継承する必要があります。
	UByteArrayHolder	System.Byte 配列を保持するホルダークラスです。
	UByteHolder	System.Byte 値を保持するホルダークラスです。

Adm

Adm の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Adm
```

説明

Adm クラスは，システム運用の管理のメソッドを提供します。

メソッドの一覧

名称	説明
CallCommand(System.String, System.Int32&, System.String&, System.String&, System.Int32)	オンライン中にコマンドを入力したときと同様に，UAP からコマンドを渡します。
CallCommand(System.String, System.Int32&, System.String&, System.Int32, System.String&, System.Int32, System.Int32)	オンライン中にコマンドを入力したときと同様に，UAP からコマンドを渡します。
Complete()	SUP.NET の開始処理が終了したことを，OpenTP1 に報告します。
GetStatus()	このメソッドを呼び出したユーザサーバの状態を報告します。

メソッドの詳細

●CallCommand

説明

オンライン中にコマンドを入力したときと同様に，UAP からコマンドを渡します。

このときのプロセスは，コマンドが処理を完了するまで待ち，コマンドの終了ステータスを戻します。コマンドの処理が終了すると，標準出力情報と標準エラー出力情報が返ります。

コマンドを実行する UAP を使う OpenTP1 には，コマンドを格納しているディレクトリをサーチパスに追加してください。

サーチパスを追加する方法を次に示します。どれかの方法を用いてください。

- プロセスサービス定義の prcsvpath オペランドにコマンドのパス名を指定
- prcpath コマンドでサーチパスを追加
- ユーザサービス定義に環境変数を"putenv PATH"と指定

宣言

【C#の場合】

```
public static int CallCommand(
  string command,
  ref int statusCode,
  ref string stdoutMessage,
  ref string stderrMessage,
  int flags
);
```

【Visual Basic の場合】

```
Public Shared Function CallCommand( _
  ByVal command As String, _
  ByRef statusCode As Integer, _
  ByRef stdoutMessage As String, _
  ByRef stderrMessage As String, _
  ByVal flags As Integer _
) As Integer
```

【J#の場合】

```
public static int CallCommand(
  System.String command,
  int statusCode,
  System.String stdoutMessage,
  System.String stderrMessage,
  int flags
);
```

パラメタ

command

実行するコマンドの文字列を設定します。

statusCode

指定したコマンドのリターン値を格納します。

stdoutMessage

指定したコマンドの標準出力に出力された文字列を格納します。

デフォルト取得サイズ（4096）を超えた分は，切り捨てられます。

stderrMessage

指定したコマンドの標準エラー出力に出力された文字列を格納します。

デフォルト取得サイズ（4096）を超えた分は，切り捨てられます。

flags

標準出力または標準エラー出力メッセージのデータを完全には取得できなかった場合の，CallCommand メソッドの動作を設定します。

- TP1ServerFlags.DCADM_DELAY
  実行したコマンドの処理を中断して，例外を発行します。

- TP1ServerFlags.DCNOFLAGS
  取得できた分のデータを格納して，エラーリターンします。

戻り値

メソッドのリターンコードを返します。

リターンコード	説明
0 (DC_OK)	正常終了しました。
-1855(DCADMER_STATNOTZERO)	コマンドの終了コードは 0 以外（コマンドの実行が異常終了）です。 標準出力および標準エラー出力のデータを格納しました。
-1856(DCADMER_MEMORY_OUT)	標準出力のデータが，デフォルト取得サイズを超えました。 デフォルトサイズ分の標準出力のデータを格納しました。
-1857(DCADMER_MEMORY_ERR)	標準エラー出力のデータが，デフォルト取得サイズを超えました。 デフォルトサイズ分の標準エラー出力のデータを格納しました。
-1858(DCADMER_MEMORY_OUTERR)	標準出力のデータと標準エラー出力のデータの両方が，デフォルト取得サイズを超えました。 デフォルトサイズ分の標準出力および標準エラー出力のデータを格納しました。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCADMER_MEMORY_ERR	標準エラー出力のデータが，デフォルトの取得サイズを超えました。
DCADMER_MEMORY_OUT	標準出力のデータが，デフォルトの取得サイズを超えました。

エラーコード	説明
DCADMER_MEMORY_OUTERR	標準出力のデータと標準エラー出力のデータの両方が，デフォルトの取得サイズを超えました。
DCADMER_PARAM	引数に設定した値が間違っています。
DCADMER_PROTO	Rpc クラスの Open メソッドを呼び出していません。
DCADMER_STATNOTZERO	コマンドの終了コードは 0 以外（コマンドの実行が異常終了）です。
DCADMER_SYSTEMCALL	システムコール（close，pipe，dup，または read）の呼び出しに失敗しました。

注意事項

サーチパスに指定したディレクトリ間で，コマンド名が重複しないように注意してください。コマンド名が重複している場合，正しいコマンドが起動されないで別のコマンドが起動されます。

また，コマンド名は，OpenTP1 が提供するコマンド群（%DCDIR%¥bin の下）のコマンド名とも重複しないようにしてください。

なお，「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

●CallCommand

説明

オンライン中にコマンドを入力したときと同様に，UAP からコマンドを渡します。

このときのプロセスは，コマンドが処理を完了するまで待ち，コマンドの終了ステータスを戻します。コマンドの処理が終了すると，標準出力情報と標準エラー出力情報が返ります。

コマンドを実行する UAP を使う OpenTP1 には，コマンドを格納しているディレクトリをサーチパスに追加してください。

サーチパスを追加する方法を次に示します。どれかの方法を用いてください。

- プロセスサービス定義の prcsvpath オペランドにコマンドのパス名を指定
- prcpath コマンドでサーチパスを追加
- ユーザサービス定義に環境変数を"putenv PATH"と指定

宣言

【C#の場合】

```
public static int CallCommand(
  string command,
  ref int statusCode,
  ref string stdoutMessage,
  int stdoutMessage_len,
  ref string stderrMessage,
  int stderrMessage_len,
  int flags
);
```

【Visual Basic の場合】

```
Public Shared Function CallCommand( _
  ByVal command As String, _
  ByRef statusCode As Integer, _
  ByRef stdoutMessage As String, _
  ByVal stdoutMessage_len As Integer, _
  ByRef stderrMessage As String, _
  ByVal stderrMessage_len As Integer, _
  ByVal flags As Integer _
) As Integer
```

【J#の場合】

```
public static int CallCommand(
  System.String command,
  int statusCode,
  System.String stdoutMessage,
  int stdoutMessage_len,
  System.String stderrMessage,
  int stderrMessage_len,
  int flags
);
```

パラメタ

command

実行するコマンドの文字列を設定します。

statusCode

指定したコマンドのリターン値を格納します。

stdoutMessage

指定したコマンドの標準出力に出力された文字列を格納します。

stdoutMessage_len パラメタで指定したサイズを超えた分は，切り捨てられます。

stdoutMessage_len

取得する標準出力のサイズを設定します。

stderrMessage

指定したコマンドの標準エラー出力に出力された文字列を格納します。

stderrMessage_len パラメタで指定したサイズを超えた分は，切り捨てられます。

stderrMessage_len

取得する標準エラー出力のサイズを指定します。

flags

標準出力または標準エラー出力メッセージのデータを完全に取得できなかった場合の，CallCommand メソッドの動作を設定します。

- TP1ServerFlags.DCADM_DELAY

  実行したコマンドの処理を中断して，例外を発行します。

- TP1ServerFlags.DCNOFLAGS
  取得できた分のデータを格納して，エラーリターンします。

戻り値

メソッドのリターンコードを返します。

リターンコード	説明
0 (DC_OK)	正常終了しました。
-1855(DCADMER_STATNOTZERO)	コマンドの終了コードは 0 以外（コマンドの実行が異常終了）です。 標準出力および標準エラー出力のデータを格納しました。
-1856(DCADMER_MEMORY_OUT)	標準出力のデータが，指定取得サイズを超えました。 指定されたサイズ分の標準出力のデータを格納しました。
-1857(DCADMER_MEMORY_ERR)	標準エラー出力のデータが，指定取得サイズを超えました。 指定されたサイズ分の標準エラー出力のデータを格納しました。
-1858(DCADMER_MEMORY_OUTERR)	標準出力のデータと標準エラー出力のデータの両方が，指定取得サイズを超えました。 指定されたサイズ分の標準出力のデータおよび標準エラー出力のデータを格納しました。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCADMER_MEMORY_ERR	標準エラー出力のデータが，デフォルトの取得サイズを超えました。
DCADMER_MEMORY_OUT	標準出力のデータが，デフォルトの取得サイズを超えました。

エラーコード	説明
DCADMER_MEMORY_OUTERR	標準出力のデータと標準エラー出力のデータの両方が，デフォルトの取得サイズを超えました。
DCADMER_PARAM	引数に設定した値が間違っています。
DCADMER_PROTO	Rpc クラスの Open メソッドを呼び出していません。
DCADMER_STATNOTZERO	コマンドの終了コードは 0 以外（コマンドの実行が異常終了）です。
DCADMER_SYSTEMCALL	システムコール（close，pipe，dup，または read）の呼び出しに失敗しました。

注意事項

サーチパスに指定したディレクトリ間で，コマンド名が重複しないように注意してください。コマンド名が重複している場合，正しいコマンドが起動されないで別のコマンドが起動されます。

また，コマンド名は，OpenTP1 が提供するコマンド群（$DCDIR/bin の下）のコマンド名とも重複しないようにしてください。

なお，「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

●Complete

説明

SUP.NET の開始処理が終了したことを，OpenTP1 に報告します。

Complete メソッドが正常に終了したことで，SUP.NET の起動は完了します。

SPP.NET では，Complete メソッドを呼び出す必要はありません。

オフラインの業務をする SUP.NET からは Complete メソッドは呼び出せません。

宣言

【C#の場合】

```
public static void Complete(
);
```

【Visual Basic の場合】

```
Public Shared Sub Complete( _
)
```

【J#の場合】

```
public static void Complete(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCADMER_COMM	プロセス間通信でエラーが発生しました。
DCADMER_PARAM	引数に設定した値が間違っています。
DCADMER_PROTO	ユーザサーバが正常開始中，または再開始中ではありません。 Rpc クラスの Open メソッドを呼び出していません。
DCADMER_STS_IO	ステータス情報の入出力エラーが発生しました。

●GetStatus

説明

このメソッドを呼び出したユーザサーバの状態を報告します。

ユーザサーバの状態はリターン値で報告されます。

宣言

【C#の場合】

```
public static int GetStatus(
);
```

【Visual Basic の場合】

```
Public Shared Function GetStatus( _
) As Integer
```

【J#の場合】

```
public static int GetStatus(
);
```

パラメタ

なし

戻り値

ユーザサーバの状態を返します。

- TP1ServerValues.DCADM_STAT_START_NORMAL
  ユーザサーバは正常開始中です。
- TP1ServerValues.DCADM_STAT_START_RECOVER
  ユーザサーバは再開始中です。
- TP1ServerValues.DCADM_STAT_ONLINE
  ユーザサーバはオンライン中です。
- TP1ServerValues.DCADM_STAT_STOP
  ユーザサーバは終了中です。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCADMER_COMM	プロセス間通信でエラーが発生しました。
DCADMER_PARAM	引数に設定した値が間違っています。
DCADMER_PROTO	ユーザサーバが正常開始中，または再開始中ではありません。 Rpc クラスの Open メソッドを呼び出していません。

エラーコード	説明
DCADMER_STS_IO	ステータス情報の入出力エラーが発生しました。

IntArrayHolder

IntArrayHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.IntArrayHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

IntArrayHolder クラスは，System.Int32 配列を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public int[] Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As Integer()
```

【J#の場合】

```
public int[] get_Value();
public void set_Value(int[]);
```

例外

なし

IntHolder

IntHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.IntHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

IntHolder クラスは，System.Int32 値を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public int Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As Integer
```

【J#の場合】

```
public int get_Value();
public void set_Value(int);
```

例外

なし

IRecord インタフェース

IRecord インタフェースの概要

名前空間

Hitachi.OpenTP1

継承関係

```
Hitachi.OpenTP1.IRecord
```

説明

IRecord インタフェースは，カスタムレコードが実装しなければならないレコード名，およびレコードの簡易説明のプロパティを規定します。

メソッドの一覧

名称	説明
GetRecordName()	レコード名の取得を行います。
GetRecordShortDescription()	レコードの簡易説明の取得を行います。
SetRecordName(System.String)	レコード名を設定します。
SetRecordShortDescription(System.String)	レコードの簡易説明を設定します。

メソッドの詳細

●GetRecordName

説明

レコード名の取得を行います。

宣言

【C#の場合】

```
public abstract string GetRecordName(
);
```

【Visual Basic の場合】

```
Public MustOverride Function GetRecordName( _
) As String
```

【J#の場合】

```
public abstract System.String GetRecordName(
);
```

パラメタ

なし

戻り値

なし

例外

なし

注意事項

設定しているレコード名を返します。

●GetRecordShortDescription

説明

レコードの簡易説明の取得を行います。

宣言

【C#の場合】

```
public abstract string GetRecordShortDescription(
);
```

【Visual Basic の場合】

```
Public MustOverride Function GetRecordShortDescription( _
) As String
```

【J#の場合】

```
public abstract System.String GetRecordShortDescription(
);
```

パラメタ

なし

戻り値

なし

例外

なし

注意事項

レコードに設定されている簡易説明を返します。

●SetRecordName

説明

レコード名を設定します。

宣言

【C#の場合】

```
public abstract void SetRecordName(
  string recordName
);
```

【Visual Basic の場合】

```
Public MustOverride Sub SetRecordName( _
  ByVal recordName As String _
)
```

【J#の場合】

```
public abstract void SetRecordName(
  System.String recordName
);
```

パラメタ

recordName

新たに設定するレコード名称。

戻り値

なし

例外

なし

●SetRecordShortDescription

説明

レコードの簡易説明を設定します。

宣言

【C#の場合】

```
public abstract void SetRecordShortDescription(
  string recordShortDescription
);
```

【Visual Basic の場合】

```
Public MustOverride Sub SetRecordShortDescription( _
  ByVal recordShortDescription As String _
)
```

【J#の場合】

```
public abstract void SetRecordShortDescription(
  System.String recordShortDescription
);
```

パラメタ

recordShortDescription

新たに設定するレコードの簡易説明。

戻り値

なし

例外

なし

Jnl 【TP1/Server Base】

Jnl の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Jnl
```

説明

Jnl クラスは，ユーザジャーナルを取得するメソッドを提供します。

Jnl クラスのメソッドは，TP1/Server Base の UAP でだけ使用できます。TP1/LiNK の UAP では，Jnl クラスのメソッドは使用できません。

メソッドの一覧

名称	説明
PutUJ(System.Byte[], System.Int32, System.Int32)	ユーザジャーナルを取得します。
PutUJ(System.Byte[], System.Int32, System.Int32, System.Int32)	ユーザジャーナルを取得します。UAP の履歴情報であるユーザジャーナル（UJ）をシステムジャーナルファイル（System_jnl_file）に取得します。

メソッドの詳細

●PutUJ

説明

ユーザジャーナルを取得します。

宣言

【C#の場合】

```
public static void PutUJ(
  byte[] data,
  int dataSize,
  int ujCode
);
```

【Visual Basic の場合】

```
Public Shared Sub PutUJ( _
  ByVal data() As Byte, _
  ByVal dataSize As Integer, _
```

```
  ByVal ujCode As Integer _
)
```

【J#の場合】

```
public static void PutUJ(
  System.Byte[] data,
  int dataSize,
  int ujCode
);
```

注意事項

この PutUJ メソッドは，PutUJ（byte[] data，int dataSize，int ujCode，int flags）メソッドの flags に TP1ServerFlags.DCNOFLAGS を指定した場合と同じ動作をします。

●PutUJ

説明

ユーザジャーナルを取得します。UAP の履歴情報であるユーザジャーナル（UJ）をシステムジャーナルファイル（system_jnl_file）に取得します。PutUJ メソッド 1 回の呼び出しで取得する UJ の単位を UJ レコードといいます。

PutUJ メソッドを呼び出しても，すぐにはシステムジャーナルファイルに出力されません。ジャーナルバッファに空きがなくなったとき，またはトランザクション処理が正常終了した同期点で，システムジャーナルファイルに UJ レコードが出力されます。

PutUJ メソッドは，SUP.NET の場合，Rpc クラスの Open メソッドの呼び出し後から Rpc クラスの Close メソッドの呼び出しまでの間で呼び出せます。SPP.NET の場合，どこからでも呼び出せます。出力済みの UJ レコードは，PutUJ メソッドを呼び出したトランザクションで障害が発生しても，ロールバック（部分回復）で無効にできません。PutUJ メソッドを呼び出したトランザクションをロールバックしても，UJ レコードはシステムジャーナルファイルに出力されます。

宣言

【C#の場合】

```
public static void PutUJ(
  byte[] data,
  int dataSize,
  int ujCode
  int flags
);
```

【Visual Basic の場合】

```
Public Shared Sub PutUJ( _
  ByVal data() As Byte, _
  ByVal dataSize As Integer, _
  ByVal ujCode As Integer _
  ByVal flagsb As Integer _
)
```

【J#の場合】

```
public static void PutUJ(
  System.Byte[] data,
  int dataSize,
  int ujCode
  int flags
);
```

パラメタ

data

取得する UAP の履歴情報を設定します。UAP の履歴情報として有効になるデータは，dataSize に設定した長さです。

dateSize

取得する UAP の履歴情報の長さを設定します。設定できる長さは，1 から（取得先のシステムジャーナルファイルサービス定義の jnl_max_datasize オペランドの値-8）までです。

ujCode

0～255 の値で設定します。

flags

UJ コードを取得する時点で，システムジャーナルファイルに UJ レコードを出力するかどうかを，次に示す形式で設定します。

- TP1ServerFlags.DCJNL_FLUSH

  UJ レコードを取得する時点で，システムジャーナルファイルに UJ レコードを出力します。トランザクション内で UJ レコードが取得されている場合は，この設定は無視されます。

- TP1ServerFlags.DCNOFLAGS

  UJ レコードを取得する時点では，システムジャーナルファイルに UJ レコードを出力しません。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合だけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCJNLER_PARAM	パラメタの形式が間違っています。
DCJNLER_SHORT	ユーザジャーナルの長さ（dataSize の値）に，0 以下のデータ長を設定しています。
DCJNLER_LONG	ユーザジャーナルの長さ（dataSize の値）に，設定できる範囲以上の値を設定しています。
DCJNLER_PROTO	UAP を開始する準備ができていません。

注意事項

トランザクション外の UJ レコードは，ジャーナルバッファに空きがなくなったとき，またはほかのアプリケーションのトランザクションが正常終了した同期点（コミットした時点）で，システムジャーナルファイルに出力されます。トランザクションが発生しないアプリケーションで UJ レコードを取得する場合は，flags に TP1ServerFlags.DCJNL_FLUSH を設定した PutUJ メソッドを適切なタイミングで呼び出してください。

Lck 【TP1/Server Base】

Lck の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Lck
```

説明

Lck クラスは，任意のユーザファイルを排他制御するメソッドを提供します。

Lck クラスのメソッドは，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，Lck クラスのメソッドは使えません。

メソッドの一覧

名称	説明
Get(System.String, System.Int32, System.Int32)	UAP で使う資源の排他を指定します。
ReleaseAll()	Get メソッドで指定した資源の排他をすべて解除します。
ReleaseByName(System.String)	Get メソッドで指定した資源の排他を，資源名称を指定して解除します。

メソッドの詳細

●Get

説明

UAP で使う資源の排他を指定します。排他の管理は，OpenTP1 のトランザクションマネジャで管理するグローバルトランザクション単位で処理されます。

ここで指定した排他は，排他を解除するメソッド（Lck.ReleaseAll メソッド，Lck.ReleaseByName メソッド）を呼び出すか，Get メソッドを呼び出したグローバルトランザクションの同期点取得後に解除されます。

宣言

【C#の場合】

```
public static void Get(
    string name,
    int lockMode,
    int flags
);
```

【Visual Basic の場合】

```
Public Shared Sub Get( _
    ByVal name As String, _
    ByVal lockMode As Integer, _
    ByVal flags As Integer _
)
```

【J#の場合】

```
public static void Get(
    System.String name,
    int lockMode,
    int flags
);
```

パラメタ

name

排他する資源の名称を，16 文字の半角英数字文字列で設定します。ここで設定した資源名称を基に，ロックサービスで排他を管理します。16 文字未満の場合は，設定した値がそのまま資源名称とみなされます。16 文字を超えた値を設定した場合は，16 文字までが資源名称とみなされ，超えた部分は切り捨てられます。

ロックサービスでは，文字列の内容についてはチェックしません。論理的に正しい名称を設定してください。資源名称に半角英数字以外の値を使った場合は，デッドロック情報，タイムアウト情報，および lckls コマンドの表示が乱れたり，資源名称が途中で切り捨てられる場合があります。

lockMode

排他制御モードを設定します。排他制御モードは次のどちらか一方を設定してください。重複して設定できません。

- TP1ServerFlags.DCLCK_PR
  資源を参照します。ほかの UAP には参照だけを許可します。
- TP1ServerFlags.DCLCK_EX
  資源を更新します。ほかの UAP には参照も更新も禁止します。

flags

資源の排他に関するフラグを設定します。設定できる値を次に示します。

- TP1ServerFlags.DCLCK_WAIT
  ほかの UAP と資源を競合した場合に，資源の解放待ちにします。このフラグが設定されてない場合に競合したときは，例外応答します。
- TP1ServerFlags.DCLCK_TEST
  資源が使えるかどうかをテストするときに設定します。このフラグを設定したときは，Get メソッドが正常終了しても，name に設定した資源は確保されていないので注意してください。
- TP1ServerFlags.DCNOFLAGS
  フラグを設定しません。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCLCKER_PARAM	引数に設定した値が間違っています。
DCLCKER_WAIT	ほかの UAP が，name に名称を設定した資源を使っています。
DCLCKER_DLOCK	デッドロックが起こりました。
DCLCKER_TIMOUT	OpenTP1 のロックサービス定義で指定した待ち時間でタイムアウトが発生したため，資源を確保できませんでした。
DCLCKER_MEMORY	排他制御用のテーブルが不足しています。
DCLCKER_OUTOFTRN	トランザクション処理でない UAP から指定しています。
DCLCKER_VERSION	OpenTP1 のライブラリとロックサービスのバージョンが一致していません。

●ReleaseAll

説明

Get メソッドで指定した資源の排他をすべて解除します。同期点取得前に排他を解除するときに，ReleaseAll メソッドを呼び出します。

排他をしたグローバルトランザクションが終了したときに，OpenTP1 のロックサービスによって自動的に排他は解除されます。このときは，UAP で排他を解除する必要はありません。

宣言

【C#の場合】

```
public static void ReleaseAll(
);
```

【Visual Basic の場合】

```
Public Shared Sub ReleaseAll( _
)
```

【J#の場合】

```
public static void ReleaseAll(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCLCKER_OUTOFTRN	トランザクション処理でない UAP から ReleaseAll メソッドを呼び出しています。
DCLCKER_NOTHING	このメソッドを呼び出したトランザクションでは，資源を確保していません。

エラーコード	説明
DCLCKER_VERSION	OpenTP1 のライブラリとロックサービスのバージョンが一致していません。

●ReleaseByName

説明

Get メソッドで指定した資源の排他を，資源名称を指定して解除します。同期点取得前に排他を解除するときに，ReleaseByName メソッドを呼び出します。

排他をしたグローバルトランザクションが終了したときに，ロックサービスによって自動的に排他は解除されます。このときは，UAP で排他を解除する必要はありません。

宣言

【C#の場合】

```
public static void ReleaseByName(
    string name
);
```

【Visual Basic の場合】

```
Public Shared Sub ReleaseByName( _
    ByVal name As String _
)
```

【J#の場合】

```
public static void ReleaseByName(
    System.String name
);
```

パラメタ

name

排他の指定を解除する資源名称を設定します。資源名称は Get メソッドで設定した名称と同じ値を設定してください。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCLCKER_PARAM	引数に設定した値が間違っています。
DCLCKER_OUTOFTRN	トランザクション処理でない UAP から ReleaseByName メソッドを呼び出しています。
DCLCKER_NOTHING	解除を指定した資源名称に該当する資源がありません。
DCLCKER_VERSION	OpenTP1 のライブラリとロックサービスのバージョンが一致していません。

Log

Log の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Log
```

説明

Log クラスは，UAP からメッセージログを出力するメソッドを提供します。

メソッドの一覧

名称	説明
Print(System.String, System.String, System.String, System.Int32)	指定した文字列に，OpenTP1 で行ヘッダ，OpenTP1 識別子，日時，要求元ノード名，要求元プログラム ID，およびメッセージ ID を付けて，メッセージログファイルに出力します。

メソッドの詳細

●Print

説明

指定した文字列に，OpenTP1 で行ヘッダ，OpenTP1 識別子，日時，要求元ノード名，要求元プログラム ID，およびメッセージ ID を付けて，メッセージログファイルに出力します。

OpenTP1 では，Print メソッドで使うメッセージ ID 用に，05000 から 06999 までの範囲の番号を割り当てています。UAP から出力するメッセージ ID の番号には，05000 から 06999 までの範囲の値を付けてください。

障害が起こって UAP からメッセージログが出力できない場合でも，Print メソッドが正常終了することがあります。そのため，メッセージログが抜ける場合がありますが，メッセージログの抜けはメッセージログに付けるメッセージログ通番で確認できます。

ひとつのプロセスから複数回 Print メソッドを呼び出した場合は，メッセージログファイルへの出力順序は保証されます。しかし，複数のプロセスから別々に Print メソッドを呼び出した場合は，呼び出した順にメッセージログファイルに出力されない場合があります。

通信障害，およびログサービス未起動のエラーが起こった場合は，UAP から出力したメッセージを，その UAP プロセス上で編集して，標準エラー出力に出力します。このとき，メッセージの終わりには，次に示すエラー要因を示すコードを付けます。

- E1

ログサービスが起動していないため，メッセージログに出力ができなかったメッセージを示します。

- E2

  通信障害のため，メッセージログファイルに出力できなかったメッセージを示します。

E1，E2 以外のエラーを検出した場合，OpenTP1 はエラーの原因を示すメッセージログに Print メソッドに指定したメッセージ ID の番号を付けて，標準エラー出力に出力します。

宣言

【C#の場合】

```
public static void Print(
  string messageID,
  string programID,
  string outputMessage,
  int displayColor
);
```

【Visual Basic の場合】

```
Public Shared Sub Print( _
  ByVal messageID As String, _
  ByVal programID As String, _
  ByVal outputMessage As String, _
  ByVal displayColor As Integer _
)
```

【J#の場合】

```
public static void Print(
  System.String messageID,
  System.String programID,
  System.String outputMessage,
  int displayColor
);
```

パラメタ

messageID

メッセージログごとに付けられる識別子（メッセージ ID）を設定します。

「KFCAn1n2n3n4n5-x」の形式（11 文字）で設定します。UAP から出力する通番（n1n2n3n4n5 の部分）には，05000 から 06999 までの数値を設定します。

programID

Print メソッドを呼び出した UAP を識別する値（要求元プログラム ID）を，ユーザが任意で設定します。英数字 2 文字で設定します。

outputMessage

メッセージログファイルにメッセージログとして出力したい任意の文字列を設定します。最大 222 文字で設定します。

displayColor

Print メソッドで設定したメッセージログを NETM の操作支援端末に出力する場合の，表示色に対応した数値を設定します。次のどれかを設定します。

1：緑

2：赤

3：白

4：青

5：紫

6：水色

7：黄色

上記以外の数値を設定した場合は，緑が仮定されます。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコード名称が出力されます。

エラーコード	説明
DCLOGER_COMM	プロセス間通信でエラーが発生しました。
DCLOGER_DEFFILE	システムの環境設定に誤りがあります。
DCLOGER_HEADER	ログサービスがメッセージログに付ける情報を取得したときに，障害が起こりました。
DCLOGER_MEMORY	メモリが不足しました。

エラーコード	説明
DCLOGER_NOT_UP	ログサービスが起動していません。
DCLOGER_PARAM_ARGS	引数に設定した値が間違っています。
DCLOGER_PROTO	Rpc クラスの Open メソッドを呼び出していません。

注意事項

ログ出力量が多い場合は，Print メソッドのリターンが遅くなります。

例えば，障害発生時にメッセージ出力量が著しく多くなると，トランザクション処理時間が延びます。スローダウンの要因になりますので，注意してください。

LongArrayHolder

LongArrayHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.LongArrayHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

LongArrayHolder クラスは，System.Int64 配列を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public long[] Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As Long()
```

【J#の場合】

```
public long[] get_Value();
public void set_Value(long[]);
```

例外

なし

LongHolder

LongHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.LongHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

LongHolder クラスは，System.Int64 値を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public long Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As Long
```

【J#の場合】

```
public long get_Value();
public void set_Value(long);
```

例外

なし

Mcf

Mcf の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Mcf
```

説明

Mcf クラスは，アプリケーション間でメッセージの送受信をするときに使う，メッセージ送受信機能を使用するためのメソッドを提供します。

メソッドの一覧

名称	説明
Send(System.Int32, System.String, System.Byte[], System.Int32)	相手システムへ一方送信メッセージを送信します。
SendReceive(System.Int32, System.String, System.Byte[], System.Int32, System.Byte[], System.Int32, System.Int32, System.Int32, System.Int32)	同期型でメッセージを送信したあと，同期型でメッセージを受信します。

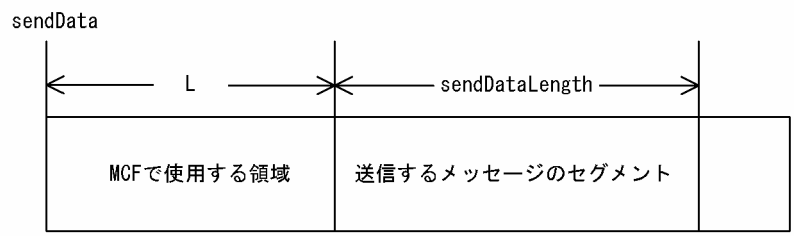
メソッドの詳細

●Send

説明

相手システムへ一方送信メッセージを送信します。一方送信メッセージは，一つのセグメントで構成されます。

セグメントを送信する領域の形式を次に示します。L はヘッダ領域の長さです。バッファ形式 1 の場合は 8 バイト，バッファ形式 2 の場合は 4 バイトになります。



宣言

【C#の場合】

```
public static void Send(
  int action,
  string terminalName,
  byte[] sendData,
  int sendDataLength
);
```

【Visual Basic の場合】

```
Public Shared Function Send( _
  ByVal action As Integer, _
  ByVal terminalName As String, _
  ByVal sendData() As Byte, _
  ByVal sendDataLength As Integer _
)
```

【J#の場合】

```
public static void Send(
  int action,
  System.String terminalName,
  ubyte[] sendData,
  int sendDataLength
);
```

パラメタ

action

単一セグメントを送信すること，優先か一般か，出力通番を付けるかどうか，および使用するバッファ形式を，次の形式で設定します。

```
DCMCFEMI 〔| {DCMCFNORM|DCMCFPRIO} 〕
 〔| {DCMCFSEQ|DCMCFNSEQ} 〕 〔| {DCMCFBUF1|DCMCFBUF2} 〕
```

- TP1ServerFlags.DCMCFEMI

  メッセージが単一セグメントであることを設定します。

- TP1ServerFlags.DCMCFNORM

  一般の一方送信メッセージとして送信する場合に設定します。

- TP1ServerFlags.DCMCFPRIO

  優先の一方送信メッセージとして送信する場合に設定します。

- TP1ServerFlags.DCMCFSEQ

  出力通番が必要な場合に設定します。

- TP1ServerFlags.DCMCFNSEQ

  出力通番が必要ない場合に設定します。

- TP1ServerFlags.DCMCFBUF1

バッファ形式 1 を使用する場合に設定します。

- TP1ServerFlags.DCMCFBUF2

  バッファ形式 2 を使用する場合に設定します。

terminalName

出力先の論理端末名称を設定します。論理端末名称の長さは最大 8 バイトです。論理端末名称の後ろにはヌル文字を付けてください。

sendData

送信するセグメントの内容を設定した領域を設定します。一つのセグメントで 32000 バイトまで送信できます。

sendDataLength

送信するセグメントの長さを設定します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）

  例外が発生する原因となった引数名が出力されます。

- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCMCFER_INVALID_ARGS	引数に設定した値が間違っています。
DCMCFER_PROTO	プロトコル不正です。 メッセージ送受信機能の使用が有効になっていません。
DCMCFRTN_71002	メッセージキューへの出力処理中に障害が発生しました。

エラーコード	説明
	メッセージキューが閉塞されています。 メッセージキューが割り当てられていません。 セグメント長に 32000 バイトを超える値を設定しています。 MCF が終了処理中のため，メッセージの送信を受け付けられません。
DCMCFRTN_71003	メッセージキューが満杯です。
DCMCFRTN_71004	メッセージを格納するバッファをメモリ上に確保できませんでした。
DCMCFRTN_71108	メッセージを送信しようとしましたが，送信先の管理テーブルが確保できませんでした。 プロセスのローカルメモリが不足しています。
DCMCFRTN_72000	トランザクションでない SPP.NET の処理から，Send メソッドを呼び出しています。
DCMCFRTN_72001	Send メソッドを呼び出せない論理端末を設定しています。 terminalName に設定した論理端末名称が間違っています。
DCMCFRTN_72016	action に設定したメッセージ種別（DCMCFNORM または DCMCFPRIO）の値が間違っています。 action に設定した値が間違っています。 引数に設定した値が間違っています。
DCMCFRTN_72017	action に設定した出力通番の要否（DCMCFSEQ または DCMCFNSEQ）の値が間違っています。
DCMCFRTN_72026	action に設定したセグメント種別（DCMCFEMI）の値が間違っています。
DCMCFRTN_72041	送信するメッセージの内容がありません。 長さが 0 バイトのセグメントを送信しています。

●SendReceive

説明

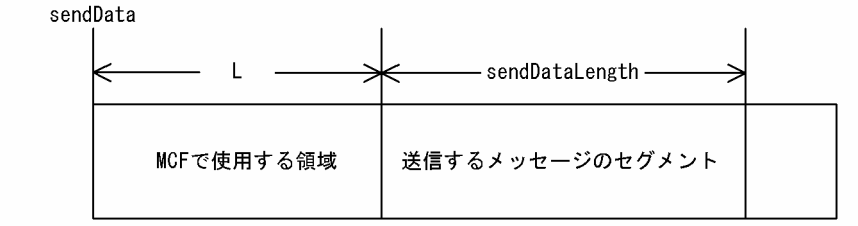
同期型でメッセージを送信したあと，同期型でメッセージを受信します。

SendReceive メソッドでは，相手システムへ送るメッセージのセグメントを送信できます。

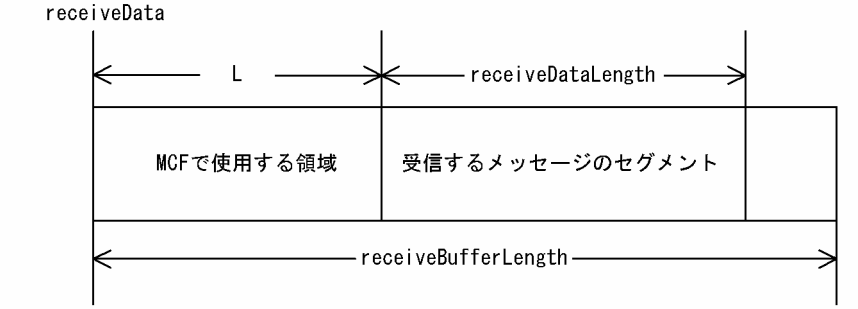
メッセージのセグメントを送信すると，SendReceive メソッドは相手システムからの応答を待ちます。応答が届くと，そのメッセージのセグメントを受信します。

セグメントを送信する領域と受信する領域の形式をそれぞれ示します。L はヘッダ領域の長さです。バッファ形式 1 の場合は 8 バイト，バッファ形式 2 の場合は 4 バイトになります。

●セグメントを送信する領域



●セグメントを受信する領域



宣言

【C#の場合】

```
public static void SendReceive(
  int action,
  string terminalName,
  byte[] sendData,
  int sendDataLength,
  byte[] receiveData,
  ref int receiveDataLength,
  int receiveBufferLength,
  ref int time,
  int watchTime
);
```

【Visual Basic の場合】

```
Public Shared Function SendReceive( _
  ByVal action As Integer, _
  ByVal terminalName As String, _
  ByVal sendData() As Byte, _
  ByVal sendDataLength As Integer, _
  ByVal receiveData() As Byte, _
  ByRef receiveDataLength As Integer, _
  ByVal receiveBufferLength As Integer, _
  ByRef time As Integer, _
  ByVal watchTime As Integer _
)
```

【J#の場合】

```
public static void SendReceive(
  int action,
  System.String terminalName,
  ubyte[] sendData,
  int sendDataLength,
```

```
  ubyte[] receiveData,
  int receiveDataLength,
  int receiveBufferLength,
  int time,
  int watchTime
);
```

パラメタ

action

メッセージの最終セグメントを送信するかどうか，および使用するバッファ形式を，次の形式で設定します。

```
DCMCFEMI 〔| {DCMCFBUF1|DCMCFBUF2} 〕
```

- TP1ServerFlags.DCMCFEMI

  最終セグメントを送信する場合に設定します。

  メッセージが単一セグメントの場合も，DCMCFEMI を設定します。メッセージの送信の終了を連絡するために，最後は必ずこの値を設定してください。この値を設定して SendReceive メソッドを呼び出すと，論理端末からの応答を待ちます。

- TP1ServerFlags.DCMCFBUF1

  バッファ形式 1 を使用する場合に設定します。

- TP1ServerFlags.DCMCFBUF2

  バッファ形式 2 を使用する場合に設定します。

terminalName

メッセージを出力して応答を入力する論理端末名称を設定します。論理端末名称の長さは最大 8 バイトです。論理端末名称の後ろにはヌル文字を付けてください。

sendData

送信するセグメントの内容を設定した領域を設定します。一つのセグメントで 32000 バイトまで送信できます。

メッセージの送信の終了を連絡する場合で，セグメントの内容がないときも，必ず設定してください。

sendDataLength

メッセージの送信の終了を連絡する場合で，セグメントの内容がないときは，0 を設定してください。

receiveData

セグメントを受信する領域を設定します。

単一セグメントまたは最終セグメントを送信する SendReceive メソッドが終了すると，受信したメッセージの先頭セグメントが返されます。

処理終了後，receiveData には OpenTP1 から値が返されます。

receiveDataLength

受信したメッセージの先頭セグメントの長さが返されます。

receiveBufferLength

セグメントを受信する領域の長さを設定します。

time

メッセージを受信した時刻が，1970 年 1 月 1 日 0 時 0 分 0 秒からの通算の秒数で返されます。

watchTime

SendReceive メソッドを呼び出してから終了するまでの最大時間を設定します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCMCFER_INVALID_ARGS	引数に設定した値が間違っています。
DCMCFER_PROTO	プロトコル不正です。 メッセージ送受信機能の使用が有効になっていません。
DCMCFRTN_71002	メッセージキューへの出力処理中に障害が発生しました。 メッセージキューが閉塞されています。 メッセージキューが割り当てられていません。 セグメント長に 32000 バイトを超える値を設定しています。 MCF が終了処理中のため，メッセージの送信を受け付けられません。
DCMCFRTN_71003	メッセージキューが満杯です。
DCMCFRTN_71004	メッセージを格納するバッファをメモリ上に確保できませんでした。

エラーコード	説明
DCMCFRTN_71108	メッセージを送信しようとしましたが，送信先の管理テーブルが確保できませんでした。 プロセスのローカルメモリが不足しています。
DCMCFRTN_72001	SendReceive メソッドを呼び出せない論理端末を設定しています。 terminalName に設定した論理端末名称が間違っています。
DCMCFRTN_72012	MCF バッファグループ定義のバッファ長が不足しました。
DCMCFRTN_72013	受信領域の長さを超えるセグメントを受信しました。受信領域の長さを超えた部分は切り捨てられました。
DCMCFRTN_72016	action に設定したメッセージ種別（DCMCFNORM または DCMCFPRIO）の値が間違っています。 action に設定した値が間違っています。 引数に設定した値が間違っています。
DCMCFRTN_72026	action に設定したセグメント種別（DCMCFEMI）の値が間違っています。
DCMCFRTN_72036	セグメントを受信する領域の長さが不足しています。バッファ形式 1 の場合は 9 バイト以上，バッファ形式 2 の場合は 5 バイト以上の領域を確保してください。
DCMCFRTN_72041	送信するメッセージの内容がありません。 長さが 0 バイトのセグメントを送信しています。
DCMCFRTN_72073	非同期メッセージを送信処理中です。
DCMCFRTN_73001	出力先の論理端末で障害が発生しました。
DCMCFRTN_73002	MCF 通信サービスで障害が発生しました。
DCMCFRTN_73003	メッセージ受信が仕掛り中です。
DCMCFRTN_73005	watchTime に設定した時間が経過しましたが，論理端末からの応答がありません。
DCMCFRTN_73010	メッセージの読み込み時に障害が発生しました。 メッセージの編集エラーが発生しました。
DCMCFRTN_73015	出力先の論理端末は，ほかの UAP で仕掛り中です。
DCMCFRTN_73018	watchTime に設定した値が間違っています。
DCMCFRTN_73019	メッセージ送信完了監視タイマのタイムアウトが発生しました。
DCMCFRTN_73020	出力先の論理端末は停止しています。

Prf

Prf の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Prf
```

説明

Prf クラスは，性能検証用トレース機能を提供するメソッドを提供します。

メソッドの一覧

名称	説明
GetTraceNum()	GetTraceNum メソッドを呼び出す前に取得した，最新の性能検証用トレース（prf トレース）のプロセス内取得通番を，メソッドの呼び出し元に通知します。
PutUTrace(System.Int32, System.Int32, System.Byte[])	ユーザ固有の性能検証用トレース（prf トレース）を取得します。

メソッドの詳細

●GetTraceNum

説明

GetTraceNum メソッドを呼び出す前に取得した，最新の性能検証用トレース（prf トレース）のプロセス内取得通番を，メソッドの呼び出し元に通知します。

GetTraceNum メソッドを呼び出したプロセスで一度も性能検証用トレースを取得していない場合，プロセス内取得番号は 0 となります。

宣言

【C#の場合】

```
public static int GetTraceNum(
);
```

【Visual Basic の場合】

```
Public Shared Function GetTraceNum( _
) As Integer
```

【J#の場合】

```
public static int GetTraceNum(
);
```

パラメタ

なし

戻り値

- 0
  一度も性能検証用トレースを取得していません。
- 正の整数
  最新の性能検証用トレースのプロセス内取得通番。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCPRFER_PARAM	引数に指定した値が間違っています。

