HITACHI
Inspire the Next

OpenTP1 Version 7
分散トランザクション処理機能

TP1/Client for .NET Framework 使用の手引

解説・手引・文法・操作書

3000-3-D68-41

前書き

■ 対象製品

P-2464-7834 uCosminexus TP1/Client for .NET Framework 07-51（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，Windows Server 2019）

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

2022 年 4 月　3000-3-D68-41

■ 著作権

変更内容

変更内容（3000-3-D68-41） uCosminexus TP1/Client for .NET Framework　07-51

追加・変更内容	変更個所
マニュアル訂正を反映しました。	－
CUP.NET とサーバ間のコネクションで確保される TCP/IP の送受信バッファのサイズを変更できるようにした。 これに伴い，Client .NET 構成定義に次の属性を追加した。 • <socket>要素の sendBufferSize 属性 • <socket>要素の receiveBufferSize 属性	3. socket

単なる誤字・脱字などはお断りなく訂正しました。

変更内容（3000-3-D68-40） uCosminexus TP1/Client for .NET Framework　07-50

追加・変更内容
アプリケーションを開発する言語から J#，および COBOL を削除した。
Client .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。
Connector .NET を使用する場合の前提ソフトウェアを変更した。
リモート API 機能を使用した RPC で DCCM3 および TMS-4V/SP が使用可能なことを追加した。
サーバダウンなどの理由によって，指定したホストと一時的に通信ができなくなった場合に，ホストを切り替えて，別のホストに通信を試みる機能（ホスト切り替え機能）を追加した。 これに伴い，Client .NET 構成定義に次の属性を追加した。 • <rpc>要素の extendOpt 属性 • <nameService>要素の hostChangeMode 属性 • <nameService>要素の sysWatchTime 属性
TCP/IP 通信機能の説明を変更した。
TCP/IP 通信機能のユースケースごとの設定方法を追加した。
ノード間負荷バランス機能についての説明を変更した。
トレースファイルについての説明を変更した。
デバッグトレースについての説明を追加した。
Client .NET 構成定義に関する注意事項を追加した。
次の Client .NET 構成定義の説明を追加した。 • <tp1Server>要素 • <rpc>要素 • <tcpip>要素

追加・変更内容
<rapService>要素の inquireTime 属性に，0 以外を指定し，オートコネクトモードで Call メソッドを使用して RPC を実行した場合，CUP 実行プロセスでも問い合わせ間隔最大時間を監視するように変更した。
常設コネクションを使用し，かつオートコネクトモードの場合に，CUP 実行プロセスで「問い合わせ間隔最大時間」を監視するようにした。 これに伴い，Client .NET 構成定義に<rapService>要素の inquireTimeCheck 属性を追加した。
ネームサービスを使用した RPC において，そのサービス要求を受け付けた TP1/Server が優先的にサービス処理できるようにした。 これに伴い，Client .NET 構成定義に次の属性を追加した。 • <nameService>要素の usePriority 属性
コネクション確立要求を格納するキューのサイズ（listen システムコールのバックログ数）を指定できるようにした。 これに伴い，Client .NET 構成定義に次の属性を追加した。 • <socket>要素の backlogCount 属性
ThreadAbortException 例外が発生した場合の注意事項を追加した。
次のクラスの説明を変更した。 • ErrFatalException • ErrServerTimedOutException
次の TP1Client クラスのメソッドの説明を変更した。 • Call • CallTo • CloseRpc • OpenRpc • SendAssembledMessage • SetRapInquireTime
DCCM3 論理端末に対して RPC を行う場合の負荷分散の説明を変更，DCCM3 論理端末に対して RPC を行う場合の注意事項の説明を追加した。
バージョンアップ時の API，定義，コマンドおよびデフォルト値の変更を記載した。

はじめに

このマニュアルは，プログラムプロダクト P-2464-7834 uCosminexus TP1/Client 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/Client for .NET Framework の機能について説明しています。

第 3 章　構成定義

TP1/Client for .NET Framework で使用する構成定義について説明しています。

第 4 章　UAP の作成と実行

TP1/Client 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　運用コマンド

TP1/Client 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 6 章　クラスリファレンス

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

第 7 章　障害運用

TP1/Client 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　DCCM3 との接続

DCCM3 と RPC 通信する方法について説明しています。

付録 B　Client .NET で利用できるクラスのフィールド

TP1/Client for .NET Framework で利用できるクラスについて説明しています。

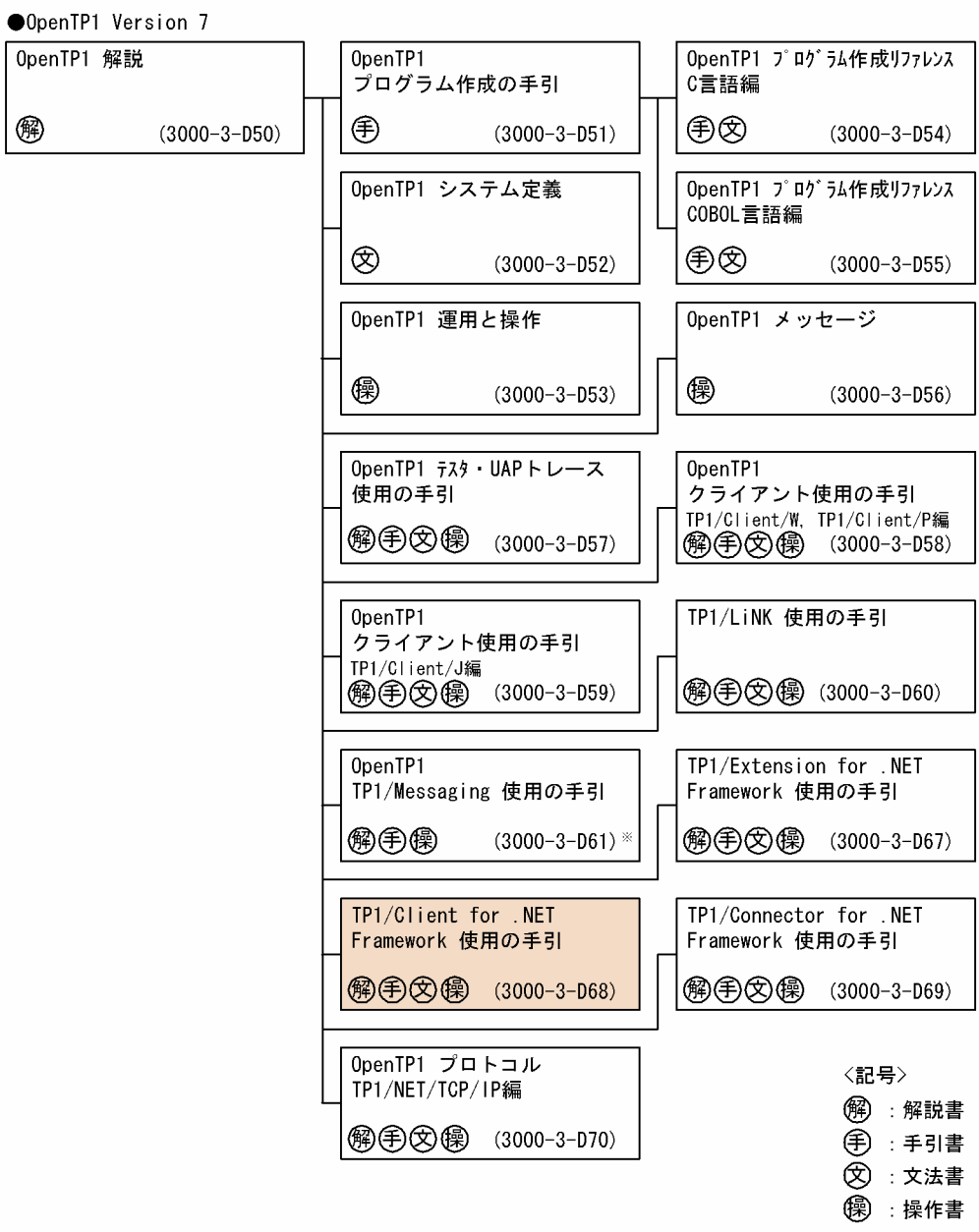
付録 C　バージョンアップ時の変更点

各バージョンでの API，定義およびメッセージの変更点について説明しています。

付録 D　用語解説

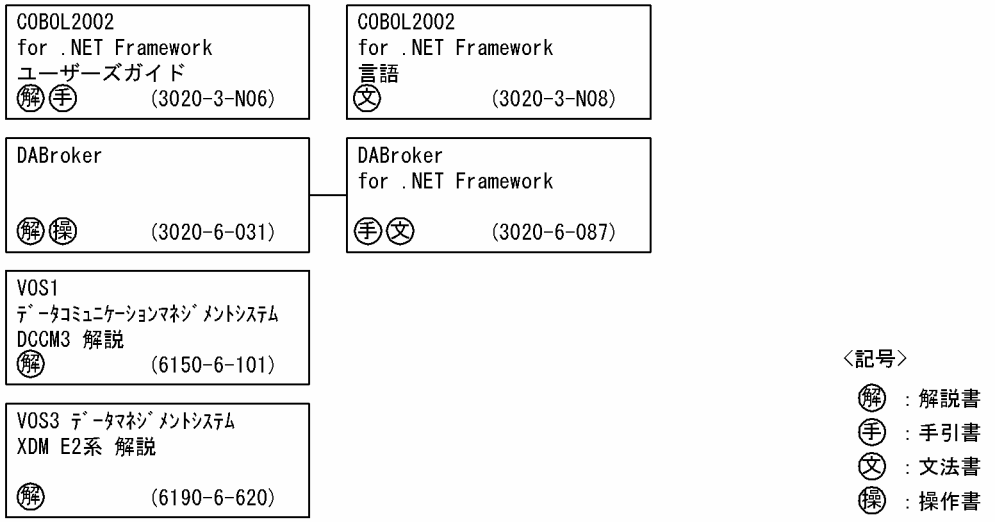
TP1/Client for .NET Framework に関する用語について説明しています。

■ 関連マニュアル



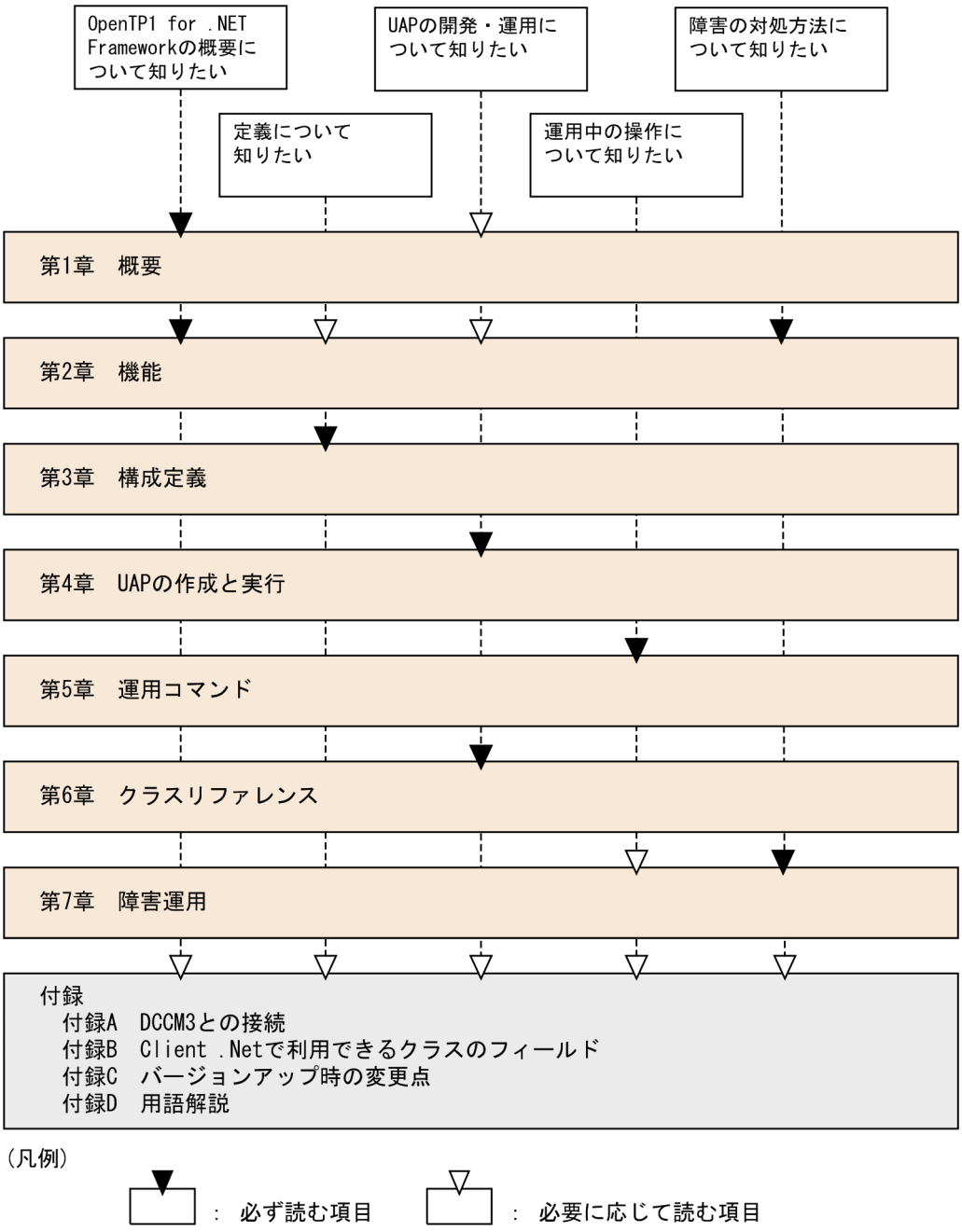
注※　このマニュアルおよびこのマニュアルが対象とする製品の発行時期についてはご確認ください。