●PutUTrace

説明

ユーザ固有の性能検証用トレース（prf トレース）を取得します。

宣言

【C#の場合】

```
public static void PutUTrace(
  int eventId,
  int dataLength,
```

```
  byte[] traceBuffer
);
```

【Visual Basic の場合】

```
Public Shared Sub PutUTrace( _
  ByVal eventId As Integer, _
  ByVal dataLength As Integer, _
  ByVal traceBuffer() As Byte _
)
```

【J#の場合】

```
public static void PutUTrace(
  int eventId,
  int dataLength,
  ubyte[] traceBuffer
);
```

パラメタ

eventId

取得するイベントのイベント ID を設定します。

使用できるイベント ID の範囲は 0x0001～0x0040 です。

dataLength

取得するトレースデータのデータ長を設定します。

設定できるデータ長は 4 バイト以上 256 バイト以下です。また，このデータ長は 4 バイトの倍数でなければなりません。

traceBuffer

取得するトレースデータの設定されているバイト配列を設定します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

例外が発生したメソッド名が出力されます。

- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCPRFER_PARAM	引数に指定した値に誤りがあります。

注意事項

PutUTrace メソッドが正常終了してもトレースが正しく取得されているとは限りません。これは，トレースの取得処理で，排他を使用しないため複数のプロセスから同時に取得要求が出された場合，データを紛失してしまうことがあるためです。

Rap

Rap の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Rap
```

説明

Rap クラスは，リモート API 機能で，コネクションの確立および解放をユーザが管理する場合に使用するメソッドを提供します。

メソッドの一覧

名称	説明
Connect(System.String, System.Int32)	rap リスナーと rap クライアントとの間にコネクションを確立します。
Disconnect(System.Int32)	rap リスナーと rap クライアントとの間に確立されているコネクションを解放します。

メソッドの詳細

●Connect

説明

rap リスナーと rap クライアントとの間にコネクションを確立します。

コネクションを確立する rap リスナーは targetHostName パラメタに指定したホスト上に targetPortNo パラメタに指定したポート番号で起動されている rap リスナーです。

宣言

【C#の場合】

```
public static int Connect(
  string targetHostName,
  int targetPortNo
);
```

【Visual Basic の場合】

```
Public Shared Function Connect( _
  ByVal targetHostName As String, _
  ByVal targetPortNo As Integer _
) As Integer
```

【J#の場合】

```
public static int Connect(
  System.String targetHostName,
  int targetPortNo
);
```

パラメタ

targetHostName

rap リスナーが起動されている OpenTP1 ノードのホスト名を設定します。

targetPortNo

rap リスナーの使用しているポート番号を設定します。

戻り値

メソッドが正常終了した場合に，サービス ID を返却します。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRAPER_ALREADY_CONNECT	すでに rap リスナーとのコネクションは確立しています。
DCRAPER_MAX_CONNECTION	一つのプロセスから Rap クラスの Connect メソッドを呼び出せる上限値を超えました。
DCRAPER_MAX_CONNECTION_SV	rap リスナーの管理する rap クライアントとのコネクション要求受け付け可能最大数を超えました。
DCRAPER_NETDOWN	rap リスナーとの通信でネットワーク障害が発生しました。

エラーコード	説明
DCRAPER_NOCONTINUE	続行できない障害が発生しました。障害の要因として次のことが考えられます。 • 予期しないメッセージを受信しました。 • 予期しない相手からのメッセージを受信しました。
DCRAPER_NOHOSTNAME	ホスト名称が解決できません。
DCRAPER_NOMEMORY	メモリが不足しました。
DCRAPER_NOMEMORY_SV	rap リスナーまたは rap サーバでメモリ不足が発生しました。
DCRAPER_NOSERVICE	rap リスナーは開始処理中，または停止処理中です。
DCRAPER_NOSOCKET	ソケット不足が発生しました。
DCRAPER_PANIC_SV	rap リスナーでシステム障害が発生しました。
DCRAPER_PARAM	引数が間違っています。
DCRAPER_PROTO	プロトコル不正です。次の要因が考えられます。 • Rpc クラスの Open メソッドが呼び出されていません。
DCRAPER_SHUTDOWN	rap リスナーは停止中です。
DCRAPER_SYSCALL	システムコールで予期しないエラーが発生しました。
DCRAPER_TIMEDOUT	rap リスナーとの通信でタイムアウトが発生しました。
DCRAPER_TIMEOUT_SV	rap リスナーサービス定義の rap_watch_time オペランドに指定したメッセージ送受信最大監視時間内にコネクションが確立できませんでした。
DCRAPER_UNKNOWN_NODE	接続されていないネットワーク上の rap リスナーに対してコネクションを確立しようとしています。

●Disconnect

説明

rap リスナーと rap クライアントとの間に確立されているコネクションを解放します。

宣言

【C#の場合】

```
public static void Disconnect(
  int serviceID
);
```

【Visual Basic の場合】

```
Public Shared Sub Disconnect( _
  ByVal serviceID As Integer _
)
```

【J#の場合】

```
public static void Disconnect(
  int serviceID
);
```

パラメタ

serviceID

Connect メソッドで受け取ったサービス ID を設定します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRAPER_NETDOWN	rap リスナーとの通信でネットワーク障害が発生しました。
DCRAPER_NOCONTINUE	続行できない障害が発生しました。障害の要因として次のことが考えられます。 • 予期しないメッセージを受信しました。 • 予期しない相手からのメッセージを受信しました。
DCRAPER_NOMEMORY	メモリが不足しました。
DCRAPER_PARAM	引数が間違っています。
DCRAPER_PROTO	プロトコル不正です。次の要因が考えられます。 • Rpc クラスの Open メソッドが呼び出されていません。
DCRAPER_SHUTDOWN	rap リスナーは停止中です。
DCRAPER_SYSCALL	システムコールで予期しないエラーが発生しました。

エラーコード	説明
DCRAPER_TIMEDOUT	rap リスナーとの通信でタイムアウトが発生しました。

注意事項

例外発生時のエラーコードが Hitachi.OpenTP1.TP1Error.DCRAPER_PARAM/ Hitachi.OpenTP1.TP1Error.DCRAPER_PROTO 以外の値で，Disconnect メソッドが異常終了した場合，rap リスナーとのコネクションは解放されています。

UAP トレースに取得されるエラー要因コードは次のとおりです。

0：エラー無し

1：RPC.Open メソッドが呼び出されていません。

3：ユーザサービス定義の rpc_rap_auto_connect オペランドに Y を指定している環境で Disconnect メソッドが呼び出されました。

Rpc

Rpc の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Rpc
```

説明

Rpc クラスは，クライアント／サーバ形態の通信をするときに使う，OpenTP1 のリモートプロシジャコールのメソッドを提供します。

メソッドの一覧

名称	説明
Call(System.String, System.String, System.Byte[], System.Int32, System.Byte[], System.Int32&, System.Int32)	SPP.NET または SPP にサービスを要求します。
CallTo(Hitachi.OpenTP1.Server.RpcBindTable, System.String, System.String, System.Byte[], System.Int32, System.Byte[], System.Int32&, System.Int32)	特定の SPP.NET または SPP にサービスを要求します。
Close()	OpenTP1 の各種メソッドを使うための環境をクローズします。
DiscardFurtherReplies()	非同期応答型 RPC（Call メソッドの flags パラメタに TP1ServerFlags.DCRPC_DOMAIN を設定）で，まだ返ってきていない応答を，これ以上受信しないことを示すメソッドです。
DiscardSpecificReply(System.Int32)	非同期応答型 RPC（Call メソッドの flags パラメタに TP1ServerFlags.DCRPC_NOWAIT を設定）で，まだ返ってきていない応答を，これ以上受信しないことを示すメソッドです。
GetCallersAddress()	クライアント UAP のプロセスが稼働するノードアドレスを，サーバ UAP で取得します。
GetErrorDescriptor()	非同期応答を特定しない PollAnyReplies メソッドで例外が発生した直後に呼び出すことで，エラーが発生した非同期応答型 RPC 要求に対応する記述子を取得します。
GetGatewayAddress()	アプリケーションゲートウェイ型ファイアウォールなどの，ゲートウェイを介してクライアント UAP からのサービス要求を受信したとき，サーバ UAP でゲートウェイのノードアドレスを取得します。
GetServicePriolity()	SetServicePriolity メソッドで設定した，サービス要求のスケジュールプライオリティを参照します。

名称	説明
GetTimeout()	現在のサービス要求の応答待ち時間を参照します。
Open()	OpenTP1 の各種メソッドを使う準備をします。
PollAnyReplies(System.Int32, System.Int32, System.Int32)	非同期応答型 RPC（Call メソッドの flags パラメタに TP1ServerFlags.DCRPC_NOWAIT を設定）でサービス要求した結果を受信します。
SetBindTable(Hitachi.OpenTP1.Server.RpcBindTable&, System.String, System.String, System.Int32)	指定されたサービス要求先ノードのノード識別子（nodeID パラメタの値），または指定されたサービス要求先ノードのホスト名（hostName パラメタの値）を RpcBindTable 構造体に設定し，CallTo メソッドの第一引数を作成します。
SetDirectSchedule(Hitachi.OpenTP1.Server.RpcBindTable&, System.String, System.Int32)	指定されたサービス要求先ノードのホスト名（hostName パラメタの値）と，指定されたスケジュールサービスのポート番号（schedulePortNo パラメタの値）を RpcBindTable 構造体に設定し，CallTo メソッドの第一引数を作成します。
SetServicePriolity(System.Int32)	サービス要求のプライオリティを設定します。
SetServiceRetry()	実行中のサービスメソッドの処理をリトライします。
SetTimeout(System.Int32)	サービス要求の応答待ち時間を変更します。

メソッドの詳細

●Call

説明

SPP.NET または SPP にサービスを要求します。Call メソッドを使うときは，要求するサービスがどのノードにあるかを意識する必要はありません。

サービスを要求するときには，サービスグループ名とサービス名を Call メソッドの引数に設定します。この名称に該当するサービスメソッドにサービスが要求されます。

Call メソッドを呼び出す UAP は，トランザクションとして実行していても，していなくてもかまいません。トランザクションとして実行している処理から Call メソッドでサービスを要求するときは，要求するサービスの処理はトランザクションブランチとして稼働します。Call メソッドを使う場合，サーバ UAP があるノードの OpenTP1 が稼働していることが前提です。

Call メソッドを実行して応答を待っている間にシグナルを受信しても，メソッドは例外応答しません。

Call メソッドの詳しい説明については，このメソッドの「注意事項」を参照してください。「注意事項」では次に示す項目について説明しています。

1. Call メソッドの引数について
2. Call メソッドがエラーになる場合
3. Call メソッドがエラーになるタイミング
4. Call メソッドがエラーになったときに再実行する指定
5. サービス要求に優先度を付ける場合

6. エラーコード Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE_GROUP と Hitachi.OpenTP1.TP1Error.DCRPCER_NET_DOWN との違い
7. エラーコード Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_TERMINATED を返させる指定
8. エラーコードと同期点処理の関係
9. サービスを要求するときの注意
10. ドメイン修飾をしてサービスを要求する場合

宣言

【C#の場合】

```
public static int Call(
  string serviceGroupName,
  string serviceName,
  byte[] inputBuffer,
  int inputBuffer_len,
  byte[] outputBuffer,
  ref int outputBuffer_len,
  int flags
);
```

【Visual Basic の場合】

```
Public Shared Function Call( _
  ByVal serviceGroupName As String, _
  ByVal serviceName As String, _
  ByVal inputBuffer() As Byte, _
  ByVal inputBuffer_len As Integer, _
  ByVal outputBuffer() As Byte, _
  ByRef outputBuffer_len As Integer, _
  ByVal flags As Integer _
) As Integer
```

【J#の場合】

```
public static int Call(
  System.String serviceGroupName,
  System.String serviceName,
  ubyte[] inputBuffer,
  int inputBuffer_len,
  ubyte[] outputBuffer,
  int outputBuffer_len,
  int flags
);
```

パラメタ

serviceGroupName

SPP.NET または SPP のサービスグループ名を，31 バイト以内の文字列で設定します。

ドメイン修飾をしてサービスを要求するときは，サービスグループ名の後ろに@（アットマーク）と DNS のドメイン名を付けて設定してください。

serviceName

SPP.NET または SPP のサービス名を，31 バイト以内の文字列で設定します。

inputBuffer

サービスの入力パラメタを設定します。

inputBuffer_len

サービスの入力パラメタ長を設定します。

outputBuffer

サービスメソッドから返ってくる応答を格納します。

outputBuffer_len

サービスの応答の長さを設定します。

非応答型 RPC の場合も，サービスの応答の長さを指定する必要があります。非応答型 RPC の場合，サービスの応答の長さには 0 を設定します。

flags

RPC の形態とオプションを，次に示す形式で設定します。

```
{TP1ServerFlags.DCNOFLAGS|
TP1ServerFlags.DCRPC_NOWAIT|
TP1ServerFlags.DCRPC_NOREPLY|
TP1ServerFlags.DCRPC_CHAINED}
 [|TP1ServerFlags.DCRPC_TPNOTRAN]
 [|TP1ServerFlags.DCRPC_DOMAIN]
```

オプション	説明
TP1ServerFlags.DCNOFLAGS	同期応答型 RPC
TP1ServerFlags.DCRPC_NOWAIT	非同期応答型 RPC
TP1ServerFlags.DCRPC_NOREPLY	非応答型 RPC
TP1ServerFlags.DCRPC_CHAINED	連鎖 RPC
TP1ServerFlags.DCRPC_TPNOTRAN	トランザクション処理からのサービス要求で，要求先の処理をトランザクションにしない場合に設定します。 TP1ServerFlags.DCRPC_TPNOTRAN オプションを設定すると，トランザクションの処理からのサービス要求でも，サービス関数およびサービスメソッドの処理はトランザクションになりません。
TP1ServerFlags.DCRPC_DOMAIN	サービスグループ名をドメイン修飾した場合に指定します。 ドメイン修飾をした RPC はトランザクションブランチにできません。 そのため，トランザクションの処理から Call メソッドを使う場合は，必ず TP1ServerFlags.DCRPC_TPNOTRAN オプションと一緒に指定してください。

TP1ServerFlags.DCRPC_TPNOTRAN オプションと TP1ServerFlags.DCRPC_DOMAIN オプションは，RPC の形態に付けて設定します。

(例 1)

同期応答型 RPC でトランザクションにしないサービス要求をする場合，flags パラメタには「TP1ServerFlags.DCNOFLAGS|TP1ServerFlags.DCRPC_TPNOTRAN」と設定します。

(例 2)

トランザクションの処理から，同期応答型 RPC でドメイン修飾をしたサービス要求をする場合，flags パラメタには「TP1ServerFlags.DCNOFLAGS|TP1ServerFlags.DCRPC_TPNOTRAN|TP1ServerFlags.DCRPC_DOMAIN」と設定します。

戻り値

非同期応答型 RPC の場合は，正の整数は記述子です。それ以外の場合は，意味を持ちません。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_INVALID_ARGS	引数に設定した値が間違っています。
DCRPCER_INVALID_REPLY	サービスメソッドが OpenTP1 に返した応答の長さが，1 から 1048576 までの範囲にありません。
DCRPCER_MESSAGE_TOO_BIG	inputBuffer_len パラメタに設定した入力パラメタ長が，最大値を超えています。
DCRPCER_NET_DOWN	ネットワークに障害が起こりました。
DCRPCER_NOT_TRN_EXTEND	トランザクション処理の連鎖 RPC を使ったあとで，flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN を設定した Call メソッドでサービスを要求しています。
DCRPCER_NO_BUFS	メモリが不足しました。

エラーコード	説明
	または，サーバのメッセージ格納領域に十分な空きがないため，サービス要求を受け付けられませんでした。
DCRPCER_NO_BUFS_AT_SERVER	設定したサービスで，メモリが不足しました。
DCRPCER_NO_BUFS_RB	メモリが不足しました。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_NO_PORT	ドメイン修飾をしてサービスを要求しましたが，ドメイン代表スケジュールサービスのポート番号が見つかりません。
DCRPCER_NO_SUCH_DOMAIN	ドメイン修飾をしたサービスグループ名の，ドメイン名が間違っています。
DCRPCER_NO_SUCH_SERVICE	serviceName パラメタに設定したサービス名は，定義されていません。
DCRPCER_NO_SUCH_SERVICE_GROUP	serviceGroupName パラメタに設定したサービスグループは，定義されていません。
DCRPCER_OLTF_INITIALIZING	サービスを要求されたノードにある OpenTP1 は，開始処理中です。
DCRPCER_OLTF_NOT_UP	serviceName パラメタに設定したサービスがあるノードの OpenTP1 が稼働していません。異常終了，停止中，終了処理中，または通信障害が起こったことが考えられます。
DCRPCER_PROTO	Open メソッドを呼び出していません。
DCRPCER_REPLY_TOO_BIG	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。
DCRPCER_REPLY_TOO_BIG_RB	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_SECCHK	サービスを要求された SPP.NET または SPP は，OpenTP1 のセキュリティ機能で保護されています。Call メソッドでサービスを要求した UAP には，SPP.NET または SPP へのアクセス権限がありません。
DCRPCER_SERVER_BUSY	サービスを要求されたソケット受信型サーバが，サービス要求を受け取れません。
DCRPCER_SERVICE_CLOSED	serviceName パラメタに設定したサービス名のサービスがあるサービスグループは，閉塞しています。
DCRPCER_SERVICE_NOT_UP	serviceName パラメタに設定したサービスの UAP プロセスが，稼働していません。 サービス要求の応答待ち時間に 0 を指定した場合に，サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。
DCRPCER_SERVICE_TERMINATED	サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。このエラーコードは，ユーザサービス定義の rpc_extend_function オペランドに"00000001"を指定したクライアント UAP の場合にだけ返されます。rpc_extend_function オペランドに"00000000"を指定，またはオペランドを省略した場合は，このエラーコードは返らないで，DCRPCER_TIMED_OUT エラーコードまたは DCRPCER_SERVICE_NOT_UP エラーコードが返されます。
DCRPCER_SERVICE_TERMINATING	serviceName パラメタに設定したサービスは，終了処理中です。

エラーコード	説明
DCRPCER_SYSERR	システムエラーが発生しました。
DCRPCER_SYSERR_AT_SERVER	設定したサービスで，システムエラーが発生しました。
DCRPCER_SYSERR_AT_SERVER_RB	設定したサービスで，システムエラーが発生しました。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_SYSERR_RB	システムエラーが発生しました。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_TESTMODE	オンラインテスタを使っている環境で，テストモードの UAP からテストモードでない SPP へサービスを要求しています。または，テストモードでない UAP からテストモードの SPP へサービスを要求しています。
DCRPCER_TIMED_OUT	Call メソッドの処理が時間切れ（タイムアウト）になりました。 サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。
DCRPCER_TRNCHK	ノード間負荷バランス機能およびノード間負荷バランス拡張機能の環境で，複数の SPP.NET または SPP のトランザクション属性が一致していません。または，負荷を分散する先のノードにある OpenTP1 のバージョンが，クライアントの OpenTP1 のバージョンよりも低いため，ノード間負荷バランス機能およびノード間負荷バランス拡張機能を実行できません。 このエラーコードは，ノード間負荷バランス機能およびノード間負荷バランス拡張機能を使っている SPP.NET または SPP にサービスを要求した場合にだけ返されます。
DCRPCER_TRNCHK_EXTEND	同時に起動できるトランザクションブランチの数を超えたため，トランザクションブランチを開始できません。 一つのトランザクションブランチから開始できる子トランザクションブランチの最大数を超えたため，トランザクションブランチを開始できません。 トランザクション内でドメイン修飾をしたサービス要求で，flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN オプションを設定していません。 リソースマネジャ（RM）でエラーが発生したため，トランザクションブランチを開始できません。

注意事項

システム共通定義の all_node オペランドで指定したドメイン以外の OpenTP1 システムにトランザクショナル RPC を行う場合，自ドメインおよび他ドメイン内のすべての OpenTP1 システムのノード識別子（システム共通定義の node_id オペランド）は一意にする必要があります。また，すべての OpenTP1 システムのバージョンは 03-02 以降にする必要があります。これらの条件を満たしていない場合，トランザクションが正しく回復できなくなることがあります。

(1)Call メソッドの引数について

Call メソッドの引数について説明します。

・サーバ UAP に渡す値

サービスを要求するときは，サービスメソッドから返ってくる応答の領域を outputBuffer パラメタに確保しておきます。クライアント UAP では，Call メソッドに次の値を設定しておきます。

- 入力パラメタ（inputBuffer パラメタ）
- 入力パラメタ長（inputBuffer_len パラメタ）
- 応答の長さ（outputBuffer_len パラメタ）

入力パラメタ，入力パラメタ長，および応答の長さは，クライアント UAP の Call メソッドで設定した値が，そのままサービスメソッドに渡されます。

文字コードや数字の表記形式を変えたい場合は，クライアント UAP，または要求されたサービスメソッドの処理で変換してください。応答を返さないサービスメソッドのサービスを要求するときは，応答の長さを設定しても無視されます。

入力パラメタ長と応答の長さの最大値は，クラスリファレンスの TP1ServerLimits の DCRPC_MAX_MESSAGE_SIZE で宣言しています。最大値を確認する場合は，クラスリファレンスの内容を参照してください。

・サーバ UAP から戻ってくる値

サービスメソッドの処理が終了して応答が戻ってくると，次の値を参照できます。

- サービスメソッドの応答（outputBuffer パラメタ）
- サービスメソッドの応答の長さ（outputBuffer_len パラメタ）

outputBuffer_len パラメタの値はサービスメソッドから実際に返ってきた応答の長さです。
outputBuffer パラメタと outputBuffer_len パラメタを参照できる場合を次に示します。

RPC の形態	outputBuffer パラメタと outputBuffer_len パラメタを参照できる場合
同期応答型 RPC，または連鎖 RPC	Call メソッドが応答を返したあと。
非同期応答型 RPC	PollAnyReplies メソッドが該当する応答を受け取って応答を返したあと。 outputBuffer_len パラメタは参照できません。
非応答型 RPC	outputBuffer パラメタと outputBuffer_len パラメタは参照できません。 Call メソッド，または PollAnyReplies メソッドが例外応答した場合には，outputBuffer パラメタと outputBuffer_len パラメタは参照できません。 返ってきた応答が，クライアント UAP で確保した応答の領域（outputBuffer パラメタの値）よりも大きい場合は，Hitachi.OpenTP1.TP1Error.DCRPCER_REPLY_TOO_BIG で例外応答します。

・flags パラメタに設定する値

flags パラメタに設定した値と Call メソッドの実行結果について説明します。

RPC の形態	Call メソッドの実行結果
同期応答型 RPC （flags パラメタに TP1ServerFlags.DCNOFLAGS を設定）	Call メソッドは，応答が返るか，通信先でエラーが起こるまで応答を返しません。

RPC の形態	Call メソッドの実行結果
非同期応答型 RPC (flags パラメタに TP1ServerFlags.DCRPC_NOWAIT を設定)	Call メソッドは，すぐに応答を返します。ただし，応答を参照できるのは，PollAnyReplies メソッドで非同期に応答を受信したあとです。 応答（outputBuffer パラメタの値）の領域は，次に示す方法で非同期応答型 RPC を終了するまで，解放しないでください。 • PollAnyReplies メソッドで応答を受け取る。 • DiscardFurtherReplies メソッドで応答の受信を拒否する。 • トランザクションの処理からサービスを要求した場合で，コミット，またはロールバックする。 トランザクションの処理で非同期応答型 RPC を使う場合，同期点処理（コミット，またはロールバック）前に PollAnyReplies メソッドで応答を受け取ってください。同期点処理後には，PollAnyReplies メソッドで応答は受信できません。PollAnyReplies メソッドが受信する応答を特定する場合は，Call メソッドが応答を返したときに返された正の整数（記述子）を，PollAnyReplies メソッドの引数に設定します。受信する応答を特定する場合は，Call メソッドのリターン値を保持しておいてください。 なお，非トランザクションの処理で，同期点処理後に応答を受け取りたい場合は，システムサービス定義の rpc_extend_function オペランドで該当するオプションを指定する必要があります。
非応答型 RPC (flags パラメタに TP1ServerFlags.DCRPC_NOREPLY を設定)	Call メソッドは，サービスメソッドの処理が終わるのを待たないで，すぐに応答を返します。サービスメソッドは応答を返さないサービスとみなされます。そのため，サービスメソッドが実行されたかどうかは，サービスを要求した UAP からはわかりません。TP1ServerFlags.DCRPC_NOREPLY を設定した場合は，応答（outputBuffer パラメタの値）と応答の長さ（outputBuffer_len パラメタの値）の値は参照できません。
連鎖 RPC (flags パラメタに TP1ServerFlags.DCRPC_CHAINED を設定)	Call メソッドは，応答が返るか，通信先でエラーが起こるまで応答を返しません。連鎖 RPC で同じサービスグループに属するサービスを複数回要求する場合は，最初の要求時と同じプロセスで実行できます。 連鎖 RPC を使う場合には，次に示す制限があります。 • 2 回目以降の Call メソッドでは，ユーザサーバおよびサービスの閉塞を検出できません。 • 2 回目以降の Call メソッドでサービスメソッドの処理で異常が起こった場合には，ユーザサーバ全体が閉塞します。サービス単位では閉塞しません。

(2)Call メソッドがエラーになる場合

Call メソッドの例外応答する理由について説明します。

サーバ UAP があるノードの OpenTP1 が稼働していない場合サービスを要求される OpenTP1 が稼働していない場合は，Call メソッドは次に示すエラーコードのどれかで例外応答します。

- Hitachi.OpenTP1.TP1Error.DCRPCER_NET_DOWN
- Hitachi.OpenTP1.TP1Error.DCRPCER_OLTF_INITIALIZING
- Hitachi.OpenTP1.TP1Error.DCRPCER_OLTF_NOT_UP
- Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_NOT_UP

・サーバ UAP が稼働していない場合

サーバ UAP がマルチサーバの場合，異常終了中，または部分回復処理中であっても，サービス要求は OpenTP1 で新しく起動されたプロセスで実行されます。ただし，次に示す場合は，Call メソッドは例外応答します。

- 閉塞している SPP.NET または SPP へはサービスを要求できません。サービスグループが閉塞されている場合は，Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_CLOSED で例外応答します。
- ユーザサーバの停止コマンド（dcsvstop コマンド）または OpenTP1 の停止コマンド（dcstop コマンド）で，SPP.NET または SPP が終了処理中もしくは終了している場合は，次に示すどれかで例外応答します。

  ・Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE_GROUP

  ・Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_CLOSED

  ・Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_TERMINATING

  上記のうち，どのエラーコードが返るかは，Call メソッドを呼び出したタイミングで決まります。
- OpenTP1 が開始処理中の場合は，Hitachi.OpenTP1.TP1Error.DCRPCER_OLTF_INITIALIZING で例外応答します。この場合は，サーバ UAP または OpenTP1 が起動完了したあとに正常にサービスを要求できることがあります。OpenTP1 は，メッセージ ID KFCA01809-I のメッセージログが出力されると起動を完了するので，このメッセージが表示されてから再びサービスを要求してください。

・ノード間負荷バランス機能およびノード間負荷バランス拡張機能の環境でサービスを要求した場合

ノード間負荷バランス機能およびノード間負荷バランス拡張機能の環境では，該当するサービスのスケジュールが閉塞していると，OpenTP1 が自動的にほかのノードにサービス要求を転送します。ただし，次に示す条件の場合，Call メソッドは Hitachi.OpenTP1.TP1Error.DCRPCER_TRNCHK で例外応答します。

- トランザクション処理の場合，転送しようとした先のノードにあるサービスのトランザクション属性が，閉塞していたサービスと一致していないとき。
- 転送しようとした先のノードにある OpenTP1 のバージョンが，サービスを要求した OpenTP1 のノードのバージョンよりも古いとき。

上記の例外応答が起こった場合は，次に示す処置をしてください。

- ノード間負荷バランス機能およびノード間負荷バランス拡張機能を構成する複数の SPP.NET または SPP のトランザクション属性を一致させてください。
- ノード間負荷バランス機能およびノード間負荷バランス拡張機能を構成する複数の OpenTP1 のバージョンを一致させてください。

・ソケット受信型サーバへサービスを要求する場合

ソケット受信型サーバでは，ユーザサービス定義の max_socket_msg オペランドおよび max_socket_msglen オペランドの指定で，データの輻輳制御をしています。そのため，定義した値を超えた場合，サービス要求を受信できないことがあります。

このとき Call メソッドは，Hitachi.OpenTP1.TP1Error.DCRPCER_SERVER_BUSY で例外応答します。この値が返った場合，クライアント UAP は，適当な時間を置いてから再実行すると，サービスを要求できることがあります。

・連鎖 RPC を使った場合

トランザクションとして処理している連鎖 RPC を使っている UAP から，同じサーバ UAP にトランザクションでない Call メソッドを呼び出すと，
Hitachi.OpenTP1.TP1Error.DCRPCER_NOT_TRN_EXTEND で例外応答します。

・オンラインテスタを使っている場合

オンラインテスタを使っている場合に，テストモードの UAP とテストモードでない UAP 間で Call メソッドを呼び出すと，Hitachi.OpenTP1.TP1Error.DCRPCER_TESTMODE で例外応答します。

・セキュリティ機能を使っている場合

Call メソッドを呼び出したときに，目的のサービスがセキュリティ機能で保護されていて，Call メソッドを呼び出したクライアント UAP に SPP.NET または SPP へのアクセス権がない場合は，
Hitachi.OpenTP1.TP1Error.DCRPCER_SECCHK で例外応答します。

(3)Call メソッドがエラーになるタイミング

サービスを要求された SPP.NET または SPP が異常終了した場合，クライアント UAP でエラーが返るタイミングについて説明します。

RPC の形態	エラーが返るタイミング
同期応答型 RPC または連鎖 RPC (flags パラメタに TP1ServerFlags.DCNOFLAGS または TP1ServerFlags.DCRPC_CHAINED を設定)	サービスを実行する SPP.NET または SPP が処理の終わる前に異常終了すると，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT で例外応答します。 クライアント UAP のユーザサービス定義の watch_time オペランドに，応答を受信するまで待ち続ける指定をしている場合は，
Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_NOT_UP で例外応答します。
非同期応答型 RPC (flags パラメタに TP1ServerFlags.DCRPC_NOWAIT を設定)	サービスを実行する SPP.NET または SPP が処理の終わる前に異常終了すると，上記のエラーコードが PollAnyReplies メソッドに例外応答します。
非応答型 RPC (flags パラメタに TP1ServerFlags.DCRPC_NOREPLY を設定)	サーバ UAP の異常終了を，クライアント UAP では検知できません。

・クライアント UAP の時間監視でエラーになる場合

次に示す場合には，クライアント UAP のユーザサービス定義の watch_time オペランドに指定した時間が経過したあとで，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT で例外応答します。

- SPP.NET または SPP があるノードの OpenTP1 全体が異常終了した場合
- サービス要求のデータをサーバ UAP が受信する前，またはサーバ UAP の処理が完了してからクライアント UAP が結果を受信する前に障害が起こった場合

(4)Call メソッドがエラーになったときに再実行する指定

開始処理中，または系切り替え中などでサービス要求先の OpenTP1 が起動していない場合でも，Call メソッドを通信エラーとしないで，OpenTP1 でサービス要求を再実行させられます。

サービス要求を再実行させる場合は，システム共通定義の rpc_retry オペランドに Y を指定してください。サービス要求の再実行回数および再実行間隔は，システム共通定義の rpc_retry_count オペランドと rpc_retry_interval オペランドで指定します。システム共通定義で指定した再実行回数を超えた場合は，Call メソッドは次に示すどれかで例外応答します。

- Hitachi.OpenTP1.TP1Error.DCRPCER_NET_DOWN
- Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE_GROUP
- Hitachi.OpenTP1.TP1Error.DCRPCER_OLTF_INITIALIZING
- Hitachi.OpenTP1.TP1Error.DCRPCER_OLTF_NOT_UP
- Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_NOT_UP

(5)サービス要求に優先度を付ける場合

サービス要求のスケジュールプライオリティは，Call メソッドを呼び出す直前に，SetServicePriolity メソッドを呼び出して設定します。

プライオリティを設定しない場合は，スケジュールサービスの省略時の解釈で，サービス要求の優先度が決まります。

(6)エラーコード Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE_GROUP と Hitachi.OpenTP1.TP1Error.DCRPCER_NET_DOWN の違い

上記のエラーコードは，該当するサービスグループ名のユーザサーバが見つからなかった場合に返されます。

- Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE_GROUP
  システム共通定義の all_node オペランドに指定した，すべてのノードを探して見つからなかったことを示します。
- Hitachi.OpenTP1.TP1Error.DCRPCER_NET_DOWN
  探している途中で，all_node オペランドに指定した一つ以上のノードとの通信で障害が起こったことを示します。これは，該当する OpenTP1 システムがなかった場合も含みます。

(7)エラーコード Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_TERMINATED を返させる指定

サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了したことを Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT または Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_NOT_UP 以外のエラーコードで判別したい場合には，ユーザサービス定義の rpc_extend_function オペランドに"00000001"を指定します。

この指定をすると，上記の例外発生時に Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_TERMINATED が返されるようになります。rpc_extend_function オペランドに"00000000"を指定するか，またはオペランドを省略した場合は，Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_TERMINATED は返らないで，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT または Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_NOT_UP が返されます。

(8)エラーコードと同期点処理の関係

Call メソッドの応答内容と同期点処理（コミット，ロールバック）の関係について説明します。ここで説明する内容は，サービス要求がトランザクション処理になる場合に該当します。トランザクションでないサービス要求（flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN を設定した場合も含む）には該当しません。

・Call メソッドが例外応答してもコミットとなる場合

サービスを要求されたサービスメソッドの異常終了，ノードの障害，ネットワーク障害などの場合でも，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT が返されることがあります。クライアント UAP がトランザクション処理でない場合は，
Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT が返っても，要求したサービスがあるサービスメソッドは正常に終了していて，データベースへの更新などが実行されていることもあります。

・ロールバック処理が必要な例外応答値

トランザクション処理から呼び出した Call メソッドが例外応答した場合，エラーコードによっては，必ずトランザクションがロールバック（サーバ UAP が rollback_only 状態）になります。この場合，コミットのメソッド，またはロールバックのメソッドのどちらを使っても，必ずロールバックになります。必ずロールバックになる Call メソッドのエラーコードを次に示します。

- Hitachi.OpenTP1.TP1Error.DCRPCER_INVALID_REPLY
- Hitachi.OpenTP1.TP1Error.DCRPCER_NO_BUFS_AT_SERVER
- Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE
- Hitachi.OpenTP1.TP1Error.DCRPCER_REPLY_TOO_BIG_RB

(9)サービスを要求するときの注意

- Call メソッドに設定するサービスグループ名およびサービス名は，サーバ UAP の環境設定で指定しておいてください。誤ったサービスグループ名またはサービス名を Call メソッドに設定してサービスを要求すると，
  Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE_GROUP または
  Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE で例外応答します。
  応答を返さないサービスメソッドの場合は，
  Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE では返しません。
- サーバ UAP は，クライアント UAP とは別のプロセスで実行します。
  そのため，通常のメソッドの呼び出しや手続き呼び出しとは，次の点が異なります。
  ・OS がクライアント UAP のプロセスに与えている属性は，サーバ UAP へは引き継がれません（例えば，環境変数，スケジュールの優先順位（nice 値）など）。
  ・クライアント UAP のノードで設定した OpenTP1 の環境設定は，サーバ UAP の OpenTP1 へは引き継がれません（例えば，トランザクション属性の指定有無，トランザクションブランチの限界経過時間，スケジュールの優先順位など）。
- 入力パラメタ（inputBuffer パラメタ）およびサービスメソッドの応答（outputBuffer パラメタ）に，同じバッファ領域を設定しないでください。

- flags パラメタに TP1ServerFlags.DCRPC_NOREPLY を設定した場合には，次に示すエラーコードは返りません。

  ・起こらないエラー

  Hitachi.OpenTP1.TP1Error.DCRPCER_INVALID_REPLY

  Hitachi.OpenTP1.TP1Error.DCRPCER_REPLY_TOO_BIG

  ・起こっても検出できないエラー

  Hitachi.OpenTP1.TP1Error.DCRPCER_NO_BUFS_AT_SERVER

  Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE

  Hitachi.OpenTP1.TP1Error.DCRPCER_OLTF_INITIALIZING

  Hitachi.OpenTP1.TP1Error.DCRPCER_SECCHK

  Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_CLOSED

  Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_TERMINATING

  Hitachi.OpenTP1.TP1Error.DCRPCER_SYSERR_AT_SERVER

  また，エラー発生時に OpenTP1 にメッセージを出力することもありません。エラーを検出しなければならない場合，flags パラメタに TP1ServerFlags.DCNOFLAGS を設定する（同期応答型 RPC）ことを検討してください。

- トランザクションの処理から Call メソッドでサービスグループを呼び出すと，トランザクションが決着するまで一つの SPP.NET または SPP を占有します。

  同じトランザクションの処理から，Call メソッドで同じサービスを複数回使う場合は，次に示すようにしてください。

  ・使う回数に応じて，ユーザサービス定義の balance_count オペランドおよび parallel_count オペランドの指定値を見積もり直す。

  ・プロセスが増えないように，連鎖 RPC でサービスを要求する。

  balance_count オペランドおよび parallel_count オペランドの指定値に誤りがあると，トランザクションが異常終了したり，デッドロックが発生したりする場合があります。

- 非同期応答型 RPC を使う場合には，PollAnyReplies メソッドですべての非同期の応答を受信するか，DiscardFurtherReplies メソッドで非同期の応答を拒否するまで，サーバ UAP（SPP.NET または SPP）を占有する場合があります。これは，トランザクションの処理の場合にもトランザクションの処理ではない場合にも起こります。非同期応答型 RPC を使う場合には，使う回数に応じて，常駐プロセス数の指定を増やしておいてください。

  非同期応答型 RPC は，SPP.NET または SPP を占有すること以外にも多くの資源を必要とします。UAP の処理効率が下がったり，必要ない SPP.NET または SPP が開始されたりすることを防ぐため，非同期応答型 RPC の Call メソッドを使ったあとには，確実に応答を受信するか，受信を拒否するかしてください。

- 非同期応答型 RPC で連続して複数回使ったあとで応答を受け取る場合には，非同期応答型 RPC のサービスを要求するときに，それぞれ異なる応答格納領域（outputBuffer パラメタ）を設定してください。同じ領域を設定すると，あとから来た応答に上書きされて，応答を正しく受け取れなくなります。

- 非同期応答型 RPC でサービス要求したサーバ UAP（SPP.NET または SPP）は，非同期応答型 RPC を実行したプロセスが PollAnyReplies メソッドを発行したかどうかに関係なく，サービスメソッド実行後にすぐに応答を送信します。PollAnyReplies メソッドを発行しないで，複数回の非同期応答型 RPC を大量に実行すると，SPP.NET または SPP から送信される応答が TCP/IP のバッファにたまり，SPP.NET または SPP の応答送信処理が失敗する場合があります。SPP.NET または SPP で応答送信処理が失敗すると，非同期応答型 RPC 発行元では，PollAnyReplies メソッドを発行しても，SPP.NET または SPP から応答を受信することはできません。
- トランザクション属性の非同期応答型 RPC,非応答型 RPC を大量に実行すると，SPP.NET または SPP から送信されるトランザクションに関する電文を受信できなくなり，トランザクションがロールバックする場合があります。

(10)ドメイン修飾をしてサービスを要求する場合

ドメイン修飾したサービスグループ名を設定すると，DNS のドメイン内にある OpenTP1 のサービスを要求できます。ドメイン修飾をしたサービスグループ名は，サービスグループ名の後ろに，@と DNS のドメイン名を付けて設定します。

・ドメイン修飾をしてサービスを要求するときの注意

- ドメイン修飾をしてサービスを要求する場合は，Call メソッドの flags パラメタに TP1ServerFlags.DCRPC_DOMAIN を設定します。TP1ServerFlags.DCRPC_DOMAIN を設定しないでドメイン修飾をしたサービスグループ名を設定すると，Call メソッドは Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE_GROUP で例外応答します。
- ドメイン修飾をした RPC の場合，Call メソッドを呼び出したプロセスがトランザクションの処理でも，トランザクションを拡張できません。そのため，トランザクションの処理からドメイン修飾をしてサービスを要求する場合は，flags パラメタに TP1ServerFlags.DCRPC_NOTRAN を指定して，トランザクションを拡張しないようにしてください。なお，ドメイン名に自ドメインを設定した場合でも，トランザクションを拡張できません。
- ドメイン修飾をした RPC では，キュー受信型サーバにだけサービスを要求できます。ソケット受信型サーバへは，ドメイン修飾をしたサービス要求はできません。
- ドメイン修飾をしたサービス要求では，namdomainsetup コマンドで登録したホストで起動されるドメイン代表スケジュールサービスへサービス要求を送信します。ドメイン代表スケジュールサービスのポート番号は，〈Windows ディレクトリ〉¥System32¥drivers¥etc¥services から取得します。サービス要求の送信で障害が起こった場合は，namdomainsetup コマンドで複数のホスト名を登録していれば，順次送信を試みます。ドメイン修飾をした RPC が正常に終了しても，ドメイン代表スケジュールサービスへの送信で障害が起こっている場合もあります。

・ドメイン修飾をしたサービス要求をする前の準備

ドメイン修飾をした RPC では，次に示す環境設定をしてください。

1. DNS のドメインデータファイルに，ドメイン代表スケジュールサービスを起動するホスト名を登録します。登録するときは，namdomainsetup コマンドを使います。

2. ドメイン修飾をしたサービス要求をする OpenTP1 を起動するホストの〈Windows ディレクトリ〉¥System32¥drivers¥etc¥services に，ドメイン代表スケジュールサービスのポート番号を設定します。ポート番号は，次の形式で記述してください。

   OpenTP1scd ポート番号/tcp

3. ドメイン代表スケジュールサービスを起動する OpenTP1 のスケジュールサービス定義の scd_port オペランドに，ドメイン代表スケジュールサービスのポートを指定します。

●CallTo

説明

特定の SPP.NET または SPP にサービスを要求します。Call メソッドと同様にサービスグループ名とサービス名を引数に設定するのに加え，ホスト名またはノード識別子を指定した RpcBindTable 構造体に指定したホスト名またはノード識別子は，サービス要求先を特定する検索キーとして使用します。この設定に該当するサービスメソッドにサービスが要求されます。

ただし，ドメイン修飾をしてサービスを要求できません。それ以外は，Call メソッドの機能と同じです。

なお，この機能は，TP1/Extension 1 をインストールしていることが前提です。TP1/Extension 1 をインストールしていない場合の動作は保証できません。

宣言

【C#の場合】

```
public static int CallTo(
  Hitachi.OpenTP1.Server.RpcBindTable direction,
  string serviceGroupName,
  string serviceName,
  byte[] inputBuffer,
  int inputBuffer_len,
  byte[] outputBuffer,
  ref int outputBuffer_len,
  int flags
);
```

【Visual Basic の場合】

```
Public Shared Function CallTo( _
  ByVal direction As Hitachi.OpenTP1.Server.RpcBindTable, _
  ByVal serviceGroupName As String, _
  ByVal serviceName As String, _
  ByVal inputBuffer() As Byte, _
  ByVal inputBuffer_len As Integer, _
  ByVal outputBuffer() As Byte, _
  ByRef outputBuffer_len As Integer, _
  ByVal flags As Integer _
) As Integer
```

【J#の場合】

```
public static int CallTo(
  Hitachi.OpenTP1.Server.RpcBindTable direction,
  System.String serviceGroupName,
  System.String serviceName,
```

```
  ubyte[] inputBuffer,
  int inputBuffer_len,
  ubyte[] outputBuffer,
  int outputBuffer_len,
  int flags
);
```

パラメタ

direction

サービス要求先を特定する検索キーを格納する RpcBindTable 構造体を指定します。検索のキーはホスト名またはノード識別子のどちらかです。

serviceGroupName

SPP.NET または SPP のサービスグループ名を設定します。

serviceName

SPP.NET または SPP のサービス名を設定します。

inputBuffer

サービスの入力パラメタを設定します。

inputBuffer_len

サービスの入力パラメタ長を設定します。

outputBuffer

サービスメソッドから返ってくる応答を格納する領域を設定します。

outputBuffer_len

サービスメソッドから返ってくる応答を格納する領域長を設定します。

flags

RPC の形態とオプションを，次に示す形式で設定します。

```
{TP1ServerFlags.DCNOFLAGS|
TP1ServerFlags.DCRPC_NOWAIT|
TP1ServerFlags.DCRPC_NOREPLY|
TP1ServerFlags.DCRPC_CHAINED}
 [|TP1ServerFlags.DCRPC_TPNOTRAN]
 [|TP1ServerFlags.DCRPC_DOMAIN]
```

オプション	説明
TP1ServerFlags.DCNOFLAGS	同期応答型 RPC
TP1ServerFlags.DCRPC_NOWAIT	非同期応答型 RPC
TP1ServerFlags.DCRPC_NOREPLY	非応答型 RPC
TP1ServerFlags.DCRPC_CHAINED	連鎖 RPC
TP1ServerFlags.DCRPC_TPNOTRAN	トランザクション処理からのサービス要求で，要求先の処理をトランザクションにしない場合に設定します。

オプション	説明
	TP1ServerFlags.DCRPC_TPNOTRAN オプションを設定すると，トランザクションの処理からのサービス要求でも，サービス関数およびサービスメソッドの処理はトランザクションになりません。
TP1ServerFlags.DCRPC_DOMAIN	サービスグループ名をドメイン修飾した場合に指定します。 ドメイン修飾をした RPC はトランザクションブランチにできません。 そのため，トランザクションの処理から Call メソッドを使う場合は，必ず TP1ServerFlags.DCRPC_TPNOTRAN オプションと一緒に指定してください。

TP1ServerFlags.DCRPC_TPNOTRAN オプションと TP1ServerFlags.DCRPC_DOMAIN オプションは，RPC の形態に付けて設定します。

(例 1)

同期応答型 RPC でトランザクションにしないサービス要求をする場合，flags パラメタには「TP1ServerFlags.DCNOFLAGS|TP1ServerFlags.DCRPC_TPNOTRAN」と設定します。

(例 2)

トランザクションの処理から，同期応答型 RPC でドメイン修飾をしたサービス要求をする場合，flags パラメタには「TP1ServerFlags.DCNOFLAGS|TP1ServerFlags.DCRPC_TPNOTRAN|TP1ServerFlags.DCRPC_DOMAIN」と設定します。

戻り値

正常に終了しました。

非同期応答型 RPC の場合は，正の整数は「記述子」です。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_INVALID_ARGS	引数に設定した値が間違っています。 RpcBindTable 構造体の HostName プロパティに指定されたホスト名を，DNS などで IP アドレスと対応づけできません。 CallTo メソッドの第一引数に指定した RpcBindTable 構造体が，SetDirectSchedule メソッドで作成されていて，SetDirectSchedule メソッドの hostName パラメタに 0 を指定しています。
DCRPCER_INVALID_REPLY	サービスメソッドが OpenTP1 に返した応答の長さが，1 から 1048576 の範囲にありません。
DCRPCER_MESSAGE_TOO_BIG	inputBuffer_len パラメタに設定した入力パラメタ長が，最大値を超えています。
DCRPCER_NET_DOWN	ネットワークに障害が起こりました。
DCRPCER_NOT_TRN_EXTEND	トランザクション処理の連鎖 RPC を使ったあとで，flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN を設定した Call メソッドでサービスを要求しています。
DCRPCER_NO_BUFS	メモリが不足しました。 または，サーバのメッセージ格納領域に十分な空きがないため，サービス要求を受け付けられませんでした。
DCRPCER_NO_BUFS_AT_SERVER	設定したサービスで，メモリが不足しました。
DCRPCER_NO_BUFS_RB	メモリが不足しました。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_NO_PORT	ドメイン修飾をしてサービスを要求しましたが，ドメイン代表スケジュールサービスのポート番号が見つかりません。
DCRPCER_NO_SUCH_DOMAIN	ドメイン修飾をしたサービスグループ名の，ドメイン名が間違っています。
DCRPCER_NO_SUCH_SERVICE	serviceName パラメタに設定したサービス名は，定義されていません。
DCRPCER_NO_SUCH_SERVICE_GROUP	serviceGroupName パラメタに設定したサービスグループは定義されていません。RpcBindTable 構造体の NodeID プロパティに指定されたノード識別子が，グローバルドメイン（システム共通定義の all_node オペランドで指定したノード名の集合）内にありません。 一つのトランザクションブランチから開始できる子トランザクションブランチの最大数を超えたため，トランザクションブランチを開始できません。 トランザクション内でドメイン修飾をしたサービス要求で，flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN を設定していません。 リソースマネジャ（RM）でエラーが発生したため，トランザクションブランチを開始できません。 SetDirectSchedule メソッドを使用して RpcBindTable 構造体を作成し，非トランザクション属性（ユーザサービス定義の atomic_update オペランドに N を指定）のユーザサーバに対し，CallTo メソッドの flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN との論理和を指定しないでサービスを要求しました。
DCRPCER_OLTF_INITIALIZING	サービスを要求されたノードにある OpenTP1 は，開始処理中です。

エラーコード	説明
DCRPCER_OLTF_NOT_UP	serviceName パラメタに設定したサービスがあるノードの OpenTP1 が稼働していません。異常終了，停止中，終了処理中，または通信障害の発生が考えられます。
DCRPCER_PROTO	Open メソッドを呼び出していません。
DCRPCER_REPLY_TOO_BIG	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。
DCRPCER_REPLY_TOO_BIG_RB	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_SECCHK	サービスを要求された SPP.NET または SPP は，OpenTP1 のセキュリティ機能で保護されています。Call メソッドでサービスを要求した UAP には，SPP.NET または SPP へのアクセス権限がありません。
DCRPCER_SERVER_BUSY	サービスを要求されたソケット受信型サーバが，サービス要求を受け取れません。
DCRPCER_SERVICE_CLOSED	serviceName パラメタに設定したサービス名のサービスがあるサービスグループは，閉塞しています。
DCRPCER_SERVICE_NOT_UP	serviceName パラメタに設定したサービスの UAP プロセスが，稼働していません。 サービス要求の応答待ち時間に 0 を指定した場合に，サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。
DCRPCER_SERVICE_TERMINATING	serviceName パラメタに設定したサービスは，終了処理中です。
DCRPCER_SYSERR	システムエラーが発生しました。
DCRPCER_SYSERR_AT_SERVER	設定したサービスで，システムエラーが発生しました。
DCRPCER_SYSERR_AT_SERVER_RB	設定したサービスで，システムエラーが発生しました。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_SYSERR_RB	システムエラーが発生しました。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_TESTMODE	オンラインテスタを使っている環境で，テストモードの UAP からテストモードでない SPP へサービスを要求しています。または，テストモードでない UAP からテストモードの SPP へサービスを要求しています。
DCRPCER_TIMED_OUT	Call メソッドの処理が時間切れ（タイムアウト）になりました。 サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。
DCRPCER_TRNCHK	ノード間負荷バランス機能およびノード間負荷バランス拡張機能の環境で，複数の SPP.NET または SPP のトランザクション属性が一致していません。または，負荷を分散する先のノードにある OpenTP1 のバージョンが，クライアントの OpenTP1 のバージョンよりも古いため，ノード間負荷バランス機能およびノード間負荷バランス拡張機能を実行できません。 このエラーコードは，ノード間負荷バランス機能およびノード間負荷バランス拡張機能を使っている SPP.NET または SPP にサービスを要求した場合だけ返されます。

エラーコード	説明
DCRPCER_TRNCHK_EXTEND	同時に起動できるトランザクションブランチの数を超えたため，トランザクションブランチを開始できません。 一つのトランザクションブランチから開始できる子トランザクションブランチの最大数を超えたため，トランザクションブランチを開始できません。 トランザクション内でドメイン修飾をしたサービス要求で，flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN を設定していません。 リソースマネジャ（RM）でエラーが発生したため，トランザクションブランチを開始できません。 SetDirectSchedule メソッドを使用して RpcBindTable 構造体を作成し，非トランザクション属性（ユーザサービス定義の atomic_update オペランドに N を指定）のユーザサーバに対し，CallTo メソッドの flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN との論理和を指定しないでサービスを要求しました。

注意事項

- 一つのマシン内に複数の LAN アダプタが接続されているマルチホームドホスト環境で，システム共通定義の my_host オペランドに指定したホスト名と異なる自マシン内のほかのホスト名を，RpcBindTable 構造体の HostName プロパティ，SetBindTable メソッドの hostName パラメタ，または SetDirectSchedule メソッドの hostName パラメタに指定してはいけません。指定した場合は，動作の保証はしません。
- ホスト名とノード識別子を同時に RpcBindTable 構造体に指定した場合，ホスト名の指定が有効になり，ノード識別子の指定は無視します。
- RpcBindTable 構造体に，ホスト名，ノード識別子ともに空文字または null（Visual Basic の場合は Nothing）を指定した場合は，Call メソッドとまったく同じ動作をします。
- スケジュールサービスの管理するユーザサーバに直接サービスを要求する場合は，必ず SetDirectSchedule メソッドを使用して RpcBindTable 構造体を作成してください。
- SetDirectSchedule メソッドを使用して RpcBindTable 構造体を作成し，ソケット受信型（ユーザサービス定義の receive_from オペランドに socket を指定）ユーザサーバへサービスを要求した場合は，CallTo メソッドは，Hitachi.OpenTP1.Server.DCRPCER_SERVICE_NOT_UP で例外応答します。
- SetDirectSchedule メソッドで作成した RpcBindTable 構造体を指定して CallTo メソッドを呼び出すときに，非トランザクション属性（ユーザサービス定義の atomic_update オペランドに N を指定）のユーザサーバへサービスを要求する場合，flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN との論理和を指定しなければなりません。指定しなかった場合，CallTo メソッドは TP1ServerFlags.DCRPCER_TRNCHK_EXTEND で例外応答します。
- SetDirectSchedule メソッドで作成した RpcBindTable 構造体を指定して CallTo メソッドを呼び出す場合，サービス要求先の OpenTP1 のバージョンは 03-02 以降でなければなりません。03-02 より前の場合，動作の保証はしません。

- ドメイン修飾をした RPC はできません。CallTo メソッドの flags パラメタに TP1ServerFlags.DCRPC_DOMAIN を指定した場合は，Hitachi.OpenTP1.Server.DCRPCER_INVALID_ARGS で例外応答します。
- 次の場合，CallTo メソッドが Hitachi.OpenTP1.Server.DCRPCER_TIMED_OUT で例外応答することがあります。

  システム共通定義の all_node オペランドで指定していないノードのサービスグループに対して，ホスト名を検索のキーに使用して CallTo メソッドを呼び出したあとに，このノード上の OpenTP1 を停止または再開始し，同じサービスグループに対して再度ホスト名を検索のキーに使用して CallTo メソッドを呼び出した場合。
- SetDirectSchedule メソッドを使用して作成した RpcBindTable 構造体を CallTo メソッドの direction パラメタに指定して，CallTo メソッドを呼び出した場合，システム共通定義の rpc_retry オペランドは無効になります。

  SetDirectSchedule メソッドを使用して作成した RpcBindTable 構造体を CallTo メソッドの direction パラメタに指定して，CallTo メソッドを呼び出した場合，性能検証用トレースは取得できますが，呼び出し先の UAP の性能検証用トレース情報とはリンクしていません。クライアント UAP で取得した性能検証用トレースの通番は，サーバ UAP に引き継ぎません。

●Close

説明

OpenTP1 の各種メソッドを使うための環境をクローズします。

SUP.NET でだけ使用します。SPP.NET では Open メソッドおよび Close メソッドを呼び出さないでください。

Close メソッドを呼び出したあとは，OpenTP1 のメソッドは使えません。

Close メソッドはプロセスで 1 回だけ呼び出してください。

Close メソッドは，OpenTP1 に SUP.NET が正常に終了したことを知らせる働きをします。

Close メソッドを呼び出さないで SUP.NET を終了すると，異常終了とみなされてサービスグループの閉塞やプロセス再起動の対象になることがあります。また，OpenTP1 で管理する各種資源が解放されないままになって，そのあとの処理に悪影響を与える場合があります。OpenTP1 で使うすべての SUP.NET では，Open メソッドを呼び出したら，Close メソッドを必ず呼び出してください。

Close メソッドは，Open メソッドが例外応答したときにも必ず呼び出してください。

Close メソッドを呼び出したあとは，同じ SUP.NET で再び Open メソッドを呼び出せません。

宣言

【C#の場合】

```
public static void Close(
);
```

【Visual Basic の場合】

```
Public Shared Sub Close( _
)
```

【J#の場合】

```
public static void Close(
);
```

パラメタ

なし

戻り値

なし

例外

なし

●DiscardFurtherReplies

説明

非同期応答型 RPC（Call メソッドの flags パラメタに TP1ServerFlags.DCRPC_DOMAIN を設定）で，まだ返ってきていない応答を，これ以上受信しないことを示すメソッドです。このメソッドを呼び出したあとは，応答が返ってきても受信しないで捨てます。

非同期応答型 RPC の結果をこれ以上受信しない場合は，必ず DiscardFurtherReplies メソッドを呼び出してください。呼び出さないと，PollAnyReplies メソッドが不要な応答を受信してしまうことがあります。

DiscardFurtherReplies メソッドを使う場合を次に示します。

- 応答待ち時間切れになったあと，結果を保持しておくバッファを解放する場合
- 非同期応答型 RPC を複数回使って，そのうち最初の応答だけ必要な場合

宣言

【C#の場合】

```
public static void DiscardFurtherReplies(
);
```

【Visual Basic の場合】

```
Public Shared Sub DiscardFurtherReplies( _
)
```

【J#の場合】

```
public static void DiscardFurtherReplies(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_PROTO	Open メソッドを呼び出していません。

●DiscardSpecificReply

説明

非同期応答型 RPC（Call メソッドの flags パラメタに TP1ServerFlags.DCRPC_NOWAIT を設定）で，まだ返ってきていない応答を，これ以上受信しないことを示すメソッドです。受信を拒否する非同期応答を特定するには，descriptor パラメタに，非同期応答型 RPC が応答で返した記述子を設定します。このメソッドを呼び出したあとに返ってきた応答の中で，設定した記述子と同じ記述子を持つ応答は受信しないで捨てます。

宣言

【C#の場合】

```
public static void DiscardSpecificReply(
  int descriptor
);
```

【Visual Basic の場合】

```
Public Shared Sub DiscardSpecificReply( _
  ByVal descriptor As Integer _
)
```

【J#の場合】

```
public static void DiscardSpecificReply(
  int descriptor
);
```

パラメタ

descriptor

非同期応答型 RPC の Call メソッドが正常に終了したときに返された記述子を設定します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_INVALID_ARGS	引数に設定した値が間違っています。
DCRPCER_INVALID_DES	descriptor パラメタに設定した記述子は存在しません。要因として次のことが考えられます。 • 設定した記述子に対応する非同期応答型 RPC を行っていない。 • 非同期応答型 RPC の応答がすでに受信されている，または受信が拒否されている。
DCRPCER_PROTO	Open メソッドを呼び出していません。

●GetCallersAddress

説明

クライアント UAP のプロセスが稼働するノードアドレスを，サーバ UAP で取得します。

GetCallersAddress メソッドで返されたアドレス値で，クライアント UAP のセキュリティチェックができます。

GetCallersAddress メソッドで返されたアドレスを使って，サービスの応答やエラーの応答などは送信できません。

GetCallersAddress メソッドは，サービスメソッドから呼び出してください。

サービスメソッド以外から呼び出した場合の処理は保証しません。

宣言

【C#の場合】

```
public static System.Net.IPAddress GetCallersAddress(
);
```

【Visual Basic の場合】

```
Public Shared Function GetCallersAddress( _
) As System.Net.IPAddress
```

【J#の場合】

```
public static System.Net.IPAddress GetCallersAddress(
);
```

パラメタ

なし

戻り値

メソッドが正常終了の場合に，クライアント UAP のノードアドレスを格納します。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_PROTO	Open メソッドを呼び出していません。

注意事項

次の条件が重なった場合，クライアント UAP のノードアドレスは，実際にクライアント UAP が通信で使用したノードアドレスと異なる場合があります。

- リモート API 機能を使用して，サービス要求を受け付けた。
- クライアント UAP が存在するホストがマルチホームドホスト環境である。

●GetErrorDescriptor

説明

非同期応答を特定しない PollAnyReplies メソッドで例外が発生した直後に呼び出すことで，エラーが発生した非同期応答型 RPC 要求に対応する記述子を取得します。

記述子を取得できるのは，SPP.NET または SPP 側でエラーが発生した場合だけです。PollAnyReplies メソッドの呼び出し側でエラーが発生した場合には，このメソッドの記述子を取得できません。

宣言

【C#の場合】

```
public static int GetErrorDescriptor(
);
```

【Visual Basic の場合】

```
Public Shared Function GetErrorDescriptor( _
) As Integer
```

【J#の場合】

```
public static int GetErrorDescriptor(
);
```

パラメタ

なし

戻り値

- 正の整数

  PollAnyReplies メソッドが返したエラーに対応する非同期応答型 RPC 要求の記述子を返します。

- 0

  PollAnyReplies メソッドが返したエラーに対応する非同期応答型 RPC 要求の記述子を取得できませんでした。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_INVALID_ARGS	引数に設定した値が間違っています。
DCRPCER_PROTO	Open メソッドを呼び出していません。

●GetGatewayAddress

説明

アプリケーションゲートウェイ型ファイアウォールなどの，ゲートウェイを介してクライアント UAP からのサービス要求を受信したとき，サーバ UAP でゲートウェイのノードアドレスを取得します。

リモート API 機能を使用してサービスを要求した場合，サーバ UAP で，ゲートウェイのノードアドレスを取得できます。

GetGatewayAddress メソッドで返されたアドレスを使って，サービスの応答やエラーの応答などの送信はできません。

GetGatewayAddress メソッドは，サービスメソッドから呼び出してください。

サービスメソッド以外から呼び出した場合の処理は保証できません。

宣言

【C#の場合】

```
public static System.Net.IPAddress GetGatewayAddress(
);
```

【Visual Basic の場合】

```
Public Shared Function GetGatewayAddress( _
) As System.Net.IPAddress
```

【J#の場合】

```
public static System.Net.IPAddress GetGatewayAddress(
);
```

パラメタ

なし

戻り値

メソッドが正常終了の場合に，ゲートウェイのノードアドレスを格納します。

リモート API 機能を使用していない場合には 0.0.0.0 が設定されます。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_INVALID_ARGS	引数に設定した値が間違っています。
DCRPCER_PROTO	サービスメソッドから呼び出されていません。 Open メソッドを呼び出していません。

●GetServicePriolity

説明

SetServicePriolity メソッドで設定した，サービス要求のスケジュールプライオリティを参照します。このメソッドが返すスケジュールプライオリティの値は，UAP が再び SetServicePriolity メソッドを呼び出すまで変わりません。

次に示す場合，GetServicePriolity メソッドは省略時仮定値（4）を返します。

- UAP で SetServicePriolity メソッドを呼び出していない場合
- SetServicePriolity メソッドの schedulePriolity パラメタに 0 を設定して呼び出した場合

宣言

【C#の場合】

```
public static int GetServicePriolity(
);
```

【Visual Basic の場合】

```
Public Shared Function GetServicePriolity( _
) As Integer
```

【J#の場合】

```
public static int GetServicePriolity(
);
```

パラメタ

なし

戻り値

SetServicePriolity メソッドで設定したスケジュールプライオリティを，1～8 の範囲で示します。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_PROTO	Open メソッドを呼び出していません。
上記以外の負の整数	プログラムの破壊などによる，予期しないエラーが発生しました。

●GetTimeout

説明

現在のサービス要求の応答待ち時間を参照します。このメソッドは，SetTimeout メソッドで応答待ち時間を一時的に変更する前に，元の値を退避するために使います。

このメソッドは，SetTimeout メソッドで変更したサービス応答待ち時間を返します。変更していない場合は，次に示す値を返します。

- TP1/Server Base の場合：システム共通定義の watch_time オペランドの値
- TP1/LiNK の場合：180 秒

このメソッドで得られる値は，OpenTP1 の Call メソッドに対して有効です。

宣言

【C#の場合】

```
public static int GetTimeout(
);
```

【Visual Basic の場合】

```
Public Shared Function GetTimeout( _
) As Integer
```

【J#の場合】

```
public static int GetTimeout(
);
```

パラメタ

なし

戻り値

現在のサービス応答待ち時間を示します。

0 の場合は，無制限に待ち続ける指定です。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_PROTO	Open メソッドを呼び出していません。
上記以外の負の整数	プログラムの破壊などによる，予期しないエラーが発生しました。

●Open

説明

OpenTP1 の各種メソッドを使う準備をします。

SUP.NET でだけ使用します。SPP.NET では Open メソッドおよび Close メソッドを呼び出さないでください。

Open メソッドはプロセスで 1 回だけ呼び出してください。

メインメソッドでの初期化手順を次に示します。

1. プロセス間通信のためのエントリポイントを開きます。
2. OpenTP1 で使う共用メモリを取得します。
3. OpenTP1 に SUP.NET の開始を通知して，プロセスの監視を要求します。
4. そのほか，設定した SUP.NET の環境に従って，使う OpenTP1 の各機能の初期化処理をします。

SUP.NET にトランザクション属性を指定している場合は，そのノードで OpenTP1 のトランザクションサービスとプロセスサービスが実行中であることが前提になります。Open メソッドは，OpenTP1 が OS の開始に伴って，または dcstart コマンドによって正常開始したあとでないと実行できません。

OpenTP1 が正常開始する前に Open メソッドを呼び出すと，Hitachi.OpenTP1.TP1Error.DCRPCER_OLTF_NOT_UP で例外応答します。この場合は，Call メソッドなどの OpenTP1 のメソッドは使えません。

UAP トレースは，Open メソッドが正常に終了したあとに呼び出したすべての OpenTP1 のメソッドについて取得されます。そのため，Open メソッドが例外応答した場合の UAP トレースは，取得されている場合も取得されていない場合もあります。

宣言

【C#の場合】

```
public static void Open(
);
```

【Visual Basic の場合】

```
Public Shared Sub Open( _
)
```

【J#の場合】

```
public static void Open(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード

発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCNJSER_SYSTEM	内部エラーが発生しました。
DCNJSER_XALIB_INVALID	トランザクション制御用ライブラリが不正です。
DCNJSER_XALIB_LOAD	トランザクション制御用ライブラリが見つかりません。
DCRPCER_FATAL	初期化に失敗しました。以降，OpenTP1 のメソッドは使えません。
DCRPCER_INVALID_ARGS	引数が間違っています。
DCRPCER_OLTF_NOT_UP	SUP.NET があるノードの OpenTP1 が実行していません。
DCRPCER_PROTO	Open メソッドはすでに呼び出しています。
DCRPCER_SEC_INIT	セキュリティ機能を使った OpenTP1 が，セキュリティ環境の初期化処理でエラーになりました。
DCRPCER_STANDBY_END	待機系のユーザサーバが，待機の終了を要求されました。または，系切り替え時にすでに終了していたため，待機の終了を要求されました。

●PollAnyReplies

説明

非同期応答型 RPC（Call メソッドの flags パラメタに TP1ServerFlags.DCRPC_NOWAIT を設定）でサービス要求した結果を受信します。

受信する非同期応答を特定する場合は，flags パラメタに TP1ServerFlags.DCRPC_SPECIFIC_MSG を設定します。このフラグを設定した場合は，descriptor パラメタに設定した記述子を設定した非同期応答型 RPC の応答を受信します。

受信する非同期応答を特定しない場合は，flags パラメタに TP1ServerFlags.DCNOFLAGS を設定します。このとき，descriptor パラメタに設定した値は無視されます。flags パラメタに TP1ServerFlags.DCNOFLAGS を設定した PollAnyReplies メソッドが正常に終了すると，受信した非同期応答の記述子と同じ値を返します。

PollAnyReplies メソッドは，次のどちらかの場合に例外応答します。

- 非同期応答型 RPC の応答を受信したとき
- メソッドのパラメタに指定した応答待ち時間切れになったとき

PollAnyReplies メソッドが正常に終了すると，非同期応答型 RPC を使った Call メソッドに設定した応答を格納する領域に，受信した応答が設定されます。

PollAnyReplies メソッドの詳しい説明については，このメソッドの「注意事項」を参照してください。「注意事項」では次に示す項目について説明しています。

(1)PollAnyReplies メソッドの timeout パラメタについて

(2)PollAnyReplies メソッドがエラーになるタイミング

(3)エラーコード Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_TERMINATED を返させる指定

(4)エラーコードと同期点処理の関係

(5)PollAnyReplies メソッドで応答が受け取れない場合

(6)PollAnyReplies メソッドを使うときの注意

宣言

【C#の場合】

```
public static int PollAnyReplies(
  int descriptor,
  int timeout,
  int flags
);
```

【Visual Basic の場合】

```
Public Shared Function PollAnyReplies( _
  ByVal descriptor As Integer, _
  ByVal timeout As Integer, _
  ByVal flags As Integer _
) As Integer
```

【J#の場合】

```
public static int PollAnyReplies(
  int descriptor,
  int timeout,
  int flags
);
```

パラメタ

descriptor

非同期応答型 RPC の Call メソッド（flags パラメタに TP1ServerFlags.DCRPC_NOWAIT）が正常に終了したときに返された，記述子を設定します。

flags パラメタに TP1ServerFlags.DCNOFLAGS を設定した場合は，ここに設定した値は無視されます。

timeout

非同期応答型 RPC の Call メソッドの結果が返ってくるまでの待ち時間を，秒単位またはミリ秒単位で設定します。指定できる値の範囲は，-1 から 2147483647 までです。

PollAnyReplies メソッドで非同期応答を受信する場合は，UAP に設定した応答待ち時間を参照しません。

0 を設定した場合，flags パラメタに TP1ServerFlags.DCNOFLAGS または TP1ServerFlags.DCRPC_SPECIFIC_MSG を設定したとき，応答が返っていないと Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT で，すぐに例外応答します。flags パラメタに TP1ServerFlags.DCRPC_WAIT_MILLISEC を設定したときは 50 ミリ秒として処理します。

-1 を設定した場合は，応答が返るまで待ち続けます。

flags

RPC の形態とオプションを，次に示す形式で設定します。

```
{TP1ServerFlags.DCNOFLAGS|
TP1ServerFlags.DCRPC_SPECIFIC_MSG}
 〔|TP1ServerFlags.DCRPC_WAIT_MILLISEC〕
```

オプション	説明
TP1ServerFlags.DCNOFLAGS	受信する非同期の応答を特定しません。
TP1ServerFlags.DCRPC_SPECIFIC_MSG	descriptor パラメタに設定した記述子をリターンした，非同期応答型 RPC の応答を受信します。
TP1ServerFlags.DCRPC_WAIT_MILLISEC	timeout パラメタで設定した待ち時間の単位をミリ秒にします。

戻り値

受信した非同期応答の記述子を示します。正の整数は，flags パラメタに TP1ServerFlags.DCNOFLAGS を設定したメソッドが正常に終了した場合に返されます。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_ALL_RECEIVED	非同期応答型 RPC で要求したサービスの処理結果は，すべて受信しました。
DCRPCER_INVALID_ARGS	引数に設定した値が間違っています。
DCRPCER_INVALID_DES	descriptor パラメタに設定した記述子は存在しません。このエラーコードは，flags パラメタに TP1ServerFlags.DCRPC_SPECIFIC_MSG を設定した場合に返されます。

エラーコード	説明
DCRPCER_INVALID_REPLY	サービスメソッドが OpenTP1 に返した応答の長さが，1 から 1048576 の範囲にありません。
DCRPCER_MESSAGE_TOO_BIG	inputBuffer_len パラメタに設定した入力パラメタ長が，最大値を超えています。
DCRPCER_NET_DOWN	ネットワークに障害が起こりました。
DCRPCER_NO_BUFS	メモリが不足しました。
DCRPCER_NO_BUFS_AT_SERVER	Call メソッドに設定したサービスで，メモリが不足しました。
DCRPCER_NO_BUFS_RB	メモリが不足しました。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_NO_PORT	ドメイン修飾をしてサービスを要求しましたが，ドメイン代表スケジュールサービスのポート番号が見つかりません。
DCRPCER_NO_SUCH_DOMAIN	ドメイン修飾をしたサービスグループ名の，ドメイン名が間違っています。
DCRPCER_NO_SUCH_SERVICE	Call メソッドの serviceName パラメタに設定したサービス名は，定義されていません。
DCRPCER_NO_SUCH_SERVICE_GROUP	Call メソッドの serviceGroupName パラメタに設定したサービスグループは，定義されていません。
DCRPCER_OLTF_INITIALIZING	サービスを要求されたノードにある OpenTP1 は，開始処理中です。
DCRPCER_OLTF_NOT_UP	Call メソッドの serviceName パラメタに設定したサービスがあるノードの OpenTP1 が稼働していません。異常終了，停止中，終了処理中，または通信障害が起こったことが考えられます。
DCRPCER_PROTO	Open メソッドを呼び出していません。
DCRPCER_REPLY_TOO_BIG	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。
DCRPCER_REPLY_TOO_BIG_RB	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_SECCHK	サービスを要求された SPP.NET または SPP は，OpenTP1 のセキュリティ機能で保護されています。Call メソッドでサービスを要求した UAP には，SPP.NET または SPP へのアクセス権限がありません。
DCRPCER_SERVER_BUSY	サービスを要求されたソケット受信型サーバが，サービス要求を受け取れません。
DCRPCER_SERVICE_CLOSED	Call メソッドの serviceName パラメタに設定したサービス名があるサービスグループは，閉塞しています。
DCRPCER_SERVICE_NOT_UP	Call メソッドの serviceName パラメタに設定したサービスの UAP プロセスが，稼働していません。 timeout パラメタに-1 を指定した場合に，サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。

エラーコード	説明
DCRPCER_SERVICE_TERMINATED	サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。このエラーコードは，ユーザサービス定義の rpc_extend_function オペランドに"00000001"を指定したクライアント UAP の場合にだけ返されます。 rpc_extend_function オペランドに"00000000"を指定，またはオペランドを省略した場合は，このエラーコードは返らないで，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT または Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_NOT_UP が返されます。
DCRPCER_SERVICE_TERMINATING	Call メソッドの serviceName パラメタに設定したサービスは，終了処理中です。
DCRPCER_SYSERR	システムエラーが発生しました。
DCRPCER_SYSERR_AT_SERVER	Call メソッドに設定したサービスで，システムエラーが発生しました。
DCRPCER_SYSERR_AT_SERVER_RB	設定したサービスで，システムエラーが発生しました。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_SYSERR_RB	システムエラーが発生しました。このエラーコードが返った場合は，トランザクションブランチをコミットできません。
DCRPCER_TESTMODE	オンラインテスタを使っている環境で，テストモードの UAP からテストモードでない SPP へサービスを要求しています。または，テストモードでない UAP からテストモードの SPP へサービスを要求しています。
DCRPCER_TIMED_OUT	Call メソッドの処理が時間切れ（タイムアウト）になりました。 サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。
DCRPCER_TRNCHK	ノード間負荷バランス機能およびノード間負荷バランス拡張機能の環境で，複数の SPP.NET または SPP のトランザクション属性が一致していません。または，負荷を分散する先のノードにある OpenTP1 のバージョンが，クライアントの OpenTP1 のバージョンよりも古いため，ノード間負荷バランス機能およびノード間負荷バランス拡張機能を実行できません。 このエラーコードは，ノード間負荷バランス機能およびノード間負荷バランス拡張機能を使っている SPP.NET または SPP にサービスを要求した場合にだけ返されます。
DCRPCER_TRNCHK_EXTEND	同時に起動できるトランザクションブランチの数を超えたため，トランザクションブランチを開始できません。 一つのトランザクションブランチから開始できる子トランザクションブランチの最大数を超えたため，トランザクションブランチを開始できません。トランザクション内でドメイン修飾をしたサービス要求で，flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN を設定していません。 リソースマネジャ（RM）でエラーが発生したため，トランザクションブランチを開始できません。

注意事項

(1)PollAnyReplies メソッドの timeout パラメタについて

非同期受信の監視時間は，応答が返るたびにリセットされます。

そのため，受信する非同期応答を特定（flags パラメタに TP1ServerFlags.DCRPC_SPECIFIC_MSG を設定）した場合は，timeout パラメタに設定した時間を経過しても，応答を受信できることがあります。また，timeout パラメタに設定した時間を経過しても，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT で例外応答しない場合もあります。

(2)PollAnyReplies メソッドがエラーになるタイミング

サービスを要求された SPP.NET または SPP が異常終了した場合，クライアント UAP でエラーが返るタイミングについて説明します。

サービスを実行する SPP.NET または SPP が，処理が終わる前に異常終了すると，PollAnyReplies メソッドは Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT で例外応答します。
PollAnyReplies メソッドの timeout パラメタに-1 を設定している場合は，Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_NOT_UP で例外応答します。

PollAnyReplies メソッドの時間監視でエラーになる場合

次に示す場合には，PollAnyReplies メソッドの timeout パラメタに設定した時間が経過したあとで，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT で例外応答します。

- SPP.NET または SPP があるノードの OpenTP1 全体が異常終了した場合
- サービス要求のデータをサーバ UAP が受信する前，またはサーバ UAP の処理が完了してからクライアント UAP が結果を受信する前に障害が起こった場合

(3)エラーコード Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_TERMINATED を返させる指定

サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了したことを Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT または Hitachi.OpenTP1.Server.DCRPCER_SERVICE_NOT_UP 以外のエラーコードで判別したい場合には，ユーザサービス定義の rpc_extend_function オペランドに"00000001"を指定します。この指定をすると，上記のエラー時に Hitachi.OpenTP1.Server.DCRPCER_SERVICE_TERMINATED が返されるようになります。
rpc_extend_function オペランドに"00000000"を指定するか，またはオペランドを省略した場合は，Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_TERMINATED は返らないで，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT または Hitachi.OpenTP1.TP1Error.DCRPCER_SERVICE_NOT_UP が返されます。

(4)エラーコードと同期点処理の関係

PollAnyReplies メソッドのエラーコードと同期点処理（コミット，ロールバック）の関係について説明します。ここで説明する内容は，サービス要求がトランザクション処理になる場合に該当します。

トランザクションでないサービス要求（Call メソッドの flags パラメタに TP1ServerFlags.DCRPC_TPNOTRAN を設定した場合も含む）には該当しません。

PollAnyReplies メソッドが例外応答してもコミットとなる場合

サービスを要求されたサービスメソッドの異常終了，ノードの障害，ネットワーク障害などの場合でも，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT が返ることがあります。クライアント UAP がトランザクション処理でない場合は，Hitachi.OpenTP1.TP1Error.DCRPCER_TIMED_OUT が返っても，要求したサービスがあるサービスメソッドは正常終了していて，データベースへの更新などが実行されているときもあります。

ロールバック処理が必要な例外応答値

トランザクション処理から呼び出した PollAnyReplies メソッドが例外応答した場合，エラーコードによっては，必ずトランザクションがロールバック（サーバ UAP が rollback_only 状態）になります。この場合，コミットのメソッド，またはロールバックのメソッドのどちらを使っても，必ずロールバックになります。必ずロールバックになる PollAnyReplies メソッドのエラーコードを次に示します。

- Hitachi.OpenTP1.TP1Error.DCRPCER_INVALID_REPLY
- Hitachi.OpenTP1.TP1Error.DCRPCER_NO_BUFS_AT_SERVER
- Hitachi.OpenTP1.TP1Error.DCRPCER_NO_SUCH_SERVICE
- Hitachi.OpenTP1.TP1Error.DCRPCER_REPLY_TOO_BIG_RB

(5)PollAnyReplies メソッドで応答が受け取れない場合

非同期応答型 RPC でサービスを要求した UAP が次に示すメソッドを呼び出すと，PollAnyReplies メソッドで応答を受け取れません。

- DiscardFurtherReplies メソッドで，非同期応答の受信を拒否した場合
- トランザクションの処理の場合，同期点処理のメソッドでコミットまたはロールバックした場合

上記のメソッドを使ったあとで返ってきた応答は，破棄されます。

非同期応答型 RPC では，上記のメソッドを呼び出す前に，必要な非同期の応答を PollAnyReplies メソッドですべて受け取ってください。

(6)PollAnyReplies メソッドを使うときの注意

- PollAnyReplies メソッドを待ち時間 0 と設定（timeout パラメタに 0 を設定）して呼び出すと，マルチスレッド環境のスケジューリングに関連して，応答が到着していても受信できない場合があります。そのため，すべての応答を受信するまで，待ち時間に 0 を設定した PollAnyReplies メソッドを呼び出す UAP は，無限ループになることがあるので注意してください。
- 記述子を特定しない PollAnyReplies メソッドが例外応答した場合，エラーとなった応答の記述子を特定できません。PollAnyReplies メソッドが例外応答したときに該当する記述子がわかるようにしておきたい場合は，flags パラメタに TP1ServerFlags.DCRPC_SPECIFIC_MSG を設定しておいてください。

●SetBindTable

説明

指定されたサービス要求先ノードのノード識別子（nodeID パラメタの値），または指定されたサービス要求先ノードのホスト名（hostName パラメタの値）を RpcBindTable 構造体に設定し，CallTo メソッドの第一引数を作成します。

宣言

【C#の場合】

```
public static void SetBindTable(
  ref Hitachi.OpenTP1.Server.RpcBindTable direction,
  string nodeID,
  string hostName,
  int portNo
);
```

【Visual Basic の場合】

```
Public Shared Sub SetBindTable( _
  ByRef direction As Hitachi.OpenTP1.Server.RpcBindTable, _
  ByVal nodeID As String, _
  ByVal hostName As String, _
  ByVal portNo As Integer _
)
```

【J#の場合】

```
public static void SetBindTable(
  Hitachi.OpenTP1.Server.RpcBindTable direction,
  System.String nodeID,
  System.String hostName,
  int portNo
);
```

パラメタ

direction

RpcBindTable 構造体を設定します。

nodeID

ノード識別子を設定します。

ノード識別子を検索のキーにしない場合には，空文字列または null（Visual Basic の場合は Nothing）を指定してください。

ノード識別子は，システム共通定義の node_id オペランドに指定した名称で，かつサービス要求先ノードのホスト名がグローバルドメイン（システム共通定義の all_node オペランドで指定したノード名の集合）内にあることが前提です。

hostName

ホスト名を設定します。

ホスト名を，サービス要求先を特定する検索のキーにしない場合には，空文字列または null（Visual Basic の場合は Nothing）を指定してください。

サービス要求先ノードのホスト名はグローバルドメイン（システム共通定義の all_node オペランドに指定したノード名の集合）に指定されていても，指定されていなくてもどちらでもかまいません。

portNo

- ホスト名を検索のキーにする場合

  サービス要求先の OpenTP1 システムのネームサービスのポート番号（システム共通定義の name_port オペランドに指定した値）を指定します。

  サービス要求先のネームサービスのポート番号が，サービス要求元のネームサービスのポート番号と同じ場合は，0 を指定します。

- ノード識別子を検索のキーとする場合

  portNo パラメタに 0 を指定します。portNo パラメタに 0 を指定した場合は，サービス要求先のネームサービスのポート番号（システム共通定義の name_port オペランドに指定した値）と，サービス要求元のネームサービスのポート番号を同じにしてください。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_PROTO	Open メソッドを呼び出していません。

注意事項

- SetBindTable メソッドは，CallTo メソッドの第一引数に指定する，RpcBindTable 構造体を設定するためのメソッドです。

- SetBindTable メソッドの引数の指定値のチェックや，指定方法によってどのような動作をするかについては，作成した RpcBindTable 構造体を指定して呼び出した CallTo メソッドの動作が示します。
- SetBindTable メソッドは，UAP トレースを取得しません。

●SetDirectSchedule

説明

指定されたサービス要求先ノードのホスト名（hostName パラメタの値）と，指定されたスケジュールサービスのポート番号（schedulePortNo パラメタの値）を RpcBindTable 構造体に設定し，CallTo メソッドの第一引数を作成します。

宣言

【C#の場合】

```
public static void SetDirectSchedule(
  ref Hitachi.OpenTP1.Server.RpcBindTable direction,
  string hostName,
  int schedulePortNo
);
```

【Visual Basic の場合】

```
Public Shared Sub SetDirectSchedule( _
  ByRef direction As Hitachi.OpenTP1.Server.RpcBindTable, _
  ByVal hostName As String, _
  ByVal schedulePortNo As Integer _
)
```

【J#の場合】

```
public static void SetDirectSchedule(
  Hitachi.OpenTP1.Server.RpcBindTable direction,
  System.String hostName,
  int schedulePortNo
);
```

パラメタ

direction

RpcBindTable 構造体を設定します。

hostName

サービス要求先ノードのホスト名を指定します。

サービス要求先ノードのホスト名はグローバルドメイン（システム共通定義の all_node オペランドに指定したノード名の集合）に指定されていても，指定されていなくても，どちらでもかまいません。

ホスト名は必ず指定してください。ホスト名に空文字列または null（Visual Basic の場合は Nothing）が指定された場合は，CallTo メソッドで例外が発生します。

schedulePortNo

サービス要求先 OpenTP1 システムのスケジュールサービスのポート番号（サービス要求先のスケジュールサービス定義の scd_port オペランドに指定した値）を指定します。0 を指定した場合は，送信先ポート番号の省略値として，サービス要求元のスケジュールサービス定義の scd_port オペランドの指定値を仮定します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_PROTO	Open メソッドを呼び出していません。