●関連製品



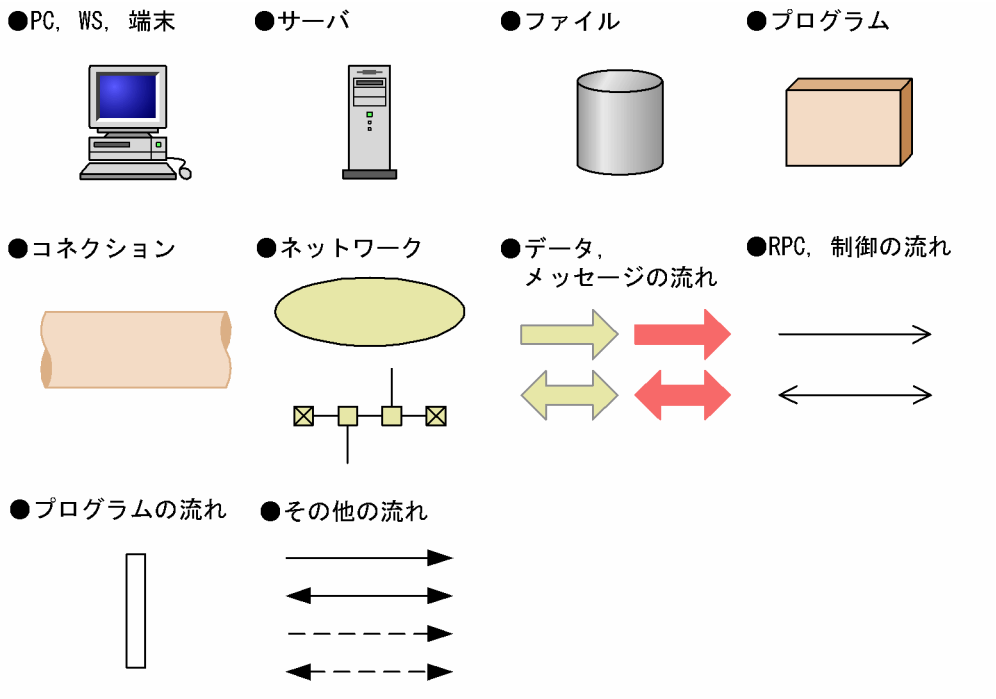
■ 読書手順

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。



■ 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。 ただし，構成定義の説明では省略できる項目にこの記号は使用しません。
…	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。

文法記述記号	意味
____（下線）	（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 .NET Framework	.NET Framework
Microsoft Visual C#	C#

製品名称	このマニュアルでの表記	
Microsoft Visual Basic	Visual Basic	
uCosminexus TP1/Client for .NET Framework	TP1/Client for .NET Framework	Client .NET
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#	
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 Visual Studio Community 2013	Visual Studio 2013	Visual Studio
Microsoft Visual Studio Professional 2013		
Microsoft Visual Studio Premium 2013		
Microsoft Visual Studio Ultimate 2013		
Microsoft Visual Studio Express 2013 for Windows Desktop	Visual Studio Express 2013 for Windows Desktop	
Microsoft Visual Studio Community 2015	Visual Studio 2015	
Microsoft Visual Studio Professional 2015		

製品名称		このマニュアルでの表記	
Microsoft Visual Studio Enterprise 2015		Visual Studio 2015	Visual Studio
Microsoft Visual Studio Express 2015 for Windows Desktop		Visual Studio Express 2015 for Windows Desktop	
Microsoft Visual Studio Community 2017		Visual Studio 2017	
Microsoft Visual Studio Professional 2017			
Microsoft Visual Studio Enterprise 2017			
Microsoft Visual Studio Community 2019		Visual Studio 2019	
Microsoft Visual Studio Professional 2019			
Microsoft Visual Studio Enterprise 2019			
Microsoft Windows Software Development Kit (v7.0) for Windows 7 and .NET Framework 3.5 SP1		Windows SDK 7.0	
Microsoft Windows Software Development Kit (v7.1) for Windows 7 and .NET Framework 4		Windows SDK 7.1	
Microsoft Windows 7 Enterprise	Windows 7	Windows 7	Windows
Microsoft Windows 7 Professional			
Microsoft Windows 7 Ultimate			
Microsoft Windows 7 Enterprise(x64)	Windows 7 x64 Edition		
Microsoft Windows 7 Professional(x64)			
Microsoft Windows 7 Ultimate(x64)			
Windows 8 Enterprise	Windows 8	Windows 8	
Windows 8 Pro			
Windows 8 Enterprise(x64)	Windows 8 x64 Edition		
Windows 8 Pro(x64)			
Windows 8.1 Enterprise	Windows 8.1	Windows 8.1	
Windows 8.1 Pro			

製品名称		このマニュアルでの表記	
Windows 8.1 Enterprise(x64)	Windows 8.1 x64 Edition	Windows 8.1	Windows
Windows 8.1 Pro (x64)			
Windows 10 Pro	Windows 10	Windows 10	
Windows 10 Enterprise			
Windows 10 Pro (x64)	Windows 10 x64 Edition		
Windows 10 Enterprise(x64)			
Microsoft Windows Server 2008 R2, Datacenter Edition	Windows Server 2008 R2		
Microsoft Windows Server 2008 R2, Enterprise Edition			
Microsoft Windows Server 2008 R2, Standard Edition			
Microsoft Windows Server 2012 Datacenter	Windows Server 2012		
Microsoft Windows Server 2012 Standard			
Microsoft Windows Server 2012 R2 Datacenter	Windows Server 2012 R2		
Microsoft Windows Server 2012 R2 Standard			
Microsoft Windows Server 2016 Datacenter	Windows Server 2016		
Microsoft Windows Server 2016 Standard			
Microsoft Windows Server 2019 Datacenter	Windows Server 2019		
Microsoft Windows Server 2019 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。

■ 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
ADO	ActiveX Data Object
ADO.NET	ActiveX Data Object for .NET Framework
AP	Application Program
API	Application Programming Interface
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
EBCDIK	Extended Binary Coded Decimal Interchange Kana Code
GAC	Global Assembly Cache
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IIS	Internet Information Services
MHP	Message Handling Program
MSDTC	MicroSoft Distributed Transaction Coordinator
OLTP	Online Transaction Processing
OS	Operating System
PC	Personal Computer
PRF	PeRFormance
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SPP	Service Providing Program

英略語	英字での表記
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
TSP	TP1 Service Proxy
UAP	User Application Program
WS	Workstation
WWW	World Wide Web
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

　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/Client for .NET Framework（Client .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 Client .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。

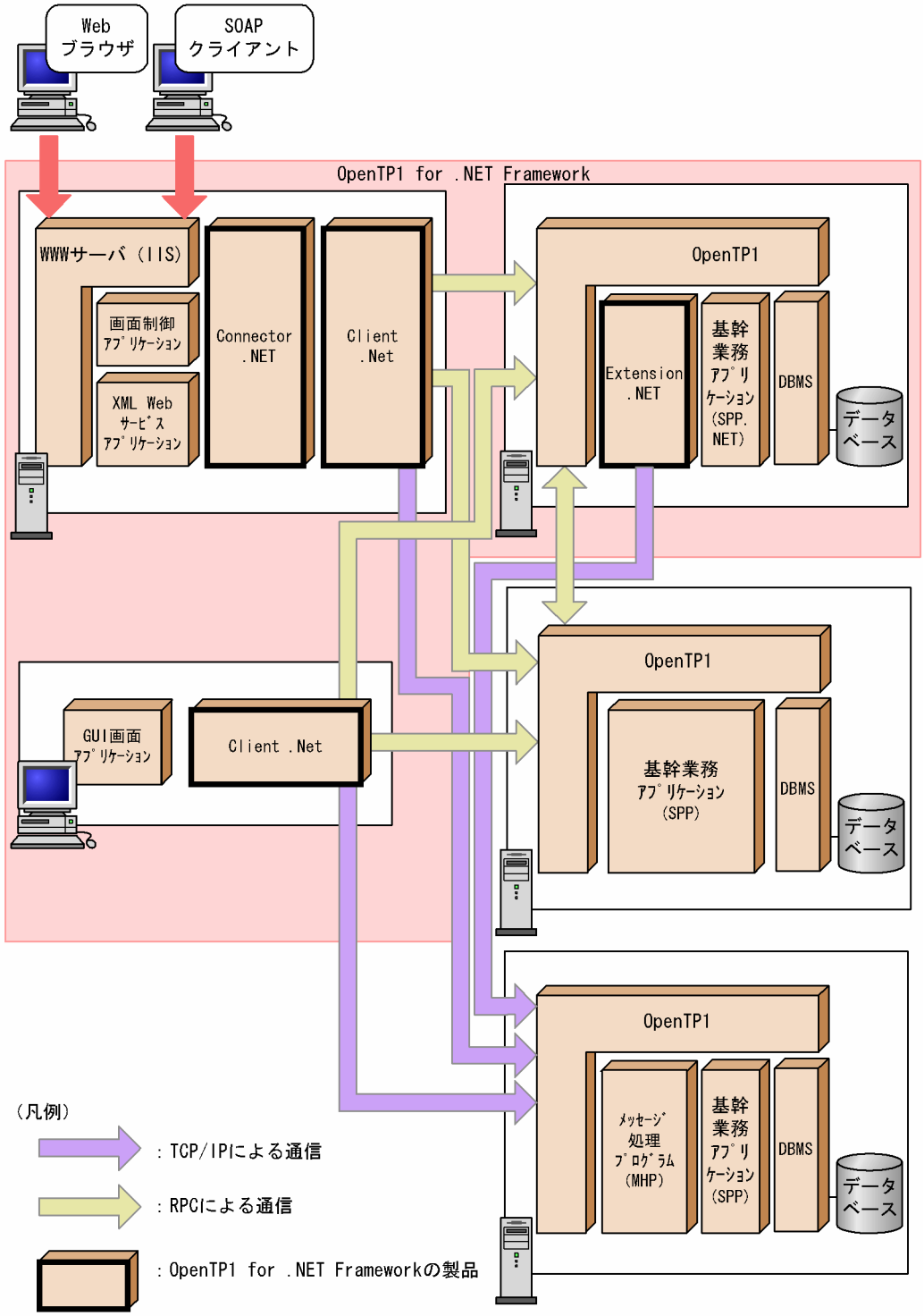
1.1 OpenTP1 for .NET Framework とは

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#，または Visual Basic から選択できます。

  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。

  また，Visual Studio と連携した，OpenTP1 for .NET Framework のアプリケーション作成ウィザードや，デバッグ支援機能が利用できます。

(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 前提条件

ここでは，システムを利用するための前提条件について説明します。

(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。

- Visual Studio 2013 で開発する場合
  Visual Studio 2013
- Visual Studio Express 2013 for Windows Desktop で開発する場合
  Visual Studio Express 2013 for Windows Desktop
- Visual Studio 2015 で開発する場合
  Visual Studio 2015
- Visual Studio Express 2015 for Windows Desktop で開発する場合
  Visual Studio Express 2015 for Windows Desktop
- Visual Studio 2017 で開発する場合
  Visual Studio 2017
- Visual Studio 2019 で開発する場合
  Visual Studio 2019
- .NET Framework 3.5 SP1 で開発する場合
  .NET Framework 3.5 SP1
- .NET Framework 4.5.2 で開発する場合
  .NET Framework 4.5.2
- .NET Framework 4.6 で開発する場合
  .NET Framework 4.6
- .NET Framework 4.6.1 で開発する場合
  .NET Framework 4.6.1
- .NET Framework 4.6.2 で開発する場合
  .NET Framework 4.6.2
- .NET Framework 4.7 で開発する場合

.NET Framework 4.7

- .NET Framework 4.7.1 で開発する場合
  .NET Framework 4.7.1
- .NET Framework 4.7.2 で開発する場合
  .NET Framework 4.7.2
- .NET Framework 4.8 で開発する場合
  .NET Framework 4.8

注

.NET Framework 4.6 以降を使用する場合の注意事項はリリースノートを参照してください。

(2) Client .NET を使用する場合

Client .NET 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。

- .NET Framework 3.5 SP1
- Visual Studio 2013 を使用する場合（同梱する Windows SDK を使用）
  Visual Studio 2013
- Visual Studio 2015 を使用する場合（同梱する Windows SDK を使用）
  Visual Studio 2015
- Visual Studio 2017 を使用する場合（同梱する Windows SDK は使用できません）
  Windows SDK 7.0
- Visual Studio 2019 を使用する場合（同梱する Windows SDK は使用できません）
  Windows SDK 7.0
- Windows SDK 7.0 を使用する場合
  Windows SDK 7.0
- Windows SDK 7.1 を使用する場合
  - Windows SDK 7.1
  - .NET Framework 4.5.2