注意事項

- SetDirectSchedule メソッドは，CallTo メソッドの第一引数に指定する RpcBindTable 構造体を設定するためのメソッドです。
- SetDirectSchedule メソッドの引数の指定値のチェックや，指定方法によってどのような動作をするかについては，作成した RpcBindTable 構造体を指定して呼び出した CallTo メソッドの動作が示します。
- SetDirectSchedule メソッドは，UAP トレースを取得しません。

●SetServicePriolity

説明

サービス要求のプライオリティを設定します。サービス要求単位でスケジュールプライオリティを制御する場合に呼び出します。

このメソッドで設定したプライオリティは，このメソッドを再び呼び出すまで更新されません。したがって，同じプライオリティでまとめてサービス要求する場合は，このメソッドを 1 回だけ呼び出します。

このメソッドで指定したプライオリティは，直後に呼び出す Call メソッドで，スケジュールキューを経由してサーバに通知されます。

このメソッドを一度も呼び出さない場合の処理は，スケジュールサービスの省略時解釈で，サービス要求のプライオリティが決定されます。

宣言

【C#の場合】

```
public static void SetServicePriolity(
  int schedulePriolity
);
```

【Visual Basic の場合】

```
Public Shared Sub SetServicePriolity( _
  ByVal schedulePriolity As Integer _
)
```

【J#の場合】

```
public static void SetServicePriolity(
  int schedulePriolity
);
```

パラメタ

schedulePriolity

サービス要求のスケジュールプライオリティを，0 から 8 の範囲で設定します。schedulePriolity パラメタの設定は省略できません。

最も高いプライオリティの値は 1 で，最も低いプライオリティの値は 8 です。

0 を設定した場合は，スケジュールサービスの省略時解釈となります。

上記以外の値を設定した場合は，SetServicePriolity メソッドは無視されます。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_PROTO	Open メソッドを呼び出していません。

注意事項

- キュー受信型サーバでは，設定したサービス要求のプライオリティは，サーバ UAP のユーザサービス定義の service_priority_control オペランドに Y（プライオリティを制御する）を指定している場合にだけ有効です。
  サービス要求する相手のサーバ UAP でプライオリティを制御していない場合は，このメソッドを呼び出しても無効になります。
- 2 回目以降の連鎖 RPC での Call メソッドと，連鎖 RPC を終了させるために呼び出す同期応答型 RPC の Call メソッド（flags パラメタに TP1ServerFlags.DCNOFLAGS を設定）のサービス要求に対して SetServicePriority メソッドを呼び出しても無効となります。
- Call メソッドは，サービス要求のプライオリティを省略値にリセットしません。サービス要求のプライオリティをリセットする場合は，schedulePriority に 0 を設定した SetServicePriority メソッドを呼び出し直してください。

●SetServiceRetry

説明

実行中のサービスメソッドの処理をリトライします。リトライする場合は，サービスメソッドで SetServiceRetry メソッドを呼び出したあとで，リトライするサービスメソッドをリターンしてください。リターンしたあと，同じプロセスで同じサービスメソッドが再起動されます。

応答型 RPC で呼ばれたサービスメソッドがリトライされた場合は，リトライ前のサービスメソッドが設定した値（応答を格納する領域および応答の長さ）は無効になります。

ユーザサービス定義の rpc_service_retry_count オペランドに指定した回数を超えた（rpc_service_retry_count オペランドに 0 を指定した場合も含む）あとで SetServiceRetry メソッドを呼び出した場合，メソッドは Hitachi.OpenTP1.TP1Error.DCRPCER_RETRY_COUNT_OVER で例外応答します。このとき，サービスメソッドはリトライされません。

応答型 RPC で呼ばれたサービスメソッドの場合は，応答を格納する領域の内容をクライアント UAP に返します。

宣言

【C#の場合】

```
public static void SetServiceRetry(
);
```

【Visual Basic の場合】

```
Public Shared Sub SetServiceRetry( _
)
```

【J#の場合】

```
public static void SetServiceRetry(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_PROTO	SetServiceRetry メソッドを呼び出す条件に誤りがあります。 次に示すことが考えられます。 • サービスメソッドの中で呼び出していません。 • グローバルトランザクションの範囲の中で呼び出しています。

エラーコード	説明
	• Open メソッドを呼び出していません。
DCRPCER_RETRY_COUNT_OVER	ユーザサービス定義の rpc_service_retry_count オペランドに指定したサービスリトライ回数最大値を超えて，SetServiceRetry メソッドを呼び出しています。これ以上サービスメソッドをリトライできません。

注意事項

- SetServiceRetry メソッドを呼び出す場合は，次に示す条件を満たしてください。これらの条件を満たしていない場合，SetServiceRetry メソッドは例外応答します。

  ・サービスメソッドの中で SetServiceRetry メソッドを呼び出していること。

  ・実行中のサービスメソッドが，グローバルトランザクションの範囲でないこと。
- SetServiceRetry メソッドを呼び出すサービスメソッドでは，クライアント UAP から渡されたデータは参照できますが，変更できません。入力データ領域の内容を変更した場合，システムの動作は保証しません。
- SetServiceRetry メソッドは，OpenTP1 独自のリモートプロシジャコール（Call メソッド）でサービスを要求されたサービスメソッドでだけ呼び出せます。それ以外のサービスメソッドの処理は，SetServiceRetry メソッドでリトライできません。

●SetTimeout

説明

サービス要求の応答待ち時間を変更します。このメソッドで変更した値は，Close メソッドを呼び出すまで有効です。

サービス要求の応答待ち時間をこのメソッドを呼び出す前の値に戻すときは，GetTimeout メソッドで返された元の値を，このメソッドで再設定してください。

このメソッドは，システム共通定義の watch_time オペランドに指定した値を変更しません。このメソッドで設定する値は，直後に呼び出す Call メソッドにだけ影響します。

宣言

【C#の場合】

```
public static void SetTimeout(
  int waitTime
);
```

【Visual Basic の場合】

```
Public Shared Sub SetTimeout( _
  ByVal waitTime As Integer _
)
```

【J#の場合】

```
public static void SetTimeout(
  int waitTime
);
```

パラメタ

waitTime

サービス応答待ち時間を設定します。0～65535 の範囲で設定します。

無制限に待ち続ける場合は，0 を設定します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）

  例外が発生する原因となった引数名が出力されます。

- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRPCER_PROTO	Open メソッドを呼び出していません。
上記以外の負の整数	プログラムの破壊などによる，予期しないエラーが発生しました。

RpcBindTable 構造体

RpcBindTable 構造体の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- System.ValueType
    +- Hitachi.OpenTP1.Server.RpcBindTable
```

説明

RpcBindTable 構造体は，サービス要求先を特定する検索キーを格納します。

プロパティの一覧

名称	説明
Flags	フラグの設定および参照を行います。
HostName	サービス要求先ノードのホスト名の設定および参照を行います。
NodeID	サービス要求先のノード識別子の設定および参照を行います。
PortNo	サービス要求先ノードのネームサービスのポート番号の設定および参照を行います。

プロパティの詳細

●Flags

説明

フラグの設定および参照を行います。

通常，TP1ServerFlags.DCNOFLAGS を設定します。

portNo パラメタの設定を有効にする場合は，TP1ServerFlags.DCRPC_NAMPORT を設定してください。

参照時に TP1ServerFlags.DCNOFLAGS であった場合，portNo パラメタに値が設定されていても有効になりません。

宣言

【C#の場合】

```
public int Flags {get; set;}
```

【Visual Basic の場合】

```
Public Property Flags As Integer
```

【J#の場合】

```
public int get_Flags();
public void set_Flags(int);
```

例外

なし

●HostName

説明

サービス要求先ノードのホスト名の設定および参照を行います。

ホスト名を検索のキーとして使用しない場合は，空文字列または null（Visual Basic の場合は Nothing）を指定してください。

参照時にホスト名が空文字列または null（Visual Basic の場合は Nothing）であった場合，ホスト名を検索のキーとして使用できません。

宣言

【C#の場合】

```
public string HostName {get; set;}
```

【Visual Basic の場合】

```
Public Property HostName As String
```

【J#の場合】

```
public System.String get_HostName();
public void set_HostName(System.String);
```

例外

なし

●NodeID

説明

サービス要求先のノード識別子の設定および参照を行います。

ノード識別子を検索のキーにしない場合には，空文字列または null（Visual Basic の場合は Nothing）を設定します。

参照時にノード識別子が空文字列または null（Visual Basic の場合は Nothing）であった場合，ノード識別子を検索のキーとして使用できません。

宣言

【C#の場合】

```
public string NodeID {get; set;}
```

【Visual Basic の場合】

```
Public Property NodeID As String
```

【J#の場合】

```
public System.String get_NodeID();
public void set_NodeID(System.String);
```

例外

なし

●PortNo

説明

サービス要求先ノードのネームサービスのポート番号の設定および参照を行います。

ホスト名を検索のキーとして使用しない場合は，0 を指定してください。

参照時にポート番号が 0 であった場合，ホスト名を検索のキーとして使用できません。

宣言

【C#の場合】

```
public int PortNo {get; set;}
```

【Visual Basic の場合】

```
Public Property PortNo As Integer
```

【J#の場合】

```
public int get_PortNo();
public void set_PortNo(int);
```

例外

なし

Rts

Rts の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Rts
```

説明

Rts クラスは，リアルタイム統計情報を取得する場合に使用するメソッドを提供します。

メソッドの一覧

名称	説明
PutUTrace(System.Int32, System.Int32)	UAP 内の任意の区間で，eventID に設定した項目の実行時間および実行回数をリアルタイム統計情報として取得します。

メソッドの詳細

●PutUTrace

説明

UAP 内の任意の区間で，eventID に設定した項目の実行時間および実行回数をリアルタイム統計情報として取得します。

宣言

【C#の場合】

```
public static void PutUTrace(
  int eventID,
  int flags
);
```

【Visual Basic の場合】

```
Public Shared Sub PutUTrace( _
  ByVal eventID As Integer, _
  ByVal flags As Integer _
)
```

【J#の場合】

```
public static void PutUTrace(
  int eventID,
```

```
  int flags
);
```

パラメタ

eventID

取得するリアルタイム統計情報の項目 ID を設定します。ID に使用できる値は，1000000～2147483647 です。

flags

PutUTrace メソッドで実行する処理を設定します。

- TP1ServerFlags.DCRTS_START
  eventID に設定した項目 ID の実行時間の計測を開始します。このフラグを設定して PutUTrace メソッドを呼び出した時点では，リアルタイム統計情報を取得しません。
- TP1ServerFlags.DCRTS_END
  eventID に設定した項目 ID の実行時間を取得して，計測を終了します。
- TP1ServerFlags.DCNOFLAGS
  eventID に設定した項目 ID の実行回数だけを取得します。実行時間は 0 秒となります。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合だけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCRTSER_PARAM	引数に設定した値に誤りがあります。
DCRTSER_PROTO	Rpc クラスの Open メソッドを呼び出していません。 すでに実行時間の計測を開始している項目 ID を eventID に設定して，flags に TP1ServerFlags.DCRTS_START を設定した PutUTrace メソッドを呼び出しました。 実行時間の計測を開始していない項目 ID を eventID に設定して flags に TP1ServerFlags.DCRTS_END を設定した PutUTrace メソッドを呼び出しました。
DCRTSER_ITEM_OVER	取得項目の数がリアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。
DCRTSER_ITEM_OVER_SRV	サーバ単位の取得項目の数が，リアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。このエラーコードが返った場合，サービス単位，またはサービス以外の処理の統計情報は取得されています。
DCRTSER_ITEM_OVER_SVC	サービス単位，またはサービス以外の処理での取得項目の数が，リアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。このエラーコードが返った場合，サーバ単位の統計情報は取得されています。
DCRTSER_NOMEM	プロセスメモリが不足したため，処理を実行できません。
DCRTSER_RTS_NOT_START	リアルタイム統計情報サービスが開始していません。
DCRTSER_NOENTRY	PutUTrace メソッドを呼び出したサーバ，およびサービスがリアルタイム統計情報の取得対象に登録されていません。
DCRTSER_VERSION	UAP が，現在稼働しているリアルタイム統計情報サービスでは稼働できないバージョンのライブラリと結合しています。

注意事項

- PutUTrace メソッドでは，システム全体のリアルタイム統計情報は取得できません。
- マルチサーバを使用している UAP では，同じ呼び出し元サービスおよび同じ eventID を設定した PutUTrace メソッドを複数プロセスから同時に呼び出した場合，プロセスによっては統計情報が取得されないことがあります。これは，統計情報の取得処理では排他制御がされないため，書き込み処理が同時に行われることが要因です。
- PutUTrace メソッドは，UAP トレースを取得しません。
- flags に TP1ServerFlags.DCRTS_START を指定した PutUTrace メソッドが例外を返し，エラーコードが DCRTSER_RTS_NOT_START または DCRTSER_NOENTRY の場合があります。その場合に，同じ eventID で flags に TP1ServerFlags.DCRTS_END を指定した PutUTrace メソッドを呼び出すまでの間にリアルタイム統計情報サービスを開始して，呼び出し元の UAP を取得対象に追加したとき，PutUTrace メソッドは例外を返し，エラーコードは DCRTSER_PROTO になります。

ShortArrayHolder

ShortArrayHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.ShortArrayHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

ShortArrayHolder クラスは，System.Int16 配列を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public short[] Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As Short()
```

【J#の場合】

```
public short[] get_Value();
public void set_Value(short[]);
```

例外

なし

ShortHolder

ShortHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.ShortHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

ShortHolder クラスは，System.Int16 値を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public short Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As Short
```

【J#の場合】

```
public short get_Value();
public void set_Value(short);
```

例外

なし

SPPBase

SPPBase の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.SPPBase
```

説明

SPPBase クラスは，SPP.NET 実装の抽象クラスです。

メソッドの一覧

名称	説明
FinalizeSPP()	このメソッドは，SPP.NET 実装の抽象クラスの解放時に呼び出されます。
InitializeSPP()	このメソッドは，SPP.NET 実装の抽象クラスの初期化時に呼び出されます。

メソッドの詳細

●FinalizeSPP

説明

このメソッドは，SPP.NET 実装の抽象クラスの解放時に呼び出されます。

SPP.NET 終了時の解放処理を追加する場合は，このメソッドをオーバライドします。

宣言

【C#の場合】

```
public virtual void FinalizeSPP(
);
```

【Visual Basic の場合】

```
Public Overridable Sub FinalizeSPP( _
)
```

【J#の場合】

```
public void FinalizeSPP(
);
```

パラメタ

なし

戻り値

なし

例外

なし

●InitializeSPP

説明

このメソッドは，SPP.NET 実装の抽象クラスの初期化時に呼び出されます。

SPP.NET 起動時の初期化処理を追加する場合は，このメソッドをオーバライドします。

宣言

【C#の場合】

```
public virtual void InitializeSPP(
);
```

【Visual Basic の場合】

```
Public Overridable Sub InitializeSPP( _
)
```

【J#の場合】

```
public void InitializeSPP(
);
```

パラメタ

なし

戻り値

なし

例外

なし

StringArrayHolder

StringArrayHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.StringArrayHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

StringArrayHolder クラスは，System.String 配列を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public string[] Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As String()
```

【J#の場合】

```
public System.String[] get_Value();
public void set_Value(System.String[]);
```

例外

なし

StringHolder

StringHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.StringHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

StringHolder クラスは，System.String を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public string Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As String
```

【J#の場合】

```
public System.String get_Value();
public void set_Value(System.String);
```

例外

なし

Tam 【TP1/Server Base】

Tam の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Tam
```

説明

Tam クラスは，TAM ファイルサービス機能を使用するメソッドを提供します。

Tam クラスのメソッドは，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，Tam クラスのメソッドは使えません。

メソッドの一覧

名称	説明
Close(System.Int32)	TAM テーブルをクローズします。
Delete(System.Int32, Hitachi.OpenTP1.Server.TamKeyTable&, System.Int32, System.Byte[], System.Int32, System.Int32)	キー値に示すレコードを，TAM テーブルから削除します。
GetInfo(System.String)	TAM テーブルの状態を取得します。
GetStatus(System.String, Hitachi.OpenTP1.Server.TamStatusTable&)	TAM テーブルの情報を，TamStatusTable 構造体に返します。
Open(System.String, System.Int32)	TAM テーブルをオープンします。
Read(System.Int32, Hitachi.OpenTP1.Server.TamKeyTable&, System.Int32, System.Byte[], System.Int32, System.Int32)	flags に設定した検索種別に従って，TAM テーブル上のレコードを参照または更新の目的で入力します。
ReadCancel(System.Int32, Hitachi.OpenTP1.Server.TamKeyTable&, System.Int32)	トランザクション内で，Tam.Read メソッドを使って排他を掛けた参照目的の入力，および更新目的の入力を取り消し，レコード排他を解除します。
Rewrite(System.Int32, Hitachi.OpenTP1.Server.TamKeyTable&, System.Int32, System.Byte[], System.Int32)	Tam.Read メソッドで入力したレコードを，更新して出力します。
Write(System.Int32, Hitachi.OpenTP1.Server.TamKeyTable&,	キー値に示すレコードを，TAM テーブル上に更新または追加します。

名称	説明
System.Int32, System.Byte[], System.Int32, System.Int32)	

メソッドの詳細

●Close

説明

TAM テーブルをクローズします。Close メソッドを呼び出したあとは，tableID パラメタに設定したテーブル記述子は使えません。

Close メソッドが例外応答した場合は，このメソッド内で確保した資源はすべて解放して，メソッドを呼び出す前の状態に戻ります。

トランザクション外で Tam.Open メソッドを呼び出した場合，Close メソッドもトランザクション外で呼び出してください。

トランザクション内で Tam.Open メソッドを呼び出した場合は，Close メソッドもトランザクション内で呼び出してください。また，トランザクション終了時までに Close メソッドを呼び出さなかった場合は，同期点で TAM テーブルがクローズされます。

サービスメソッドの中で，トランザクション外のオープンに対する Close メソッドを呼び出す場合，クローズさせる TAM テーブルにアクセスしている同一プロセス上のトランザクションは，すべて終了させてください。このことに関するエラーチェックはしないので，トランザクションを終了させないでこのメソッドを呼び出した場合の動作については保証しません。

宣言

【C#の場合】

```
public static void Close(
    int tableID
);
```

【Visual Basic の場合】

```
Public Shared Sub Close( _
    ByVal tableID As Integer _
)
```

【J#の場合】

```
public static void Close(
    int tableID
);
```

パラメタ

tableID

クローズする TAM テーブルの，テーブル記述子を設定します。テーブル記述子は，Tam.Open メソッドで返された値です。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTAMER_PARAM_TID	tableID に設定したテーブル記述子が間違っています。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_TRNOPN	Tam.Open メソッドはトランザクション外で呼び出しています。
DCTAMER_NOOPEN	TAM テーブルがオープン状態ではありません。
DCTAMER_MEMORY	メモリが不足しました。
DCTAMER_IO	入出力エラーが発生しました。

●Delete

説明

キー値に示すレコードを，TAM テーブルから削除します。削除するレコードをバッファに退避することもできます。ただし，このメソッドが例外応答した場合には，バッファの内容は保証されません。

レコード排他で TAM テーブルがオープンしている場合，更新排他でテーブル排他を確保します。

Delete メソッドが例外応答した場合は，このメソッド内で設定した資源はすべて解放して，メソッドを呼び出す前の状態に戻ります。ただし，メソッドを呼び出す前に参照排他で確保されていた TAM テーブルを削除しようとした場合は更新排他となり，参照排他には戻りません。

複数のレコードを設定して削除する場合，それらのレコードのうち一つでもエラーが起こったときは，このメソッドで設定した全レコードの処理をエラーとして，メソッドを呼び出す前の状態に戻ります。

宣言

【C#の場合】

```
public static void Delete(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo,
    byte[] buffer,
    int bufferSize,
    int flags
);
```

【Visual Basic の場合】

```
Public Shared Sub Delete( _
    ByVal tableID As Integer, _
    ByVal key() As Hitachi.OpenTP1.Server.TamKeyTable, _
    ByVal keyNo As Integer, _
    ByVal buffer() As Byte, _
    ByVal bufferSize As Integer, _
    ByVal flags As Integer _
)
```

【J#の場合】

```
public static void Delete(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo,
    ubyte[] buffer,
    int bufferSize,
    int flags
);
```

パラメタ

tableID

レコードを削除する TAM テーブルの，テーブル記述子を設定します。テーブル記述子は，Tam.Open メソッドで返された値です。

key

削除するレコードのキー値を持つ，構造体の配列を設定します。

keyNo

要求レコード数（key で設定する構造体の数）を設定します。

buffer

削除するレコードをバッファに退避する場合に，そのバッファを設定します。flags に TP1ServerFlags.DCTAM_NOOUTREC（削除するレコードを退避しません）を設定した場合は，この設定は無効です。

bufferSize

削除するレコードをバッファに退避する場合に，そのバッファ長を設定します。返却バッファ長は，「レコード長×要求レコード数」以上にします。flags に TP1ServerFlags.DCTAM_NOOUTREC（削除するレコードを退避しません）を設定した場合は，この設定は無効です。

flags

レコードのアクセス種別，資源の競合が起こった場合の排他解除待ち種別を，次の形式で設定します。

```
{TP1ServerFlags.DCTAM_NOOUTREC|
TP1ServerFlags.DCTAM_OUTREC}
 〔|{TP1ServerFlags.DCTAM_WAIT|
TP1ServerFlags.DCTAM_NOWAIT}〕
```

- フラグ 1

  フラグ 1 には，レコードアクセス種別を指定します。

  レコードのアクセス種別の設定は省略できません。次のアクセス種別のどちらかを指定してください。

  TP1ServerFlags.DCTAM_NOOUTREC … 削除するレコードを退避しません。

  TP1ServerFlags.DCTAM_OUTREC … 削除するレコードを退避します。

- フラグ 2

  フラグ 2 には，排他解除待ち種別を指定します。

  次の排他解除待ち種別のどちらかを指定してください。排他解除待ち種別を省略した場合は，排他解除待ちをしないで例外応答します。

  TP1ServerFlags.DCTAM_WAIT … 排他解除待ちをします。

  TP1ServerFlags.DCTAM_NOWAIT … 待たないで，例外応答します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

"OpenTP1 提供関数実行時にエラーが発生しました。"
それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTAMER_PARAM_TID	tableID に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	key に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	keyNo に設定した値が間違っています。
DCTAMER_PARAM_BFA	buffer に設定した値が間違っています。
DCTAMER_PARAM_BFS	bufferSize に設定したバッファ長が短過ぎます。
DCTAMER_PARAM_FLG	flags に設定した値が間違っています。
DCTAMER_NOTTAM	tableID に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_RMTBL	TAM テーブルが削除されています。
DCTAMER_NOLOAD	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	TAM テーブルがオープン状態ではありません。
DCTAMER_LOGHLD	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	TAM テーブルが障害閉塞状態です。
DCTAMER_ACSATL	TAM サービス定義で指定した TAM テーブルのアクセス形態では実行できません。
DCTAMER_NOREC	指定されたレコードは存在しません。

エラーコード	説明
DCTAMER_LOCK	排他エラーが発生しました。flags に TP1ServerFlags.DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間でタイムアウトが発生したため，資源を確保できませんでした。
DCTAMER_DLOCK	デッドロックが起こりました。
DCTAMER_TBLVR	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_RECOBS	レコードが破壊されています。
DCTAMER_TRNNUM	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	TAM ファイルに対するアクセス権がありません。
DCTAMER_MEMORY	メモリが不足しました。
DCTAMER_IO	入出力エラーが発生しました。
DCTAMER_TMERR	トランザクションサービスでエラーが発生しました。
DCTAMER_ACCESS	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。Delete メソッドを呼び出した UAP には，アクセス権限がありません。

●GetInfo

説明

TAM テーブルの状態を取得します。取得する TAM テーブルの状態を次に示します。

- オープン状態
- クローズ状態
- 論理閉塞状態
- 障害閉塞状態

GetInfo メソッドは，トランザクション内でもトランザクション外でも呼び出せます。

GetInfo メソッドを呼び出したプロセスで Tam.Open メソッドを呼び出していなくても，設定した TAM テーブルにほかのプロセスで Tam.Open メソッドを呼び出している場合は，TAM テーブルはオープン状態としてリターンします。

宣言

【C#の場合】

```
public static int GetInfo(
    string tableName
);
```

【Visual Basic の場合】

```
Public Shared Function GetInfo( _
    ByVal tableName As String _
) As Integer
```

【J#の場合】

```
public static int GetInfo(
    System.String tableName
);
```

パラメタ

tableName

状態を取得する TAM テーブル名を設定します。TAM テーブル名は 32 文字以内の半角英数字文字列で設定してください。

戻り値

- TP1ServerValues.DCTAM_STS_OPN
  オープン状態であることを示します。
- TP1ServerValues.DCTAM_STS_CLS
  クローズ状態であることを示します。
- TP1ServerValues.DCTAM_STS_LHLD
  論理閉塞状態であることを示します。
- TP1ServerValues.DCTAM_STS_OHLD
  障害閉塞状態であることを示します。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。

- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTAMER_PARAM_TBL	tableName に設定した値が間違っています。
DCTAMER_UNDEF	TAM テーブルが定義されていません。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_TAMVR	UAP が，現在稼働している TAM サービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_NO_ACL	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。
DCTAMER_ACCESS	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。GetInfo メソッドを呼び出した UAP には，アクセス権限がありません。

●GetStatus

説明

TAM テーブルの情報が，TamStatusTable 構造体で返されます。リターンする値を次に示します。

- TAM ファイル名
- TAM テーブルの状態
- 使用中のレコード数
- 最大レコード数
- インデクス種別
- アクセス形態
- ローディング契機
- TAM レコード長

- キー長
- キー開始位置
- セキュリティ属性

宣言

【C#の場合】

```
public static Hitachi.OpenTP1.Server.TamStatusTable GetStatus(
    string tableName
);
```

【Visual Basic の場合】

```
Public Shared Function GetStatus( _
    ByVal tableName As String, _
) As Hitachi.OpenTP1.Server.TamStatusTable _
```

【J#の場合】

```
public static Hitachi.OpenTP1.Server.TamStatusTable GetStatus(
    System.String tableName
);
```

パラメタ

tableName

情報を取得する TAM テーブル名を設定します。TAM テーブル名は，32 文字以内の半角英数字文字列で設定します。

戻り値

TAM テーブルの情報が，TamStatusTable 構造体で返されます。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名

  例外が発生したクラス名が出力されます。
- メソッド名

  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）

  例外が発生する原因となった引数名が出力されます。

- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTAMER_PARAM_TBL	tableName に設定した値が間違っています。
DCTAMER_NOTTAM	tableName に設定した名称は，TAM テーブルではありません。
DCTAMER_UNDEF	TAM テーブルが定義されていません。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_TBLVR	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_TAMVR	UAP が，現在稼働している TAM サービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_OPENNUM	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	スペシャルファイルに対するアクセス権がありません。
DCTAMER_MEMORY	メモリが不足しました。
DCTAMER_IO	入出力エラーが発生しました。
DCTAMER_NO_ACL	情報を取得しようとした TAM テーブルは，セキュリティ機能で保護されています。該当する TAM テーブルに対する ACL がありません。
DCTAMER_ACCESS	情報を取得しようとした TAM テーブルは，セキュリティ機能で保護されています。GetStatus メソッドを呼び出した UAP には，アクセス権限がありません。

●Open

説明

TAM テーブルをオープンします。Open メソッドは，トランザクション内でもトランザクション外でも呼び出せます。

トランザクション内で呼び出して，排他種別にテーブル排他を設定した場合，更新排他でテーブル排他を確保します。

Open メソッドが例外応答した場合は，このメソッド内で確保した資源はすべて解放して，メソッドを呼び出す前の状態に戻ります。

宣言

【C#の場合】

```
public static int Open(
    string tableName,
    int flags
);
```

【Visual Basic の場合】

```
Public Shared Function Open( _
    ByVal tableName As String, _
    ByVal flags As Integer _
) As Integer
```

【J#の場合】

```
public static int Open(
    System.String tableName,
    int flags
);
```

パラメタ

tableName

オープンする TAM テーブル名を設定します。TAM テーブル名は 32 文字以内の半角英数字文字列で設定してください。

flags

テーブル排他を掛けるかレコード排他を掛けるかを，次の形式で設定します。

```
 〔{TP1ServerFlags.DCTAM_TBL_EXCLUSIVE
 〔|{TP1ServerFlags.DCTAM_WAIT|
TP1ServerFlags.DCTAM_NOWAIT}〕 |
TP1ServerFlags.DCTAM_REC_EXCLUSIVE}〕
```

- フラグ 1

  フラグ 1 には，排他種別を指定します。

  テーブル排他の場合は，更新排他で確保します。レコード排他の場合は，レコードのアクセスメソッド内で排他を確保します。Open メソッドをトランザクション外で呼び出す場合は，テーブル排他は指定できません。

  次の排他種別のどちらかを指定してください。このフラグの指定を省略した場合は，TP1ServerFlags.DCTAM_REC_EXCLUSIVE が仮定されます。

  TP1ServerFlags.DCTAM_TBL_EXCLUSIVE … テーブル排他です。

  TP1ServerFlags.DCTAM_REC_EXCLUSIVE … レコード排他です。

- フラグ 2

  テーブル排他の場合は，フラグ 2 に資源の競合が起こったときの排他解除待ち種別を指定します。

  次の排他解除待ち種別のどちらかを指定してください。このフラグの指定を省略した場合は，TP1ServerFlags.DCTAM_NOWAIT が仮定されます。

TP1ServerFlags.DCTAM_WAIT … 排他解除待ちをします。

TP1ServerFlags.DCTAM_NOWAIT … 待たないで，例外応答します。

戻り値

テーブル記述子を示します。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名

  例外が発生したクラス名が出力されます。
- メソッド名

  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）

  例外が発生する原因となった引数名が出力されます。
- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTAMER_PARAM_TBL	tableName に設定した値が間違っています。
DCTAMER_PARAM_FLG	flags に設定した値が間違っています。
DCTAMER_NOTTAM	tableName に設定したテーブルは TAM テーブルではありません。
DCTAMER_UNDEF	TAM テーブルが定義されていません。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_NOLOAD	TAM テーブルがロードされていません。
DCTAMER_OPENED	TAM テーブルがオープン済みです。

エラーコード	説明
DCTAMER_LOGHLD	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	TAM テーブルが障害閉塞状態です。
DCTAMER_LOCK	排他エラーが発生しました。flags に TP1ServerFlags.DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間でタイムアウトが発生したため，資源を確保できませんでした。
DCTAMER_DLOCK	デッドロックが起こりました。
DCTAMER_TBLVR	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_TAMVR	UAP が，現在稼働している TAM サービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_RECOBS	レコードが破壊されています。
DCTAMER_TRNNUM	TAM サービスで管理できるトランザクション数を超えています。
DCTAMER_OPENNUM	キャラクタ型スペシャルファイルのオープン数の制限値を超えています。
DCTAMER_ACCESSS	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	TAM ファイルに対するアクセス権がありません。
DCTAMER_MEMORY	メモリが不足しました。
DCTAMER_IO	入出力エラーが発生しました。
DCTAMER_TMERR	トランザクションサービスでエラーが発生しました。
DCTAMER_NO_ACL	オープンしようとした TAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

●Read

説明

flags に設定した検索種別に従って，TAM テーブル上のレコードを参照または更新の目的で入力します。

参照目的の入力で排他を掛ける場合，参照排他でテーブル排他とレコード排他を確保します。レコード排他でオープンした TAM テーブルを更新目的で入力する場合は，参照排他でテーブル排他を確保し，更新排他でレコード排他を確保します。

Read メソッドが例外応答した場合は，このメソッド内で設定した資源は すべて解放して，メソッドを呼び出す前の状態に戻ります。ただし，メソッドを呼び出す前に参照排他で確保されていたレコードを更新目的で入力した場合は，更新排他となり参照排他には戻りません。また，例外応答した場合には，バッファの内容は保証できません。

複数のレコードを設定して入力する場合，それらのレコードのうち一つでもエラーが発生したときは，このメソッドで設定した全レコードの処理をエラーとします。

宣言

【C#の場合】

```
public static void Read(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo,
    byte[] buffer,
    int bufferSize,
    int flags
);
```

【Visual Basic の場合】

```
Public Shared Sub Read( _
    ByVal tableID As Integer, _
    ByVal key() As Hitachi.OpenTP1.Server.TamKeyTable, _
    ByVal keyNo As Integer, _
    ByVal buffer() As Byte, _
    ByVal bufferSize As Integer, _
    ByVal flags As Integer _
)
```

【J#の場合】

```
public static void Read(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo,
    ubyte[] buffer,
    int bufferSize,
    int flags
);
```

パラメタ

tableID

レコードを入力する TAM テーブルの，テーブル記述子を設定します。テーブル記述子は Tam.Open メソッドで返された値です。

key

レコードを検索するための，キー値を持つ構造体の配列を設定します。

keyNo

要求レコード数（key で設定する構造体の数）を設定します。

buffer

レコードを入力するバッファを設定します。

bufferSize

レコードを入力するバッファの長さを設定します。バッファ長は「レコード長×要求レコード数」以上にします。

flags

レコードの検索種別，アクセス種別，参照目的の排他のときの排他要否種別を設定します。また，排他を掛ける場合に，資源の競合が起こったときの排他解除待ち種別を次の形式で設定します。

```
{フラグ1}|{TP1ServerFlags.DCTAM_REFERENCE
 〔|{TP1ServerFlags.DCTAM_EXCLUSIVE|
TP1ServerFlags.DCTAM_NOEXCLUSIVE}〕 |
TP1ServerFlags.DCTAM_MODIFY}
 〔|{TP1ServerFlags.DCTAM_WAIT|
TP1ServerFlags.DCTAM_NOWAIT}〕
```

- フラグ 1

  フラグ 1 には，次のレコードの検索種別のうちどれか一つを指定してください。

  レコードの検索種別の設定は省略できません。また，検索種別は重複して設定できません。

  TP1ServerFlags.DCTAM_EQLSRC … 'キー値＝'を検索します（ハッシュ，ツリー）。

  TP1ServerFlags.DCTAM_GRTEQLSRC … 'キー値＜＝'を検索します（ツリー）。

  TP1ServerFlags.DCTAM_GRTSRC … 'キー値＜'を検索します（ツリー）。

  TP1ServerFlags.DCTAM_LSSEQLSRC … 'キー値＞＝'を検索します（ツリー）。

  TP1ServerFlags.DCTAM_LSSSRC … 'キー値＞'を検索します（ツリー）。

  TP1ServerFlags.DCTAM_FIRSTSRC … 先頭から検索します（ハッシュ）。

  TP1ServerFlags.DCTAM_NEXTSRC … 設定したキー値の，次のレコードから検索します（ハッシュ）。

- フラグ 2

  フラグ 2 には，レコードアクセス種別を指定します。

  レコードのアクセス種別の指定は省略できません。

  次のアクセス種別のどちらかを指定してください。

  TP1ServerFlags.DCTAM_REFERENCE … 参照目的の排他です。

  TP1ServerFlags.DCTAM_MODIFY … 更新目的の排他です。

- フラグ 3

  参照目的の排他の場合は，フラグ 3 に排他するかどうかを指定します。

  次のどちらかを指定してください。このフラグの設定を省略した場合は，TP1ServerFlags.DCTAM_NOEXCLUSIVE が仮定されます。

  TP1ServerFlags.DCTAM_EXCLUSIVE … 排他します。

  TP1ServerFlags.DCTAM_NOEXCLUSIVE … 排他をしません。

- フラグ 4

  フラグ 4 には，排他解除待ち種別を指定します。

  次のどちらかを指定してください。このフラグの設定を省略した場合は，TP1ServerFlags.DCTAM_NOWAIT が仮定されます。

  TP1ServerFlags.DCTAM_WAIT … 排他解除待ちをします。

  TP1ServerFlags.DCTAM_NOWAIT … 待たないで，例外応答します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTAMER_PARAM_TID	tableID に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	key に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	keyNo に設定した値が間違っています。
DCTAMER_PARAM_BFA	buffer に設定した値が間違っています。
DCTAMER_PARAM_BFS	bufferSize に設定したバッファ長が短過ぎます。
DCTAMER_PARAM_FLG	flags に設定した値が間違っています。
DCTAMER_NOTTAM	tableID に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_RMTBL	TAM テーブルが削除されています。

エラーコード	説明
DCTAMER_NOLOAD	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	TAM テーブルがオープン状態ではありません。
DCTAMER_LOGHLD	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	TAM テーブルが障害閉塞状態です。
DCTAMER_IDXTYP	TAM テーブルファイルの初期作成で設定した TAM テーブルのインデクス種別では実行できません。
DCTAMER_ACSATL	TAM サービス定義で設定した TAM テーブルのアクセス形態では実行できません。
DCTAMER_NOREC	flags に設定した検索条件を満たすレコードがありません。
DCTAMER_LOCK	排他エラーが発生しました。flags に TP1ServerFlags.DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間でタイムアウトが発生したため，資源を確保できませんでした。
DCTAMER_DLOCK	デッドロックが起こりました。
DCTAMER_TBLVR	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_RECOBS	レコードが破壊されています。
DCTAMER_TRNNUM	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	TAM ファイルに対するアクセス権がありません。
DCTAMER_MEMORY	メモリが不足しました。
DCTAMER_IO	入出力エラーが発生しました。
DCTAMER_TMERR	トランザクションサービスでエラーが発生しました。
DCTAMER_ACCESS	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。Read メソッドを呼び出した UAP には，アクセス権限がありません。

●ReadCancel

説明

トランザクション内で，Tam.Read メソッドを使って排他を掛けた参照目的の入力，および更新目的の入力を取り消し，レコード排他を解除します。

更新，または追加済みのレコードには，排他を掛けた参照目的の入力を取り消すことはできません。また，Tam.Rewrite メソッドで更新したレコードの，更新目的の入力の取り消しはできません。

更新，または追加済みのレコード，およびテーブル排他でオープンした，TAM テーブル上のレコードに対する更新目的の入力の取り消しでは，排他を解除しません。

ReadCancel メソッドで入力を取り消したあとも，トランザクションが終了するまでは，入力した TAM テーブルに対して，ほかのトランザクションからレコードの追加，削除はできません。

ReadCancel メソッドが例外を返した場合，このメソッドで解放した資源は再確保しないで，メソッドを呼び出す前の状態には戻しません。また，複数のレコードを設定してアクセスを要求した場合，それらのレコードのうち，一つでもエラーが発生したら処理を中断して，例外応答します。

宣言

【C#の場合】

```
public static void ReadCancel(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo
);
```

【Visual Basic の場合】

```
Public Shared Sub ReadCancel( _
    ByVal tableID As Integer, _
    ByVal key() As Hitachi.OpenTP1.Server.TamKeyTable, _
    ByVal keyNo As Integer _
)
```

【J#の場合】

```
public static void ReadCancel(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo
);
```

パラメタ

tableID

レコードの入力を取り消す TAM テーブルの，テーブル記述子を設定します。テーブル記述子は Tam.Open メソッドで返された値です。

key

入力を取り消すレコードの，キー値を持つ構造体の配列を設定します。

keyNo

要求レコード数（key で設定する構造体の数）を設定します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTAMER_PARAM_TID	tableID に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	key に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	keyNo に設定した値が間違っています。
DCTAMER_NOTTAM	tableID に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_RMTBL	TAM テーブルが削除されています。
DCTAMER_NOLOAD	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	TAM テーブルがオープン状態ではありません。
DCTAMER_LOGHLD	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	TAM テーブルが障害閉塞状態です。
DCTAMER_NOREC	指定されたレコードは存在しません。
DCTAMER_SEQENCE	Tam.Read メソッドを呼び出していません。
DCTAMER_EXWRITE	tableID に設定したテーブル記述子は Tam.Write メソッドで更新または追加したレコードです。

エラーコード	説明
DCTAMER_EXREWRT	tableID に設定したテーブル記述子は，Tam.Rewrite メソッドで更新済みです。
DCTAMER_TBLVR	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_TRNNUM	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	TAM テーブルファイルに対するアクセス権がありません。
DCTAMER_MEMORY	メモリが不足しました。
DCTAMER_IO	入出力エラーが発生しました。
DCTAMER_TMERR	トランザクションサービスでエラーが発生しました。

●Rewrite

説明

Tam.Read メソッドで入力したレコードを，更新して出力します。

更新目的の入力の Tam.Read メソッドを 1 回呼び出せば，その後トランザクションの同期点まで，何度でも Rewrite メソッドを使えます。ただし，Tam.Delete メソッド，Tam.ReadCancel メソッドを呼び出したあとには，Rewrite メソッドを使えません。

Rewrite メソッドが例外応答した場合は，このメソッド内で設定した資源はすべて解放して，メソッドを呼び出す前の状態に戻ります。

複数のレコードを設定して更新を要求する場合，それらのレコードのうち一つでもエラーが発生したときは，このメソッドで設定した全レコードの処理をエラーとします。

更新データ内のキー値の格納位置，およびキー領域長は，TAM テーブルファイルの初期作成時の tamcre コマンドに設定した値です。

TAM テーブルファイルの初期作成時，データ部にキー値を付けている（tamcre コマンドに-s オプションを指定していない）場合は，Rewrite メソッドに設定したキー値が更新データ内になければ，例外応答します。また，データ部にキー値を付けていない（tamcre コマンドに-s オプションを指定）場合は，更新データの内容をチェックしません。

宣言

【C#の場合】

```
public static void Rewrite(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo,
    byte[] data,
```

```
    int dataSize
);
```

【Visual Basic の場合】

```
Public Shared Sub Rewrite( _
    ByVal tableID As Integer, _
    ByVal key() As Hitachi.OpenTP1.Server.TamKeyTable, _
    ByVal keyNo As Integer, _
    ByVal data() As Byte, _
    ByVal dataSize As Integer _
)
```

【J#の場合】