(3) Connector .NET を使用する場合

Connector .NET 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。Visual Studio 2017 または Visual Studio 2019 を使用する場合は Windows SDK 7.0 を使用してください。

- Client .NET 07-50 以降
- .NET Framework 3.5 SP1

- Visual Studio 2013 を使用する場合（同梱する Windows SDK を使用）
  Visual Studio 2013
- Visual Studio 2015 を使用する場合（同梱する Windows SDK を使用）
  Visual Studio 2015
- Visual Studio 2017 を使用する場合（同梱する Windows SDK は使用できません）
  Windows SDK 7.0
- Visual Studio 2019 を使用する場合（同梱する Windows SDK は使用できません）
  Windows SDK 7.0
- Windows SDK 7.0 を使用する場合
  Windows SDK 7.0
- Windows SDK 7.1 を使用する場合
  - Windows SDK 7.1
  - .NET Framework 4.5.2

1.2.2 クライアント/サーバシステムの形態

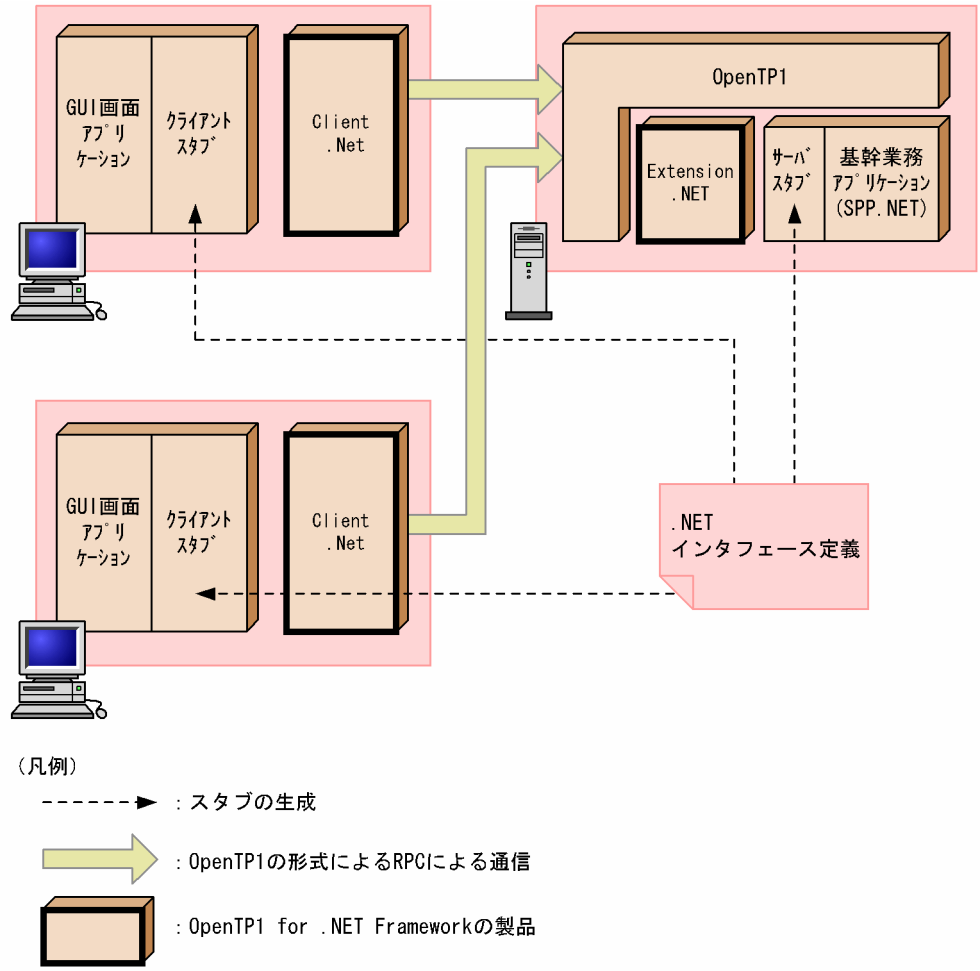
Client .NET を使用することで，.NET Framework 環境で動作する OpenTP1 のクライアントアプリケーション（GUI 画面アプリケーションなど）を作成できます。

また，Extension .NET 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

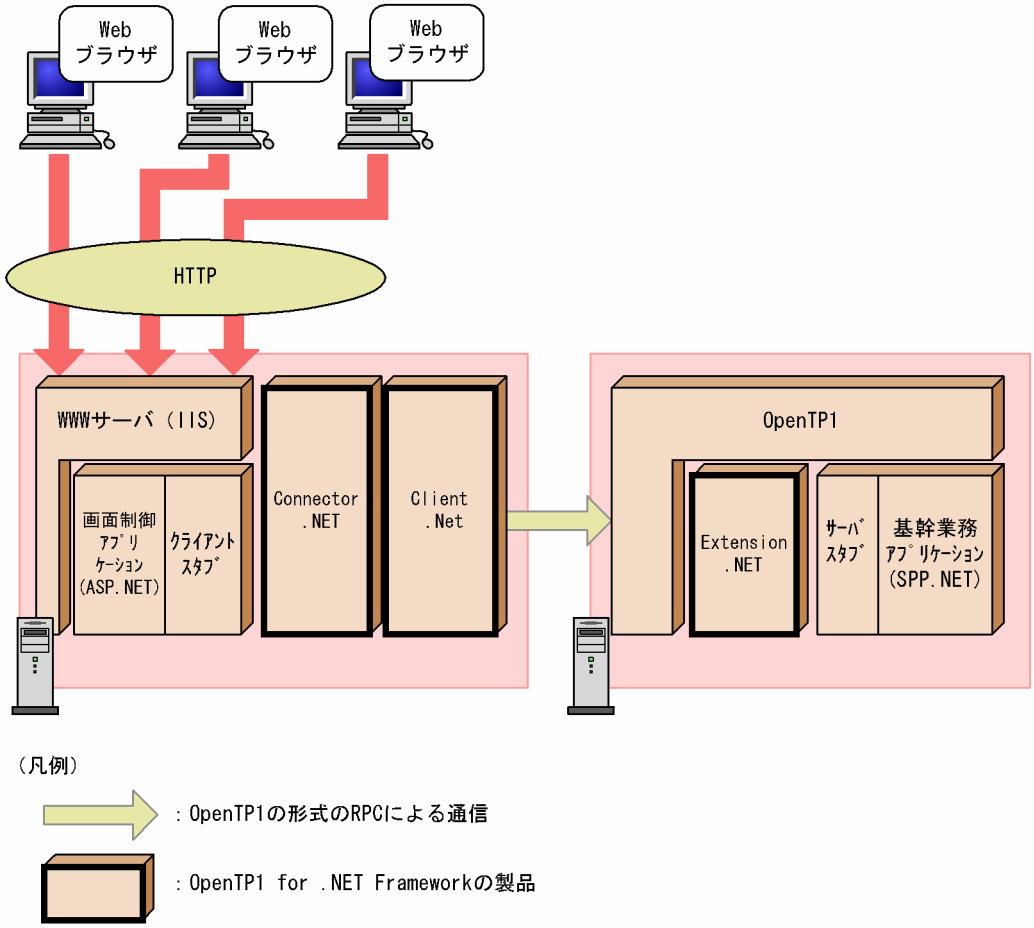
クライアントとなる Web ブラウザから，WWW サーバ（IIS）を経由して OpenTP1 サーバ上の基幹業務アプリケーションにアクセスできます。

WWW サーバ上で Client .NET と Connector .NET 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。

Connector .NET 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



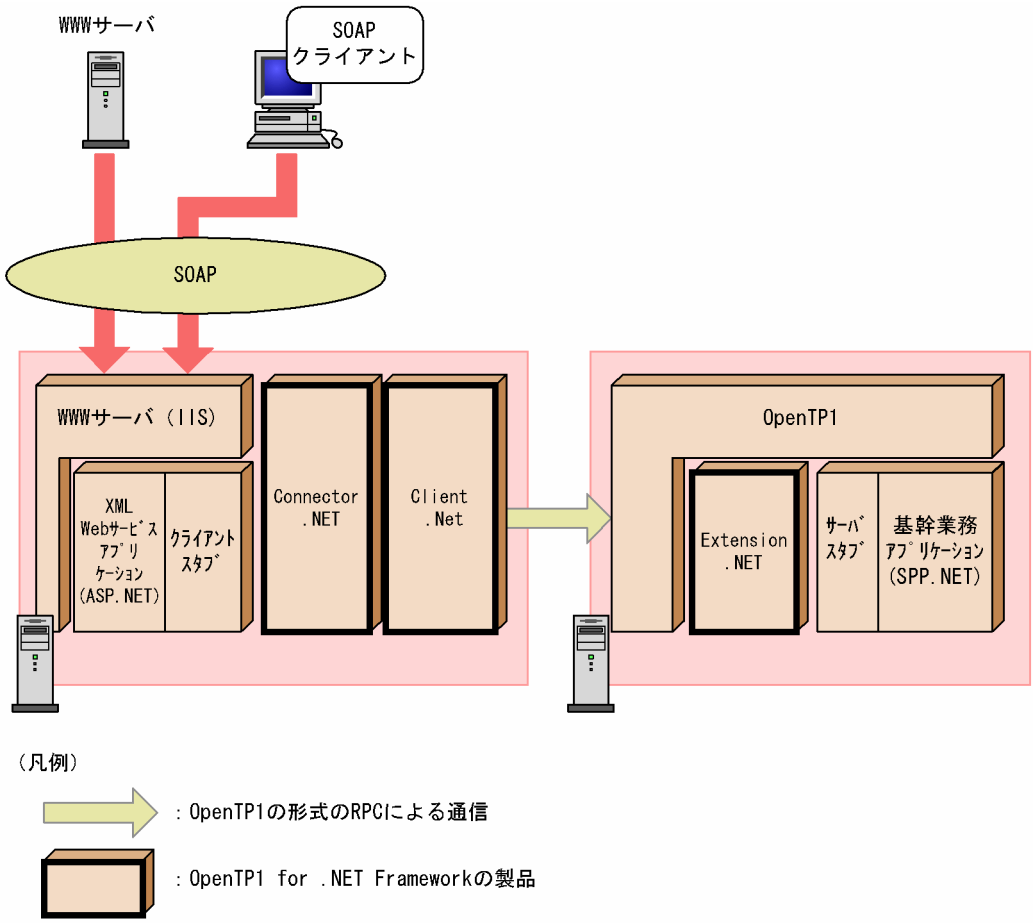
1.2.4　ASP.NET XML Web サービスとの接続

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 Client .NET と Connector .NET 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。

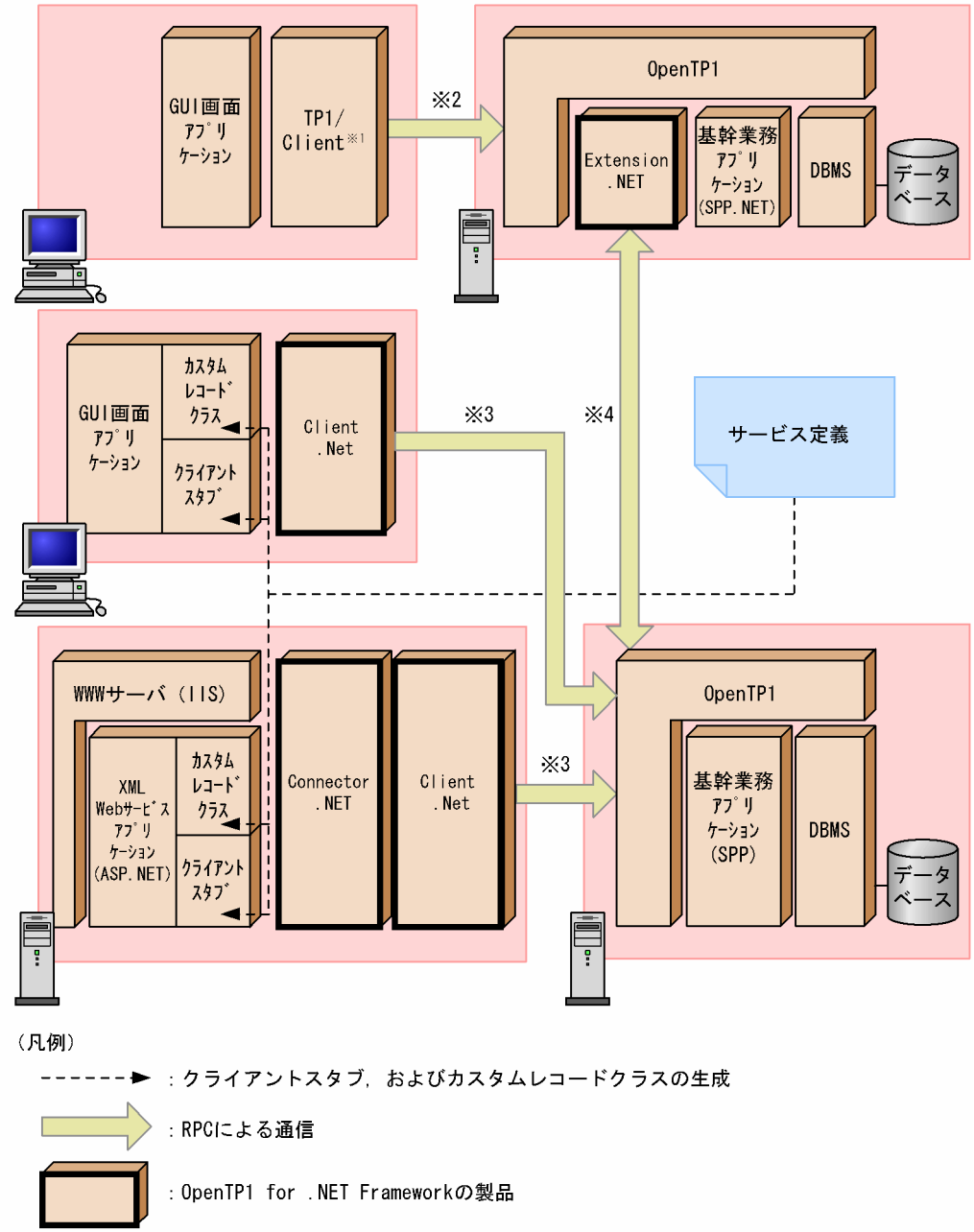
1.2.5　OpenTP1 システムとの接続

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，Extension .NET 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，Extension .NET 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，マニュアル「TP1/Extension for .NET Framework 使用の手引」の SPP.NET についての記述を参照してください。