```
public static void Rewrite(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo,
    ubyte[] data,
    int dataSize
);
```

パラメタ

tableID

レコードを更新する TAM テーブルの，テーブル記述子を設定します。テーブル記述子は Tam.Open メソッドで返された値です。

key

更新するレコードの，キー値を持つ構造体の配列を設定します。

keyNo

要求レコード数（key で設定する構造体の数）を設定します。

data

更新データを設定します。

dataSize

更新データ長を設定します。更新データ長は，「レコード長×要求レコード数」以上にしてください。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）
  例外が発生する原因となった引数名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTAMER_PARAM_TID	tableID に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	key に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	keyNo に設定した値が間違っています。
DCTAMER_PARAM_DTA	data に設定した値が間違っています。
DCTAMER_PARAM_DTS	dataSize に設定したデータ長が短過ぎます。
DCTAMER_NOTTAM	tableID に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_RMTBL	TAM テーブルが削除されています。
DCTAMER_NOLOAD	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	TAM テーブルがオープン状態ではありません。
DCTAMER_LOGHLD	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	TAM テーブルが障害閉塞状態です。
DCTAMER_NOREC	指定されたレコードは存在しません。
DCTAMER_SEQENCE	Tam.Read メソッドを呼び出していません。
DCTAMER_TBLVR	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。

エラーコード	説明
DCTAMER_RECOBS	レコードが破壊されています。
DCTAMER_TRNNUM	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	TAM テーブルファイルに対するアクセス権がありません。
DCTAMER_MEMORY	メモリが不足しました。
DCTAMER_IO	入出力エラーが発生しました。
DCTAMER_TMERR	トランザクションサービスでエラーが発生しました。

●Write

説明

キー値に示すレコードを，TAM テーブル上に更新または追加します。

レコード排他で TAM テーブルがオープンしている場合，次のように排他を確保します。

- アクセス種別が「更新」の場合 （flags に TP1ServerFlags.DCTAM_WRITE を設定）
  参照排他でテーブル排他を確保して，更新排他でレコード排他を確保します。ただし，TAM サービス定義の「アクセス時のテーブル排他モード」に，「テーブル排他なしモード」を指定している場合は，アクセス形態が「参照型」「追加・削除できない更新型」のテーブルには，テーブル排他を確保しません。
- アクセス種別が「更新または追加」，「追加」の場合（flags に TP1ServerFlags.DCTAM_WRTADD，または TP1ServerFlags.DCTAM_ADD を設定）
  更新排他でテーブル排他を確保します。

Write メソッドが例外を返した場合，このメソッド内で確保した資源はすべて解放して，メソッドを呼び出す前の状態に戻します。ただし，このメソッドを呼び出す前に，参照排他で確保されていた TAM テーブルを更新または追加した場合は，更新排他となり参照排他には戻りません。

複数のレコードを設定して更新または追加する場合，それらのレコードのうち一つでもエラーが発生したら，このメソッドで設定した全レコードの処理をエラーとします。

更新または追加するデータ内のキー値の格納位置，およびキー領域長は，TAM テーブルファイルの初期作成時の tamcre コマンドに設定した値です。

TAM テーブルファイルの初期作成時，データ部にキー値を付けている（tamcre コマンドに-s オプションを指定していない）場合は，Write メソッドに設定したキー値が更新または追加するデータ内になければ，例外応答します。また，データ部にキー値を付けていない（tamcre コマンドに-s オプションを指定）場合は，更新または追加するデータの内容をチェックしません。

宣言

【C#の場合】

```
public static void Write(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo,
    byte[] data,
    int dataSize,
    int flags
);
```

【Visual Basic の場合】

```
Public Shared Sub Write( _
    ByVal tableID As Integer, _
    ByVal key() As Hitachi.OpenTP1.Server.TamKeyTable, _
    ByVal keyNo As Integer, _
    ByVal data() As Byte, _
    ByVal dataSize As Integer, _
    ByVal flags As Integer _
)
```

【J#の場合】

```
public static void Write(
    int tableID,
    Hitachi.OpenTP1.Server.TamKeyTable[] key,
    int keyNo,
    ubyte[] data,
    int dataSize,
    int flags
);
```

パラメタ

tableID

レコードを更新または追加する TAM テーブルの，テーブル記述子を設定します。テーブル記述子は Tam.Open メソッドで返された値です。

key

更新または追加するレコードのキー値を持つ，構造体の配列を設定します。

keyNo

要求レコード数（key で設定する構造体の数）を設定します。

data

更新または追加するデータを設定します。

dataSize

更新または追加するデータ長を設定します。更新または追加するデータ長は「レコード長×要求レコード数」以上にしてください。

flags

レコードのアクセス種別，資源が競合した場合の排他解除待ち種別を設定します。

```
{TP1ServerFlags.DCTAM_WRITE|
TP1ServerFlags.DCTAM_WRTADD|
TP1ServerFlags.DCTAM_ADD}
 〔|{TP1ServerFlags.DCTAM_WAIT|
TP1ServerFlags.DCTAM_NOWAIT}〕
```

- フラグ 1

  フラグ 1 には，レコードアクセス種別を指定します。

  レコードのアクセス種別の設定は省略できません。次のアクセス種別のどちらかを指定してください。

  TP1ServerFlags.DCTAM_WRITE … 更新します。

  TP1ServerFlags.DCTAM_WRTADD … 更新または追加します。

  TP1ServerFlags.DCTAM_ADD … 追加します。

- フラグ 2

  フラグ 2 には，排他解除待ち種別を指定します。

  次の排他解除待ち種別のどちらかを指定してください。このフラグの指定を省略した場合は，TP1ServerFlags.DCTAM_NOWAIT が仮定されます。

  TP1ServerFlags.DCTAM_WAIT … 排他解除待ちをします。

  TP1ServerFlags.DCTAM_NOWAIT … 排他解除を待たないで，例外応答します。

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- 引数名（OpenTP1 提供関数呼び出し前の引数チェックでエラーになった場合にだけ出力）

  例外が発生する原因となった引数名が出力されます。

- エラーコード

発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTAMER_PARAM_TID	tableID に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	key に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	keyNo に設定した値が間違っています。
DCTAMER_PARAM_DTA	data に設定した値が間違っています。
DCTAMER_PARAM_DTS	dataSize に設定したデータ長が短過ぎます。
DCTAMER_PARAM_FLG	flags に設定した値が間違っています。
DCTAMER_NOTTAM	tableID に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_RMTBL	TAM テーブルが削除されています。
DCTAMER_NOLOAD	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	TAM テーブルがオープン状態ではありません。
DCTAMER_LOGHLD	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	TAM テーブルが障害閉塞状態です。
DCTAMER_ACSATL	TAM サービス定義で設定した TAM テーブルのアクセス形態では実行できません。
DCTAMER_NOREC	指定されたレコードは存在しません。
DCTAMER_EXKEY	key に設定したキー値が TAM テーブルに存在するので，レコードの追加はできません。
DCTAMER_LOCK	排他エラーが発生しました。flags に TP1ServerFlags.DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間でタイムアウトが発生したため，資源を確保できませんでした。
DCTAMER_DLOCK	デッドロックが起こりました。
DCTAMER_TBLVR	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。

エラーコード	説明
DCTAMER_NOAREA	TAM テーブルに空きレコードがありません。
DCTAMER_RECOBS	レコードが破壊されています。
DCTAMER_TRNNUM	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	TAM テーブルファイルに対するアクセス権がありません。
DCTAMER_MEMORY	メモリが不足しました。
DCTAMER_IO	入出力エラーが発生しました。
DCTAMER_TMERR	トランザクションサービスでエラーが発生しました。
DCTAMER_ACCESS	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。Write メソッドを呼び出した UAP には，アクセス権限がありません。

TamKeyTable 構造体 【TP1/Server Base】

TamKeyTable 構造体の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- System.ValueType
    +- Hitachi.OpenTP1.Server.TamKeyTable
```

説明

TamKeyTable 構造体は，レコードのキー値を格納します。

KeyName プロパティ，KeyName2 プロパティのどちらでもキー値を設定できますが，設定しない方のプロパティを null（Visual Basic の場合は Nothing）にしてください。両方にキー値を設定した場合は，KeyName プロパティのキー値を有効なキー値とします。

TamKeyTable 構造体配列にした場合，各要素のプロパティは TamKeyTable 構造体配列の先頭の TamKeyTable 構造体のプロパティと同一の型のプロパティを使用してください。

プロパティの一覧

名称	説明
KeyName	string 型でキー値を設定します。
KeyName2	byte 配列型でキー値を設定します。

プロパティの詳細

●KeyName

説明

string 型でキー値を設定します。

宣言

【C#の場合】

```
public string KeyName {get; set;}
```

【Visual Basic の場合】

```
Public Property KeyName As String
```

【J#の場合】

```
public System.String get_KeyName();
public void set_KeyName(System.String);
```

例外

なし

注意事項

KeyName プロパティに設定した文字列は ANSI 文字列（マルチバイト文字列）に変換されます。変換後の文字列の長さが tamcre コマンドの-l オプションで設定したキー領域長と一致するようにしてください。変換後の文字列の長さがキー領域長より短い場合は，変換後の文字列の後ろに不正な値が付加され，不正な文字列になります。変換後の文字列の長さがキー領域長より長い場合は，キー値として有効な文字列はキー領域長分になります。

●KeyName2

説明

byte 配列型でキー値を設定します。

宣言

【C#の場合】

```
public byte[] KeyName2 {get; set;}
```

【Visual Basic の場合】

```
Public Property KeyName2() As Byte
```

【J#の場合】

```
public ubyte[] get_KeyName2();
public void set_KeyName2(ubyte[]);
```

例外

なし

注意事項

- KeyName2 プロパティに設定したバイト配列の長さが tamcre コマンドの-l オプションで設定したキー領域長と一致するようにしてください。バイト配列の長さがキー領域長より短い場合は，設定したバイナリデータの後ろに不正な値が付加され，不正なバイナリデータになります。バイト配列の長さがキー領域長より長い場合は，キー値として有効なバイナリデータはキー領域長分になります。
- KeyName2 プロパティを使用する場合は，TamKeyTable 構造体配列で指定したすべての KeyName2 プロパティに値を指定してください。null（Visual Basic の場合は Nothing）または長さが 0 のバイト配列がある場合は，例外 TP1Error.DCTAMER_PARAM_KEY が返ります。
- KeyName2 プロパティを使用する場合は，TamKeyTable 構造体配列で指定したすべての KeyName2 プロパティの長さの総和が 2147483647 バイト以下になるようにしてください。2147483647 バイトより長い場合は，例外 TP1Error.DCTAMER_PARAM_KEY が返ります。

TamStatusTable 構造体 【TP1/Server Base】

TamStatusTable 構造体の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- System.ValueType
    +- Hitachi.OpenTP1.Server.TamStatusTable
```

説明

TamStatusTable 構造体は，TAM テーブルの情報を格納します。

TamStatusTable 構造体は，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，TamStatusTable 構造体は使えません。

プロパティの一覧

名称	説明
AccessType	TAM テーブルのアクセス形態が返されます。
FileName	TAM ファイル名が返されます。
IndexType	TAM テーブルのインデクス種別が返されます。
KeyLength	TAM テーブルのキー長が返されます。
KeyPosition	TAM テーブルのデータ内のキー開始位置が返されます。
LoadType	TAM テーブルのローディング契機が返されます。
RecordLength	TAM テーブルのレコード長が返されます。
RecordMaxNumber	TAM テーブルの最大レコード数が返されます。
TableSecurity	TAM サービス定義で指定した TAM テーブルのセキュリティ属性が返されます。
TableStatus	TAM テーブルの状態が返されます。
UseRecordNumber	TAM テーブルで現在使っているレコードの数が返されます。

プロパティの詳細

●AccessType

説明

TAM テーブルのアクセス形態が，次に示す値のどれかで返されます。

- TP1ServerValues.DCTAM_STS_READ
  参照型であることを示します。
- TP1ServerValues.DCTAM_STS_REWRITE
  追加・削除できない更新型であることを示します。
- TP1ServerValues.DCTAM_STS_WRITE
  追加・削除できる更新型であることを示します。
- TP1ServerValues.DCTAM_STS_RECLCK
  テーブル排他を確保しない，追加・削除できる更新型であることを示します。

宣言

【C#の場合】

```
public byte AccessType {get;}
```

【Visual Basic の場合】

```
Public Property AccessType As Byte
```

【J#の場合】

```
public byte get_AccessType();
```

例外

なし

●FileName

説明

TAM ファイル名が返されます。

宣言

【C#の場合】

```
public string FileName {get;}
```

【Visual Basic の場合】

```
Public Property FileName As String
```

【J#の場合】

```
public System.String get_FileName();
```

例外

なし

●IndexType

説明

TAM テーブルのインデクス種別が，次に示す値のどちらかで返されます。

- TP1ServerValues.DCTAM_STS_HASH
  ハッシュ形式であることを示します。
- TP1ServerValues.DCTAM_STS_TREE
  ツリー形式であることを示します。

宣言

【C#の場合】

```
public byte IndexType {get;}
```

【Visual Basic の場合】

```
Public Property IndexType As Byte
```

【J#の場合】

```
public byte get_IndexType();
```

例外

なし

●KeyLength

説明

TAM テーブルのキー長が返されます。

宣言

【C#の場合】

```
public int KeyLength {get;}
```

【Visual Basic の場合】

```
Public Property KeyLength As Integer
```

【J#の場合】

```
public int get_KeyLength();
```

例外

なし

●KeyPosition

説明

TAM テーブルのデータ内のキー開始位置が返されます。

宣言

【C#の場合】

```
public int KeyPosition {get;}
```

【Visual Basic の場合】

```
Public Property KeyPosition As Integer
```

【J#の場合】

```
public int get_KeyPosition();
```

例外

なし

●LoadType

説明

TAM テーブルのローディング契機が，次に示す値のどれかで返されます。

- TP1ServerValues.DCTAM_STS_START
  TAM サービスの開始時であることを示します。
- TP1ServerValues.DCTAM_STS_LIB
  Tam.Open メソッドで TAM テーブルをオープンしたときであることを示します。
- TP1ServerValues.DCTAM_STS_CMD
  tamload コマンドを実行したときであることを示します。

宣言

【C#の場合】

```
public byte LoadType {get;}
```

【Visual Basic の場合】

```
Public Property LoadType As Byte
```

【J#の場合】

```
public byte get_LoadType();
```

例外

なし

●RecordLength

説明

TAM テーブルのレコード長が返されます。

宣言

【C#の場合】

```
public int RecordLength {get;}
```

【Visual Basic の場合】

```
Public Property RecordLength As Integer
```

【J#の場合】

```
public int get_RecordLength();
```

例外

なし

●RecordMaxNumber

説明

TAM テーブルの最大レコード数が返されます。

宣言

【C#の場合】

```
public int RecordMaxNumber {get;}
```

【Visual Basic の場合】

```
Public Property RecordMaxNumber As Integer
```

【J#の場合】

```
public int get_RecordMaxNumber();
```

例外

なし

●TableSecurity

説明

TAM サービス定義で指定した TAM テーブルのセキュリティ属性が，次に示す値のどちらかで返されます。

- TP1ServerValues.DCTAM_STS_NOSEC
  セキュリティの指定がないことを示します。

- TP1ServerValues.DCTAM_STS_SEC
  セキュリティの指定があることを示します。

宣言

【C#の場合】

```
public int TableSecurity {get;}
```

【Visual Basic の場合】

```
Public Property TableSecurity As Integer
```

【J#の場合】

```
public int get_TableSecurity();
```

例外

なし

●TableStatus

説明

TAM テーブルの状態が，次に示す値のどれかで返されます。

- TP1ServerValues.DCTAM_STS_OPN
  オープン状態であることを示します。
- TP1ServerValues.DCTAM_STS_CLS
  クローズ状態であることを示します。
- TP1ServerValues.DCTAM_STS_LHLD
  論理閉塞状態であることを示します。
- TP1ServerValues.DCTAM_STS_OHLD
  障害閉塞状態であることを示します。

宣言

【C#の場合】

```
public int TableStatus {get;}
```

【Visual Basic の場合】

```
Public Property TableStatus As Integer
```

【J#の場合】

```
public int get_TableStatus();
```

例外

なし

●UseRecordNumber

説明

TAM テーブルで現在使っているレコードの数が返されます。ただし，Tam.GetStatus メソッドを呼び出したあとでレコードの追加や削除があった場合は，値を保証しません。

宣言

【C#の場合】

```
public int UseRecordNumber {get;}
```

【Visual Basic の場合】

```
Public Property UseRecordNumber As Integer
```

【J#の場合】

```
public int get_UseRecordNumber();
```

例外

なし

TP1Error

TP1Error の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.TP1Error
```

説明

TP1Error クラスは，クラスライブラリの各メソッドで返されるエラーや例外に設定されるエラーの値を定義したクラスです。

フィールドの一覧

名称	説明
DCADMER_COMM	プロセス間の通信エラーが発生しました。
DCADMER_DEF	マルチノード構成定義に指定した値に誤りがあります。
DCADMER_MEMORY	メモリが不足しました。
DCADMER_MEMORY_ERR	標準エラー出力のデータが，領域に入り切りませんでした。
DCADMER_MEMORY_OUT	標準出力のデータが，領域に入り切りませんでした。
DCADMER_MEMORY_OUTERR	標準出力のデータと標準エラー出力のデータの両方が，領域に入り切りませんでした。
DCADMER_MULTI_DEF	システム共通定義の multi_node_option オペランドに N を指定しています。 または，システムに TP1/Multi が組み込まれていません。
DCADMER_NODE_NOT_EXIST	ノード識別子に該当する OpenTP1 ノードはありません。
DCADMER_NO_MORE_ENTRY	OpenTP1 ノードは，これ以上ありません。
DCADMER_PARAM	引数に設定した値が間違っています。
DCADMER_PROTO	プロトコル不正です。
DCADMER_REMOTE	指定した OpenTP1 ノードではマルチノード機能は使用できません。
DCADMER_STATNOTZERO	標準出力および標準エラー出力のデータを領域に格納しました。
DCADMER_STS_IO	ステータス情報の入出力エラーが発生しました。
DCADMER_SUBAREA_NOT_EXIST	メソッドのパラメタに設定した名称に該当するマルチノードサブエリアはありません。

名称	説明
DCADMER_SWAP	系切り替えが起こっているため，ユーザサーバの状態を取得できません。
DCADMER_SYSTEMCALL	システムコール（close，pipe，dup，または read）の呼び出しに失敗しました。
DCJNLER_LONG	ユーザジャーナルの長さに，設定できる範囲以上の値を設定しています。
DCJNLER_MODE	PutUJ メソッドを，トランザクションでない UAP の処理から呼び出しています。このフィールドは，下位互換のためのフィールドです。実際にエラーが発生することはありません。
DCJNLER_PARAM	パラメタの形式が間違っています。
DCJNLER_PROTO	UAP を開始する準備ができていません。
DCJNLER_SHORT	ユーザジャーナルの長さに，0 以下のデータ長を設定しています。
DCLCKER_DLOCK	デッドロックが起こりました。
DCLCKER_MEMORY	排他制御用のテーブルが不足しています。
DCLCKER_NOTHING	このメソッドを呼び出したトランザクションでは，資源を確保していません。
DCLCKER_OUTOFTRN	トランザクション処理でない UAP から指定しています。
DCLCKER_PARAM	引数に設定した値が間違っています。
DCLCKER_TIMOUT	OpenTP1 のロックサービス定義で指定した待ち時間でタイムアウトが発生したため，資源を確保できませんでした。
DCLCKER_VERSION	OpenTP1 のライブラリとロックサービスのバージョンが一致していません。
DCLCKER_WAIT	ほかの UAP が，name に名称を設定した資源を使っています。
DCLOGER_COMM	通信障害が発生しました。
DCLOGER_DEFFILE	システムの環境設定に誤りがあります。
DCLOGER_HEADER	ログサービスがメッセージログに付ける情報を取得したときに，障害が発生しました。
DCLOGER_MEMORY	メモリが不足しました。
DCLOGER_NOT_UP	ログサービスが起動していません。
DCLOGER_PARAM_ARGS	引数に設定した値が間違っています。
DCLOGER_PROTO	プロトコル不正です。
DCLOGER_TIMEOUT	メソッドのパラメタに設定した時間を超えましたが，メッセージログが通知されません。
DCMCFER_INVALID_ARGS	引数に設定した値が間違っています。
DCMCFER_PROTO	プロトコル不正です。

名称	説明
DCMCFRTN_71002	メッセージキューへの出力処理中に障害が発生しました。 メッセージキューが閉塞されています。 メッセージキューが割り当てられていません。 セグメント長に 32000 バイトを超える値を設定しています。 MCF が終了処理中のため，メッセージの送信を受け付けられません。
DCMCFRTN_71003	メッセージキューが満杯です。
DCMCFRTN_71004	メッセージを格納するバッファをメモリ上に確保できませんでした。
DCMCFRTN_71108	メッセージを送信しようとしましたが，送信先の管理テーブルが確保できませんでした。 プロセスのローカルメモリが不足しています。
DCMCFRTN_72000	トランザクションでない SPP.NET の処理から，Send メソッドを呼び出しています。
DCMCFRTN_72001	Send メソッドまたは SendReceive メソッドを呼び出せない論理端末を設定しています。 terminalName に設定した論理端末名称が間違っています。
DCMCFRTN_72012	MCF バッファグループ定義のバッファ長が不足しました。
DCMCFRTN_72013	受信領域の長さを超えるセグメントを受信しました。 受信領域の長さを超えた部分は切り捨てられました。
DCMCFRTN_72016	action に設定したメッセージ種別（TP1ServerFlags.DCMCFNORM または TP1ServerFlags.DCMCFPRIO）の値が間違っています。 action に設定した値が間違っています。 引数に設定した値が間違っています。
DCMCFRTN_72017	action に設定した出力通番の要否（TP1ServerFlags.DCMCFSEQ または TP1ServerFlags.DCMCFNSEQ）の値が間違っています。
DCMCFRTN_72026	Send メソッドの action に設定したセグメント種別（TP1ServerFlags.DCMCFEMI）の値が間違っています。または，SendReceive メソッドの action に設定した値が間違っています。 引数に設定した値が間違っています。
DCMCFRTN_72036	セグメントを受信する領域の長さが不足しています。バッファ形式 1 の場合は 9 バイト以上，バッファ形式 2 の場合は 5 バイト以上の領域を確保してください。
DCMCFRTN_72041	送信するメッセージの内容がありません。 Send メソッドで長さが 0 バイトのセグメントを送信しています。または，単一セグメントを送信する SendReceive メソッドで，長さが 0 バイトのセグメントを送信しています。

名称	説明
DCMCFRTN_72073	非同期メッセージを送信処理中です。
DCMCFRTN_73001	出力先の論理端末で障害が発生しました。
DCMCFRTN_73002	MCF 通信サービスで障害が発生しました。
DCMCFRTN_73003	MCF バッファグループ定義のバッファ長が不足しました。
DCMCFRTN_73005	watchTime に設定した時間が経過しましたが，論理端末からの応答がありません。
DCMCFRTN_73010	メッセージの読み込み時に障害が発生しました。 メッセージの編集エラーが発生しました。
DCMCFRTN_73015	出力先の論理端末は，ほかの UAP で仕掛り中です。
DCMCFRTN_73018	watchTime に設定した値が間違っています。
DCMCFRTN_73019	メッセージ送信完了監視タイマのタイムアウトが発生しました。
DCMCFRTN_73020	出力先の論理端末は停止しています。
DCNJS2ER_INTERNAL	マーシャリング処理で内部エラーが発生しました。
DCNJS2ER_INVALID_ARGS	カスタムレコードのメンバに不正な値が指定されました。
DCNJS2ER_INVALID_DATA	マーシャリング処理で不正なデータを検出しました。
DCNJS2ER_XML_ANALYSIS_ERR	XML ドキュメントの解析処理でエラーが発生しました。
DCNJSER_SYSTEM	内部エラーが発生しました。
DCNJSER_XALIB_INVALID	トランザクション制御用ライブラリが不正です。
DCNJSER_XALIB_LOAD	トランザクション制御用ライブラリが見つかりません。
DCPRFER_PARAM	引数に指定した値に誤りがあります。
DCRAPER_ALREADY_CONNECT	すでに rap リスナーとのコネクションは確立しています。
DCRAPER_MAX_CONNECTION	一つのプロセスから Rap クラスの Connect メソッドを呼び出せる上限値を超えました。
DCRAPER_MAX_CONNECTION_SV	rap リスナーの管理する rap クライアントとのコネクション要求受け付け可能最大数を超えました。
DCRAPER_NETDOWN	rap リスナーとの通信でネットワーク障害が発生しました。
DCRAPER_NOCONTINUE	続行できない障害が発生しました。
DCRAPER_NOHOSTNAME	ホスト名称が解決できません。
DCRAPER_NOMEMORY	メモリ不足が発生しました。
DCRAPER_NOMEMORY_SV	rap リスナーまたは rap サーバでメモリ不足が発生しました。
DCRAPER_NOSERVICE	rap リスナーは開始処理中，または停止処理中です。
DCRAPER_NOSOCKET	ソケット不足が発生しました。

名称	説明
DCRAPER_PANIC_SV	rap リスナーでシステム障害が発生しました。
DCRAPER_PARAM	引数が間違っています。
DCRAPER_PROTO	プロトコル不正です。
DCRAPER_SHUTDOWN	rap リスナーは停止中です。
DCRAPER_SYSCALL	システムコールで予期しないエラーが発生しました。
DCRAPER_TIMEDOUT	rap リスナーとの通信でタイムアウトが発生しました。
DCRAPER_TIMEOUT_SV	rap リスナーサービス定義に指定したメッセージ送受信最大監視時間内にコネクションが確立できませんでした。
DCRAPER_UNKNOWN_NODE	接続されていないネットワーク上の rap リスナーに対してコネクションを確立しようとしています。
DCRPCER_ALL_RECEIVED	非同期応答型 RPC で要求したサービスの処理結果は，すべて受信しました。
DCRPCER_FATAL	初期化に失敗しました。
DCRPCER_INVALID_ARGS	引数に設定した値が間違っています。
DCRPCER_INVALID_DES	設定した記述子は存在しません。
DCRPCER_INVALID_REPLY	サービスメソッドまたはサービス関数が OpenTP1 に返した応答の長さが定義の範囲外です。
DCRPCER_MESSAGE_TOO_BIG	設定した電文長が，最大値を超えています。
DCRPCER_NET_DOWN	ネットワークに障害が発生しました。
DCRPCER_NOT_TRN_EXTEND	トランザクション処理の連鎖 RPC を使ったあとで，flags パラメタに DCRPC_TPNOTRAN を設定してサービスを要求しています。
DCRPCER_NO_BUFS	メモリが不足しました。または，サーバのメッセージ格納領域に十分な空きがないため，サービス要求を受け付けられませんでした。
DCRPCER_NO_BUFS_AT_SERVER	設定したサービスで，メモリが不足しました。
DCRPCER_NO_BUFS_RB	メモリが不足しました。このリターン値が返った場合は，トランザクションブランチをコミットできません。
DCRPCER_NO_PORT	サービスのポート番号が見つかりません。
DCRPCER_NO_SUCH_DOMAIN	ドメイン修飾をしたサービスグループ名の，ドメイン名が間違っています。
DCRPCER_NO_SUCH_SERVICE	設定したサービス名は，定義されていません。
DCRPCER_NO_SUCH_SERVICE_GROUP	設定したサービスグループは，定義されていません。
DCRPCER_OLTF_INITIALIZING	サービスを要求されたノードにある OpenTP1 は，開始処理中です。

名称	説明
DCRPCER_OLTF_NOT_UP	設定したサービスの UAP プロセスが，稼働していません。
DCRPCER_PROTO	プロトコル不正です。
DCRPCER_REPLY_TOO_BIG	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。
DCRPCER_REPLY_TOO_BIG_RB	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。このリターン値が返った場合は，トランザクションブランチをコミットできません。
DCRPCER_RETRY_COUNT_OVER	指定したサービスリトライ回数最大値を超えています。
DCRPCER_SECCHK	サービスを要求された SPP.NET または SPP は，OpenTP1 のセキュリティ機能で保護されているか SPP.NET または SPP へのアクセス権がありません。
DCRPCER_SEC_INIT	セキュリティ機能を使った OpenTP1 が，セキュリティ環境の初期化処理でエラーになりました。
DCRPCER_SERVER_BUSY	ソケット受信型サーバが，サービス要求を受け取れません。
DCRPCER_SERVICE_CLOSED	設定したサービス名のサービスがあるサービスグループは，閉塞しています。
DCRPCER_SERVICE_NOT_UP	指定したサービスは実行していません。
DCRPCER_SERVICE_TERMINATED	サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。
DCRPCER_SERVICE_TERMINATING	設定したサービスは，終了処理中です。
DCRPCER_STANDBY_END	待機系のユーザサーバが，待機の終了を要求されました。または，系切り替え時にすでに終了していたため，待機の終了を要求されました。
DCRPCER_SYSERR	システムエラーが発生しました。
DCRPCER_SYSERR_AT_SERVER	設定したサービスで，システムエラーが発生しました。
DCRPCER_SYSERR_AT_SERVER_RB	設定したサービスで，システムエラーが発生しました。このリターン値が返った場合は，トランザクションブランチをコミットできません。
DCRPCER_SYSERR_RB	システムエラーが発生しました。このリターン値が返った場合は，トランザクションブランチをコミットできません。
DCRPCER_TESTMODE	テストモードの UAP からテストモードでない SPP へ，またはテストモードでない UAP からテストモードの SPP へサービスを要求しています。
DCRPCER_TIMED_OUT	処理を完了する前にタイムアウトして異常終了しました。
DCRPCER_TRNCHK	複数の SPP.NET または SPP のトランザクション属性が一致していません。

名称	説明
DCRPCER_TRNCHK_EXTEND	同時に起動できるトランザクションブランチの数を超えたため，トランザクションブランチを開始できません。
DCRTSER_ITEM_OVER	取得項目の数がリアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。
DCRTSER_ITEM_OVER_SRV	サーバ単位の取得項目の数が，リアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。このエラーコードが返った場合，サービス単位またはサービス以外の処理の統計情報は取得されています。
DCRTSER_ITEM_OVER_SVC	サービス単位またはサービス以外の処理での取得項目の数が，リアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。このエラーコードが返った場合，サーバ単位の統計情報は取得されています。
DCRTSER_NOENTRY	Rts クラスの PutUTrace メソッドを呼び出したサーバ，およびサービスがリアルタイム統計情報の取得対象に登録されていません。
DCRTSER_NOMEM	プロセスメモリが不足したため，処理を実行できません。
DCRTSER_PARAM	引数に設定した値に誤りがあります。
DCRTSER_PROTO	Rpc クラスの Open メソッドを呼び出していません。 すでに実行時間の計測を開始している項目 ID を eventID に設定して，flags に TP1ServerFlags.DCRTS_START を設定した PutUTrace メソッドを呼び出しました。 実行時間の計測を開始していない項目 ID を eventID に設定して，flags に TP1ServerFlags.DCRTS_END を設定した PutUTrace メソッドを呼び出しました。
DCRTSER_RTS_NOT_START	リアルタイム統計情報サービスが開始していません。
DCRTSER_VERSION	UAP が，現在稼働しているリアルタイム統計情報サービスでは稼働できないバージョンのライブラリと結合しています。
DCTAMER_ACCESS	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。メソッドを呼び出した UAP には，アクセス権限がありません。
DCTAMER_ACCESSF	TAM ファイルに対するアクセス権がありません。
DCTAMER_ACCESSS	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACSATL	TAM サービス定義で指定した TAM テーブルのアクセス形態では実行できません。
DCTAMER_DLOCK	デッドロックが起こりました。
DCTAMER_EXKEY	key に設定したキー値が TAM テーブルに存在するので，レコードの追加はできません。

名称	説明
DCTAMER_EXREWRT	tableID に設定したテーブル記述子は，Tam.Rewrite メソッドで更新済みです。
DCTAMER_EXWRITE	tableID に設定したテーブル記述子は Tam.Write メソッドで更新または追加したレコードです。
DCTAMER_FLSVR	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_IDXTYP	TAM テーブルファイルの初期作成で設定した TAM テーブルのインデクス種別では実行できません。
DCTAMER_IO	入出力エラーが発生しました。
DCTAMER_LOCK	排他エラーが発生しました。flags に TP1ServerFlags.DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間でタイムアウトが発生したため，資源を確保できませんでした。
DCTAMER_LOGHLD	TAM テーブルが論理閉塞状態です。
DCTAMER_MEMORY	メモリが不足しました。
DCTAMER_NO_ACL	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。
DCTAMER_NOAREA	TAM テーブルに空きレコードがありません。
DCTAMER_NOLOAD	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	TAM テーブルがオープン状態ではありません。
DCTAMER_NOREC	指定されたレコードは存在しません。
DCTAMER_NOTTAM	tableID に設定したテーブルは TAM テーブルではありません。
DCTAMER_OBSHLD	TAM テーブルが障害閉塞状態です。
DCTAMER_OPENED	TAM テーブルがオープン済みです。
DCTAMER_OPENNUM	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_PARAM_BFA	buffer に設定した値が間違っています。
DCTAMER_PARAM_BFS	bufferSize に設定したバッファ長が短過ぎます。
DCTAMER_PARAM_DTA	data に設定した値が間違っています。
DCTAMER_PARAM_DTS	dataSize に設定したデータ長が短過ぎます。
DCTAMER_PARAM_FLG	flags に設定した値が間違っています。
DCTAMER_PARAM_KEY	key に設定しキー値が間違っています。
DCTAMER_PARAM_KNO	keyNo に設定した値が間違っています。
DCTAMER_PARAM_TBL	tableName に設定した値が間違っています。

名称	説明
DCTAMER_PARAM_TID	tableID に設定したテーブル記述子が間違っています。
DCTAMER_PROTO	• TAM テーブルへアクセスする順序が間違っています。 • UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。 • メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。
DCTAMER_RECOBS	レコードが破壊されています。
DCTAMER_RMTBL	TAM テーブルが削除されています。
DCTAMER_SEQENCE	Tam.Read メソッドを呼び出していません。
DCTAMER_TAMEND	TAM サービスが終了中です。
DCTAMER_TAMVR	UAP が，現在稼働している TAM サービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_TBLVR	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_TMERR	トランザクションサービスでエラーが発生しました。
DCTAMER_TRNNUM	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_TRNOPN	Tam.Open メソッドはトランザクション外で呼び出しています。
DCTAMER_UNDEF	TAM テーブルが定義されていません。
DCTRNER_HAZARD	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。 しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。
DCTRNER_HAZARD_NO_BEGIN	新しいトランザクションは開始できませんでした。 リターン値が返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_HEURISTIC	ヒューリスティック決定のため，あるトランザクションブランチはコミットし，あるトランザクションブランチはロールバックしました。
DCTRNER_HEURISTIC_NO_BEGIN	新しいトランザクションは開始できませんでした。 リターン値が返ったあと，プロセスはトランザクション下にはありません。
DCTRNER_NO_BEGIN	新しいトランザクションは開始できません。
DCTRNER_PROTO	プロトコル不正です。

名称	説明
DCTRNER_RM	リソースマネジャでエラーが発生しました。トランザクションは開始できませんでした。
DCTRNER_ROLLBACK	現在のトランザクションは，コミットできないでロールバックしました。
DCTRNER_ROLLBACK_NO_BEGIN	コミットしようとしたトランザクションは，コミットできないでロールバックしました。
DCTRNER_TM	トランザクションサービスでエラーが発生しました。

フィールドの詳細

●DCADMER_COMM

説明

プロセス間の通信エラーが発生しました。

宣言

【C#の場合】

```
public const int DCADMER_COMM
```

【Visual Basic の場合】

```
Public Const DCADMER_COMM As Integer
```

【J#の場合】

```
public static final int DCADMER_COMM
```

●DCADMER_DEF

説明

マルチノード構成定義に指定した値に誤りがあります。

宣言

【C#の場合】

```
public const int DCADMER_DEF
```

【Visual Basic の場合】

```
Public Const DCADMER_DEF As Integer
```

【J#の場合】

```
public static final int DCADMER_DEF
```

●DCADMER_MEMORY

説明

メモリが不足しました。

宣言

【C#の場合】

```
public const int DCADMER_MEMORY
```

【Visual Basic の場合】

```
Public Const DCADMER_MEMORY As Integer
```

【J#の場合】

```
public static final int DCADMER_MEMORY
```

●DCADMER_MEMORY_ERR

説明

標準エラー出力のデータが，領域に入り切りませんでした。

宣言

【C#の場合】

```
public const int DCADMER_MEMORY_ERR
```

【Visual Basic の場合】

```
Public Const DCADMER_MEMORY_ERR As Integer
```

【J#の場合】

```
public static final int DCADMER_MEMORY_ERR
```

●DCADMER_MEMORY_OUT

説明

標準出力のデータが，領域に入り切りませんでした。

宣言

【C#の場合】

```
public const int DCADMER_MEMORY_OUT
```

【Visual Basic の場合】

```
Public Const DCADMER_MEMORY_OUT As Integer
```

【J#の場合】

```
public static final int DCADMER_MEMORY_OUT
```

●DCADMER_MEMORY_OUTERR

説明

標準出力のデータと標準エラー出力のデータの両方が，領域に入り切りませんでした。

宣言

【C#の場合】

```
public const int DCADMER_MEMORY_OUTERR
```

【Visual Basic の場合】

```
Public Const DCADMER_MEMORY_OUTERR As Integer
```

【J#の場合】

```
public static final int DCADMER_MEMORY_OUTERR
```

●DCADMER_MULTI_DEF

説明

システム共通定義の multi_node_option オペランドに N を指定しています。

または，システムに TP1/Multi が組み込まれていません。

宣言

【C#の場合】

```
public const int DCADMER_MULTI_DEF
```

【Visual Basic の場合】

```
Public Const DCADMER_MULTI_DEF As Integer
```

【J#の場合】

```
public static final int DCADMER_MULTI_DEF
```

●DCADMER_NODE_NOT_EXIST

説明

ノード識別子に該当する OpenTP1 ノードはありません。

宣言

【C#の場合】

```
public const int DCADMER_NODE_NOT_EXIST
```

【Visual Basic の場合】

```
Public Const DCADMER_NODE_NOT_EXIST As Integer
```

【J#の場合】

```
public static final int DCADMER_NODE_NOT_EXIST
```

●DCADMER_NO_MORE_ENTRY

説明

OpenTP1 ノードは，これ以上ありません。

宣言

【C#の場合】

```
public const int DCADMER_NO_MORE_ENTRY
```

【Visual Basic の場合】

```
Public Const DCADMER_NO_MORE_ENTRY As Integer
```

【J#の場合】

```
public static final int DCADMER_NO_MORE_ENTRY
```

●DCADMER_PARAM

説明

引数に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCADMER_PARAM
```

【Visual Basic の場合】

```
Public Const DCADMER_PARAM As Integer
```

【J#の場合】

```
public static final int DCADMER_PARAM
```

●DCADMER_PROTO

説明

プロトコル不正です。

宣言

【C#の場合】

```
public const int DCADMER_PROTO
```

【Visual Basic の場合】

```
Public Const DCADMER_PROTO As Integer
```

【J#の場合】

```
public static final int DCADMER_PROTO
```

●DCADMER_REMOTE

説明

指定した OpenTP1 ノードではマルチノード機能は使用できません。

宣言

【C#の場合】

```
public const int DCADMER_REMOTE
```

【Visual Basic の場合】

```
Public Const DCADMER_REMOTE As Integer
```

【J#の場合】

```
public static final int DCADMER_REMOTE
```

●DCADMER_STATNOTZERO

説明

標準出力および標準エラー出力のデータを領域に格納しました。

宣言

【C#の場合】

```
public const int DCADMER_STATNOTZERO
```

【Visual Basic の場合】

```
Public Const DCADMER_STATNOTZERO As Integer
```

【J#の場合】

```
public static final int DCADMER_STATNOTZERO
```

●DCADMER_STS_IO

説明

ステータス情報の入出力エラーが発生しました。

宣言

【C#の場合】

```
public const int DCADMER_STS_IO
```

【Visual Basic の場合】

```
Public Const DCADMER_STS_IO As Integer
```

【J#の場合】

```
public static final int DCADMER_STS_IO
```

●DCADMER_SUBAREA_NOT_EXIST

説明

メソッドのパラメタに設定した名称に該当するマルチノードサブエリアはありません。

宣言

【C#の場合】

```
public const int DCADMER_SUBAREA_NOT_EXIST
```

【Visual Basic の場合】

```
Public Const DCADMER_SUBAREA_NOT_EXIST As Integer
```

【J#の場合】

```
public static final int DCADMER_SUBAREA_NOT_EXIST
```

●DCADMER_SWAP

説明

系切り替えが起こっているため，ユーザサーバの状態を取得できません。

宣言

【C#の場合】

```
public const int DCADMER_SWAP
```

【Visual Basic の場合】

```
Public Const DCADMER_SWAP As Integer
```

【J#の場合】

```
public static final int DCADMER_SWAP
```

●DCADMER_SYSTEMCALL

説明

システムコール（close，pipe，dup，または read）の呼び出しに失敗しました。

宣言

【C#の場合】

```
public const int DCADMER_SYSTEMCALL
```

【Visual Basic の場合】

```
Public Const DCADMER_SYSTEMCALL As Integer
```

【J#の場合】

```
public static final int DCADMER_SYSTEMCALL
```

●DCJNLER_LONG

説明

ユーザジャーナルの長さに，設定できる範囲以上の値を設定しています。

宣言

【C#の場合】

```
public const int DCJNLER_LONG
```

【Visual Basic の場合】

```
Public Const DCJNLER_LONG As Integer
```

【J#の場合】

```
public static final int DCJNLER_LONG
```

●DCJNLER_MODE

説明

PutUJ メソッドを，トランザクションでない UAP の処理から呼び出しています。このフィールドは，下位互換のためのフィールドです。実際にこのエラーが発生することはありません。

宣言

【C#の場合】

```
public const int DCJNLER_MODE
```

【Visual Basic の場合】

```
Public Const DCJNLER_MODE As Integer
```

【J#の場合】

```
public static final int DCJNLER_MODE
```

●DCJNLER_PARAM

説明

パラメタの形式が間違っています。

宣言

【C#の場合】

```
public const int DCJNLER_PARAM
```

【Visual Basic の場合】

```
Public Const DCJNLER_PARAM As Integer
```

【J#の場合】

```
public static final int DCJNLER_PARAM
```

●DCJNLER_PROTO

説明

UAP を開始する準備ができていません。

宣言

【C#の場合】

```
public const int DCJNLER_PROTO
```

【Visual Basic の場合】

```
Public Const DCJNLER_PROTO As Integer
```

【J#の場合】

```
public static final int DCJNLER_PROTO
```

●DCJNLER_SHORT

説明

ユーザジャーナルの長さに，0 以下のデータ長を設定しています。

宣言

【C#の場合】

```
public const int DCJNLER_SHORT
```

【Visual Basic の場合】

```
Public Const DCJNLER_SHORT As Integer
```

【J#の場合】

```
public static final int DCJNLER_SHORT
```

●DCLCKER_DLOCK

説明

デッドロックが起こりました。

宣言

【C#の場合】

```
public const int DCLCKER_DLOCK
```

【Visual Basic の場合】

```
Public Const DCLCKER_DLOCK As Integer
```

【J#の場合】

```
public static final int DCLCKER_DLOCK
```

●DCLCKER_MEMORY

説明

排他制御用のテーブルが不足しています。

宣言

【C#の場合】

```
public const int DCLCKER_MEMORY
```

【Visual Basic の場合】

```
Public Const DCLCKER_MEMORY As Integer
```

【J#の場合】

```
public static final int DCLCKER_MEMORY
```

●DCLCKER_NOTHING

説明

このメソッドを呼び出したトランザクションでは，資源を確保していません。

宣言

【C#の場合】