(2) SUP.NET

SUP.NET は，Extension .NET 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，マニュアル「TP1/Extension for .NET Framework 使用の手引」の SUP.NET についての記述を参照してください。

(3) CUP.NET

CUP.NET は，Client .NET または Connector .NET 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

それぞれの RPC インタフェースについて説明します。

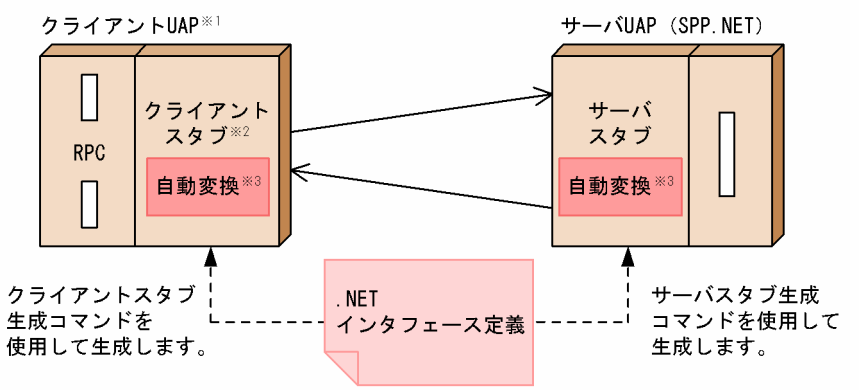
(1) .NET インタフェース定義を使用した RPC

.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（Connector .NET 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。

(2) 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。

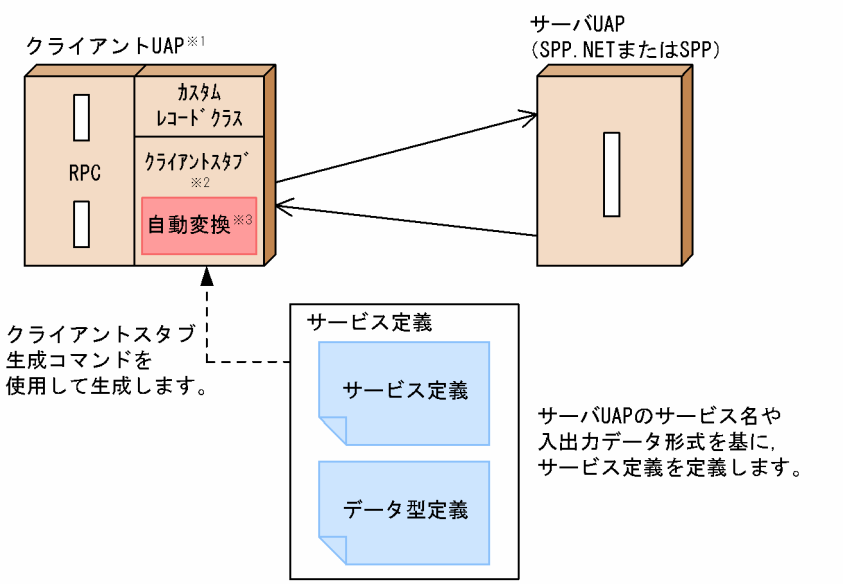
このサービス定義から運用コマンドでカスタムレコードとクライアントスタブを生成して使用します。

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（Connector .NET 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。

(3)　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。

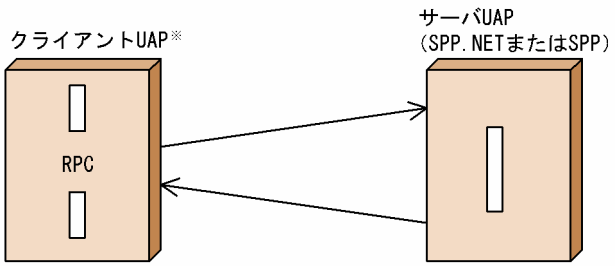
Connector .NET で，バイナリデータを使用した RPC を実行する場合は，インデクスドレコードを使用します。

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（Connector .NET 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，次の図に示します。

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）

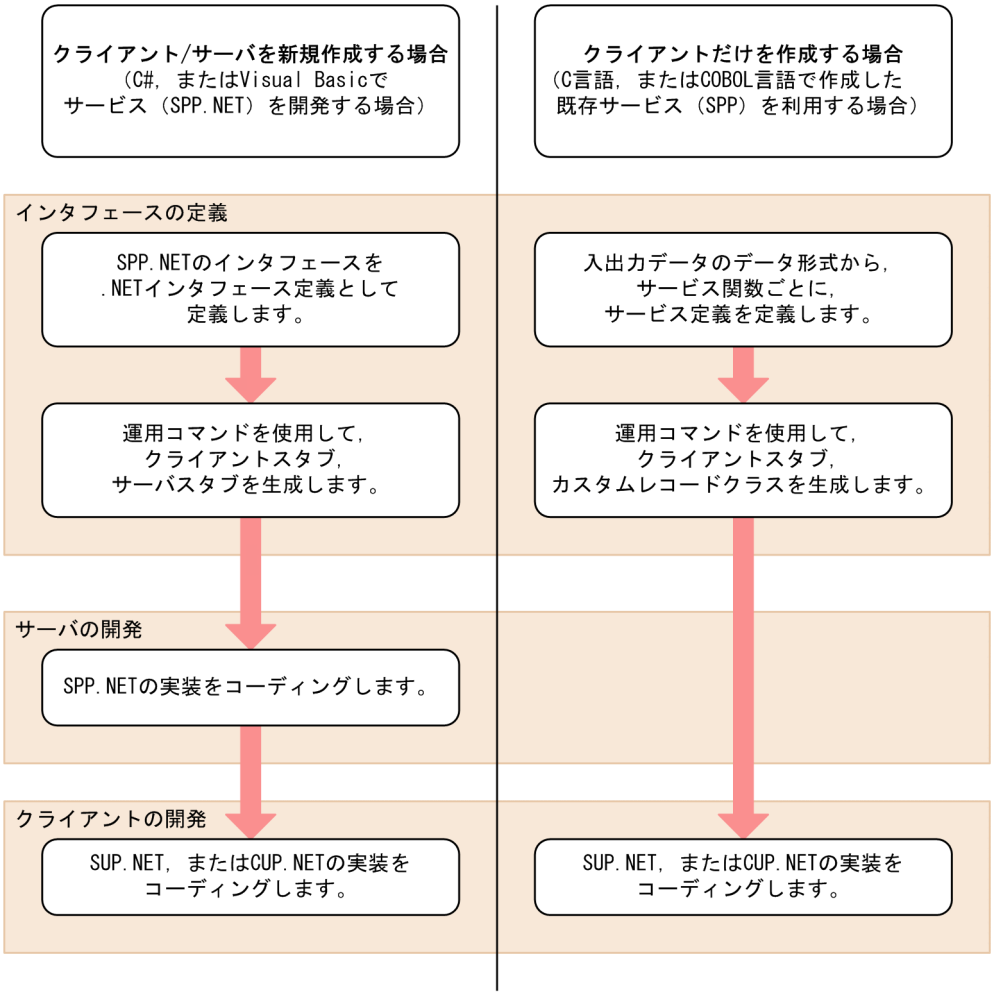
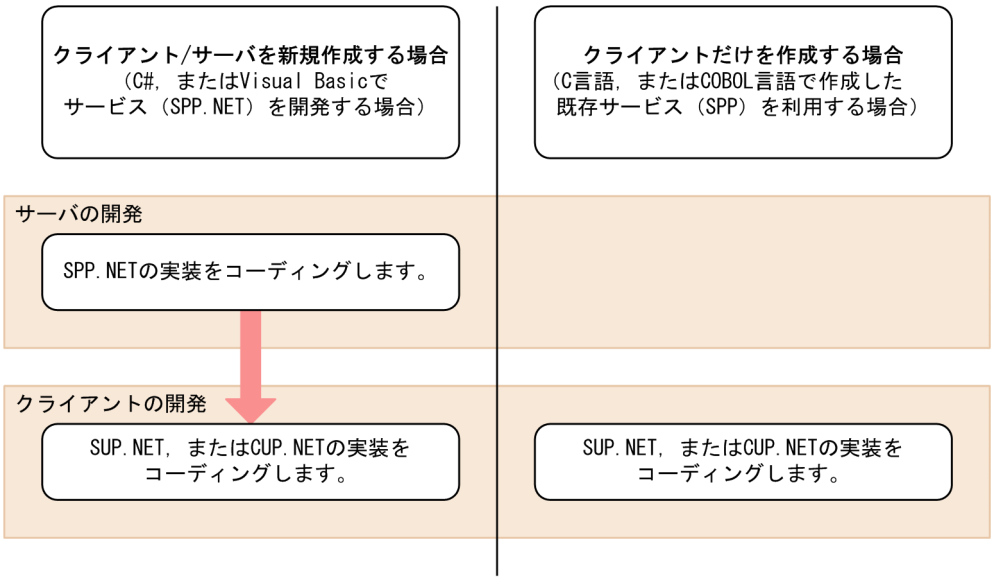


図 1-10　アプリケーション開発の流れ（RPC インタフェースにバイナリデータを使用する場合）



2 機能

この章では，Client .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　リモートプロシジャコール（RPC）

CUP.NET は，SPP.NET または SPP とリモートプロシジャコール（RPC）を使って通信できます。

CUP.NET からサービスを要求するメソッドを呼び出し，RPC によって SPP.NET または SPP にサービスを要求します。サービスを要求する方法には，クライアントスタブを利用する方法とクライアントスタブを利用しない方法があります。

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。

2.1.1　RPC の形態

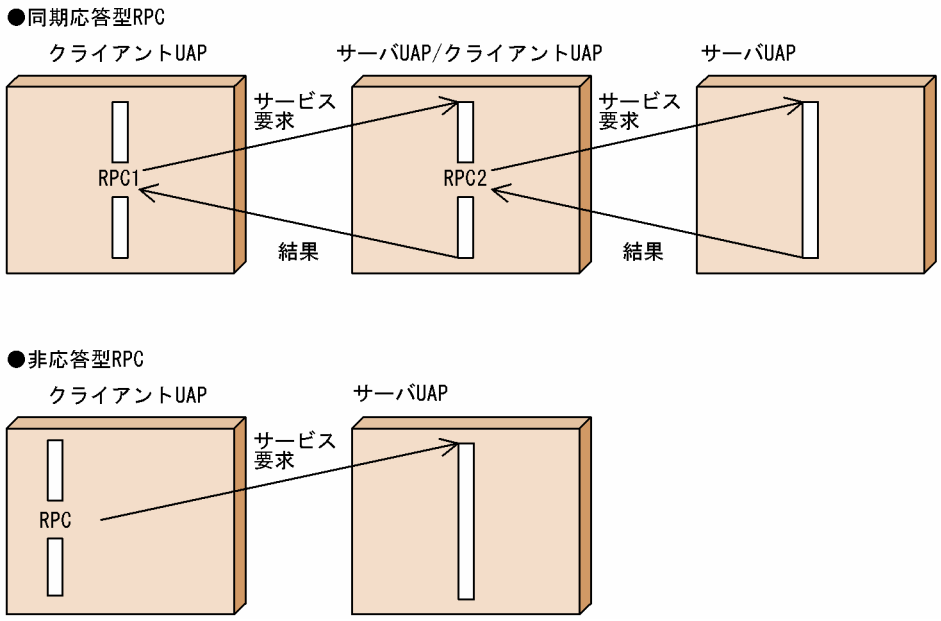
Client .NET の RPC には，次に示す 2 種類の形態があります。

- 同期応答型 RPC
- 非応答型 RPC

Extension .NET を使用すると，非同期応答型 RPC を利用できます。非同期応答型 RPC については，マニュアル「TP1/Extension for .NET Framework 使用の手引」を参照してください。

RPC の形態別の処理概要を次の図に示します。

図 2-1　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。

2.1.2　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。

2.1.3　RPC の形態による RPC インタフェースの使用可否

CUP.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「2.1.1　RPC の形態」および「2.1.2　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。