```
public const int DCLCKER_NOTHING
```

【Visual Basic の場合】

```
Public Const DCLCKER_NOTHING As Integer
```

【J#の場合】

```
public static final int DCLCKER_NOTHING
```

●DCLCKER_OUTOFTRN

説明

トランザクション処理でない UAP から指定しています。

宣言

【C#の場合】

```
public const int DCLCKER_OUTOFTRN
```

【Visual Basic の場合】

```
Public Const DCLCKER_OUTOFTRN As Integer
```

【J#の場合】

```
public static final int DCLCKER_OUTOFTRN
```

●DCLCKER_PARAM

説明

引数に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCLCKER_PARAM
```

【Visual Basic の場合】

```
Public Const DCLCKER_PARAM As Integer
```

【J#の場合】

```
public static final int DCLCKER_PARAM
```

●DCLCKER_TIMOUT

説明

OpenTP1 のロックサービス定義で指定した待ち時間でタイムアウトが発生したため，資源を確保できませんでした。

宣言

【C#の場合】

```
public const int DCLCKER_TIMOUT
```

【Visual Basic の場合】

```
Public Const DCLCKER_TIMOUT As Integer
```

【J#の場合】

```
public static final int DCLCKER_TIMOUT
```

●DCLCKER_VERSION

説明

OpenTP1 のライブラリとロックサービスのバージョンが一致していません。

宣言

【C#の場合】

```
public const int DCLCKER_VERSION
```

【Visual Basic の場合】

```
Public Const DCLCKER_VERSION As Integer
```

【J#の場合】

```
public static final int DCLCKER_VERSION
```

●DCLCKER_WAIT

説明

ほかの UAP が，name に名称を設定した資源を使っています。

宣言

【C#の場合】

```
public const int DCLCKER_WAIT
```

【Visual Basic の場合】

```
Public Const DCLCKER_WAIT As Integer
```

【J#の場合】

```
public static final int DCLCKER_WAIT
```

●DCLOGER_COMM

説明

通信障害が発生しました。

宣言

【C#の場合】

```
public const int DCLOGER_COMM
```

【Visual Basic の場合】

```
Public Const DCLOGER_COMM As Integer
```

【J#の場合】

```
public static final int DCLOGER_COMM
```

●DCLOGER_DEFFILE

説明

システムの環境設定に誤りがあります。

宣言

【C#の場合】

```
public const int DCLOGER_DEFFILE
```

【Visual Basic の場合】

```
Public Const DCLOGER_DEFFILE As Integer
```

【J#の場合】

```
public static final int DCLOGER_DEFFILE
```

●DCLOGER_HEADER

説明

ログサービスがメッセージログに付ける情報を取得したときに，障害が発生しました。

宣言

【C#の場合】

```
public const int DCLOGER_HEADER
```

【Visual Basic の場合】

```
Public Const DCLOGER_HEADER As Integer
```

【J#の場合】

```
public static final int DCLOGER_HEADER
```

●DCLOGER_MEMORY

説明

メモリが不足しました。

宣言

【C#の場合】

```
public const int DCLOGER_MEMORY
```

【Visual Basic の場合】

```
Public Const DCLOGER_MEMORY As Integer
```

【J#の場合】

```
public static final int DCLOGER_MEMORY
```

●DCLOGER_NOT_UP

説明

ログサービスが起動していません。

宣言

【C#の場合】

```
public const int DCLOGER_NOT_UP
```

【Visual Basic の場合】

```
Public Const DCLOGER_NOT_UP As Integer
```

【J#の場合】

```
public static final int DCLOGER_NOT_UP
```

●DCLOGER_PARAM_ARGS

説明

引数に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCLOGER_PARAM_ARGS
```

【Visual Basic の場合】

```
Public Const DCLOGER_PARAM_ARGS As Integer
```

【J#の場合】

```
public static final int DCLOGER_PARAM_ARGS
```

●DCLOGER_PROTO

説明

プロトコル不正です。

宣言

【C#の場合】

```
public const int DCLOGER_PROTO
```

【Visual Basic の場合】

```
Public Const DCLOGER_PROTO As Integer
```

【J#の場合】

```
public static final int DCLOGER_PROTO
```

●DCLOGER_TIMEOUT

説明

メソッドのパラメタに設定した時間を超えましたが，メッセージログが通知されません。

宣言

【C#の場合】

```
public const int DCLOGER_TIMEOUT
```

【Visual Basic の場合】

```
Public Const DCLOGER_TIMEOUT As Integer
```

【J#の場合】

```
public static final int DCLOGER_TIMEOUT
```

●DCMCFER_INVALID_ARGS

説明

引数に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCMCFER_INVALID_ARGS
```

【Visual Basic の場合】

```
Public Const DCMCFER_INVALID_ARGS As Integer
```

【J#の場合】

```
public static final int DCMCFER_INVALID_ARGS
```

●DCMCFER_PROTO

説明

プロトコル不正です。

宣言

【C#の場合】

```
public const int DCMCFER_PROTO
```

【Visual Basic の場合】

```
Public Const DCMCFER_PROTO As Integer
```

【J#の場合】

```
public static final int DCMCFER_PROTO
```

●DCMCFRTN_71002

説明

メッセージキューへの出力処理中に障害が発生しました。

メッセージキューが閉塞されています。

メッセージキューが割り当てられていません。

セグメント長に 32000 バイトを超える値を設定しています。

MCF が終了処理中のため，メッセージの送信を受け付けられません。

宣言

【C#の場合】

```
public const int DCMCFRTN_71002
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_71002 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_71002
```

●DCMCFRTN_71003

説明

メッセージキューが満杯です。

宣言

【C#の場合】

```
public const int DCMCFRTN_71003
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_71003 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_71003
```

●DCMCFRTN_71004

説明

メッセージを格納するバッファをメモリ上に確保できませんでした。

宣言

【C#の場合】

```
public const int DCMCFRTN_71004
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_71004 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_71004
```

●DCMCFRTN_71108

説明

メッセージを送信しようとしましたが，送信先の管理テーブルが確保できませんでした。
プロセスのローカルメモリが不足しています。

宣言

【C#の場合】

```
public const int DCMCFRTN_71108
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_71108 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_71108
```

●DCMCFRTN_72000

説明

トランザクションでない SPP.NET の処理から，Send メソッドを呼び出しています。

宣言

【C#の場合】

```
public const int DCMCFRTN_72000
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72000 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72000
```

●DCMCFRTN_72001

説明

Send メソッドまたは SendReceive メソッドを呼び出せない論理端末を設定しています。
terminalName に設定した論理端末名称が間違っています。

宣言

【C#の場合】

```
public const int DCMCFRTN_72001
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72001 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72001
```

●DCMCFRTN_72012

説明

MCF バッファグループ定義のバッファ長が不足しました。

宣言

【C#の場合】

```
public const int DCMCFRTN_72012
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72012 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72012
```

●DCMCFRTN_72013

説明

受信領域の長さを超えるセグメントを受信しました。

受信領域の長さを超えた部分は切り捨てられました。

宣言

【C#の場合】

```
public const int DCMCFRTN_72013
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72013 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72013
```

●DCMCFRTN_72016

説明

action に設定したメッセージ種別（TP1ServerFlags.DCMCFNORM または TP1ServerFlags.DCMCFPRIO）の値が間違っています。

action に設定した値が間違っています。

引数に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCMCFRTN_72016
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72016 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72016
```

●DCMCFRTN_72017

説明

action に設定した出力通番の要否（TP1ServerFlags.DCMCFSEQ または TP1ServerFlags.DCMCFNSEQ）の値が間違っています。

宣言

【C#の場合】

```
public const int DCMCFRTN_72017
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72017 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72017
```

●DCMCFRTN_72026

説明

Send メソッドの action に設定したセグメント種別（TP1ServerFlags.DCMCFEMI）の値が間違っています。または，SendReceive メソッドの action に設定した値が間違っています。

引数に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCMCFRTN_72026
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72026 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72026
```

●DCMCFRTN_72036

説明

セグメントを受信する領域の長さが不足しています。バッファ形式 1 の場合は 9 バイト以上，バッファ形式 2 の場合は 5 バイト以上の領域を確保してください。

宣言

【C#の場合】

```
public const int DCMCFRTN_72036
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72036 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72036
```

●DCMCFRTN_72041

説明

送信するメッセージの内容がありません。

Send メソッドで長さが 0 バイトのセグメントを送信しています。または，単一セグメントを送信する SendReceive メソッドで，長さが 0 バイトのセグメントを送信しています。

宣言

【C#の場合】

```
public const int DCMCFRTN_72041
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72041 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72041
```

●DCMCFRTN_72073

説明

非同期メッセージを送信処理中です。

宣言

【C#の場合】

```
public const int DCMCFRTN_72073
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_72073 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_72073
```

●DCMCFRTN_73001

説明

出力先の論理端末で障害が発生しました。

宣言

【C#の場合】

```
public const int DCMCFRTN_73001
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_73001 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_73001
```

●DCMCFRTN_73002

説明

MCF 通信サービスで障害が発生しました。

宣言

【C#の場合】

```
public const int DCMCFRTN_73002
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_73002 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_73002
```

●DCMCFRTN_73003

説明

MCF バッファグループ定義のバッファ長が不足しました。

宣言

【C#の場合】

```
public const int DCMCFRTN_73003
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_73003 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_73003
```

●DCMCFRTN_73005

説明

watchTime に設定した時間が経過しましたが，論理端末からの応答がありません。

宣言

【C#の場合】

```
public const int DCMCFRTN_73005
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_73005 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_73005
```

●DCMCFRTN_73010

説明

メッセージの読み込み時に障害が発生しました。

メッセージの編集エラーが発生しました。

宣言

【C#の場合】

```
public const int DCMCFRTN_73010
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_73010 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_73010
```

●DCMCFRTN_73015

説明

出力先の論理端末は，ほかの UAP で仕掛り中です。

宣言

【C#の場合】

```
public const int DCMCFRTN_73015
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_73015 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_73015
```

●DCMCFRTN_73018

説明

watchTime に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCMCFRTN_73018
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_73018 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_73018
```

●DCMCFRTN_73019

説明

メッセージ送信完了監視タイマのタイムアウトが発生しました。

宣言

【C#の場合】

```
public const int DCMCFRTN_73019
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_73019 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_73019
```

●DCMCFRTN_73020

説明

出力先の論理端末は停止しています。

宣言

【C#の場合】

```
public const int DCMCFRTN_73020
```

【Visual Basic の場合】

```
Public Const DCMCFRTN_73020 As Integer
```

【J#の場合】

```
public static final int DCMCFRTN_73020
```

●DCNJS2ER_INTERNAL

説明

マーシャリング処理で内部エラーが発生しました。

宣言

【C#の場合】

```
public const int DCNJS2ER_INTERNAL
```

【Visual Basic の場合】

```
Public Const DCNJS2ER_INTERNAL As Integer
```

【J#の場合】

```
public static final int DCNJS2ER_INTERNAL
```

●DCNJS2ER_INVALID_ARGS

説明

カスタムレコードのメンバに不正な値が指定されました。

宣言

【C#の場合】

```
public const int DCNJS2ER_INVALID_ARGS
```

【Visual Basic の場合】

```
Public Const DCNJS2ER_INVALID_ARGS As Integer
```

【J#の場合】

```
public static final int DCNJS2ER_INVALID_ARGS
```

●DCNJS2ER_INVALID_DATA

説明

マーシャリング処理で不正なデータを検出しました。

宣言

【C#の場合】

```
public const int DCNJS2ER_INVALID_DATA
```

【Visual Basic の場合】

```
Public Const DCNJS2ER_INVALID_DATA As Integer
```

【J#の場合】

```
public static final int DCNJS2ER_INVALID_DATA
```

●DCNJS2ER_XML_ANALYSIS_ERR

説明

XML ドキュメントの解析処理でエラーが発生しました。

宣言

【C#の場合】

```
public const int DCNJS2ER_XML_ANALYSIS_ERR
```

【Visual Basic の場合】

```
Public Const DCNJS2ER_XML_ANALYSIS_ERR As Integer
```

【J#の場合】

```
public static final int DCNJS2ER_XML_ANALYSIS_ERR
```

●DCNJSER_SYSTEM

説明

内部エラーが発生しました。

宣言

【C#の場合】

```
public const int DCNJSER_SYSTEM
```

【Visual Basic の場合】

```
Public Const DCNJSER_SYSTEM As Integer
```

【J#の場合】

```
public static final int DCNJSER_SYSTEM
```

●DCNJSER_XALIB_INVALID

説明

トランザクション制御用ライブラリが不正です。

宣言

【C#の場合】

```
public const int DCNJSER_XALIB_INVALID
```

【Visual Basic の場合】

```
Public Const DCNJSER_XALIB_INVALID As Integer
```

【J#の場合】

```
public static final int DCNJSER_XALIB_INVALID
```

●DCNJSER_XALIB_LOAD

説明

トランザクション制御用ライブラリが見つかりません。

宣言

【C#の場合】

```
public const int DCNJSER_XALIB_LOAD
```

【Visual Basic の場合】

```
Public Const DCNJSER_XALIB_LOAD As Integer
```

【J#の場合】

```
public static final int DCNJSER_XALIB_LOAD
```

●DCPRFER_PARAM

説明

引数に指定した値に誤りがあります。

宣言

【C#の場合】

```
public const int DCPRFER_PARAM
```

【Visual Basic の場合】

```
Public Const DCPRFER_PARAM As Integer
```

【J#の場合】

```
public static final int DCPRFER_PARAM
```

●DCRAPER_ALREADY_CONNECT

説明

すでに rap リスナーとのコネクションは確立しています。

宣言

【C#の場合】

```
public const int DCRAPER_ALREADY_CONNECT
```

【Visual Basic の場合】

```
Public Const DCRAPER_ALREADY_CONNECT As Integer
```

【J#の場合】

```
public static final int DCRAPER_ALREADY_CONNECT
```

●DCRAPER_MAX_CONNECTION

説明

一つのプロセスから Rap クラスの Connect メソッドを呼び出せる上限値を超えました。

宣言

【C#の場合】

```
public const int DCRAPER_MAX_CONNECTION
```

【Visual Basic の場合】

```
Public Const DCRAPER_MAX_CONNECTION As Integer
```

【J#の場合】

```
public static final int DCRAPER_MAX_CONNECTION
```

●DCRAPER_MAX_CONNECTION_SV

説明

rap リスナーの管理する rap クライアントとのコネクション要求受け付け可能最大数を超えました。

宣言

【C#の場合】

```
public const int DCRAPER_MAX_CONNECTION_SV
```

【Visual Basic の場合】

```
Public Const DCRAPER_MAX_CONNECTION_SV As Integer
```

【J#の場合】

```
public static final int DCRAPER_MAX_CONNECTION_SV
```

●DCRAPER_NETDOWN

説明

rap リスナーとの通信でネットワーク障害が発生しました。

宣言

【C#の場合】

```
public const int DCRAPER_NETDOWN
```

【Visual Basic の場合】

```
Public Const DCRAPER_NETDOWN As Integer
```

【J#の場合】

```
public static final int DCRAPER_NETDOWN
```

●DCRAPER_NOCONTINUE

説明

続行できない障害が発生しました。

宣言

【C#の場合】

```
public const int DCRAPER_NOCONTINUE
```

【Visual Basic の場合】

```
Public Const DCRAPER_NOCONTINUE As Integer
```

【J#の場合】

```
public static final int DCRAPER_NOCONTINUE
```

●DCRAPER_NOHOSTNAME

説明

ホスト名称が解決できません。

宣言

【C#の場合】

```
public const int DCRAPER_NOHOSTNAME
```

【Visual Basic の場合】

```
Public Const DCRAPER_NOHOSTNAME As Integer
```

【J#の場合】

```
public static final int DCRAPER_NOHOSTNAME
```

●DCRAPER_NOMEMORY

説明

メモリ不足が発生しました。

宣言

【C#の場合】

```
public const int DCRAPER_NOMEMORY
```

【Visual Basic の場合】

```
Public Const DCRAPER_NOMEMORY As Integer
```

【J#の場合】

```
public static final int DCRAPER_NOMEMORY
```

●DCRAPER_NOMEMORY_SV

説明

rap リスナーまたは rap サーバでメモリ不足が発生しました。

宣言

【C#の場合】

```
public const int DCRAPER_NOMEMORY_SV
```

【Visual Basic の場合】

```
Public Const DCRAPER_NOMEMORY_SV As Integer
```

【J#の場合】

```
public static final int DCRAPER_NOMEMORY_SV
```

●DCRAPER_NOSERVICE

説明

rap リスナーは開始処理中，または停止処理中です。

宣言

【C#の場合】

```
public const int DCRAPER_NOSERVICE
```

【Visual Basic の場合】

```
Public Const DCRAPER_NOSERVICE As Integer
```

【J#の場合】

```
public static final int DCRAPER_NOSERVICE
```

●DCRAPER_NOSOCKET

説明

ソケット不足が発生しました。

宣言

【C#の場合】

```
public const int DCRAPER_NOSOCKET
```

【Visual Basic の場合】

```
Public Const DCRAPER_NOSOCKET As Integer
```

【J#の場合】

```
public static final int DCRAPER_NOSOCKET
```

●DCRAPER_PANIC_SV

説明

rap リスナーでシステム障害が発生しました。

宣言

【C#の場合】

```
public const int DCRAPER_PANIC_SV
```

【Visual Basic の場合】

```
Public Const DCRAPER_PANIC_SV As Integer
```

【J#の場合】

```
public static final int DCRAPER_PANIC_SV
```

●DCRAPER_PARAM

説明

引数が間違っています。

宣言

【C#の場合】

```
public const int DCRAPER_PARAM
```

【Visual Basic の場合】

```
Public Const DCRAPER_PARAM As Integer
```

【J#の場合】

```
public static final int DCRAPER_PARAM
```

●DCRAPER_PROTO

説明

プロトコル不正です。

宣言

【C#の場合】

```
public const int DCRAPER_PROTO
```

【Visual Basic の場合】

```
Public Const DCRAPER_PROTO As Integer
```

【J#の場合】

```
public static final int DCRAPER_PROTO
```

●DCRAPER_SHUTDOWN

説明

rap リスナーは停止中です。

宣言

【C#の場合】

```
public const int DCRAPER_SHUTDOWN
```

【Visual Basic の場合】

```
Public Const DCRAPER_SHUTDOWN As Integer
```

【J#の場合】

```
public static final int DCRAPER_SHUTDOWN
```

●DCRAPER_SYSCALL

説明

システムコールで予期しないエラーが発生しました。

宣言

【C#の場合】

```
public const int DCRAPER_SYSCALL
```

【Visual Basic の場合】

```
Public Const DCRAPER_SYSCALL As Integer
```

【J#の場合】

```
public static final int DCRAPER_SYSCALL
```

●DCRAPER_TIMEDOUT

説明

rap リスナーとの通信でタイムアウトが発生しました。

宣言

【C#の場合】

```
public const int DCRAPER_TIMEDOUT
```

【Visual Basic の場合】

```
Public Const DCRAPER_TIMEDOUT As Integer
```

【J#の場合】

```
public static final int DCRAPER_TIMEDOUT
```

●DCRAPER_TIMEOUT_SV

説明

rap リスナーサービス定義に指定したメッセージ送受信最大監視時間内にコネクションが確立できませんでした。

宣言

【C#の場合】

```
public const int DCRAPER_TIMEOUT_SV
```

【Visual Basic の場合】

```
Public Const DCRAPER_TIMEOUT_SV As Integer
```

【J#の場合】

```
public static final int DCRAPER_TIMEOUT_SV
```

●DCRAPER_UNKNOWN_NODE

説明

接続されていないネットワーク上の rap リスナーに対してコネクションを確立しようとしています。

宣言

【C#の場合】

```
public const int DCRAPER_UNKNOWN_NODE
```

【Visual Basic の場合】

```
Public Const DCRAPER_UNKNOWN_NODE As Integer
```

【J#の場合】

```
public static final int DCRAPER_UNKNOWN_NODE
```

●DCRPCER_ALL_RECEIVED

説明

非同期応答型 RPC で要求したサービスの処理結果は，すべて受信しました。

宣言

【C#の場合】

```
public const int DCRPCER_ALL_RECEIVED
```

【Visual Basic の場合】

```
Public Const DCRPCER_ALL_RECEIVED As Integer
```

【J#の場合】

```
public static final int DCRPCER_ALL_RECEIVED
```

●DCRPCER_FATAL

説明

初期化に失敗しました。

宣言

【C#の場合】

```
public const int DCRPCER_FATAL
```

【Visual Basic の場合】

```
Public Const DCRPCER_FATAL As Integer
```

【J#の場合】

```
public static final int DCRPCER_FATAL
```

●DCRPCER_INVALID_ARGS

説明

引数に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCRPCER_INVALID_ARGS
```

【Visual Basic の場合】

```
Public Const DCRPCER_INVALID_ARGS As Integer
```

【J#の場合】

```
public static final int DCRPCER_INVALID_ARGS
```

●DCRPCER_INVALID_DES

説明

設定した記述子は存在しません。

宣言

【C#の場合】

```
public const int DCRPCER_INVALID_DES
```

【Visual Basic の場合】

```
Public Const DCRPCER_INVALID_DES As Integer
```

【J#の場合】

```
public static final int DCRPCER_INVALID_DES
```

●DCRPCER_INVALID_REPLY

説明

サービスメソッドまたはサービス関数が OpenTP1 に返した応答の長さが定義の範囲外です。

宣言

【C#の場合】

```
public const int DCRPCER_INVALID_REPLY
```

【Visual Basic の場合】

```
Public Const DCRPCER_INVALID_REPLY As Integer
```

【J#の場合】

```
public static final int DCRPCER_INVALID_REPLY
```

●DCRPCER_MESSAGE_TOO_BIG

説明

設定した電文長が，最大値を超えています。

宣言

【C#の場合】

```
public const int DCRPCER_MESSAGE_TOO_BIG
```

【Visual Basic の場合】

```
Public Const DCRPCER_MESSAGE_TOO_BIG As Integer
```

【J#の場合】

```
public static final int DCRPCER_MESSAGE_TOO_BIG
```

●DCRPCER_NET_DOWN

説明

ネットワークに障害が発生しました。

宣言

【C#の場合】

```
public const int DCRPCER_NET_DOWN
```

【Visual Basic の場合】

```
Public Const DCRPCER_NET_DOWN As Integer
```

【J#の場合】

```
public static final int DCRPCER_NET_DOWN
```

●DCRPCER_NOT_TRN_EXTEND

説明

トランザクション処理の連鎖 RPC を使ったあとで，flags パラメタに DCRPC_TPNOTRAN を設定してサービスを要求しています。

宣言

【C#の場合】

```
public const int DCRPCER_NOT_TRN_EXTEND
```

【Visual Basic の場合】

```
Public Const DCRPCER_NOT_TRN_EXTEND As Integer
```

【J#の場合】

```
public static final int DCRPCER_NOT_TRN_EXTEND
```

●DCRPCER_NO_BUFS

説明

メモリが不足しました。または，サーバのメッセージ格納領域に十分な空きがないため，サービス要求を受け付けられませんでした。

宣言

【C#の場合】

```
public const int DCRPCER_NO_BUFS
```

【Visual Basic の場合】

```
Public Const DCRPCER_NO_BUFS As Integer
```

【J#の場合】

```
public static final int DCRPCER_NO_BUFS
```

●DCRPCER_NO_BUFS_AT_SERVER

説明

設定したサービスで，メモリが不足しました。

宣言

【C#の場合】

```
public const int DCRPCER_NO_BUFS_AT_SERVER
```

【Visual Basic の場合】

```
Public Const DCRPCER_NO_BUFS_AT_SERVER As Integer
```

【J#の場合】

```
public static final int DCRPCER_NO_BUFS_AT_SERVER
```

●DCRPCER_NO_BUFS_RB

説明

メモリが不足しました。このリターン値が返った場合は，トランザクションブランチをコミットできません。

宣言

【C#の場合】

```
public const int DCRPCER_NO_BUFS_RB
```

【Visual Basic の場合】

```
Public Const DCRPCER_NO_BUFS_RB As Integer
```

【J#の場合】

```
public static final int DCRPCER_NO_BUFS_RB
```

●DCRPCER_NO_PORT

説明

サービスのポート番号が見つかりません。

宣言

【C#の場合】

```
public const int DCRPCER_NO_PORT
```

【Visual Basic の場合】

```
Public Const DCRPCER_NO_PORT As Integer
```

【J#の場合】

```
public static final int DCRPCER_NO_PORT
```

●DCRPCER_NO_SUCH_DOMAIN

説明

ドメイン修飾をしたサービスグループ名の，ドメイン名が間違っています。

宣言

【C#の場合】

```
public const int DCRPCER_NO_SUCH_DOMAIN
```

【Visual Basic の場合】

```
Public Const DCRPCER_NO_SUCH_DOMAIN As Integer
```

【J#の場合】

```
public static final int DCRPCER_NO_SUCH_DOMAIN
```

●DCRPCER_NO_SUCH_SERVICE

説明

設定したサービス名は，定義されていません。

宣言

【C#の場合】

```
public const int DCRPCER_NO_SUCH_SERVICE
```

【Visual Basic の場合】

```
Public Const DCRPCER_NO_SUCH_SERVICE As Integer
```

【J#の場合】

```
public static final int DCRPCER_NO_SUCH_SERVICE
```

●DCRPCER_NO_SUCH_SERVICE_GROUP

説明

設定したサービスグループは，定義されていません。

宣言

【C#の場合】

```
public const int DCRPCER_NO_SUCH_SERVICE_GROUP
```

【Visual Basic の場合】

```
Public Const DCRPCER_NO_SUCH_SERVICE_GROUP As Integer
```

【J#の場合】

```
public static final int DCRPCER_NO_SUCH_SERVICE_GROUP
```

●DCRPCER_OLTF_INITIALIZING

説明

サービスを要求されたノードにある OpenTP1 は，開始処理中です。

宣言

【C#の場合】

```
public const int DCRPCER_OLTF_INITIALIZING
```

【Visual Basic の場合】

```
Public Const DCRPCER_OLTF_INITIALIZING As Integer
```

【J#の場合】

```
public static final int DCRPCER_OLTF_INITIALIZING
```

●DCRPCER_OLTF_NOT_UP

説明

設定したサービスの UAP プロセスが，稼働していません。

宣言

【C#の場合】

```
public const int DCRPCER_OLTF_NOT_UP
```

【Visual Basic の場合】

```
Public Const DCRPCER_OLTF_NOT_UP As Integer
```

【J#の場合】

```
public static final int DCRPCER_OLTF_NOT_UP
```

●DCRPCER_PROTO

説明

プロトコル不正です。

宣言

【C#の場合】

```
public const int DCRPCER_PROTO
```

【Visual Basic の場合】

```
Public Const DCRPCER_PROTO As Integer
```

【J#の場合】

```
public static final int DCRPCER_PROTO
```

●DCRPCER_REPLY_TOO_BIG

説明

返ってきた応答が，クライアント UAP で用意した領域に入り切りません。

宣言

【C#の場合】

```
public const int DCRPCER_REPLY_TOO_BIG
```

【Visual Basic の場合】

```
Public Const DCRPCER_REPLY_TOO_BIG As Integer
```

【J#の場合】

```
public static final int DCRPCER_REPLY_TOO_BIG
```

●DCRPCER_REPLY_TOO_BIG_RB

説明

返ってきた応答が，クライアント UAP で用意した領域に入り切りません。このリターン値が返った場合は，トランザクションブランチをコミットできません。

宣言

【C#の場合】

```
public const int DCRPCER_REPLY_TOO_BIG_RB
```

【Visual Basic の場合】

```
Public Const DCRPCER_REPLY_TOO_BIG_RB As Integer
```

【J#の場合】

```
public static final int DCRPCER_REPLY_TOO_BIG_RB
```

●DCRPCER_RETRY_COUNT_OVER

説明

指定したサービスリトライ回数最大値を超えています。

宣言

【C#の場合】

```
public const int DCRPCER_RETRY_COUNT_OVER
```

【Visual Basic の場合】

```
Public Const DCRPCER_RETRY_COUNT_OVER As Integer
```

【J#の場合】

```
public static final int DCRPCER_RETRY_COUNT_OVER
```

●DCRPCER_SECCHK

説明

サービスを要求された SPP.NET または SPP は，OpenTP1 のセキュリティ機能で保護されているか SPP.NET または SPP へのアクセス権がありません。

宣言

【C#の場合】

```
public const int DCRPCER_SECCHK
```

【Visual Basic の場合】

```
Public Const DCRPCER_SECCHK As Integer
```

【J#の場合】

```
public static final int DCRPCER_SECCHK
```

●DCRPCER_SEC_INIT

説明

セキュリティ機能を使った OpenTP1 が，セキュリティ環境の初期化処理でエラーになりました。

宣言

【C#の場合】

```
public const int DCRPCER_SEC_INIT
```

【Visual Basic の場合】

```
Public Const DCRPCER_SEC_INIT As Integer
```

【J#の場合】

```
public static final int DCRPCER_SEC_INIT
```

●DCRPCER_SERVER_BUSY

説明

ソケット受信型サーバが，サービス要求を受け取れません。

宣言

【C#の場合】

```
public const int DCRPCER_SERVER_BUSY
```

【Visual Basic の場合】

```
Public Const DCRPCER_SERVER_BUSY As Integer
```

【J#の場合】

```
public static final int DCRPCER_SERVER_BUSY
```

●DCRPCER_SERVICE_CLOSED

説明

設定したサービス名のサービスがあるサービスグループは，閉塞しています。

宣言

【C#の場合】

```
public const int DCRPCER_SERVICE_CLOSED
```

【Visual Basic の場合】

```
Public Const DCRPCER_SERVICE_CLOSED As Integer
```

【J#の場合】

```
public static final int DCRPCER_SERVICE_CLOSED
```

●DCRPCER_SERVICE_NOT_UP

説明

指定したサービスは実行していません。

宣言

【C#の場合】

```
public const int DCRPCER_SERVICE_NOT_UP
```

【Visual Basic の場合】

```
Public Const DCRPCER_SERVICE_NOT_UP As Integer
```

【J#の場合】

```
public static final int DCRPCER_SERVICE_NOT_UP
```

●DCRPCER_SERVICE_TERMINATED

説明

サービスを要求された SPP.NET または SPP が，処理を完了する前に異常終了しました。

宣言

【C#の場合】

```
public const int DCRPCER_SERVICE_TERMINATED
```

【Visual Basic の場合】

```
Public Const DCRPCER_SERVICE_TERMINATED As Integer
```

【J#の場合】

```
public static final int DCRPCER_SERVICE_TERMINATED
```

●DCRPCER_SERVICE_TERMINATING

説明

設定したサービスは，終了処理中です。

宣言

【C#の場合】

```
public const int DCRPCER_SERVICE_TERMINATING
```

【Visual Basic の場合】

```
Public Const DCRPCER_SERVICE_TERMINATING As Integer
```

【J#の場合】

```
public static final int DCRPCER_SERVICE_TERMINATING
```

●DCRPCER_STANDBY_END

説明

待機系のユーザサーバが，待機の終了を要求されました。または，系切り替え時にすでに終了していたため，待機の終了を要求されました。

宣言

【C#の場合】

```
public const int DCRPCER_STANDBY_END
```

【Visual Basic の場合】

```
Public Const DCRPCER_STANDBY_END As Integer
```

【J#の場合】

```
public static final int DCRPCER_STANDBY_END
```

●DCRPCER_SYSERR

説明

システムエラーが発生しました。

宣言

【C#の場合】

```
public const int DCRPCER_SYSERR
```

【Visual Basic の場合】

```
Public Const DCRPCER_SYSERR As Integer
```

【J#の場合】

```
public static final int DCRPCER_SYSERR
```

●DCRPCER_SYSERR_AT_SERVER

説明

設定したサービスで，システムエラーが発生しました。

宣言

【C#の場合】

```
public const int DCRPCER_SYSERR_AT_SERVER
```

【Visual Basic の場合】

```
Public Const DCRPCER_SYSERR_AT_SERVER As Integer
```

【J#の場合】

```
public static final int DCRPCER_SYSERR_AT_SERVER
```

●DCRPCER_SYSERR_AT_SERVER_RB

説明

設定したサービスで，システムエラーが発生しました。このリターン値が返った場合は，トランザクションブランチをコミットできません。

宣言

【C#の場合】

```
public const int DCRPCER_SYSERR_AT_SERVER_RB
```

【Visual Basic の場合】

```
Public Const DCRPCER_SYSERR_AT_SERVER_RB As Integer
```

【J#の場合】

```
public static final int DCRPCER_SYSERR_AT_SERVER_RB
```

●DCRPCER_SYSERR_RB

説明

システムエラーが発生しました。このリターン値が返った場合は，トランザクションブランチをコミットできません。

宣言

【C#の場合】

```
public const int DCRPCER_SYSERR_RB
```

【Visual Basic の場合】

```
Public Const DCRPCER_SYSERR_RB As Integer
```

【J#の場合】

```
public static final int DCRPCER_SYSERR_RB
```

●DCRPCER_TESTMODE

説明

テストモードの UAP からテストモードでない SPP へ，またはテストモードでない UAP からテストモードの SPP へサービスを要求しています。

宣言

【C#の場合】

```
public const int DCRPCER_TESTMODE
```

【Visual Basic の場合】

```
Public Const DCRPCER_TESTMODE As Integer
```

【J#の場合】

```
public static final int DCRPCER_TESTMODE
```

●DCRPCER_TIMED_OUT

説明

処理を完了する前にタイムアウトして異常終了しました。

宣言

【C#の場合】

```
public const int DCRPCER_TIMED_OUT
```

【Visual Basic の場合】

```
Public Const DCRPCER_TIMED_OUT As Integer
```

【J#の場合】

```
public static final int DCRPCER_TIMED_OUT
```

●DCRPCER_TRNCHK

説明

複数の SPP.NET または SPP のトランザクション属性が一致していません。

宣言

【C#の場合】

```
public const int DCRPCER_TRNCHK
```

【Visual Basic の場合】

```
Public Const DCRPCER_TRNCHK As Integer
```

【J#の場合】

```
public static final int DCRPCER_TRNCHK
```

●DCRPCER_TRNCHK_EXTEND

説明

同時に起動できるトランザクションブランチの数を超えたため，トランザクションブランチを開始できません。

宣言

【C#の場合】

```
public const int DCRPCER_TRNCHK_EXTEND
```

【Visual Basic の場合】

```
Public Const DCRPCER_TRNCHK_EXTEND As Integer
```

【J#の場合】

```
public static final int DCRPCER_TRNCHK_EXTEND
```

●DCRTSER_ITEM_OVER

説明

取得項目の数が，リアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。

宣言

【C#の場合】

```
public const int DCRTSER_ITEM_OVER
```

【Visual Basic の場合】

```
Public Const DCRTSER_ITEM_OVER As Integer
```

【J#の場合】

```
public static final int DCRTSER_ITEM_OVER
```

●DCRTSER_ITEM_OVER_SRV

説明

サーバ単位の取得項目の数が，リアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。このエラーコードが返った場合，サービス単位またはサービス以外の処理の統計情報は取得されています。

宣言

【C#の場合】

```
public const int DCRTSER_ITEM_OVER_SRV
```

【Visual Basic の場合】

```
Public Const DCRTSER_ITEM_OVER_SRV As Integer
```

【J#の場合】

```
public static final int DCRTSER_ITEM_OVER_SRV
```

●DCRTSER_ITEM_OVER_SVC

説明

サービス単位またはサービス以外の処理での取得項目の数が，リアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。このエラーコードが返った場合，サーバ単位の統計情報は取得されています。

宣言

【C#の場合】

```
public const int DCRTSER_ITEM_OVER_SVC
```

【Visual Basic の場合】

```
Public Const DCRTSER_ITEM_OVER_SVC As Integer
```

【J#の場合】

```
public static final int DCRTSER_ITEM_OVER_SVC
```

●DCRTSER_NOENTRY

説明

Rts クラスの PutUTrace メソッドを呼び出したサーバ，およびサービスがリアルタイム統計情報の取得対象に登録されていません。

宣言

【C#の場合】

```
public const int DCRTSER_NOENTRY
```

【Visual Basic の場合】

```
Public Const DCRTSER_NOENTRY As Integer
```

【J#の場合】

```
public static final int DCRTSER_NOENTRY
```

●DCRTSER_NOMEM

説明

プロセスメモリが不足したため，処理を実行できません。

宣言

【C#の場合】

```
public const int DCRTSER_NOMEM
```

【Visual Basic の場合】

```
Public Const DCRTSER_NOMEM As Integer
```

【J#の場合】

```
public static final int DCRTSER_NOMEM
```

●DCRTSER_PARAM

説明

引数に設定した値に誤りがあります。

宣言

【C#の場合】

```
public const int DCRTSER_PARAM
```

【Visual Basic の場合】

```
Public Const DCRTSER_PARAM As Integer
```

【J#の場合】

```
public static final int DCRTSER_PARAM
```

●DCRTSER_PROTO

説明

Rpc クラスの Open メソッドを呼び出していません。

すでに実行時間の計測を開始している項目 ID を eventID に設定して，flags に TP1ServerFlags.DCRTS_START を設定した PutUTrace メソッドを呼び出しました。

実行時間の計測を開始していない項目 ID を eventID に設定して，flags に TP1ServerFlags.DCRTS_END を設定した PutUTrace メソッドを呼び出しました。

宣言

【C#の場合】

```
public const int DCRTSER_PROTO
```

【Visual Basic の場合】

```
Public Const DCRTSER_PROTO As Integer
```

【J#の場合】

```
public static final int DCRTSER_PROTO
```

●DCRTSER_RTS_NOT_START

説明

リアルタイム統計情報サービスが開始していません。

宣言

【C#の場合】

```
public const int DCRTSER_RTS_NOT_START
```

【Visual Basic の場合】

```
Public Const DCRTSER_RTS_NOT_START As Integer
```

【J#の場合】

```
public static final int DCRTSER_RTS_NOT_START
```

●DCRTSER_VERSION

説明

UAP が，現在稼働しているリアルタイム統計情報サービスでは稼働できないバージョンのライブラリと結合しています。

宣言

【C#の場合】

```
public const int DCRTSER_VERSION
```

【Visual Basic の場合】

```
Public Const DCRTSER_VERSION As Integer
```

【J#の場合】

```
public static final int DCRTSER_VERSION
```

●DCTAMER_ACCESS

説明

アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。メソッドを呼び出した UAP には，アクセス権限がありません。

宣言

【C#の場合】

```
public const int DCTAMER_ACCESS
```

【Visual Basic の場合】

```
Public Const DCTAMER_ACCESS As Integer
```

【J#の場合】

```
public static final int DCTAMER_ACCESS
```

●DCTAMER_ACCESSF

説明

TAM ファイルに対するアクセス権がありません。

宣言

【C#の場合】

```
public const int DCTAMER_ACCESSF
```

【Visual Basic の場合】

```
Public Const DCTAMER_ACCESSF As Integer
```

【J#の場合】

```
public static final int DCTAMER_ACCESSF
```

●DCTAMER_ACCESSS

説明

スペシャルファイルに対するアクセス権がありません。

宣言

【C#の場合】

```
public const int DCTAMER_ACCESSS
```

【Visual Basic の場合】

```
Public Const DCTAMER_ACCESSS As Integer
```

【J#の場合】

```
public static final int DCTAMER_ACCESSS
```

●DCTAMER_ACSATL

説明

TAM サービス定義で指定した TAM テーブルのアクセス形態では実行できません。

宣言

【C#の場合】

```
public const int DCTAMER_ACSATL
```

【Visual Basic の場合】

```
Public Const DCTAMER_ACSATL As Integer
```

【J#の場合】

```
public static final int DCTAMER_ACSATL
```

●DCTAMER_DLOCK

説明

デッドロックが起こりました。

宣言

【C#の場合】

```
public const int DCTAMER_DLOCK
```

【Visual Basic の場合】

```
Public Const DCTAMER_DLOCK As Integer
```

【J#の場合】

```
public static final int DCTAMER_DLOCK
```

●DCTAMER_EXKEY

説明

key に設定したキー値が TAM テーブルに存在するので，レコードの追加はできません。

宣言

【C#の場合】

```
public const int DCTAMER_EXKEY
```

【Visual Basic の場合】

```
Public Const DCTAMER_EXKEY As Integer
```

【J#の場合】

```
public static final int DCTAMER_EXKEY
```

●DCTAMER_EXREWRT

説明

tableID に設定したテーブル記述子は，すでに Tam.Rewrite メソッドで更新済みです。

宣言

【C#の場合】

```
public const int DCTAMER_EXREWRT
```

【Visual Basic の場合】

```
Public Const DCTAMER_EXREWRT As Integer
```

【J#の場合】

```
public static final int DCTAMER_EXREWRT
```

●DCTAMER_EXWRITE

説明

tableID に設定したテーブル記述子は Tam.Write メソッドで更新または追加したレコードです。

宣言

【C#の場合】

```
public const int DCTAMER_EXWRITE
```

【Visual Basic の場合】

```
Public Const DCTAMER_EXWRITE As Integer
```

【J#の場合】

```
public static final int DCTAMER_EXWRITE
```

●DCTAMER_FLSVR

説明

UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。

宣言

【C#の場合】

```
public const int DCTAMER_FLSVR
```

【Visual Basic の場合】

```
Public Const DCTAMER_FLSVR As Integer
```

【J#の場合】

```
public static final int DCTAMER_FLSVR
```

●DCTAMER_IDXTYP

説明

TAM テーブルファイルの初期作成で設定した TAM テーブルのインデクス種別では実行できません。

宣言

【C#の場合】

```
public const int DCTAMER_IDXTYP
```

【Visual Basic の場合】

```
Public Const DCTAMER_IDXTYP As Integer
```

【J#の場合】

```
public static final int DCTAMER_IDXTYP
```

●DCTAMER_IO

説明

入出力エラーが発生しました。

宣言

【C#の場合】

```
public const int DCTAMER_IO
```

【Visual Basic の場合】

```
Public Const DCTAMER_IO As Integer
```

【J#の場合】

```
public static final int DCTAMER_IO
```

●DCTAMER_LOCK

説明

排他エラーが発生しました。flags に TP1ServerFlags.DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間でタイムアウトが発生したため，資源を確保できませんでした。

宣言

【C#の場合】

```
public const int DCTAMER_LOCK
```

【Visual Basic の場合】

```
Public Const DCTAMER_LOCK As Integer
```

【J#の場合】

```
public static final int DCTAMER_LOCK
```

●DCTAMER_LOGHLD

説明

TAM テーブルが論理閉塞状態です。

宣言

【C#の場合】

```
public const int DCTAMER_LOGHLD
```

【Visual Basic の場合】

```
Public Const DCTAMER_LOGHLD As Integer
```

【J#の場合】

```
public static final int DCTAMER_LOGHLD
```

●DCTAMER_MEMORY

説明

メモリが不足しました。

宣言

【C#の場合】

```
public const int DCTAMER_MEMORY
```

【Visual Basic の場合】

```
Public Const DCTAMER_MEMORY As Integer
```

【J#の場合】

```
public static final int DCTAMER_MEMORY
```

●DCTAMER_NO_ACL

説明

アクセスしようとしたTAMファイルは，セキュリティ機能で保護されています。該当するファイルに対するACLがありません。

宣言

【C#の場合】

```
public const int DCTAMER_NO_ACL
```

【Visual Basicの場合】

```
Public Const DCTAMER_NO_ACL As Integer
```

【J#の場合】

```
public static final int DCTAMER_NO_ACL
```

●DCTAMER_NOAREA

説明

TAMテーブルに空きレコードがありません。

宣言

【C#の場合】

```
public const int DCTAMER_NOAREA
```

【Visual Basicの場合】

```
Public Const DCTAMER_NOAREA As Integer
```

【J#の場合】

```
public static final int DCTAMER_NOAREA
```

●DCTAMER_NOLOAD

説明

TAMテーブルがロードされていません。

宣言

【C#の場合】

```
public const int DCTAMER_NOLOAD
```

【Visual Basicの場合】

```
Public Const DCTAMER_NOLOAD As Integer
```

【J#の場合】

```
public static final int DCTAMER_NOLOAD
```

●DCTAMER_NOOPEN

説明

TAM テーブルがオープン状態ではありません。

宣言

【C#の場合】

```
public const int DCTAMER_NOOPEN
```

【Visual Basic の場合】

```
Public Const DCTAMER_NOOPEN As Integer
```

【J#の場合】

```
public static final int DCTAMER_NOOPEN
```

●DCTAMER_NOREC

説明

指定されたレコードは存在しません。

宣言

【C#の場合】

```
public const int DCTAMER_NOREC
```

【Visual Basic の場合】

```
Public Const DCTAMER_NOREC As Integer
```

【J#の場合】

```
public static final int DCTAMER_NOREC
```

●DCTAMER_NOTTAM

説明

tableID に設定したテーブルは TAM テーブルではありません。

宣言

【C#の場合】

```
public const int DCTAMER_NOTTAM
```

【Visual Basic の場合】

```
Public Const DCTAMER_NOTTAM As Integer
```

【J#の場合】

```
public static final int DCTAMER_NOTTAM
```

●DCTAMER_OBSHLD

説明

TAM テーブルが障害閉塞状態です。

宣言

【C#の場合】

```
public const int DCTAMER_OBSHLD
```

【Visual Basic の場合】

```
Public Const DCTAMER_OBSHLD As Integer
```

【J#の場合】

```
public static final int DCTAMER_OBSHLD
```

●DCTAMER_OPENED

説明

TAM テーブルがオープン済みです。

宣言

【C#の場合】

```
public const int DCTAMER_OPENED
```

【Visual Basic の場合】

```
Public Const DCTAMER_OPENED As Integer
```

【J#の場合】

```
public static final int DCTAMER_OPENED
```

●DCTAMER_OPENNUM

説明

キャラクタ型スペシャルファイルのオープン数の制限値を超えました。

宣言

【C#の場合】

```
public const int DCTAMER_OPENNUM
```

【Visual Basic の場合】

```
Public Const DCTAMER_OPENNUM As Integer
```

【J#の場合】

```
public static final int DCTAMER_OPENNUM
```

●DCTAMER_PARAM_BFA

説明

buffer に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCTAMER_PARAM_BFA
```

【Visual Basic の場合】

```
Public Const DCTAMER_PARAM_BFA As Integer
```

【J#の場合】

```
public static final int DCTAMER_PARAM_BFA
```

●DCTAMER_PARAM_BFS

説明

bufferSize に設定したバッファ長が短過ぎます。

宣言

【C#の場合】

```
public const int DCTAMER_PARAM_BFS
```

【Visual Basic の場合】

```
Public Const DCTAMER_PARAM_BFS As Integer
```

【J#の場合】

```
public static final int DCTAMER_PARAM_BFS
```

●DCTAMER_PARAM_DTA

説明

data に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCTAMER_PARAM_DTA
```

【Visual Basic の場合】

```
Public Const DCTAMER_PARAM_DTA As Integer
```

【J#の場合】

```
public static final int DCTAMER_PARAM_DTA
```

●DCTAMER_PARAM_DTS

説明

dataSize に設定したデータ長が短過ぎます。

宣言

【C#の場合】

```
public const int DCTAMER_PARAM_DTS
```

【Visual Basic の場合】

```
Public Const DCTAMER_PARAM_DTS As Integer
```

【J#の場合】

```
public static final int DCTAMER_PARAM_DTS
```

●DCTAMER_PARAM_FLG

説明

flags に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCTAMER_PARAM_FLG
```

【Visual Basic の場合】

```
Public Const DCTAMER_PARAM_FLG As Integer
```

【J#の場合】

```
public static final int DCTAMER_PARAM_FLG
```

●DCTAMER_PARAM_KEY

説明

key に設定したキー値が間違っています。

宣言

【C#の場合】

```
public const int DCTAMER_PARAM_KEY
```

【Visual Basic の場合】

```
Public Const DCTAMER_PARAM_KEY As Integer
```

【J#の場合】

```
public static final int DCTAMER_PARAM_KEY
```

●DCTAMER_PARAM_KNO

説明

keyNo に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCTAMER_PARAM_KNO
```

【Visual Basic の場合】

```
Public Const DCTAMER_PARAM_KNO As Integer
```

【J#の場合】

```
public static final int DCTAMER_PARAM_KNO
```

●DCTAMER_PARAM_TBL

説明

tableName に設定した値が間違っています。

宣言

【C#の場合】

```
public const int DCTAMER_PARAM_TBL
```

【Visual Basic の場合】

```
Public Const DCTAMER_PARAM_TBL As Integer
```

【J#の場合】

```
public static final int DCTAMER_PARAM_TBL
```

●DCTAMER_PARAM_TID

説明

tableID に設定したテーブル記述子が間違っています。

宣言

【C#の場合】

```
public const int DCTAMER_PARAM_TID
```

【Visual Basic の場合】

```
Public Const DCTAMER_PARAM_TID As Integer
```

【J#の場合】

```
public static final int DCTAMER_PARAM_TID
```

●DCTAMER_PROTO

説明

- TAM テーブルへアクセスする順序が間違っています。
- UAP のユーザサービス定義に指定したトランザクション制御用ライブラリのリソースマネジャ登録が間違っています。または，UAP のユーザサービス定義にトランザクション制御用ライブラリを指定していません。
- メソッドを呼び出した UAP のユーザサービス定義に，非トランザクション属性（atomic_update オペランドに N を指定）を指定しています。

宣言

【C#の場合】

```
public const int DCTAMER_PROTO
```

【Visual Basic の場合】

```
Public Const DCTAMER_PROTO As Integer
```

【J#の場合】

```
public static final int DCTAMER_PROTO
```

●DCTAMER_RECOBS

説明

レコードが破壊されています。

宣言

【C#の場合】

```
public const int DCTAMER_RECOBS
```

【Visual Basic の場合】

```
Public Const DCTAMER_RECOBS As Integer
```

【J#の場合】

```
public static final int DCTAMER_RECOBS
```

●DCTAMER_RMTBL

説明

TAM テーブルが削除されています。

宣言

【C#の場合】

```
public const int DCTAMER_RMTBL
```

【Visual Basic の場合】

```
Public Const DCTAMER_RMTBL As Integer
```

【J#の場合】

```
public static final int DCTAMER_RMTBL
```

●DCTAMER_SEQENCE

説明

Tam.Read メソッドを呼び出していません。

宣言

【C#の場合】

```
public const int DCTAMER_SEQENCE
```

【Visual Basic の場合】

```
Public Const DCTAMER_SEQENCE As Integer
```

【J#の場合】

```
public static final int DCTAMER_SEQENCE
```

●DCTAMER_TAMEND

説明

TAM サービスが終了中です。

宣言

【C#の場合】

```
public const int DCTAMER_TAMEND
```

【Visual Basic の場合】

```
Public Const DCTAMER_TAMEND As Integer
```

【J#の場合】

```
public static final int DCTAMER_TAMEND
```

●DCTAMER_TAMVR

説明

UAP が，現在稼働している TAM サービスでは動作できないバージョンの TAM ライブラリと結合されています。

宣言

【C#の場合】

```
public const int DCTAMER_TAMVR
```

【Visual Basic の場合】

```
Public Const DCTAMER_TAMVR As Integer
```

【J#の場合】

```
public static final int DCTAMER_TAMVR
```

●DCTAMER_TBLVR

説明

UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。

宣言

【C#の場合】

```
public const int DCTAMER_TBLVR
```

【Visual Basic の場合】

```
Public Const DCTAMER_TBLVR As Integer
```

【J#の場合】

```
public static final int DCTAMER_TBLVR
```

●DCTAMER_TMERR

説明

トランザクションサービスでエラーが発生しました。

宣言

【C#の場合】

```
public const int DCTAMER_TMERR
```

【Visual Basic の場合】

```
Public Const DCTAMER_TMERR As Integer
```

【J#の場合】

```
public static final int DCTAMER_TMERR
```

●DCTAMER_TRNNUM

説明

TAM サービスで管理できるトランザクション数を超えました。

宣言

【C#の場合】

```
public const int DCTAMER_TRNNUM
```

【Visual Basic の場合】

```
Public Const DCTAMER_TRNNUM As Integer
```

【J#の場合】

```
public static final int DCTAMER_TRNNUM
```

●DCTAMER_TRNOPN

説明

Tam.Open メソッドはトランザクション外で呼び出しています。

宣言

【C#の場合】

```
public const int DCTAMER_TRNOPN
```

【Visual Basic の場合】

```
Public Const DCTAMER_TRNOPN As Integer
```

【J#の場合】

```
public static final int DCTAMER_TRNOPN
```

●DCTAMER_UNDEF

説明

TAM テーブルが定義されていません。

宣言

【C#の場合】

```
public const int DCTAMER_UNDEF
```

【Visual Basic の場合】

```
Public Const DCTAMER_UNDEF Integer
```

【J#の場合】

```
public static final int DCTAMER_UNDEF
```

●DCTRNER_HAZARD

説明

グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。

しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。

宣言

【C#の場合】

```
public const int DCTRNER_HAZARD
```

【Visual Basic の場合】

```
Public Const DCTRNER_HAZARD As Integer
```

【J#の場合】

```
public static final int DCTRNER_HAZARD
```

●DCTRNER_HAZARD_NO_BEGIN

説明

新しいトランザクションは開始できませんでした。

リターン値が返ったあと，このプロセスはトランザクション下にはありません。

宣言

【C#の場合】

```
public const int DCTRNER_HAZARD_NO_BEGIN
```

【Visual Basic の場合】

```
Public Const DCTRNER_HAZARD_NO_BEGIN As Integer
```

【J#の場合】

```
public static final int DCTRNER_HAZARD_NO_BEGIN
```

●DCTRNER_HEURISTIC

説明

ヒューリスティック決定のため，あるトランザクションブランチはコミットし，あるトランザクションブランチはロールバックしました。

宣言

【C#の場合】

```
public const int DCTRNER_HEURISTIC
```

【Visual Basic の場合】

```
Public Const DCTRNER_HEURISTIC As Integer
```

【J#の場合】

```
public static final int DCTRNER_HEURISTIC
```

●DCTRNER_HEURISTIC_NO_BEGIN

説明

新しいトランザクションは開始できませんでした。

リターン値が返ったあと，プロセスはトランザクション下にはありません。

宣言

【C#の場合】

```
public const int DCTRNER_HEURISTIC_NO_BEGIN
```

【Visual Basic の場合】

```
Public Const DCTRNER_HEURISTIC_NO_BEGIN As Integer
```

【J#の場合】

```
public static final int DCTRNER_HEURISTIC_NO_BEGIN
```

●DCTRNER_NO_BEGIN

説明

新しいトランザクションは開始できません。

宣言

【C#の場合】

```
public const int DCTRNER_NO_BEGIN
```

【Visual Basic の場合】

```
Public Const DCTRNER_NO_BEGIN As Integer
```

【J#の場合】

```
public static final int DCTRNER_NO_BEGIN
```

●DCTRNER_PROTO

説明

プロトコル不正です。

宣言

【C#の場合】

```
public const int DCTRNER_PROTO
```

【Visual Basic の場合】

```
Public Const DCTRNER_PROTO As Integer
```

【J#の場合】

```
public static final int DCTRNER_PROTO
```

●DCTRNER_RM

説明

リソースマネジャでエラーが発生しました。トランザクションは開始できませんでした。

宣言

【C#の場合】

```
public const int DCTRNER_RM
```

【Visual Basic の場合】

```
Public Const DCTRNER_RM As Integer
```

【J#の場合】

```
public static final int DCTRNER_RM
```

●DCTRNER_ROLLBACK

説明

現在のトランザクションは，コミットできないでロールバックしました。

宣言

【C#の場合】

```
public const int DCTRNER_ROLLBACK
```

【Visual Basic の場合】

```
Public Const DCTRNER_ROLLBACK As Integer
```

【J#の場合】

```
public static final int DCTRNER_ROLLBACK
```

●DCTRNER_ROLLBACK_NO_BEGIN

説明

コミットしようとしたトランザクションは，コミットできないでロールバックしました。

宣言

【C#の場合】

```
public const int DCTRNER_ROLLBACK_NO_BEGIN
```

【Visual Basic の場合】

```
Public Const DCTRNER_ROLLBACK_NO_BEGIN As Integer
```

【J#の場合】

```
public static final int DCTRNER_ROLLBACK_NO_BEGIN
```

●DCTRNER_TM

説明

トランザクションサービスでエラーが発生しました。

宣言

【C#の場合】

```
public const int DCTRNER_TM
```

【Visual Basic の場合】

```
Public Const DCTRNER_TM As Integer
```

【J#の場合】

```
public static final int DCTRNER_TM
```

TP1Exception

TP1Exception の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

（例外の共通基底クラス）

TP1Exception クラスは，OpenTP1 for .NET Framework で使用する Exception クラスの基底となるクラスです。

プロパティの一覧

名称	説明
ErrorCode	エラーコードを返します。

メソッドの一覧

名称	説明
ToString()	この Exception クラスを表す文字列を返します。

プロパティの詳細

●ErrorCode

説明

エラーコードを返します。

宣言

【C#の場合】

```
public int ErrorCode {get;}
```

【Visual Basic の場合】

```
Public ReadOnly Property ErrorCode As Integer
```

【J#の場合】

```
public int get_ErrorCode();
```

例外

なし

メソッドの詳細

●ToString

説明

この Exception クラスを表す，次の形式の文字列を返します。

Exception クラスの名称: Message プロパティの値

ErrorCode ＝ ErrorCode プロパティの値

InnerException プロパティの ToString メソッドの値

StackTrace プロパティの値

宣言

【C#の場合】

```
public virtual string ToString(
);
```

【Visual Basic の場合】

```
Public Overridable Function ToString( _
) As String
```

【J#の場合】

```
public System.String ToString(
);
```

パラメタ

なし

戻り値

この Exception クラスを表す文字列。

例外

なし

TP1MarshalException

TP1MarshalException の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.TP1MarshalException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

TP1MarshalException クラスは，読み込みおよび書き込みできないバイト配列のインデクスが参照された場合，またはデータの内容が不正な場合に出力される例外です。

TP1RemoteException

TP1RemoteException の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.TP1RemoteException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

（リモート受信例外）

TP1RemoteException クラスは，SPP.NET で例外が発生したことを SPP.NET，SUP.NET，または CUP.NET に知らせる例外です。

プロパティの一覧

名称	説明
ExceptionMessage	SPP.NET で発生した例外のエラーメッセージの文字列を返します。
ExceptionName	SPP.NET で発生した例外の名前を返します。

メソッドの一覧

名称	説明
ToString()	この Exception クラスを表す文字列を返します。

プロパティの詳細

●ExceptionMessage

説明

SPP.NET で発生した例外のエラーメッセージの文字列を返します。

宣言

【C#の場合】

```
public string ExceptionMessage {get;}
```

【Visual Basic の場合】

```
Public ReadOnly Property ExceptionMessage As String
```

【J#の場合】

```
public System.String get_ExceptionMessage();
```

例外

なし

●ExceptionName

説明

SPP.NET で発生した例外の名前を返します。

宣言

【C#の場合】

```
public string ExceptionName {get;}
```

【Visual Basic の場合】

```
Public ReadOnly Property ExceptionName As String
```

【J#の場合】

```
public System.String get_ExceptionName();
```

例外

なし

メソッドの詳細

●ToString

説明

この Exception クラスを表す，次の形式の文字列を返します。

Exception クラスの名称: Message プロパティの値

ErrorCode = ErrorCode プロパティの値

ExceptionName = ExceptionName プロパティの値

ExceptionMessage = ExceptionMessage プロパティの値

StackTrace プロパティの値

宣言

【C#の場合】

```
public virtual string ToString(
);
```

【Visual Basic の場合】

```
Public Overridable Function ToString( _
) As String
```

【J#の場合】

```
public System.String ToString(
);
```

パラメタ

なし

戻り値

この Exception クラスを表す文字列。

例外

なし

TP1RpcMethod

TP1RpcMethod の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- System.Attribute
    +- Hitachi.OpenTP1.TP1RpcMethod
```

説明

サービスメソッドカスタム属性です。

コンストラクタの一覧

名称	説明
TP1RpcMethod()	TP1RpcMethod のコンストラクタです。

コンストラクタの詳細

●TP1RpcMethod

説明

TP1RpcMethod のコンストラクタです。

宣言

【C#の場合】

```
public TP1RpcMethod(
);
```

【Visual Basic の場合】

```
Public New( _
)
```

【J#の場合】

```
public TP1RpcMethod(
);
```

パラメタ

なし

例外

なし

TP1ServerException

TP1ServerException の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Server.TP1ServerException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

TP1/Extension for .NET Framework が返す例外のクラスです。クラスライブラリの各メソッドでエラーを検知した場合に発生します。

TP1ServerFlags

TP1ServerFlags の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.TP1ServerFlags
```

説明

TP1ServerFlags クラスは，OpenTP1 の各種フラグを提供します。

フィールドの一覧

名称	説明
DCADM_DELAY	実行したコマンドの処理を中断して，処理を中止します。
DCADM_STAT_NOT_UP	指定した OpenTP1 ノードとは，次に示す理由で通信できません。 • OpenTP1 ノードの OpenTP1 を，dcsetup コマンドで登録するか，または登録し直す必要があります。 • マルチノード物理定義に指定した値が間違っています。 • 通信障害が起こりました。
DCADM_STAT_ONLINE	ユーザサーバはオンライン中です。
DCADM_STAT_START_NORMAL	ユーザサーバは正常開始中です。
DCADM_STAT_START_RECOVER	ユーザサーバは再開始中です。
DCADM_STAT_STOP	ユーザサーバは正常終了処理中です。
DCADM_STAT_STOPA	OpenTP1 ノードは計画停止 A で終了処理中です。
DCADM_STAT_STOPB	OpenTP1 ノードは計画停止 B で終了処理中です。
DCADM_STAT_SWAP	系切り替えが起こっています。
DCADM_STAT_TERM	OpenTP1 ノードが停止中です。または異常終了中です。
DCJNL_FLUSH	UJ レコードを取得する時点で，システムジャーナルファイルに UJ レコードを出力します。トランザクション内で UJ レコードが取得されている場合，この設定は無視されます。
DCLCK_EX	資源を更新します。ほかの UAP には参照も更新も禁止します。
DCLCK_PR	資源を参照します。ほかの UAP には参照だけを許可します。
DCLCK_TEST	資源が使えるかどうかをテストするときに設定します。
DCLCK_WAIT	ほかの UAP と資源を競合した場合に，資源の解放待ちにします。

名称	説明
DCMCFBUF1	バッファ形式 1 を使用する場合に設定します。
DCMCFBUF2	バッファ形式 2 を使用する場合に設定します。
DCMCFEMI	最終セグメントを送信する場合に設定します。
DCMCFNORM	一般の一方送信メッセージとして送信する場合に設定します。
DCMCFNSEQ	出力通番が必要ない場合に設定します。
DCMCFPRIO	優先の一方送信メッセージとして送信する場合に設定します。
DCMCFSEQ	出力通番が必要な場合に設定します。
DCNOFLAGS	オプションフラグを指定しません。
DCRPC_CHAINED	連鎖 RPC です。
DCRPC_DOMAIN	サービスグループ名をドメイン修飾した場合に指定します。
DCRPC_NAMPORT	ホスト名を検索のキーにする場合に指定します。
DCRPC_NOREPLY	非応答型 RPC です。
DCRPC_NOWAIT	非同期応答型 RPC です。
DCRPC_SPECIFIC_MSG	引数に設定した記述子をリターンした，非同期応答型 RPC の応答を受信します。
DCRPC_TPNOTRAN	トランザクション処理からのサービス要求で，要求先の処理をトランザクションにしない場合に設定します。
DCRPC_WAIT_MILLISEC	引数で設定した待ち時間の単位をミリ秒にします。
DCRTS_END	eventID に設定した項目 ID の実行時間を取得して，計測を終了します。
DCRTS_START	eventID に設定した項目 ID の実行時間の計測を開始します。このフラグを設定して PutUTrace メソッドを呼び出した時点では，リアルタイム統計情報を取得しません。
DCTAM_ADD	追加します。
DCTAM_EQLSRC	'キー値='を検索します（ハッシュ，ツリー）。
DCTAM_EXCLUSIVE	排他します。
DCTAM_FIRSTSRC	先頭から検索します（ハッシュ）。
DCTAM_GRTEQLSRC	'キー値<='を検索します（ツリー）。
DCTAM_GRTSRC	'キー値<'を検索します（ツリー）。
DCTAM_LSSEQLSRC	'キー値>='を検索します（ツリー）。
DCTAM_LSSSRC	'キー値>'を検索します（ツリー）。
DCTAM_MODIFY	更新目的の排他です。
DCTAM_NEXTSRC	設定したキー値の，次のレコードから検索します（ハッシュ）。

名称	説明
DCTAM_NOEXCLUSIVE	排他をしません。
DCTAM_NOOUTREC	削除するレコードを退避しません。
DCTAM_NOWAIT	排他解除待ちをしません。
DCTAM_OUTREC	削除するレコードを退避します。
DCTAM_REC_EXCLUSIVE	レコード排他です。
DCTAM_REFERENCE	参照目的の排他です。
DCTAM_TBL_EXCLUSIVE	テーブル排他です。
DCTAM_WAIT	排他解除待ちをします。
DCTAM_WRITE	更新します。
DCTAM_WRTADD	更新または追加します。

フィールドの詳細

●DCADM_DELAY

説明

実行したコマンドの処理を中断して，処理を中止します。

宣言

【C#の場合】

```
public const int DCADM_DELAY
```

【Visual Basic の場合】

```
Public Const DCADM_DELAY As Integer
```

【J#の場合】

```
public static final int DCADM_DELAY
```

●DCADM_STAT_NOT_UP

説明

指定した OpenTP1 ノードとは，次に示す理由で通信できません。

- OpenTP1 ノードの OpenTP1 を，dcsetup コマンドで登録するか，または登録し直す必要があります。
- マルチノード物理定義に指定した値が間違っています（OpenTP1 ノードを登録していないか，指定したホスト名またはポート番号が間違っています）。

- 通信障害が起こりました（OpenTP1 ノードのマシンの電源を入れていないか，またはネットワーク障害が起こりました）。

このフィールドは前バージョンとの互換性を保つために定義してあります。

TP1ServerValues.DCADM_STAT_NOT_UP を使用してください。

宣言

【C#の場合】

```
public const int DCADM_STAT_NOT_UP
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_NOT_UP As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_NOT_UP
```

●DCADM_STAT_ONLINE

説明

ユーザサーバはオンライン中です。

このフィールドは前バージョンとの互換性を保つために定義してあります。

TP1ServerValues.DCADM_STAT_ONLINE を使用してください。

宣言

【C#の場合】

```
public const int DCADM_STAT_ONLINE
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_ONLINE As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_ONLINE
```

●DCADM_STAT_START_NORMAL

説明

ユーザサーバは正常開始中です。

このフィールドは前バージョンとの互換性を保つために定義してあります。

TP1ServerValues.DCADM_STAT_START_NORMAL を使用してください。

宣言

【C#の場合】

```
public const int DCADM_STAT_START_NORMAL
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_START_NORMAL As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_START_NORMAL
```

●DCADM_STAT_START_RECOVER

説明

ユーザサーバは再開始中です。

このフィールドは前バージョンとの互換性を保つために定義してあります。

TP1ServerValues.DCADM_STAT_START_RECOVER を使用してください。

宣言

【C#の場合】

```
public const int DCADM_STAT_START_RECOVER
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_START_RECOVER As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_START_RECOVER
```

●DCADM_STAT_STOP

説明

ユーザサーバは正常終了処理中です。

このフィールドは前バージョンとの互換性を保つために定義してあります。

TP1ServerValues.DCADM_STAT_STOP を使用してください。

宣言

【C#の場合】

```
public const int DCADM_STAT_STOP
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_STOP As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_STOP
```

●DCADM_STAT_STOPA

説明

OpenTP1 ノードは計画停止 A で終了処理中です。

このフィールドは前バージョンとの互換性を保つために定義してあります。

TP1ServerValues.DCADM_STAT_STOPA を使用してください。

宣言

【C#の場合】

```
public const int DCADM_STAT_STOPA
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_STOPA As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_STOPA
```

●DCADM_STAT_STOPB

説明

OpenTP1 ノードは計画停止 B で終了処理中です。

このフィールドは前バージョンとの互換性を保つために定義してあります。

TP1ServerValues.DCADM_STAT_STOPB を使用してください。

宣言

【C#の場合】

```
public const int DCADM_STAT_STOPB
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_STOPB As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_STOPB
```

●DCADM_STAT_SWAP

説明

系切り替えが起こっています。

このフィールドは前バージョンとの互換性を保つために定義してあります。

TP1ServerValues.DCADM_STAT_SWAP を使用してください。

宣言

【C#の場合】

```
public const int DCADM_STAT_SWAP
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_SWAP As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_SWAP
```

●DCADM_STAT_TERM

説明

OpenTP1 ノードが停止中です。または異常終了中です。

このフィールドは前バージョンとの互換性を保つために定義してあります。

TP1ServerValues.DCADM_STAT_TERM を使用してください。

宣言

【C#の場合】

```
public const int DCADM_STAT_TERM
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_TERM As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_TERM
```

●DCJNL_FLUSH

説明

UJ レコードを取得する時点で，システムジャーナルファイルに UJ レコードを出力します。トランザクション内で UJ レコードが取得されている場合，この設定は無視されます。

宣言

【C#の場合】

```
public const int DCJNL_FLUSH
```

【Visual Basic の場合】

```
Public Const DCJNL_FLUSH As Integer
```

【J#の場合】

```
public static final int DCJNL_FLUSH
```

●DCLCK_EX

説明

資源を更新します。ほかの UAP には参照も更新も禁止します。

宣言

【C#の場合】

```
public const int DCLCK_EX
```

【Visual Basic の場合】

```
Public Const DCLCK_EX As Integer
```

【J#の場合】

```
public static final int DCLCK_EX
```

●DCLCK_PR

説明

資源を参照します。ほかの UAP には参照だけを許可します。

宣言

【C#の場合】

```
public const int DCLCK_PR
```

【Visual Basic の場合】

```
Public Const DCLCK_PR As Integer
```

【J#の場合】

```
public static final int DCLCK_PR
```

●DCLCK_TEST

説明

資源が使えるかどうかをテストするときに設定します。

宣言

【C#の場合】

```
public const int DCLCK_TEST
```

【Visual Basic の場合】

```
Public Const DCLCK_TEST As Integer
```

【J#の場合】

```
public static final int DCLCK_TEST
```

●DCLCK_WAIT

説明

ほかの UAP と資源を競合した場合に，資源の解放待ちにします。このフラグが設定されてない場合に競合したときは，例外応答します。

宣言

【C#の場合】

```
public const int DCLCK_WAIT
```

【Visual Basic の場合】

```
Public Const DCLCK_WAIT As Integer
```

【J#の場合】

```
public static final int DCLCK_WAIT
```

●DCMCFBUF1

説明

バッファ形式 1 を使用する場合に設定します。

宣言

【C#の場合】

```
public const int DCMCFBUF1
```

【Visual Basic の場合】

```
Public Const DCMCFBUF1 As Integer
```

【J#の場合】

```
public static final int DCMCFBUF1
```

●DCMCFBUF2

説明

バッファ形式 2 を使用する場合に設定します。

宣言

【C#の場合】

```
public const int DCMCFBUF2
```

【Visual Basic の場合】

```
Public Const DCMCFBUF2 As Integer
```

【J#の場合】

```
public static final int DCMCFBUF2
```

●DCMCFEMI

説明

最終セグメントを送信する場合に設定します。

メッセージが単一セグメントの場合も，DCMCFEMI を設定します。

メッセージの送信の終了を連絡するために，最後は必ずこの値を設定してください。

この値を設定して SendReceive メソッドを呼び出すと，論理端末からの応答を待ちます。

宣言

【C#の場合】

```
public const int DCMCFEMI
```

【Visual Basic の場合】

```
Public Const DCMCFEMI As Integer
```

【J#の場合】

```
public static final int DCMCFEMI
```

●DCMCFNORM

説明

一般の一方送信メッセージとして送信する場合に設定します。

宣言

【C#の場合】

```
public const int DCMCFNORM
```

【Visual Basic の場合】

```
Public Const DCMCFNORM As Integer
```

【J#の場合】

```
public static final int DCMCFNORM
```

●DCMCFNSEQ

説明

出力通番が必要ない場合に設定します。

宣言

【C#の場合】

```
public const int DCMCFNSEQ
```

【Visual Basic の場合】

```
Public Const DCMCFNSEQ As Integer
```

【J#の場合】

```
public static final int DCMCFNSEQ
```

●DCMCFPRIO

説明

優先の一方送信メッセージとして送信する場合に設定します。

宣言

【C#の場合】

```
public const int DCMCFPRIO
```

【Visual Basic の場合】

```
Public Const DCMCFPRIO As Integer
```

【J#の場合】

```
public static final int DCMCFPRIO
```

●DCMCFSEQ

説明

出力通番が必要な場合に設定します。

宣言

【C#の場合】

```
public const int DCMCFSEQ
```

【Visual Basic の場合】

```
Public Const DCMCFSEQ As Integer
```

【J#の場合】

```
public static final int DCMCFSEQ
```

●DCNOFLAGS

説明

オプションフラグを指定しません。

宣言

【C#の場合】

```
public const int DCNOFLAGS
```

【Visual Basic の場合】

```
Public Const DCNOFLAGS As Integer
```

【J#の場合】

```
public static final int DCNOFLAGS
```

●DCRPC_CHAINED

説明

連鎖 RPC です。

宣言

【C#の場合】

```
public const int DCRPC_CHAINED
```

【Visual Basic の場合】

```
Public Const DCRPC_CHAINED As Integer
```

【J#の場合】

```
public static final int DCRPC_CHAINED
```

●DCRPC_DOMAIN

説明

サービスグループ名をドメイン修飾した場合に指定します。ドメイン修飾をした RPC はトランザクションブランチにできません。そのため，トランザクションの処理から Call メソッドを使う場合は，必ず DCRPC_TPNOTRAN と一緒に指定してください。

宣言

【C#の場合】

```
public const int DCRPC_DOMAIN
```

【Visual Basic の場合】

```
Public Const DCRPC_DOMAIN As Integer
```

【J#の場合】

```
public static final int DCRPC_DOMAIN
```

●DCRPC_NAMPORT

説明

ホスト名を検索のキーにする場合に指定します。

宣言

【C#の場合】

```
public const int DCRPC_NAMPORT
```

【Visual Basic の場合】

```
Public Const DCRPC_NAMPORT As Integer
```

【J#の場合】

```
public static final int DCRPC_NAMPORT
```

●DCRPC_NOREPLY

説明

非応答型 RPC です。

宣言

【C#の場合】

```
public const int DCRPC_NOREPLY
```

【Visual Basic の場合】

```
Public Const DCRPC_NOREPLY As Integer
```

【J#の場合】

```
public static final int DCRPC_NOREPLY
```

●DCRPC_NOWAIT

説明

非同期応答型 RPC です。

宣言

【C#の場合】

```
public const int DCRPC_NOWAIT
```

【Visual Basic の場合】

```
Public Const DCRPC_NOWAIT As Integer
```

【J#の場合】

```
public static final int DCRPC_NOWAIT
```

●DCRPC_SPECIFIC_MSG

説明

引数に設定した記述子をリターンした，非同期応答型 RPC の応答を受信します。

宣言

【C#の場合】

```
public const int DCRPC_SPECIFIC_MSG
```

【Visual Basic の場合】

```
Public Const DCRPC_SPECIFIC_MSG As Integer
```

【J#の場合】

```
public static final int DCRPC_SPECIFIC_MSG
```

●DCRPC_TPNOTRAN

説明

トランザクション処理からのサービス要求で，要求先の処理をトランザクションにしない場合に設定します。

宣言

【C#の場合】

```
public const int DCRPC_TPNOTRAN
```

【Visual Basic の場合】

```
Public Const DCRPC_TPNOTRAN As Integer
```

【J#の場合】

```
public static final int DCRPC_TPNOTRAN
```

●DCRPC_WAIT_MILLISEC

説明

引数で設定した待ち時間の単位をミリ秒にします。

宣言

【C#の場合】

```
public const int DCRPC_WAIT_MILLISEC
```

【Visual Basic の場合】

```
Public Const DCRPC_WAIT_MILLISEC As Integer
```

【J#の場合】

```
public static final int DCRPC_WAIT_MILLISEC
```

●DCRTS_END

説明

eventID に設定した項目 ID の実行時間を取得して，計測を終了します。

宣言

【C#の場合】

```
public const int DCRTS_END
```

【Visual Basic の場合】

```
Public Const DCRTS_END As Integer
```

【J#の場合】

```
public static final int DCRTS_END
```

●DCRTS_START

説明

eventID に設定した項目 ID の実行時間の計測を開始します。このフラグを設定して PutUTrace メソッドを呼び出した時点では，リアルタイム統計情報を取得しません。

宣言

【C#の場合】

```
public const int DCRTS_START
```

【Visual Basic の場合】

```
Public Const DCRTS_START As Integer
```

【J#の場合】

```
public static final int DCRTS_START
```

●DCTAM_ADD

説明

追加します。

宣言

【C#の場合】

```
public const int DCTAM_ADD
```

【Visual Basic の場合】

```
Public Const DCTAM_ADD As Integer
```

【J#の場合】

```
public static final int DCTAM_ADD
```

●DCTAM_EQLSRC

説明

'キー値='を検索します（ハッシュ，ツリー）。

宣言

【C#の場合】

```
public const int DCTAM_EQLSRC
```

【Visual Basic の場合】

```
Public Const DCTAM_EQLSRC As Integer
```

【J#の場合】

```
public static final int DCTAM_EQLSRC
```

●DCTAM_EXCLUSIVE

説明

排他します。

宣言

【C#の場合】

```
public const int DCTAM_EXCLUSIVE
```

【Visual Basic の場合】

```
Public Const DCTAM_EXCLUSIVE As Integer
```

【J#の場合】

```
public static final int DCTAM_EXCLUSIVE
```

●DCTAM_FIRSTSRC

説明

先頭から検索します（ハッシュ）。

宣言

【C#の場合】

```
public const int DCTAM_FIRSTSRC
```

【Visual Basic の場合】

```
Public Const DCTAM_FIRSTSRC As Integer
```

【J#の場合】

```
public static final int DCTAM_FIRSTSRC
```

●DCTAM_GRTEQLSRC

説明

'キー値<='を検索します（ツリー）。

宣言

【C#の場合】

```
public const int DCTAM_GRTEQLSRC
```

【Visual Basic の場合】

```
Public Const DCTAM_GRTEQLSRC As Integer
```

【J#の場合】

```
public static final int DCTAM_GRTEQLSRC
```

●DCTAM_GRTSRC

説明

'キー値<'を検索します（ツリー）。

宣言

【C#の場合】

```
public const int DCTAM_GRTSRC
```

【Visual Basic の場合】

```
Public Const DCTAM_GRTSRC As Integer
```

【J#の場合】

```
public static final int DCTAM_GRTSRC
```

●DCTAM_LSSEQLSRC

説明

'キー値>='を検索します（ツリー）。

宣言

【C#の場合】

```
public const int DCTAM_LSSEQLSRC
```

【Visual Basic の場合】

```
Public Const DCTAM_LSSEQLSRC As Integer
```

【J#の場合】

```
public static final int DCTAM_LSSEQLSRC
```

●DCTAM_LSSSRC

説明

'キー値>'を検索します（ツリー）。

宣言

【C#の場合】

```
public const int DCTAM_LSSSRC
```

【Visual Basic の場合】

```
Public Const DCTAM_LSSSRC As Integer
```

【J#の場合】

```
public static final int DCTAM_LSSSRC
```

●DCTAM_MODIFY

説明

更新目的の排他です。

宣言

【C#の場合】

```
public const int DCTAM_MODIFY
```

【Visual Basic の場合】

```
Public Const DCTAM_MODIFY As Integer
```

【J#の場合】

```
public static final int DCTAM_MODIFY
```

●DCTAM_NEXTSRC

説明

設定したキー値の，次のレコードから検索します（ハッシュ）。

宣言

【C#の場合】

```
public const int DCTAM_NEXTSRC
```

【Visual Basic の場合】

```
Public Const DCTAM_NEXTSRC As Integer
```

【J#の場合】

```
public static final int DCTAM_NEXTSRC
```

●DCTAM_NOEXCLUSIVE

説明

排他をしません。

宣言

【C#の場合】

```
public const int DCTAM_NOEXCLUSIVE
```

【Visual Basic の場合】

```
Public Const DCTAM_NOEXCLUSIVE As Integer
```

【J#の場合】

```
public static final int DCTAM_NOEXCLUSIVE
```

●DCTAM_NOOUTREC

説明

削除するレコードを退避しません。

宣言

【C#の場合】

```
public const int DCTAM_NOOUTREC
```

【Visual Basic の場合】

```
Public Const DCTAM_NOOUTREC As Integer
```

【J#の場合】

```
public static final int DCTAM_NOOUTREC
```

●DCTAM_NOWAIT

説明

排他解除待ちをしません。

宣言

【C#の場合】

```
public const int DCTAM_NOWAIT
```

【Visual Basic の場合】

```
Public Const DCTAM_NOWAIT As Integer
```

【J#の場合】

```
public static final int DCTAM_NOWAIT
```

●DCTAM_OUTREC

説明

削除するレコードを退避します。

宣言

【C#の場合】

```
public const int DCTAM_OUTREC
```

【Visual Basic の場合】

```
Public Const DCTAM_OUTREC As Integer
```

【J#の場合】

```
public static final int DCTAM_OUTREC
```

●DCTAM_REC_EXCLUSIVE

説明

レコード排他です。

宣言

【C#の場合】

```
public const int DCTAM_REC_EXCLUSIVE
```

【Visual Basic の場合】

```
Public Const DCTAM_REC_EXCLUSIVE As Integer
```

【J#の場合】

```
public static final int DCTAM_REC_EXCLUSIVE
```

●DCTAM_REFERENCE

説明

参照目的の排他です。

宣言

【C#の場合】

```
public const int DCTAM_REFERENCE
```

【Visual Basic の場合】

```
Public Const DCTAM_REFERENCE As Integer
```

【J#の場合】

```
public static final int DCTAM_REFERENCE
```

●DCTAM_TBL_EXCLUSIVE

説明

テーブル排他です。

宣言

【C#の場合】

```
public const int DCTAM_TBL_EXCLUSIVE
```

【Visual Basic の場合】

```
Public Const DCTAM_TBL_EXCLUSIVE As Integer
```

【J#の場合】

```
public static final int DCTAM_TBL_EXCLUSIVE
```

●DCTAM_WAIT

説明

排他解除待ちをします。

宣言

【C#の場合】

```
public const int DCTAM_WAIT
```

【Visual Basic の場合】

```
Public Const DCTAM_WAIT As Integer
```

【J#の場合】

```
public static final int DCTAM_WAIT
```

●DCTAM_WRITE

説明

更新します。

宣言

【C#の場合】

```
public const int DCTAM_WRITE
```

【Visual Basic の場合】

```
Public Const DCTAM_WRITE As Integer
```

【J#の場合】

```
public static final int DCTAM_WRITE
```

●DCTAM_WRTADD

説明

更新または追加します。

宣言

【C#の場合】

```
public const int DCTAM_WRTADD
```

【Visual Basic の場合】

```
Public Const DCTAM_WRTADD As Integer
```

【J#の場合】

```
public static final int DCTAM_WRTADD
```

TP1ServerLimits

TP1ServerLimits の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.TP1ServerLimits
```

説明

OpenTP1 で使用する長さの制限などを定義します。

フィールドの一覧

名称	説明
DCRPC_MAX_MESSAGE_SIZE	RPC の送受信メッセージの最大長です。

フィールドの詳細

●DCRPC_MAX_MESSAGE_SIZE

説明

RPC の送受信メッセージの最大長です。

宣言

【C#の場合】

```
public const long DCRPC_MAX_MESSAGE_SIZE
```

【Visual Basic の場合】

```
Public Const DCRPC_MAX_MESSAGE_SIZE As Long
```

【J#の場合】

```
public static final long DCRPC_MAX_MESSAGE_SIZE
```

TP1ServerValues

TP1ServerValues の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.TP1ServerValues
```

説明

TP1ServerValues クラスは，OpenTP1 の各種値を提供します。

フィールドの一覧

名称	説明
DCADM_STAT_NOT_UP	指定した OpenTP1 ノードとは，次に示す理由で通信できません。 • OpenTP1 ノードの OpenTP1 を，dcsetup コマンドで登録するか，または登録し直す必要があります。 • マルチノード物理定義に指定した値が間違っています。 • 通信障害が起こりました。
DCADM_STAT_ONLINE	ユーザサーバはオンライン中です。
DCADM_STAT_START_NORMAL	ユーザサーバは正常開始中です。
DCADM_STAT_START_RECOVER	ユーザサーバは再開始中です。
DCADM_STAT_STOP	ユーザサーバは正常終了処理中です。
DCADM_STAT_STOPA	OpenTP1 ノードは計画停止 A で終了処理中です。
DCADM_STAT_STOPB	OpenTP1 ノードは計画停止 B で終了処理中です。
DCADM_STAT_SWAP	系切り替えが起こっています。
DCADM_STAT_TERM	OpenTP1 ノードが停止中です。または異常終了中です。
DCTAM_STS_CLS	TAM テーブルがクローズ状態であることを示します。
DCTAM_STS_CMD	tamload コマンドを実行したときであることを示します。
DCTAM_STS_HASH	ハッシュ形式であることを示します。
DCTAM_STS_LHLD	TAM テーブルが論理閉塞状態であることを示します。
DCTAM_STS_LIB	Tam.Open メソッドで TAM テーブルをオープンしたときであることを示します。
DCTAM_STS_NOSEC	セキュリティの指定がないことを示します。