2.1.4　RPC の種類

Client .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 機能を使って，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ことができます。

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。

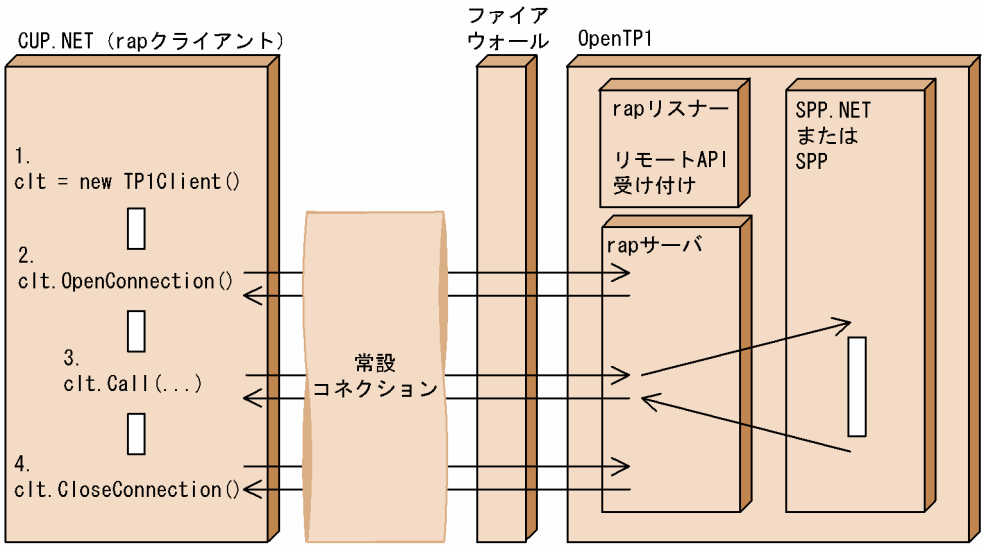
リモート API 機能を使用した RPC では，OpenTP1 上の rap サーバを経由してサービスを要求します。リモート API 機能を使用すると，Client .NET から確立したコネクションを常に使用してサービスの要求や応答の受信ができます。したがって，ファイアウォールの内側にある UAP に対しても，サービスを要求できます。なお，リモート API 機能を使用した RPC は，TP1/Server のほかに，DCCM3 および TMS-4V/SP を使用できます。

リモート API 機能を使用した RPC を行う場合は，必ず常設コネクションを確立してください。常設コネクションについては，「2.2　常設コネクション」を参照してください。

リモート API 機能を使用した RPC を行う場合で，OpenRpc メソッドを呼び出すときは，Client .NET 構成定義の<rpc>要素の use 属性に rap を指定し，かつ Client .NET 構成定義の<rapService>要素を指定します。

リモート API 機能を使用したサービス要求の流れを次に示します。

図 2-2　リモート API 機能を使用したサービス要求の流れ



1. TP1Client クラスのインスタンスを作成します。
2. OpenConnection メソッドを呼び出し，OpenTP1 上の rap リスナーを経由して，rap サーバとの間に常設コネクションを確立します。
3. Call メソッドを呼び出して，rap サーバ経由で該当する SPP.NET または SPP にサービスを要求します。
4. CloseConnection メソッドを呼び出して，OpenTP1 上の rap サーバとの間の常設コネクションを解放します。

(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 を発行できます。

ネームサービスを使用した RPC では，サービスを要求する前に OpenRpc メソッドを呼び出す必要があります。また，CUP.NET の実行の最後に CloseRpc メソッドを呼び出す必要があります。

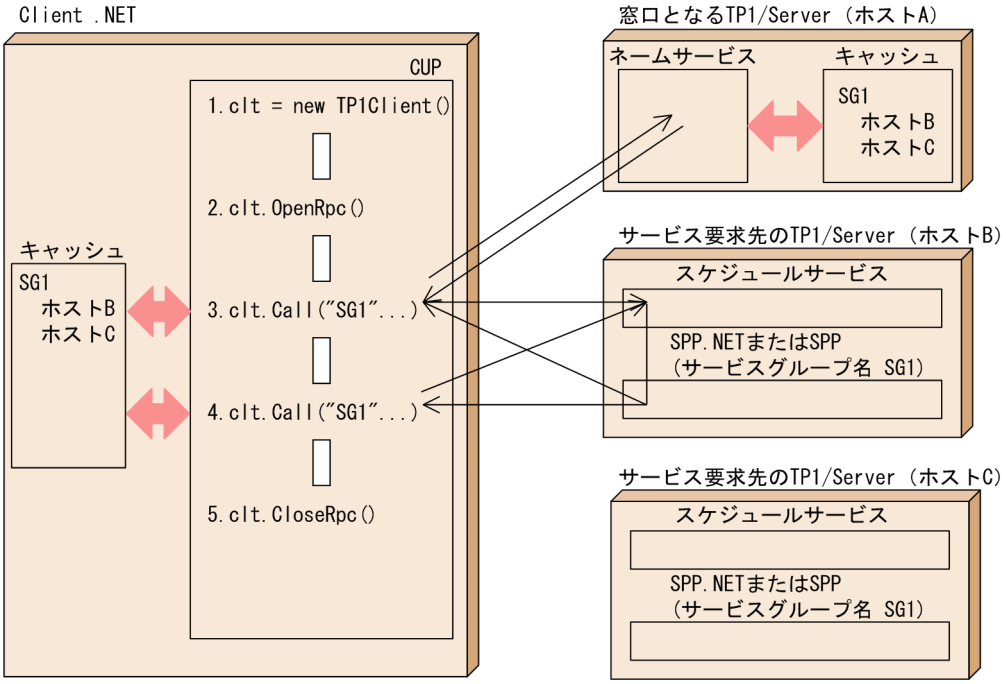
ネームサービスを使用した RPC の流れと，RPC ごとにサービス要求先スケジューラを分散させる場合の定義について説明します。

(a)　ネームサービスを使用した RPC の流れ

ネームサービスを使用した RPC を行う場合は，Client .NET 構成定義の<rpc>要素の use 属性に nam を指定し，かつ Client .NET 構成定義の<nameService>要素を指定します。

ネームサービスを使用したサービス要求の流れを次に示します。

図 2-3　ネームサービスを使用したサービス要求の流れ



1. TP1Client クラスのインスタンスを作成します。
2. OpenRpc メソッドを呼び出して，CUP.NET の RPC 環境を初期化します。
3. Call メソッドを呼び出して，該当する SPP.NET または SPP にサービスを要求します。※
4. Call メソッドを呼び出して，Client .NET のキャッシュを検索し，該当する SPP.NET または SPP にサービスを要求します。
5. CloseRpc メソッドを呼び出して，RPC 環境を解放します。

注※

Call メソッド内部の処理の流れを次に示します。

1. 指定された nam サーバに対してコネクションを確立します。
2. nam サーバにサービス情報取得要求を送信し，nam サーバとの間で確立したコネクションを解放します。
3. nam サーバから応答メッセージ送信用のコネクション確立要求を受信後，コネクションを確立してサービス情報を受信します。
4. サービス情報の受信後，nam サーバとのコネクションを解放します。

5. nam サーバからの応答メッセージ中のサービス情報を Client .NET のキャッシュに格納し，その情報を基に，サービスを実行している OpenTP1 上の scd サーバに対してコネクションを確立します。
6. scd サーバとのコネクション確立後，サービス要求メッセージを送信してコネクションを切断します。
7. SPP.NET または SPP から応答メッセージ送信用のコネクション確立要求を受信後，コネクションを確立して応答メッセージを受信します。
8. SPP.NET または SPP との間で確立したコネクションを解放します。

(b) RPC ごとにサービス要求先スケジューラを分散させる場合の定義

ネームサービスを使用した RPC では，サービス要求先スケジューラの情報をネームサーバからキャッシュに格納し，キャッシュの情報を参照して RPC ごとにサービス要求先スケジューラを分散させることができます。RPC ごとにサービス要求先スケジューラを分散させる場合の Client .NET 側，TP1/Server 側の関連する定義について説明します。

■ Client .NET 側の定義

RPC ごとにサービス要求先スケジューラを分散させる場合は，<nameService>要素の loadBalance 属性に true を指定します。

これによって，Client .NET は，ネームサーバから複数のサービス要求先スケジューラの情報を取得し，負荷レベルが最も低いサービス要求先スケジューラの情報をキャッシュに格納します。キャッシュに格納されるサービス要求先スケジューラは，一つだけではなく，複数の場合もあります。

キャッシュに格納されたサービス要求先スケジューラが複数ある場合，最初のサービス要求先スケジューラは，ランダムに選択されます。RPC でのサービス要求が 2 度目以降の場合は，サービス要求先スケジューラの情報を取得するためにキャッシュを参照し，RPC ごとにラウンドロビン方式でサービス要求先スケジューラを切り替え，分散させます。

キャッシュの有効期限の指定

<nameService>要素の cacheTime 属性で，サービス要求先スケジューラの情報を保持する，キャッシュの有効期限を指定できます。

キャッシュにサービス要求先スケジューラの情報を格納したあとで，キャッシュの有効期限を満了した場合，そのサービス要求先スケジューラの情報を破棄します。その後，サービス要求先スケジューラの情報を再度取得して，その情報をキャッシュに格納することで，キャッシュを更新します。

<nameService>要素の cacheTime 属性は，<nameService>要素の loadBalance 属性に true を指定した場合だけ有効になります。

■ TP1/Server 側の定義

次に示す TP1/Server 側の定義のオペランドを指定すると，ネームサーバからサービス要求先スケジューラの情報を取得するときに，スケジューラの負荷情報も同時に取得します。

スケジュールサービス定義

- scd_announce_server_status=Y（デフォルト）※1

ユーザサービス定義，またはユーザサービスデフォルト定義

- loadcheck_interval
- levelup_queue_count※2
- leveldown_queue_count※2

注※1

このオペランドに N を指定した場合，スケジューラの負荷レベルは常に LEVEL0（負荷が低い状態）になります。負荷レベルが常に LEVEL0 のスケジューラが，Client .NET のサービス要求先スケジューラになった場合，実際の負荷状態に関係なく，常にサービス要求先スケジューラとして選択されます。

注※2

負荷レベルがサービス要求の滞留数で決まる方式とする場合に指定します。

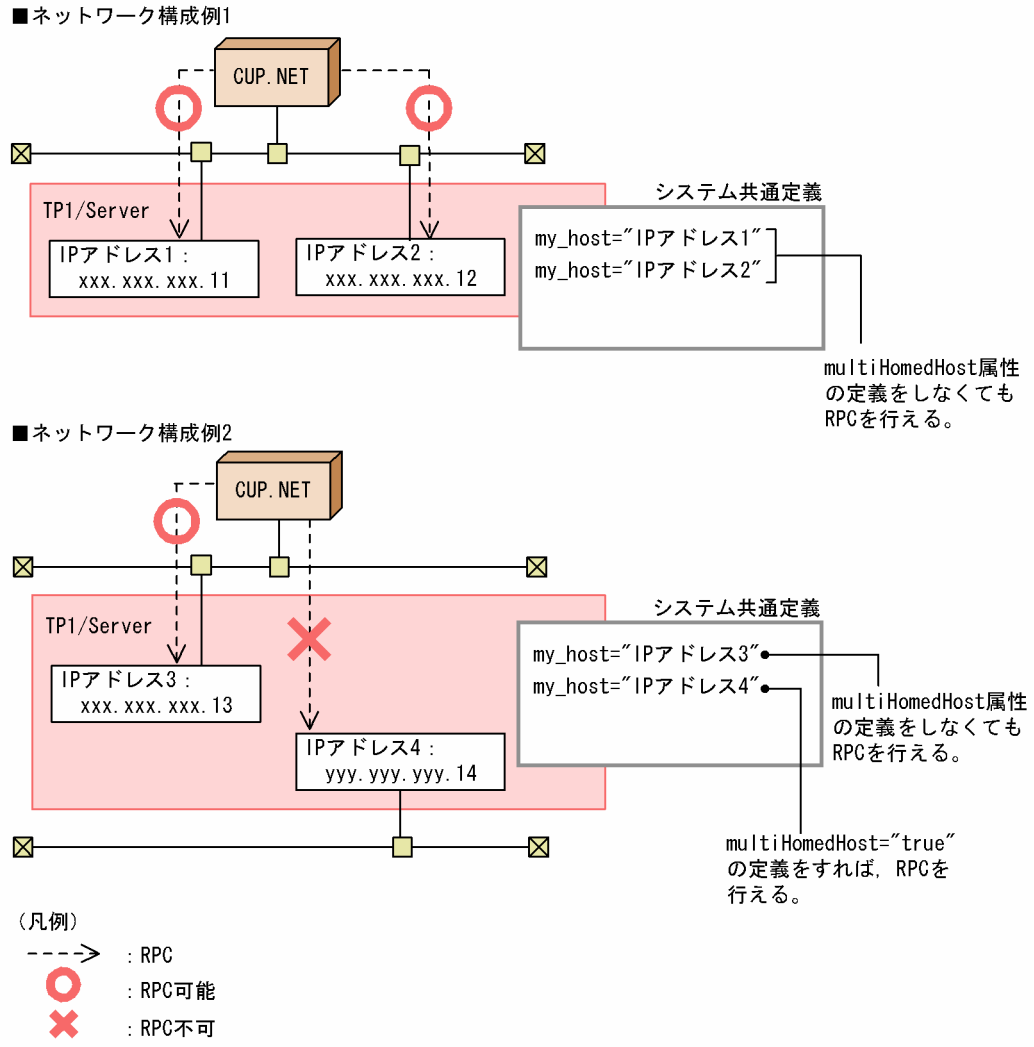
上記の定義の詳細については，マニュアル「OpenTP1 システム定義」を参照してください。