名称	説明
DCTAM_STS_OHLD	TAM テーブルが障害閉塞状態であることを示します。
DCTAM_STS_OPN	TAM テーブルがオープン状態であることを示します。
DCTAM_STS_READ	参照型であることを示します。
DCTAM_STS_RECLCK	テーブル排他を確保しない，追加・削除できる更新型であることを示します。
DCTAM_STS_REWRITE	追加・削除できない更新型であることを示します。
DCTAM_STS_SEC	セキュリティの指定があることを示します。
DCTAM_STS_START	TAM サービスの開始時であることを示します。
DCTAM_STS_TREE	ツリー形式であることを示します。
DCTAM_STS_WRITE	追加・削除できる更新型であることを示します。

フィールドの詳細

●DCADM_STAT_NOT_UP

説明

指定した OpenTP1 ノードとは，次に示す理由で通信できません。

- OpenTP1 ノードの OpenTP1 を，dcsetup コマンドで登録するか，または登録し直す必要があります。
- マルチノード物理定義に指定した値が間違っています（OpenTP1 ノードを登録していない，もしくは指定したホスト名またはポート番号が間違っています)。
- 通信障害が起こりました（OpenTP1 ノードのマシンの電源を入れていないか，またはネットワーク障害が起こりました)。

宣言

【C#の場合】

```
public const int DCADM_STAT_NOT_UP
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_NOT_UP As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_NOT_UP
```

●DCADM_STAT_ONLINE

説明

ユーザサーバはオンライン中です。

宣言

【C#の場合】

```
public const int DCADM_STAT_ONLINE
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_ONLINE As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_ONLINE
```

●DCADM_STAT_START_NORMAL

説明

ユーザサーバは正常開始中です。

宣言

【C#の場合】

```
public const int DCADM_STAT_START_NORMAL
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_START_NORMAL As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_START_NORMAL
```

●DCADM_STAT_START_RECOVER

説明

ユーザサーバは再開始中です。

宣言

【C#の場合】

```
public const int DCADM_STAT_START_RECOVER
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_START_RECOVER As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_START_RECOVER
```

●DCADM_STAT_STOP

説明

ユーザサーバは正常終了処理中です。

宣言

【C#の場合】

```
public const int DCADM_STAT_STOP
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_STOP As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_STOP
```

●DCADM_STAT_STOPA

説明

OpenTP1 ノードは計画停止 A で終了処理中です。

宣言

【C#の場合】

```
public const int DCADM_STAT_STOPA
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_STOPA As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_STOPA
```

●DCADM_STAT_STOPB

説明

OpenTP1 ノードは計画停止 B で終了処理中です。

宣言

【C#の場合】

```
public const int DCADM_STAT_STOPB
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_STOPB As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_STOPB
```

●DCADM_STAT_SWAP

説明

系切り替えが起こっています。

宣言

【C#の場合】

```
public const int DCADM_STAT_SWAP
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_SWAP As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_SWAP
```

●DCADM_STAT_TERM

説明

OpenTP1 ノードが停止中です。または異常終了中です。

宣言

【C#の場合】

```
public const int DCADM_STAT_TERM
```

【Visual Basic の場合】

```
Public Const DCADM_STAT_TERM As Integer
```

【J#の場合】

```
public static final int DCADM_STAT_TERM
```

●DCTAM_STS_CLS

説明

TAM テーブルがクローズ状態であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_CLS
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_CLS As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_CLS
```

●DCTAM_STS_CMD

説明

tamload コマンドを実行したときであることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_CMD
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_CMD As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_CMD
```

●DCTAM_STS_HASH

説明

ハッシュ形式であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_HASH
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_HASH As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_HASH
```

●DCTAM_STS_LHLD

説明

TAM テーブルが論理閉塞状態であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_LHLD
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_LHLD As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_LHLD
```

●DCTAM_STS_LIB

説明

Tam.Open メソッドで TAM テーブルをオープンしたときであることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_LIB
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_LIB As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_LIB
```

●DCTAM_STS_NOSEC

説明

セキュリティの指定がないことを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_NOSEC
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_NOSEC As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_NOSEC
```

●DCTAM_STS_OHLD

説明

TAM テーブルが障害閉塞状態であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_OHLD
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_OHLD As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_OHLD
```

●DCTAM_STS_OPN

説明

TAM テーブルがオープン状態であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_OPN
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_OPN As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_OPN
```

●DCTAM_STS_READ

説明

参照型であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_READ
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_READ As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_READ
```

●DCTAM_STS_RECLCK

説明

テーブル排他を確保しない，追加・削除できる更新型であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_RECLCK
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_RECLCK As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_RECLCK
```

●DCTAM_STS_REWRITE

説明

追加・削除できない更新型であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_REWRITE
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_REWRITE As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_REWRITE
```

●DCTAM_STS_SEC

説明

セキュリティの指定があることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_SEC
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_SEC As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_SEC
```

●DCTAM_STS_START

説明

TAM サービスの開始時であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_START
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_START As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_START
```

●DCTAM_STS_TREE

説明

ツリー形式であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_TREE
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_TREE As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_TREE
```

●DCTAM_STS_WRITE

説明

追加・削除できる更新型であることを示します。

宣言

【C#の場合】

```
public const int DCTAM_STS_WRITE
```

【Visual Basic の場合】

```
Public Const DCTAM_STS_WRITE As Integer
```

【J#の場合】

```
public static final int DCTAM_STS_WRITE
```

TP1UserException

TP1UserException の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.TP1UserException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

(ユーザアプリケーションプログラム例外)

TP1UserException クラスは，SPP.NET のサービスメソッド内でユーザがスローする例外です。

サービス呼び出し元の SUP.NET および CUP.NET では例外情報から復元してユーザにスローします。

コンストラクタの一覧

名称	説明
TP1UserException(System.Int32, System.String)	TP1UserException クラスのコンストラクタです。
TP1UserException(System.Int32, System.String, System.String)	エラーコード，エラー情報，およびこの例外の原因を説明するメッセージを指定するコンストラクタです。

プロパティの一覧

名称	説明
Information	エラー情報の文字列を返します。

メソッドの一覧

名称	説明
ToString()	この Exception クラスを表す文字列を返します。

コンストラクタの詳細

●TP1UserException

説明

TP1UserException クラスのコンストラクタです。

宣言

【C#の場合】

```
public TP1UserException(
  int errCode,
  string argInfo
);
```

【Visual Basic の場合】

```
Public New( _
  ByVal errCode As Integer, _
  ByVal argInfo As String _
)
```

【J#の場合】

```
public TP1UserException(
  int errCode,
  System.String argInfo
);
```

パラメタ

errCode

エラーコード。

argInfo

エラー情報。

例外

なし

●TP1UserException

説明

エラーコード，エラー情報，およびこの例外の原因を説明するメッセージを指定するコンストラクタです。

宣言

【C#の場合】

```
public TP1UserException(
  int errCode,
  string argInfo,
```

```
  string message
);
```

【Visual Basic の場合】

```
Public New( _
  ByVal errCode As Integer, _
  ByVal argInfo As String, _
  ByVal message As String _
)
```

【J#の場合】

```
public TP1UserException(
  int errCode,
  System.String argInfo,
  System.String message
);
```

パラメタ

errCode

エラーコード。

argInfo

エラー情報。

message

例外の原因を説明するエラーメッセージ。

例外

なし

プロパティの詳細

●Information

説明

エラー情報の文字列を返します。

宣言

【C#の場合】

```
public string Information {get;}
```

【Visual Basic の場合】

```
Public ReadOnly Property Information As String
```

【J#の場合】

```
public System.String get_Information();
```

例外

なし

メソッドの詳細

●ToString

説明

この Exception クラスを表す，次の形式の文字列を返します。

Exception クラスの名称: Message プロパティの値

ErrorCode ＝ ErrorCode プロパティの値

Information ＝ Information プロパティの値

StackTrace プロパティの値

宣言

【C#の場合】

```
public virtual string ToString(
);
```

【Visual Basic の場合】

```
Public Overridable Function ToString( _
) As String
```

【J#の場合】

```
public System.String ToString(
);
```

パラメタ

なし

戻り値

この Exception クラスを表す文字列。

例外

なし

TP1UserStruct

TP1UserStruct の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.TP1UserStruct
```

説明

TP1 ユーザ構造体を利用するには，このクラスを継承する必要があります。

Trn

Trn の概要

名前空間

Hitachi.OpenTP1.Server

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Server.Trn
```

説明

Trn クラスは，OpenTP1 独自のトランザクション制御をするメソッドを提供します。

メソッドの一覧

名称	説明
Begin()	グローバルトランザクションを，このメソッドを呼び出したプロセスから開始します。
Commit()	グローバルトランザクションの正常終了（コミット）を，トランザクションを構成するトランザクションブランチ，トランザクションサービス，およびリソースマネジャに知らせます。
CommitChained()	トランザクションの同期点を取得します。グローバルトランザクションのルートトランザクションブランチとして，処理が正常に終了したこと（コミット）を，トランザクションを構成するトランザクションブランチの UAP，トランザクションサービス，およびリソースマネジャに知らせます。
GetInfo()	GetInfo メソッドを呼び出した UAP が，現在トランザクションとして起動しているかどうかを返します。
Rollback()	トランザクションをロールバックします。非連鎖モードでロールバックしたあとには，トランザクションは続けて開始しません。
RollbackChained()	トランザクションをロールバックします。RollbackChained メソッドを呼び出したあとには，続けてトランザクションが開始します。

メソッドの詳細

●Begin

説明

グローバルトランザクションを，このメソッドを呼び出したプロセスから開始します。

Begin メソッドを呼び出したプロセスは，グローバルトランザクションのルートトランザクションブランチになります。

Begin メソッドを呼び出した UAP は，実行環境の設定でトランザクション属性を指定しておいてください。また，すでに Begin メソッドを呼び出しているグローバルトランザクションの中では，どのト

ランザクションブランチからも再び Begin メソッドを呼び出せません。一つのグローバルトランザクション中で重複して呼び出した場合は例外が発生します。

宣言

【C#の場合】

```
public static void Begin(
);
```

【Visual Basic の場合】

```
Public Shared Sub Begin( _
)
```

【J#の場合】

```
public static void Begin(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTRNER_PROTO	Begin メソッドを不正なコンテキスト（例えば，すでにトランザクション内にいる）から呼び出しています。 Rpc クラスの Open メソッドを呼び出していません。
DCTRNER_RM	リソースマネジャ（RM）でエラーが発生しました。

エラーコード	説明
	トランザクションは開始できませんでした。
DCTRNER_TM	トランザクションサービスでエラーが起こったので，トランザクションは開始できませんでした。このエラーコードが戻った場合は，再び Begin メソッドを実行すると成功する可能性が高いので，再実行してください。

●Commit

説明

グローバルトランザクションの正常終了（コミット）を，トランザクションを構成するトランザクションブランチ，トランザクションサービス，およびリソースマネジャに知らせます。

Commit メソッドが正常に終了したあとで，新しいグローバルトランザクションは開始しません。

グローバルトランザクションが複数のトランザクションブランチから構成されるとき（メソッドを呼び出した UAP だけでないとき）は，それぞれのトランザクションブランチの処理結果がコミットとならないかぎり，コミットされません。

Commit メソッドを呼び出せるのは，ルートトランザクションブランチ（トランザクションを開始した UAP）だけです。それ以外の UAP から呼び出した場合は，例外が発生します。

Commit メソッドを呼び出すプロセスは，このクラスリファレンスの記述に従って正しく作成された UAP を稼働させたものでなければなりません。

Commit メソッドは，同期点処理が完了したときに，正常に終了，またはエラーリターンのどちらかで返ります。Commit メソッドを呼び出すサービスが正常に終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが前提となります。

宣言

【C#の場合】

```
public static void Commit(
);
```

【Visual Basic の場合】

```
Public Shared Sub Commit( _
)
```

【J#の場合】

```
public static void Commit(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTRNER_HAZARD	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このエラーコードが返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。プロセスはグローバルトランザクションの範囲外です。
DCTRNER_HEURISTIC	Commit メソッドを呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットし，あるトランザクションブランチはロールバックしました。 このエラーコードは，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合に返します。 このエラーコードが返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。プロセスはグローバルトランザクションの範囲外です。
DCTRNER_PROTO	Commit メソッドを不正なコンテキスト（例えば，すでにトランザクション中にいない）で呼び出しています。 トランザクションモードに対する影響はありません。 Rpc クラスの Open メソッドを呼び出していません。
DCTRNER_ROLLBACK	現在のトランザクションは，コミットできないでロールバックしました。プロセスはトランザクションの範囲外です。

●CommitChained

説明

トランザクションの同期点を取得します。グローバルトランザクションのルートトランザクションブランチとして，処理が正常に終了したこと（コミット）を，トランザクションを構成するトランザクションブランチの UAP，トランザクションサービス，およびリソースマネジャに知らせます。

CommitChained メソッドが正常に終了すると，新しいグローバルトランザクションが開始され，メソッドを呼び出したプロセスはこのトランザクションの範囲内になります。しかし，このことは，このメソッドを呼び出した UAP 以外のトランザクションモードに対しての指定を意味しません。

グローバルトランザクションが複数のトランザクションブランチから構成されるとき（メソッドを呼び出した UAP だけでないとき）は，それぞれのトランザクションブランチの処理結果がコミットとならないかぎりコミットされません。

CommitChained メソッドを呼び出せるのはルートトランザクションブランチ（Begin メソッドを呼び出した UAP）だけです。それ以外の UAP から呼び出した場合は，例外が発生します。

CommitChained メソッドを呼び出すプロセスは，このクラスリファレンスの記述に従って正しく作成された UAP を稼働させたものでなければなりません。

CommitChained メソッドは，同期点処理が完了したときに正常に終了，または例外応答のどちらかで返ります。CommitChained メソッドが正常に終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが必要です。

宣言

【C#の場合】

```
public static void CommitChained(
);
```

【Visual Basic の場合】

```
Public Shared Sub CommitChained( _
)
```

【J#の場合】

```
public static void CommitChained(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

例外の内容が出力されます。

OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

"OpenTP1 提供関数実行時にエラーが発生しました。"

それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名

  例外が発生したメソッド名が出力されます。

- エラーコード

  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTRNER_HAZARD	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このエラーコードが返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このエラーコードが返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_HAZARD	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このエラーコードが返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このエラーコードが返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_HAZARD_NO_BEGIN	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このエラーコードが返される原因となった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。新しいトランザクションは開始できませんでした。このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_HEURISTIC	CommitChained メソッドを呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットし，あるトランザクションブランチはロールバックしました。 このエラーコードは，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンします。 このエラーコードが返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。

エラーコード	説明
	このエラーコードが返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_HEURISTIC_NO_BEGIN	CommitChained メソッドを呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットし，あるトランザクションブランチはロールバックしました。 このエラーコードは，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンされます。 このエラーコードが返される原因となった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。新しいトランザクションは開始できませんでした。このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_NO_BEGIN	コミット処理は正常に終了しましたが，新しいトランザクションは開始できませんでした。このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_PROTO	CommitChained メソッドを不正なコンテキスト（例えば，すでにトランザクション中にいない）で呼び出しています。 トランザクションモードに対する影響はありません。 Rpc クラスの Open メソッドを呼び出していません。
DCTRNER_ROLLBACK	現在のトランザクションは，コミットできないでロールバックしました。 このエラーコードが返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。

●GetInfo

説明

GetInfo メソッドを呼び出した UAP が，現在トランザクションとして起動しているかどうかを返します。

GetInfo メソッドを呼び出すプロセスは，このクラスリファレンスの記述に従って正しく作成された UAP を稼働させたものでなければなりません。

GetInfo メソッドを呼び出すサービスが正常終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが前提です。

宣言

【C#の場合】

```
public static int GetInfo(
);
```

【Visual Basic の場合】

```
Public Shared Function GetInfo( _
) As Integer
```

【J#の場合】

```
public static int GetInfo(
);
```

パラメタ

なし

戻り値

- 1
  GetInfo メソッドを呼び出したプロセスは，トランザクションとして起動しています。
- 0
  GetInfo メソッドを呼び出したプロセスは，トランザクションとして起動していません。

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTRNER_PROTO	Rpc クラスの Open メソッドを呼び出していません。

注意事項

この API は UAP トレースを取得していません。

●Rollback

説明

トランザクションをロールバックします。非連鎖モードでロールバックしたあとには，トランザクションは続けて開始しません。

Rollback メソッドを呼び出すことで，トランザクションブランチ，トランザクションサービス，およびリソースマネジャにロールバックを知らせます。

Rollback メソッドは，グローバルトランザクションのどのトランザクションブランチからでも呼び出せます。

ルートトランザクションブランチから呼び出した場合，Rollback メソッドが正常終了したあとには，新しいトランザクションは開始しません。

ルートトランザクションブランチ以外から呼び出した場合は，そのトランザクションブランチを rollback_only 状態にします。

この場合，ルートトランザクションブランチの同期点処理が完了するまで，Rollback メソッドを呼び出したトランザクションブランチはトランザクションの範囲内です。

Rollback メソッドを呼び出すプロセスは，このクラスリファレンスの記述に従って正しく作成された UAP を稼働させたものでなければなりません。

Rollback メソッドを呼び出すサービスが正常に終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが前提です。

宣言

【C#の場合】

```
public static void Rollback(
);
```

【Visual Basic の場合】

```
Public Shared Sub Rollback( _
)
```

【J#の場合】

```
public static void Rollback(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ

  例外の内容が出力されます。

  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。

  "OpenTP1 提供関数実行時にエラーが発生しました。"

  それ以外の場合は，各エラーに対応したメッセージが出力されます。

- クラス名

  例外が発生したクラス名が出力されます。

- メソッド名
  例外が発生したメソッド名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTRNER_HAZARD	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このエラーコードが返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このエラーコードが返ったあと，このプロセスはトランザクション下になく，グローバルトランザクションの範囲外です。
DCTRNER_HAZARD_NO_BEGIN	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このエラーコードが返される原因となった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。新しいトランザクションは開始できませんでした。このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_HEURISTIC	Rollback メソッドを呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットし，あるトランザクションブランチはロールバックしました。 このエラーコードは，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合に返します。 このエラーコードが返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このエラーコードが返ったあと，このプロセスはトランザクション下になく，グローバルトランザクションの範囲外です。
DCTRNER_HEURISTIC_NO_BEGIN	RollbackChained メソッドを呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットし，あるトランザクションブランチはロールバックしました。 このエラーコードは，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンされます。 このエラーコードが返される原因となった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。新しいトランザクションは開始できませんでした。このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_NO_BEGIN	ロールバック処理は正常終了しましたが，新しいトランザクションは開始できませんでした。このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。

エラーコード	説明
DCTRNER_PROTO	Rollback メソッドを不正なコンテキスト（例えば，すでにトランザクション中にいない）で呼び出しています。 トランザクションモードに対する影響はありません。 Rpc クラスの Open メソッドを呼び出していません。

●RollbackChained

説明

トランザクションをロールバックします。RollbackChained メソッドを呼び出したあとには，続けてトランザクションが開始します。

RollbackChained メソッドを呼び出すことで，ルートトランザクションブランチから，トランザクションブランチ，トランザクションサービス，およびリソースマネジャにロールバックを知らせます。

RollbackChained メソッドが正常に終了すると，メソッドを呼び出したプロセスはロールバックしてリターンします。

そのあとで新しいグローバルトランザクションが開始します。

メソッドを呼び出したプロセスはこのトランザクションの範囲内です。

ただし，このメソッドを呼び出した UAP 以外のトランザクションモードに対しての指定を意味しません。

RollbackChained メソッドを呼び出せるのは，ルートトランザクションブランチ（Begin メソッドを呼び出した UAP）からだけです。

それ以外の UAP から呼び出した場合は，例外が発生します。

RollbackChained メソッドを呼び出すプロセスは，このクラスリファレンスの記述に従って正しく作成された UAP を稼働させたものでなければなりません。

RollbackChained メソッドは，同期点処理が完了したときに，正常終了，または例外応答のどちらかで返ります。

RollbackChained メソッドを呼び出すサービスが正常終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが前提です。

宣言

【C#の場合】

```
public static void RollbackChained(
);
```

【Visual Basic の場合】

```
Public Shared Sub RollbackChained( _
)
```

【J#の場合】

```
public static void RollbackChained(
);
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Server.TP1ServerException

次の情報が出力されます。

- メッセージ
  例外の内容が出力されます。
  OpenTP1 提供関数内でエラーが発生した場合は，次のように出力されます。
  "OpenTP1 提供関数実行時にエラーが発生しました。"
  それ以外の場合は，各エラーに対応したメッセージが出力されます。
- クラス名
  例外が発生したクラス名が出力されます。
- メソッド名
  例外が発生したメソッド名が出力されます。
- エラーコード
  発生原因に応じ，次のエラーコードが出力されます。

エラーコード	説明
DCTRNER_HAZARD	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このエラーコードが返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このエラーコードが返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_HAZARD_NO_BEGIN	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このエラーコードが返される原因となった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。新しいトランザクションは開始できませんでした。このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_HEURISTIC	RollbackChained メソッドを呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットし，あるトランザクションブランチはロールバックしました。

エラーコード	説明
	このエラーコードは，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンします。 このエラーコードが返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このエラーコードが返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_HEURISTIC_NO_BEGIN	RollbackChained メソッドを呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットし，あるトランザクションブランチはロールバックしました。 このエラーコードは，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンされます。 このエラーコードが返される原因となった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。新しいトランザクションは開始できませんでした。このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_NO_BEGIN	ロールバック処理は正常に終了しましたが，新しいトランザクションは開始できませんでした。このエラーコードが返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_PROTO	RollbackChained メソッドを不正なコンテキスト（例えば，すでにトランザクション中にいない）で呼び出しています。 トランザクションモードに対する影響はありません。 Rpc クラスの Open メソッドを呼び出していません。

UByteArrayHolder

UByteArrayHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.UByteArrayHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

UByteArrayHolder クラスは，System.Byte 配列を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public byte[] Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As Byte()
```

【J#の場合】

```
public ubyte[] get_Value();
public void set_Value(ubyte[]);
```

例外

なし

UByteHolder

UByteHolder の概要

名前空間

Hitachi.OpenTP1

継承関係

```
System.Object
  +- Hitachi.OpenTP1.UByteHolder
```

実装インタフェース

Hitachi.OpenTP1.Common.IHolder

説明

UByteHolder クラスは，System.Byte 値を保持するホルダークラスです。

プロパティの一覧

名称	説明
Value	保持している変数に対して値の設定および参照を行います。

プロパティの詳細

●Value

説明

保持している変数に対して値の設定および参照を行います。

宣言

【C#の場合】

```
public byte Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As Byte
```

【J#の場合】

```
public ubyte get_Value();
public void set_Value(ubyte);
```

例外

なし

8 障害運用

この章では，TP1/Extension for .NET Framework（Extension .NET）で障害が発生した場合の対処方法を説明します。

8.1　障害の種類と対処方法

Extension .NET で発生するおそれがある障害の種類とユーザが取る処置を次の表に示します。

表 8-1　障害の種類とユーザの取る処置

障害の種類	障害の内容	Extension .NET の処理	ユーザが取る処置
通信障害	・サーバ障害 ・ネットワーク障害 など	1. 障害発生時の情報をメッセージログファイルに出力します。 2. 例外を UAP に通知します。	障害原因を調査して取り除くか，または障害回復を待ち，再度処理を実行してください。 必要に応じて OpenTP1 の定義の内容を見直してください。
ユーザサーバ障害	・ユーザサーバの起動失敗，または停止失敗 ・ユーザサーバダウン	1. 障害発生時の情報をメッセージログファイルに出力します。 2. ユーザサーバを異常終了させます。	SPP.NET および SUP.NET の処理，または環境を見直してください。 必要に応じて SPP.NET および SUP.NET のユーザサービス定義，および OpenTP1 の定義の内容を見直してください。
サーバアプリケーション障害※1	SPP.NET での例外発生（ユーザ例外）	1. 障害発生時の詳細情報を.NET エラーログファイルに出力します。 2. TP1UserException をクライアント UAP に通知します。	SPP.NET の処理，入力データなどを見直してください。
	SPP.NET での例外発生（ユーザ例外以外）	1. 障害発生時の詳細情報を.NET エラーログファイルに出力します。 2. TP1RemoteException をクライアント UAP に通知します。	SPP.NET の処理，実行環境などを見直してください。
データ変換障害	・サーバスタブでのデータ変換エラー ・クライアントスタブでのデータ変換エラー	1. 障害発生時の詳細情報を.NET エラーログファイルに出力します。 2. TP1MarshalException をクライアント UAP に通知します。	使用している.NET インタフェース定義の内容を見直してください。または，サービス定義およびデータ型定義の内容を見直してください。
マシン環境障害	・ファイル障害 ・メモリ不足 など	1. 障害発生時の情報をメッセージログファイルに出力します。※2 2. 例外を UAP に通知します。	障害の要因を取り除いてください。

注※1

サーバアプリケーションが.NET インタフェース定義を使用した SPP.NET の場合にだけ発生します。それ以外の場合は通信障害として通知されます。

注※2

メッセージログファイルへの出力や.NET エラーログファイルへの出力で障害が発生した場合は，メッセージログファイルへの出力や.NET エラーログファイルへの出力のエラーを無視して処理を続行します。以降，ログは出力されません。

8.2　障害時に取得する情報

障害が発生した場合は，次に示す情報を取得してください。また，必要に応じて，取得した情報をシステム管理者に提供してください。

なお，以降の説明の「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

例外が発生した場合

　例外の情報を ToString メソッドで取得して，ファイルに出力するか画面に表示してください。必要に応じて%DCDIR%¥spool ディレクトリ下のファイルもあわせて取得してください。

コマンドがエラーとなった場合

　コマンドが出力したエラーメッセージ，コマンドの入力形式，入力ファイルなどを保存してください。

その他の障害の場合

　%DCDIR%¥spool ディレクトリ下のファイルを保存してください。

8.3　.NET エラーログファイル

.NET エラーログファイルには，SPP.NET，SUP.NET などのサーバアプリケーションで障害，または例外が発生した場合に詳細情報が出力されます。特に例外が発生した場合は例外のスタックトレースが取得されるため，UAP の障害要因の特定に役立ちます。

.NET エラーログは次に示すファイルにテキスト形式で出力されます。

%DCDIR%¥spool¥njslog1

%DCDIR%¥spool¥njslog2

ファイルの最大サイズは 1 メガバイトです。

njslog1，njslog2 が交代で使用されます。

なお，「%DCDIR%」は OpenTP1 インストールディレクトリを表しています。

.NET エラーログの出力形式を次に示します。

```
Day Mon DD HH:MM:SS.UUU YYYY
サーバ名：UU...UU サービスグループ名：GG...GG サービス名：SS...SS　プロセスＩＤ：PPPPP
KFCAxxxxx-E 〈エラーメッセージ〉
〈スタックトレース情報〉
```

Day Mon DD HH:MM:SS.UUU YYYY：障害の発生時刻をミリ秒単位で出力します。

UU...UU：障害が発生したユーザサーバ名を出力します。

GG...GG：障害が発生したサービスグループ名を出力します。

SS...SS：障害が発生したサービス名を出力します。

PPPPP：障害が発生したプロセス ID を出力します。

KFCAxxxxx-E：メッセージ ID を示します。詳細については，マニュアル「OpenTP1　メッセージ」を参照してください。

<エラーメッセージ>：エラーメッセージの内容を示します。詳細については，マニュアル「OpenTP1 メッセージ」を参照してください。

<スタックトレース情報>：例外が発生した場合に出力されます。例外発生時のスタックトレース情報が出力されます。

.NET エラーログの出力例を次に示します。

```
Fri Apr 16 22:12:41.486 2004
サーバ名：CSSPPBIN サービスグループ名：EXTNET_SVG_CS_SPPBIN サービス名：******************
プロセスＩＤ：10706
KFCA32216-E SPP.NET実行サービス(CSSPPBIN)はSPP.NET実装クラスから例外を受け取りました。詳細情
報＝SPP.NETの実装クラス内のメソッド実行時に例外が発生しました。実装クラス名＝CSSample.CSSPPB
IN, メソッド名＝InitializeSPP, 詳細情報＝System.Exception: 例外のメッセージです
   at CSSample.CSSPPBIN.InitializeSPP()
   at Hitachi.OpenTP1.Server.Internal.ServerInstance.InitializeSPP(String assemblyName,
String className, Int32& methodRC)
```

付録

付録 A　クラス名およびメソッド名と C 言語の関数名の対応

SPP.NET および SUP.NET で利用できるクラス名およびメソッド名と C 言語の関数名の対応を次の表に示します。

表 A-1　SPP.NET および SUP.NET で利用できるクラス名およびメソッド名と C 言語の関数名の対応

名前空間	クラス名	メソッド名	対応する C 言語の関数名
Hitachi.OpenTP1.Server	Adm	CallCommand	dc_adm_call_command
		Complete	dc_adm_complete
		GetStatus	dc_adm_status
	Jnl※1	PutUJ	dc_jnl_ujput
	Lck※1	Get	dc_lck_get
		ReleaseAll	dc_lck_release_all
		ReleaseByName	dc_lck_release_byname
	Log	Print	dc_logprint
	Mcf	Send	dc_mcf_send
		SendReceive	dc_mcf_sendrecv
	Prf	GetTraceNum	dc_prf_get_trace_num
		PutUTrace	dc_prf_utrace_put
	Rap	Connect	dc_rap_connect
		Disconnect	dc_rap_disconnect
	Rpc	Call	dc_rpc_call
		CallTo	dc_rpc_call_to
		Close	dc_rpc_close
		DiscardFurtherReplies	dc_rpc_discard_further_replies
		DiscardSpecificReply	dc_rpc_discard_specific_reply
		GetCallersAddress	dc_rpc_get_callers_address
		GetErrorDescriptor	dc_rpc_get_error_descriptor
		GetGatewayAddress	dc_rpc_get_gateway_address
		GetServicePriolity	dc_rpc_get_service_prio
		GetTimeout	dc_rpc_get_watch_time
		Open	dc_rpc_open

名前空間	クラス名	メソッド名	対応する C 言語の関数名
		PollAnyReplies	dc_rpc_poll_any_replies
		SetBindTable	DCRPC_BINDTBL_SET
		SetDirectSchedule	DCRPC_DIRECT_SCHEDULE
		SetServicePriolity	dc_rpc_set_service_prio
		SetServiceRetry	dc_rpc_service_retry
		SetTimeout	dc_rpc_set_watch_time
	RpcBindTable 構造体	−	DCRPC_BINDING_TBL
	Rts	PutUTrace	dc_rts_utrace_put
	Tam※1	Close	dc_tam_close
		Delete	dc_tam_delete
		GetInfo	dc_tam_get_inf
		GetStatus	dc_tam_status
		Open	dc_tam_open
		Read	dc_tam_read
		ReadCancel	dc_tam_read_cancel
		Rewrite	dc_tam_rewrite
		Write	dc_tam_write
	TamKeyTable 構造体※1	−	DC_TAMKEY
	TamStatusTable 構造体※1	−	DC_TAMSTAT
	TP1ServerException	−	TP1/Server 系共通例外
	TP1ServerFlags	−	TP1/Server 系フラグ値定義
	TP1ServerLimits	−	TP1/Server 系制限値定義
	TP1ServerValues	−	TP1/Server 系ステータス値定義
	Trn	Begin	dc_trn_begin
		Commit	dc_trn_unchained_commit
		CommitChained	dc_trn_chained_commit

名前空間	クラス名	メソッド名	対応する C 言語の関数名
		GetInfo	dc_trn_info
		Rollback	dc_trn_unchained_rollback
		RollbackChained	dc_trn_chained_rollback
	SPPBase	FinalizeSPP	―※2
		InitializeSPP	

（凡例）

－：該当しません。

注※1

TP1/Server Base の場合だけ使用できます。TP1/LiNK の場合は使用できません。

注※2

SPP.NET では Main 関数を記述しないため dc_rpc_mainloop 関数に相当するメソッドはありません。代わりに，必要に応じて InitializeSPP メソッド，FinalizeSPP メソッドをオーバライドします。

付録 B　DCCM3 と接続する場合の注意事項

Extension .NET では，RPC やメッセージ送受信機能を使用して，VOS3 や VOS1 上の DCCM3 と接続できます。実際の通信は，Extension .NET の前提製品である TP1/Server Base または TP1/LiNK，メッセージ送受信機能使用時の前提製品である TP1/NET/TCP/IP または TP1/Messaging が行います。通信時の前提条件などについては，次のマニュアルを参照してください。

- 「OpenTP1　解説」
- 「TP1/LiNK　使用の手引」
- 「OpenTP1　プロトコル　TP1/NET/TCP/IP 編」
- 「TP1/Messaging　使用の手引」

Extension .NET を使用して DCCM3 と接続する場合の注意事項を次に示します。

RPC を使用する場合

- 連鎖 RPC は使用できません。
- .NET インタフェース定義を使用した RPC は使用できません。
- サービス定義を使用した RPC は，次の条件をすべて満たしている場合だけ使用できます。
  - データ型定義によってデータの形式をメッセージの単位に定義できる。
  - 文字列を含まないデータ，または.NET Framework がサポートしているエンコード方式でエンコードおよびデコードできる文字列のデータを送受信する。

    なお，.NET Framework がサポートしているエンコード方式については，.NET Framework のドキュメントを参照してください。
- トランザクション制御機能は使用できません。
- 文字列のデータを含むメッセージを送受信する場合，あらかじめメッセージ中の文字コードを通信先システムと決めた上で，必要に応じて UAP で文字コード変換を行ってください。.NET Framework 上では，Unicode（リトルエンディアン）が文字列データのメモリ上の内部表現として使用されます。ただし，COBOL2002 for .NET Framework を使用した場合の文字列データのメモリ内部表現については，マニュアル「COBOL2002 for .NET Framework　ユーザーズガイド」を参照してください。
- マニュアル「OpenTP1　解説」またはマニュアル「TP1/LiNK　使用の手引」の注意事項もあわせて参照してください。

メッセージ送受信機能を使用する場合

- 文字列のデータを含むメッセージを送受信する場合，あらかじめメッセージ中の文字コードを通信先システムと決めた上で，必要に応じて UAP で文字コード変換を行ってください。.NET Framework 上では，Unicode（リトルエンディアン）が文字列データのメモリ上の内部表現として使用されます。ただし，COBOL2002 for .NET Framework を使用した場合の文字列データのメモリ内部表現については，マニュアル「COBOL2002 for .NET Framework　ユーザーズガイド」を参照してください。

- マニュアル「OpenTP1　プロトコル　TP1/NET/TCP/IP 編」またはマニュアル「TP1/Messaging　使用の手引」の注意事項もあわせて参照してください。

付録 C　バージョンアップ時の変更点

各バージョンでの変更点を次に示す分類ごとに示します。

- API，定義およびコマンドの追加と削除
- 動作の変更
- API，定義およびコマンドのデフォルト値の変更

付録 C.1　07-50 での変更点

Extension .NET 07-50 での API，定義およびコマンドの追加と削除を次の表に示します。

表 C–1　Extension .NET 07-50 での API，定義およびコマンドの追加と削除

種別	分類	内容
追加	API	なし
	定義	なし
	コマンド	なし
削除	API	なし
	定義	なし
	コマンド	なし
	その他	前提ソフトウェアのサポート終了に伴い，次の言語のサンプルプログラムを削除 • J# • COBOL 言語

Extension .NET 07-50 での動作の変更点を次の表に示します。

表 C–2　Extension .NET 07-50 での動作の変更点

分類	内容
API	なし
定義	なし
コマンド	if2cstub コマンド，if2sstub コマンド，if2tsdl コマンド，および spp2cstub コマンドに指定できるプログラム言語の種類から J#を削除
メッセージ	KFCA32271-I，KFCA32275-I • -l オプションの指定値から vjs を削除
その他	前提ソフトウェアのサポート終了に伴い，使用可能な開発言語から次の言語を除外 • J# • COBOL 言語

分類	内容
	適用 OS を Windows 7，および Windows Server 2008 R2 以降に変更
	Extension .NET を使用する場合の前提ソフトウェアを.NET Framework 3.5 Service Pack 1 に変更

Extension .NET 07-50 での API，定義およびコマンドのデフォルト値の変更はありません。

付録 D　用語解説

Extension .NET に関する用語について説明します。その他の用語については，必要に応じて次のマニュアルおよびドキュメントを参照してください。

- 「OpenTP1　解説」
- 「TP1/LiNK　使用の手引」
- .NET Framework のドキュメント

(記号)

.NET インタフェース定義

SPP.NET の RPC インタフェース情報を.NET Framework に対応したプログラム言語で定義したものです。

(英字)

ADO.NET

.NET Framework が提供するデータアクセスの技術セットです。

ASP.NET

.NET Framework が提供する Web アプリケーションや XML Web サービスを構築するための技術セットです。

CLR（Common Language Runtime）

→共通言語ランタイムの説明を参照してください。

CLS（Common Language Specification）

→共通言語仕様の説明を参照してください。

CTS（Common Type System）

→共通型システムの説明を参照してください。

CUP.NET（Client User Program for .NET Framework）

TP1/Client for .NET Framework を利用して，SPP.NET や SPP にサービスを要求するクライアントアプリケーションのことです。

さまざまなアプリケーション形態が可能であり，アプリケーション形態に応じたランタイムホストで実行されます。

GAC（Global Assembly Cache）

→グローバルアセンブリキャッシュの説明を参照してください。

SPP.NET（Service Providing Program for .NET Framework）

OpenTP1 for .NET Framework の UAP のうち，ファイルへのアクセスなどサーバの役割をするプログラムのことです。SPP.NET は，クライアント UAP から要求されたサービスを実行するサービスメソッドから構成されます。

OpenTP1 for .NET Framework が提供するランタイムホストで実行されます。

SPP.NET 実行コンテナ

TP1/Extension for .NET Framework が提供する SPP.NET 実行用のランタイムホストです。

SUP.NET（Service Using Program for .NET Framework）

OpenTP1 for .NET Framework の UAP のうち，SPP.NET または SPP に処理要求をするだけの，クライアントアプリケーションのことです。ほかの UAP にサービスを提供するためのメソッドは持ちません。

TSP（TP1 Service Proxy）

TSP は，コネクションを意識しないで OpenTP1 のサービスを利用できるプロキシクラスです。

XML Web サービス

XML，HTTP，SOAP などのインターネット標準技術を利用して，ネットワーク上に分散した異なるプラットフォーム上のアプリケーションを連携させる技術，または連携したアプリケーション（サービス）のことです。

（ア行）

アセンブリ

.NET Framework のアプリケーションの配置やバージョン管理などを行う場合の基本単位です。アセンブリは，リソースやファイルの論理的な集合として，一つの機能を構成します。

アプリケーションドメイン

共通言語ランタイムがアプリケーション間を分離するために使用する処理単位です。

同じアプリケーションの有効範囲内（アプリケーションスコープ）で作成されたオブジェクトの集合の境界を示します。

アプリケーションベースディレクトリ

アプリケーションドメインに読み込まれる DLL ファイルまたは EXE ファイルを格納するディレクトリのことです。

インデクスドレコード

Connector .NET でバイナリデータを使用した，RPC を発行する場合および TCP/IP 通信をする場合に使用するインタフェースのことです。

（カ行）

カスタムレコードクラス

データ型定義で定義したデータ型を.NET Framework のデータ型に対応づけて生成したデータ変換クラスのことです。

完全限定名

名前空間の名前から指定するオブジェクトの参照方法です。完全限定名を使用すると，使用するオブジェクトを特定できるため，名前の競合が発生しません。

共通型システム（CTS）

共通言語ランタイムでの型の宣言，使用，および管理方法を定義したものです。

共通言語仕様（CLS）

共通言語ランタイムによってサポートされる言語機能のサブセットです。

共通言語ランタイム（CLR）

.NET Framework に対応するプログラムが使用する共通の動作環境（ランタイム）です。

クライアントスタブ

→スタブの説明を参照してください。

グローバルアセンブリキャッシュ（GAC）

複数のアプリケーションで共有するアセンブリを格納するコードキャッシュのことです。

コネクション

Connector .NET は，Client .NET が提供する TP1Client クラスのインスタンスにハンドルを関連づけて管理します。TP1Client クラスのインスタンスに関連づけられたハンドルのことを，コネクションと呼びます。

コネクションプーリング機能

一度生成されたコネクションをプーリングする機能のことです。コネクションプーリング機能を使用すると，コネクションプール内のコネクションを再利用できるため，資源を効率的に使用できます。

コネクションプール

コネクションプーリング機能使用時に，生成されたコネクションをプーリングするための領域のことです。

（サ行）

サーバスタブ

→スタブの説明を参照してください。

サービス定義

SPP.NET や SPP のサービス名と入出力データの定義を対応づけて定義したものです。

サービス定義ファイルとデータ型定義ファイルから構成されます。

出力データ用 XML スキーマ

クライアントスタブのサービスメソッドの戻り値（出力データ）である XML 文書の構造が定義された XML スキーマのことです。

出力データ用 XML スキーマファイル

出力データ用 XML スキーマが記述されたファイルのことです。

スタブ

.NET インタフェース定義を使用する RPC や，サービス定義を使用する RPC の場合に，RPC で使用する入出力データの文字コード変換やエンディアンの識別などを自動的に行うプログラムのことです。

サーバで使用するスタブをサーバスタブ，クライアントで使用するスタブをクライアントスタブと呼びます。

スタブは運用コマンドで生成します。.NET インタフェース定義を使用する RPC の場合は，クライアントスタブとサーバスタブが生成されます。サービス定義を使用する RPC の場合は，クライアントスタブが生成されます。

（タ行）

データ型定義

RPC メッセージのユーザデータの形式を C 言語の構造体のような形式で定義したものです。

（ナ行）

名前空間

名前を一意に識別するための概念です。名前空間と名前を組み合わせることによって，オブジェクトなどを特定できます。

また，名前空間によってクラスなどを論理的にグループ化できます。

入力データ用 XML スキーマ

クライアントスタブのサービスメソッドの引数（入力データ）である XML 文書の構造が定義された XML スキーマのことです。

入力データ用 XML スキーマファイル

入力データ用 XML スキーマが記述されたファイルのことです。

（ハ行）

バッファプーリング機能

RPC 要求に使用するメッセージを格納するバッファ（バイト配列）をプーリングする機能です。

バッファプーリング機能でバッファをプーリングすると，バッファの生成や破棄が少なくなり性能が向上します。

バッファプール

バッファプーリング機能使用時に，生成されたバッファを同じバッファサイズのバッファごとに管理するプールのことです。

索引

E

F

G

H

S

T

U

V

W

X

あ

い

う

え

お

か

き

く