(c) マルチホームドホスト形態の TP1/Server に対して RPC を行う場合の定義

ネームサービスを使用した RPC では，通信先 TP1/Server がマルチホームドホスト形態の場合に通信障害が発生することがあります。これは，CUP.NET が存在するネットワークから，通信先 TP1/Server のシステム共通定義の my_host オペランドで指定したホスト名（IP アドレス）に対応するネットワークアダプタに通信できないことが原因です。

これを回避するためには，Client .NET 側で，<nameService>要素の multiHomedHost 属性に true を指定します。ネットワークの構成例を次の図に二つ示し，それぞれのネットワーク構成での multiHomedHost 属性の定義の要否を示します。

図 2-4　マルチホームドホスト形態の TP1/Server に対して RPC を行う場合の例



ネットワーク構成例 2 の CUP.NET が IP アドレス 4 と通信するには，<nameService>要素の multiHomedHost 属性に true の指定が必要です。この指定がない場合，CUP.NET と IP アドレス 4 との間では通信障害が発生します。

通信先 TP1/Server がマルチホームドホスト形態で，かつ同じ TP1/Server 上にサービス要求先の SPP が存在しているときは，<nameService>要素の multiHomedHost 属性に true を指定することで RPC が行えるようになります。通信先 TP1/Server のシステム共通定義の all_node オペランドに，CUP.NET が接続されていないネットワーク上に存在する TP1/Server が指定されている場合，この TP1/Server 上の SPP に対して RPC を行うと通信障害が発生します。なお，通信先 TP1/Server がマルチホームドホスト形態でない場合，multiHomedHost 属性の指定に関係なく，RPC を行えます。

(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を使用すると，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ことができます。

スケジューラダイレクト機能を使用した RPC では，サービスを要求する前に OpenRpc メソッドを呼び出す必要があります。また，CUP.NET の実行の最後に CloseRpc メソッドを呼び出す必要があります。

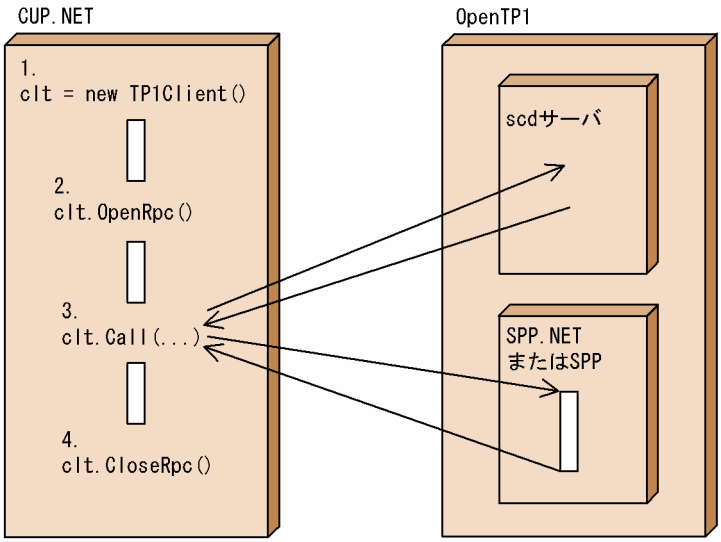
スケジューラダイレクト機能を使用した RPC の流れと，RPC ごとにサービス要求先スケジューラを分散させる場合の定義について説明します。

(a)　スケジューラダイレクト機能を使用した RPC の流れ

スケジューラダイレクト機能を使用した RPC を行う場合は，Client .NET 構成定義の<rpc>要素の use 属性に scd を指定し，かつ Client .NET 構成定義の<scheduleService>要素を指定します。

スケジューラダイレクト機能を使用したサービス要求の流れを次に示します。

図 2-5　スケジューラダイレクト機能を使用したサービス要求の流れ



1. TP1Client クラスのインスタンスを作成します。
2. OpenRpc メソッドを呼び出して，CUP.NET の RPC 環境を初期化します。
3. Call メソッドを呼び出して，該当する SPP.NET または SPP にサービスを要求します。※
4. CloseRpc メソッドを呼び出して，RPC 環境を解放します。

注※

Call メソッド内部の処理の流れを次に示します。

1. 指定された scd サーバに対してコネクションを確立します。
2. scd サーバとのコネクション確立後，サービス要求メッセージを送信してコネクションを切断します。
3. SPP.NET または SPP から応答メッセージ送信用のコネクション確立要求を受信後，コネクションを確立して応答メッセージを受信します。
4. SPP.NET または SPP との間で確立したコネクションを解放します。

(b) RPC ごとにサービス要求先スケジューラを分散させる場合の定義

スケジューラダイレクト機能を使用した RPC では，RPC ごとにラウンドロビン方式でサービス要求先スケジューラを切り替え，分散させることができます。RPC ごとにサービス要求先スケジューラを分散させる場合は，<scheduleService>要素の hostChange 属性に true を指定してください。

(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 を発行できます。

通信先を指定した RPC では，サービス要求先のノード（スケジュールサービス）を指定してサービスを要求できます。サービス要求先のノードの状態に関係なく，サービス要求処理は指定したノードで行われます。

通信先を指定した RPC を行う場合は，CallTo メソッドを使用します。

なお，この RPC は，リモート API 機能を使用した RPC，およびクライアントスタブを使用した RPC との併用はできません。

2.1.5 RPC の機能

(1) 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，TP1Client クラスの Call メソッドおよび CallTo メソッドで送受信できるユーザメッセージの最大長を，1～8 メガバイトで指定できます。

この機能を使用する場合，<rpc>要素の maxMessageSize 属性を指定してください。ただし，<rpc>要素の maxMessageSize 属性を指定した場合，次の機能は使用しないでください。通信先の TP1/Server ノードでエラーが発生します。

- スケジューラダイレクト機能を使用した RPC（<rpc>要素の use 属性に scd を指定）
- 通信先を指定した RPC（CallTo メソッドを使用）

システム共通定義の rpc_max_message_size オペランドをサポートしている TP1/Server（バージョン 06-02 以降）以外の TP1/Server にサービス要求した場合の Call メソッドの動作については，次の表を参照してください。

表 2-2 <rpc>要素の maxMessageSize 属性に 2 以上を指定した場合の Call メソッドの動作

メソッド	RPC の種類（<rpc>要素の use 属性）	窓口となる TP1/Server のバージョン	ノードのバージョン※1	
			06-02 より前の場合	06-02 以降の場合※2
Call	rap	06-02 より前	×※3	×※3
		06-02 以降	×※4	○

メソッド	RPC の種類（<rpc>要素の use 属性）	窓口となる TP1/Server のバージョン	ノードのバージョン※1	
			06-02 より前の場合	06-02 以降の場合※2
Call	nam	06-02 より前	×※4	○
		06-02 以降	×※4	○

（凡例）

○：サービスを要求できます。

×：サービスを要求できません。

注※1

窓口となる TP1/Server に指定されている all_node オペランドで定義された，SPP を起動しているノードのバージョンです。

注※2

システム共通定義に rpc_max_message_size オペランドを指定している場合。

注※3

入力データ長が 1 メガバイトより大きい場合，ErrMessageTooBigException が発生します。

入力データ長が 1 メガバイト以下，かつ応答データ格納領域長が 1 メガバイトより大きい場合，ErrInvalidArgsException が発生します。

注※4

ErrNoSuchServiceGroupException が発生します。

2.1.6 マルチスケジューラ機能を使用した RPC

CUP.NET からスケジュールキューを使う SPP（キュー受信型サーバ）にサービスを要求した場合，要求先 SPP があるノードのスケジューラデーモンが，いったんサービス要求メッセージを受信し，該当する SPP のスケジュールキューに格納します。スケジューラデーモンとは，スケジュールサービスを提供するシステムデーモンのことです。

長大なサービス要求メッセージは，一定の長さに分割してスケジューラデーモンに送信します。スケジューラデーモンは，サービス要求メッセージを組み立ててキュー受信型サーバのスケジュールキューに格納します。スケジューラデーモンは，OpenTP1 システムごとに 1 プロセスです。そのため，分割されたサービス要求メッセージの受信処理が完了するまで，スケジューラデーモンはほかのサービス要求メッセージを受信できません。通信速度が遅い回線を使用して，長大なサービス要求メッセージを送信した場合，ほかのサービス要求のスケジューリングが遅延することがあります。また，システムの大規模化，マシンやネットワークの高性能化などに伴って，効率良くスケジューリングできないことがあります。この場合，従来のスケジューラデーモンとは別に，サービス要求受信専用デーモンを複数プロセス起動し，サービス要求メッセージ受信処理を並行動作させることによって，スケジューリング遅延を回避できます。この機能を**マルチスケジューラ機能**といいます。以降，従来のスケジューラデーモンを**マスタスケジューラデーモン**，サービス要求受信専用デーモンを**マルチスケジューラデーモン**と呼びます。

マルチスケジューラ機能の検討が必要なシステム構成については，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。

(1) マルチスケジューラデーモンをランダムに選択する方法

マルチスケジューラ機能を使用することで，複数起動されているマルチスケジューラデーモンの中から，利用できるマルチスケジューラデーモンをランダムに選択してサービス要求を送信できます。マルチスケジューラデーモンをランダムに選択して，スケジューラダイレクト機能，またはネームサービスを使用した RPC を実行できます。

(a) スケジューラダイレクト機能を使用した RPC

Client .NET 構成定義の<rpc>要素の use 属性に scd を指定して，スケジューラダイレクト機能を使用した RPC を行う場合について説明します。

Client .NET では，窓口となる TP1/Server のネームサービスへ問い合わせることなく，マルチスケジューラデーモンをランダムに選択できるため，通信回数が削減されます。これによって，ネームサービスの負荷の軽減もできます。

サービス要求を送信するマルチスケジューラデーモンのポート番号は，次に示す範囲の値からランダムに選択されます。

- 下限値：Client .NET 構成定義の<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性に指定したポート番号の値
- 上限値：下限値 + Client .NET 構成定義の<scheduleService>要素の multiSchedulerCount 属性に指定したプロセス数 - 1

Client .NET 構成定義の<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性には，マルチスケジューラデーモンのポート番号を指定します。マスタスケジューラデーモンとマルチスケジューラデーモンのベースとなるポート番号が連続している場合は，マスタスケジューラデーモンのポート番号を指定することもできます。Client .NET 構成定義の<scheduleService>要素の multiSchedulerCount 属性には，TP1/Server で起動しているスケジューラデーモンのプロセス数を指定します。Client .NET 構成定義の<scheduleService>要素の multiSchedulerCount 属性に指定するプロセス数は，<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性に指定するポート番号によって，次のように異なります。

表 2-3 Client .NET 構成定義の<scheduleService>要素の multiSchedulerCount 属性に指定するプロセス数の違い

<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性の指定値	<scheduleService>要素の multiSchedulerCount 属性の指定値
マルチスケジューラデーモンのベースとなるポート番号	マルチスケジューラデーモンのプロセス数と同じ値，またはそれ以下の値を指定します。
マルチスケジューラデーモンの任意のポート番号	「マルチスケジューラデーモンのポート番号の最大 - <tp1Server>要素の port 属性，または<scheduleService>要素の port 属性に指定したポート番号 + 1」の値，またはそれ以下の値を指定します。

<tp1Server>要素の port 属性，または <scheduleService>要素の port 属性の指定値	<scheduleService>要素の multiSchedulerCount 属性の指定値	
マスタスケジューラデーモンのポート番号	マスタスケジューラデーモンとマルチスケジューラデーモンのベースとなるポート番号が連続している場合	「マルチスケジューラデーモンのプロセス数 ＋ 1」の値，またはそれ以下の値を指定します。
	上記以外	1 を指定するか，または指定を省略します（マルチスケジューラデーモンは使用できません）。

なお，Client .NET 構成定義の<tp1Server>要素で指定した窓口となる TP1/Server 間で，スケジュールサービス定義の scdmulti で指定した値を統一する必要があります。

(b) ネームサービスを使用した RPC

マルチスケジューラ機能を使用して，ネームサービスを使用した RPC を行う場合について説明します。サービス情報を一時的に格納する領域に，該当するサービス情報がないときはネームサービスにサービス情報を問い合わせます。サービス情報を基にマルチスケジューラデーモンをランダムに選択してサービス要求を送信します。

(2) Client .NET 構成定義とサービス要求を送信するスケジューラデーモンの関連

マルチスケジューラ機能を使用した場合，サービス要求を送信するスケジューラデーモンは Client .NET 構成定義の指定によって異なります。

スケジューラダイレクト機能を使用した RPC の場合の，Client .NET 構成定義の指定とスケジューラデーモンの関連を次の表に示します。

表 2-4 Client .NET 構成定義の指定とスケジューラデーモンの関連（スケジューラダイレクト機能を使用した RPC）

Client .NET 構成定義の指定			サービス要求を送信するスケジューラデーモン
<rpc>要素の use 属性	<rpc>要素の useMultiScheduler 属性	<scheduleService>要素の multiSchedulerCount 属性	
scd	true	2 以上の値	ランダムに選択したスケジューラデーモン※
		1，または指定を省略する	Client .NET 構成定義の<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性に指定したポート番号で起動されているスケジューラデーモン
	false	無効	Client .NET 構成定義の<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性に指定したポート番号で起動されているスケジューラデーモン

注※

スケジューラデーモンのポート番号は，次に示す範囲の値から選択されます。

下限値：Client .NET 構成定義の<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性に指定したポート番号の値

上限値：下限値 + Client .NET 構成定義の<scheduleService>要素の multiSchedulerCount 属性に指定したプロセス数 - 1

ネームサービスを使用した RPC の場合の，Client .NET 構成定義の指定とスケジューラデーモンの関連を次の表に示します。

表 2-5　Client .NET 構成定義の指定とスケジューラデーモンの関連（ネームサービスを使用した RPC）

Client .NET 構成定義の指定			サービス要求を送信するスケジューラデーモン
<rpc>要素の use 属性	<rpc>要素の useMultiScheduler 属性	<scheduleService>要素の multiSchedulerCount 属性	
nam	true	無効	サービス情報を基にランダムに選択したスケジューラデーモン※
	false		マスタスケジューラデーモン※

注※

ネームサービスへの問い合わせが発生します。

2.2 常設コネクション

Client .NET では，CUP.NET と TP1/Server の rap リスナー，rap サーバと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。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。

リモート API 機能を使用した RPC を実行する場合は，必ず常設コネクションを確立してください。

常設コネクションを確立すると，コネクション確立・解放のための制御用パケットを減らすことができ，通信の効率が向上します。

また，常設コネクションを使用して DCCM3 論理端末と通信する場合，端末識別情報を DCCM3 論理端末に通知し，CUP.NET に割り当てられる DCCM3 の論理端末を固定することができます。

2.2.1 常設コネクションの確立・解放

常設コネクションを確立するためには，OpenConnection メソッドを呼び出します。接続要求先の rap サーバの位置は，OpenConnection メソッドの引数で指定するか，構成ファイルの構成定義で指定します。ただし，CUP.NET と rap リスナー，rap サーバとの間に仮想アドレスを持つネットワーク機器がある場合は，そのアドレスを指定してください。

常設コネクションは，TP1Client クラスの 1 インスタンスに対して 1 コネクションだけ確立できます。

常設コネクションの管理方法には，非オートコネクトモードとオートコネクトモードの二つのモードがあります。どちらのモードを使用するかは，Client .NET 構成定義の<rapService>要素の autoConnect 属性で指定するか，または SetRpcExtend メソッドで指定してください。

(1) 非オートコネクトモード

非オートコネクトモードは，CUP.NET で明示的に OpenConnection メソッドを呼び出して TP1/Server の rap リスナー，rap サーバとの常設コネクションを確立します。OpenConnection メソッドを呼び出さないでサービスを要求した場合は，例外が発生します。常設コネクションを解放するには，CloseConnection メソッドを呼び出します。CloseConnection メソッドを呼び出して常設コネクションを解放したあとは，OpenConnection メソッドで再び常設コネクションを確立するまでサービスは要求できません。なお，CloseConnection メソッドが失敗した場合でも常設コネクションは解放されます。また，CloseConnection メソッドを呼び出さないまま CloseRpc メソッドを呼び出した場合には，自動的に常設コネクションは解放されます。

非オートコネクトモードを使用する場合のコーディング例を次に示します。

非オートコネクトモードのコーディング例 1

```
public class Sample1 {
  public static void Main(string args[]) {
            :
    TP1Client clt = new TP1Client();
```

```
            :
    clt.OpenConnection(...);
            :
    clt.Call(...);
            :
    clt.CloseConnection(...);
  }
}
```

非オートコネクトモードのコーディング例 2

```
public class Sample2 {
  public static void Main(string args[]) {
            :
    TP1Client clt = new TP1Client();
            :
    clt.OpenRpc();
            :
    clt.OpenConnection(...);
            :
    clt.Call(...);
            :
    clt.CloseConnection(...);
            :
    clt.CloseRpc();
  }
}
```

(2) オートコネクトモード

オートコネクトモードは，Client .NET で常設コネクションを管理します。

サービスを要求したときに常設コネクションが確立されていなかった場合は，自動的に rap リスナー，rap サーバとの常設コネクションを確立します。このため，OpenConnection メソッドを呼び出すことはできません。呼び出した場合は例外が発生します。常設コネクションを解放するには CloseRpc メソッドを呼び出します。ただし，Client .NET で管理している常設コネクションを強制的に解放したい場合は，CloseConnection メソッドを呼び出します。

オートコネクトモードを使用する場合のコーディング例を次に示します。

オートコネクトモードのコーディング例 1

```
public class Sample1 {
  public static void Main(string args[]) {
            :
    TP1Client clt = new TP1Client();
            :
    clt.OpenRpc(...);
            :
    clt.Call(...);
            :
    clt.CloseRpc();
  }
}
```

オートコネクトモードのコーディング例 2

```
public class Sample2 {
  public static void Main(string args[]) {
              :
    TP1Client clt = new TP1Client();
              :
    clt.OpenRpc(...);
              :
    clt.Call(...);
              :
    clt.CloseConnection();
              :
    clt.Call(...);
              :
    clt.CloseRpc();
  }
}
```

2.2.2 常設コネクションを使用する場合に関連する定義

常設コネクションを使用する場合，必要に応じて次の定義を設定してください。

- Client .NET 構成定義
  <rpc>要素の use 属性
  <rapService>要素の port 属性，autoConnect 属性，および inquireTime 属性
  <tp1Server>要素の host 属性
- rap リスナーサービス定義
  rap リスナーサービス定義の詳細については，マニュアル「OpenTP1 システム定義」を参照してください。

2.2.3 DCCM3 論理端末への端末識別情報の通知

常設コネクションを使用して DCCM3 論理端末と通信する場合，端末識別情報を DCCM3 論理端末に通知し，CUP.NET に割り当てられる DCCM3 の論理端末を固定することができます。

(1) DCCM3 論理端末でのメッセージ送受信方法

DCCM3 論理端末では，通信相手となる Client .NET を IP アドレスと DCCM3 論理端末のポート番号で区別する論理端末として定義し，論理端末ごとにメッセージの送受信を行っています。

したがって，複数の CUP.NET を同一マシンから起動した場合，どの CUP.NET からの要求でも，DCCM3 論理端末から見ると IP アドレスが同じになってしまいます。複数の CUP.NET から同じ DCCM3 論理端末のポートにサービスを要求すると，DCCM3 の定義では区別が付きません。そのため，該当する DCCM3 の論理端末が複数定義されていた場合，DCCM3 のどの論理端末に CUP.NET が割り当てられるのかが不

定になってしまいます。サービス要求を受け付ける DCCM3 の論理端末が異なると，DCCM3 側のサーバ処理の順番が保証されなくなるため，業務によっては問題となることがあります。

(2) 端末識別情報の通知

CUP.NET が DCCM3 論理端末と常設コネクションを確立するときに，端末識別情報を DCCM3 論理端末に通知することで，CUP.NET に割り当てられる DCCM3 の論理端末を固定することができます。これを**端末識別情報設定機能**といいます。この機能を使用することで，CUP.NET を常に同じ DCCM3 の論理端末に割り当てることができます。なお，DCCM3 側では，この機能を**端末固定割り当て機能**といいます。

端末識別情報設定機能を使用していない場合と使用している場合の CUP.NET と DCCM3 論理端末の関係を，次の図に示します。

図 2-6 CUP.NET と DCCM3 論理端末の関係（端末識別情報設定機能を使用していない場合）

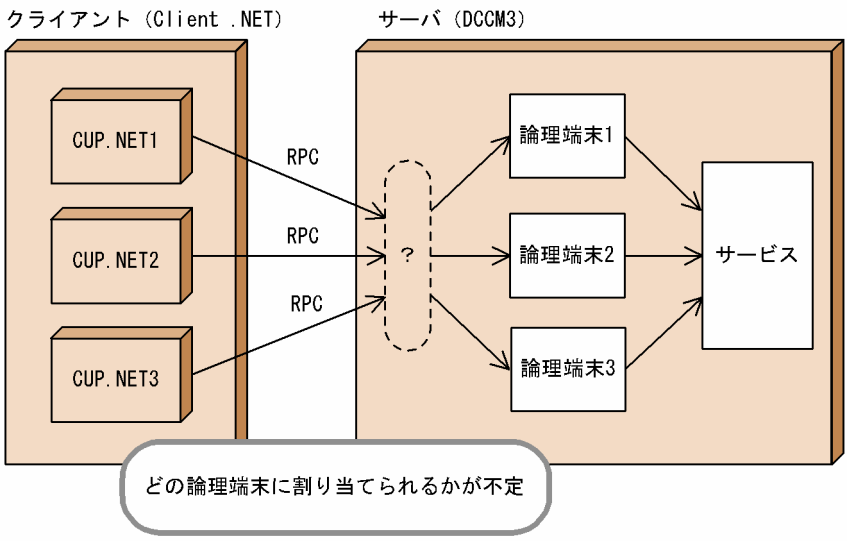
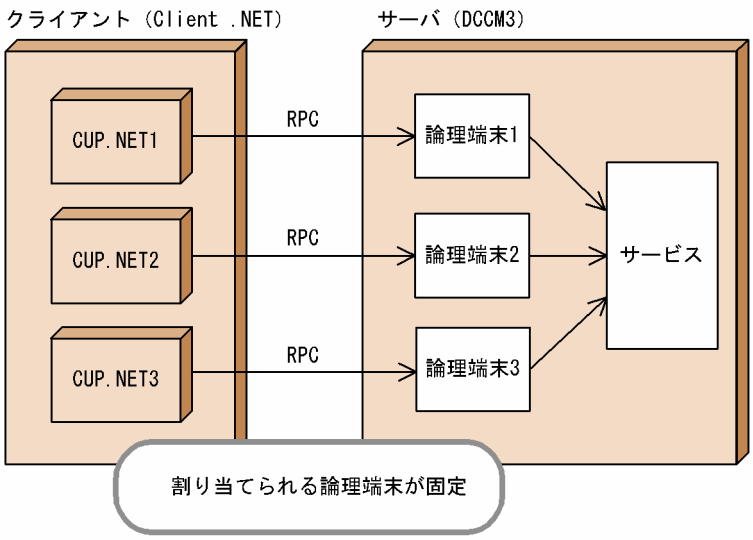


図 2-7 CUP.NET と DCCM3 論理端末の関係（端末識別情報設定機能を使用している場合）



端末識別情報設定機能は，次のどちらかの方法で使用できます。

方法 1

1. Client .NET 構成定義の<tp1Server>要素の host 属性に DCCM3 論理端末のホスト名を指定します。
2. Client .NET 構成定義の<tp1Server>要素の port 属性または<rapService>要素の port 属性に DCCM3 論理端末のポート番号を指定します。
3. SetConnectInformation メソッドに端末識別情報を設定し，メソッドを呼び出します。
4. 次のどちらかの方法で DCCM3 論理端末との常設コネクションを確立します。

   ・ OpenConnection メソッドを呼び出します。引数有りの OpenConnection メソッドの場合，引数 host に DCCM3 論理端末のホスト名，引数 port に DCCM3 論理端末のポート番号を指定します。

   ・ Client .NET 構成定義の<rapService>要素の autoConnect 属性に true を指定し，Call メソッドを呼び出します。

方法 2

1. Client .NET 構成定義の<tp1Server>要素の host 属性に DCCM3 論理端末のホスト名を指定します。
2. Client .NET 構成定義の<tp1Server>要素の port 属性または<rapService>要素の port 属性に DCCM3 論理端末のポート番号を指定します。
3. Client .NET 構成定義の<rapService>要素の connectInformation 属性に端末識別情報を設定します。
4. 次のどちらかの方法で DCCM3 論理端末との常設コネクションを確立します。

   ・ OpenConnection メソッドを呼び出します。引数有りの OpenConnection メソッドの場合，引数 host に DCCM3 論理端末のホスト名，引数 port に DCCM3 論理端末のポート番号を指定します。

   ・ Client .NET 構成定義の<rapService>要素の autoConnect 属性に true を指定し，Call メソッドを呼び出します。

注意事項

Client .NET 構成定義の<rapService>要素の connectInformation 属性に端末識別情報を設定して，SetConnectInformation メソッドに端末識別情報を設定した場合は，SetConnectInformation メソッドの設定が有効になります。<rapService>要素の connectInformation 属性に設定した値は，SetConnectInformation メソッドを呼び出したあと，再び OpenRpc メソッドを呼び出すまで無視されます。

(3) DCCM3 論理端末に端末識別情報を通知する場合の注意事項

- 端末固定割り当て機能を使用した DCCM3 の論理端末名称と，Client .NET で定義した端末識別情報とが一致していない場合に，次のメソッドを呼び出したときは ErrNetDownAtClientException 例外を返します。

- OpenConnection メソッド
- Call メソッド（ただし，Client .NET 構成定義の<rapService>要素の autoConnect 属性に true を指定した場合）

- 端末固定割り当て機能を使用した DCCM3 の論理端末に対して，Client .NET で端末識別情報を設定しないで次のメソッドを呼び出した場合，ErrNetDownAtClientException 例外を返します。
  - OpenConnection メソッド
  - Call メソッド（ただし，Client .NET 構成定義の<rapService>要素の autoConnect 属性に true を指定した場合）
- 端末固定割り当て機能を使用していない DCCM3 の論理端末に対して，Client .NET から端末識別情報を設定して常設コネクションの確立を要求した場合，DCCM3 は Client .NET が設定した端末識別情報の設定内容を無視します。
- Client .NET から端末識別情報を設定して，TP1/Server の rap サーバと常設コネクションを確立する場合，rap サーバは Client .NET が設定した端末識別情報の設定内容を無視します。また，Client .NET が rap サーバを介して DCCM3 へ RPC を発行する場合，Client .NET で設定した端末識別情報は DCCM3 に伝達されません。
- 端末識別情報は，リモート API 機能を使用した RPC の場合だけ有効となります。ネームサービスを使用した RPC，またはスケジューラダイレクト機能を使用した RPC の場合は，端末識別情報を設定しても無視されます。

2.2.4 常設コネクションを使用するときの注意事項

トランザクション内からは常設コネクションを確立できません。常設コネクションを確立したあと，トランザクションを生成してください。

2.3　トランザクション制御機能

Client .NET から OpenTP1 のトランザクションを制御できます。

トランザクション制御機能は，リモート API 機能を使用している場合にだけ使用できます。

2.3.1　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)　ローカルトランザクション

OpenTP1 が管理するトランザクションです。Client .NET を使用する CUP.NET 上でのリソースマネジャへのアクセスをトランザクションに含めることはできません。

CUP.NET のプロセスはルートトランザクションブランチになることはできません。

TP1Client クラスの Begin メソッドによってトランザクションを開始した場合，rap サーバのプロセスがルートトランザクションブランチになります。

(4) 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。

(5) 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。

2.3.2 トランザクション制御機能を使用する場合の設定

トランザクションを制御する場合，Client .NET 構成定義およびサーバ側の TP1/Server Base の各サービス定義を次のように指定しておく必要があります。

(1) Client .NET 構成定義

オートコネクトモードでリモート API 機能を使用するように指定します。

【指定例】

```
…
<tp1Server host=""/>
<rpc use="rap" watchTime="0"/>
<rapService port="10020" autoConnect="true"/>
…
```

(2)　TP1/Server Base のユーザサービス定義

サーバ側の TP1/Server Base のトランザクションとして実行する SPP は，ユーザサービス定義の atomic_update オペランドに Y を指定します。

2.3.3　トランザクションの開始と同期点取得

CUP.NET から TP1Client 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。

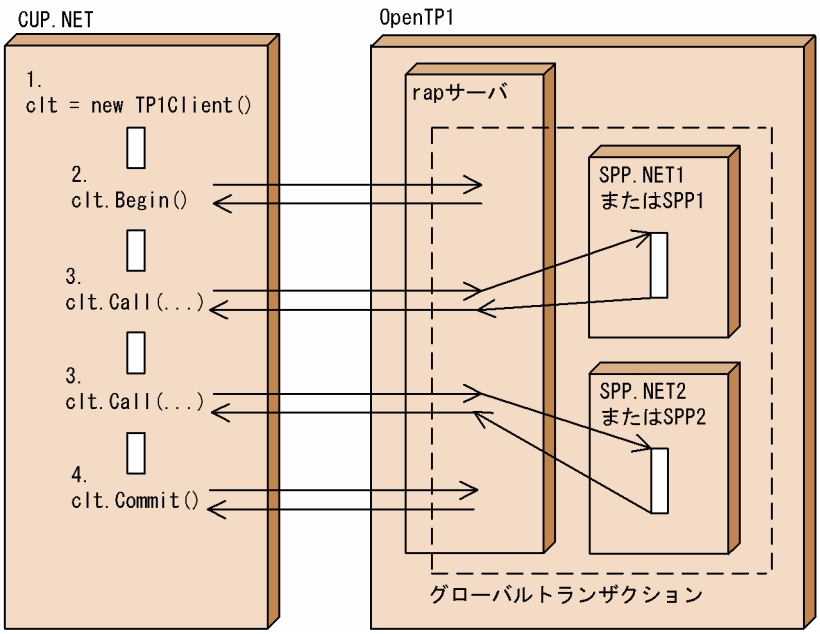
Begin メソッドを呼び出してから同期点取得（コミット）までが，グローバルトランザクションの範囲となります。Begin メソッドを呼び出したあと，そのグローバルトランザクションの中で新たな Begin メソッドは呼び出せません。

CUP.NET から SPP.NET または SPP に対して RPC 要求を実行すると，CUP.NET は rap サーバでルートトランザクションブランチを生成し，呼び出した SPP.NET または SPP はトランザクションブランチとして実行されます。

なお，Begin メソッドの呼び出しから，RPC 要求，同期点取得の間は，同一の TP1Client オブジェクトである必要があります。

トランザクションと RPC の関係を次の図に示します。

図 2-8　トランザクションと RPC の関係



1. TP1Client クラスのインスタンスを生成します。
2. TP1Client 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。
   rap サーバは，dc_trn_begin 関数を発行し，rap サーバのプロセスからグローバルトランザクションを開始します。

TP1Client のインスタンスは rap サーバのプロセスと 1 対 1 で結び付けられます。

3. TP1Client クラスの Call メソッドを呼び出して SPP.NET または SPP にサービスを要求します。コミットまたはロールバックが発生するまで，rap サーバと SPP.NET または SPP は関連づけられます。
4. TP1Client クラスの Commit メソッドを呼び出して非連鎖モードのコミットをします。
   この時点で TP1Client クラスのインスタンス，rap サーバのプロセス，および要求先の SPP.NET または SPP との関連づけはなくなります。

2.3.4 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，CUP.NET から TP1Client クラスの Commit メソッドを呼び出して要求します。

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，TP1Client 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，TP1Client 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。

- TP1Client クラスの Commit メソッドを呼び出さないで CUP.NET が終了したとき
- TP1Client クラスの Commit メソッドを呼び出す前に CUP.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，TP1Client 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見なして，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 CUP.NET の判断でロールバックしたいときは，CUP.NET からロールバック要求のメソッドを呼び出して行います。

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，TP1Client 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 CUP.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，TP1Client 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 CUP.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。

(3)　トランザクションの処理時間について

トランザクションに関する次に示す時間を Client.NET 構成定義で指定できます。詳細については，「3. 構成定義」の「transaction」を参照してください。

- トランザクション同期点処理時の最大通信待ち時間
- トランザクションブランチ最大実行可能時間

2.3.5　障害発生時のトランザクションの同期点を検証する方法

CUP.NET から開始したトランザクションで障害が発生した場合，そのトランザクションブランチがコミットしたかどうかを検証できます。

この場合，トランザクション開始後に，必ず TP1Client クラスの GetTransactionID メソッドを呼び出して，トランザクショングローバル識別子，およびトランザクションブランチ識別子を取得しておく必要があります。

Client .NET で取得しておいたトランザクショングローバル識別子と，サーバ側のメッセージログファイルに出力されるトランザクションの結果を突き合わせることによって，CUP.NET から開始したトランザクションがコミットしたかどうか検証できます。

2.3.6 TP1/Server 側での定義に対する注意事項

Client .NET と TP1/Server 間でトランザクション連携をする場合で，かつ set rpc_extend_function オペランドを指定する場合は，次の注意が必要です。

- ユーザサービスデフォルト定義の set rpc_extend_function オペランドに，00000002 ビットが ON になるような設定をしないでください。ユーザサービスデフォルト定義の set rpc_extend_function オペランドに，00000002 ビットが ON になるような設定をした場合，CUP.NET 側が管理しているトランザクション中のステータスと OpenTP1 側が保持しているトランザクション中のステータスが不一致になるときがあります。set rpc_extend_function オペランドの 00000002 ビットが ON になっている場合の動作は保証できません。
- ユーザサービスデフォルト定義の set rpc_extend_function オペランドの 00000002 ビットが ON になっている場合は，rap リスナーサービス定義の set rpc_extend_function オペランドで 00000002 ビットが OFF になるように定義し，rapdfgen コマンドで rap リスナー用ユーザサービス定義，および rap サーバ用ユーザサービス定義を再作成してください。

  また，再作成後 rap リスナー，および rap サーバを再起動してください。

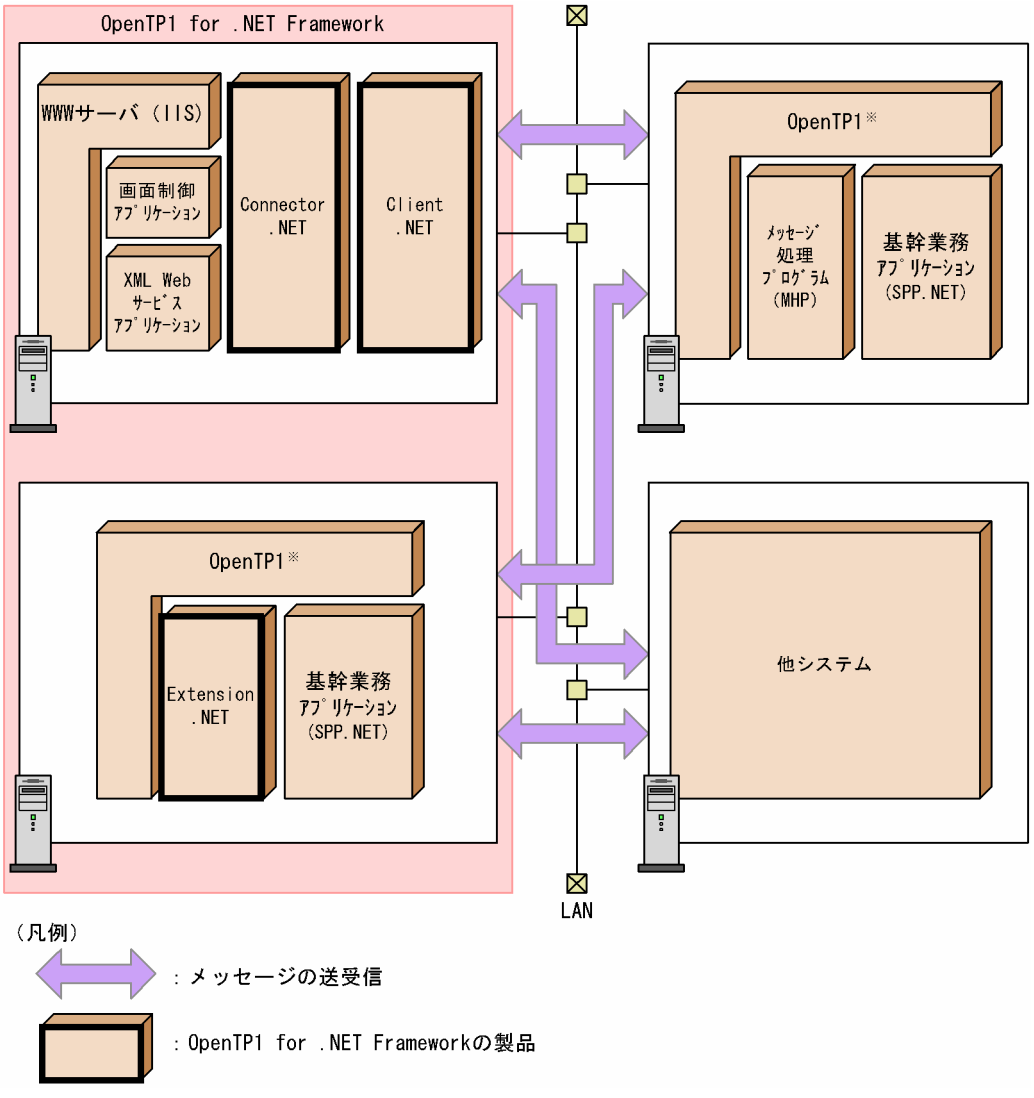
2.4　TCP/IP 通信機能

Client .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。

Client .NET で使用できる通信プロトコルは，TCP/IP だけです。そのため，Client .NET でのメッセージ送受信機能を，**TCP/IP 通信機能**と呼びます。TCP/IP 通信機能を使用する場合，Client .NET 構成定義の<tcpip>要素の use 属性に true を指定します。

TCP/IP 通信機能による通信の例を次に示します。

図 2-9　TCP/IP 通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。

2.4.1　Client .NET での TCP/IP 通信機能

Client .NET は，TCP/IP での通信機能を提供します。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。

CUP.NET がクライアントとなる場合は，CUP.NET 側から定義や引数で指定した通信相手に対してコネクションを確立して通信をします。CUP.NET がサーバとなる場合は，定義で指定したポートで通信相手がコネクションを確立したあと通信をします。コネクション確立後は，引数に指定された領域に対してデータの送受信をします。

2.4.2　通信形態

TCP/IP 通信機能を使用したメッセージの送受信には，次の 3 種類があります。

- CUP.NET から MHP へのメッセージの一方送信
- MHP から CUP.NET へのメッセージの一方受信
- MHP と CUP.NET との間のメッセージの送受信

(1)　メッセージの一方送信

CUP.NET から MHP 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。

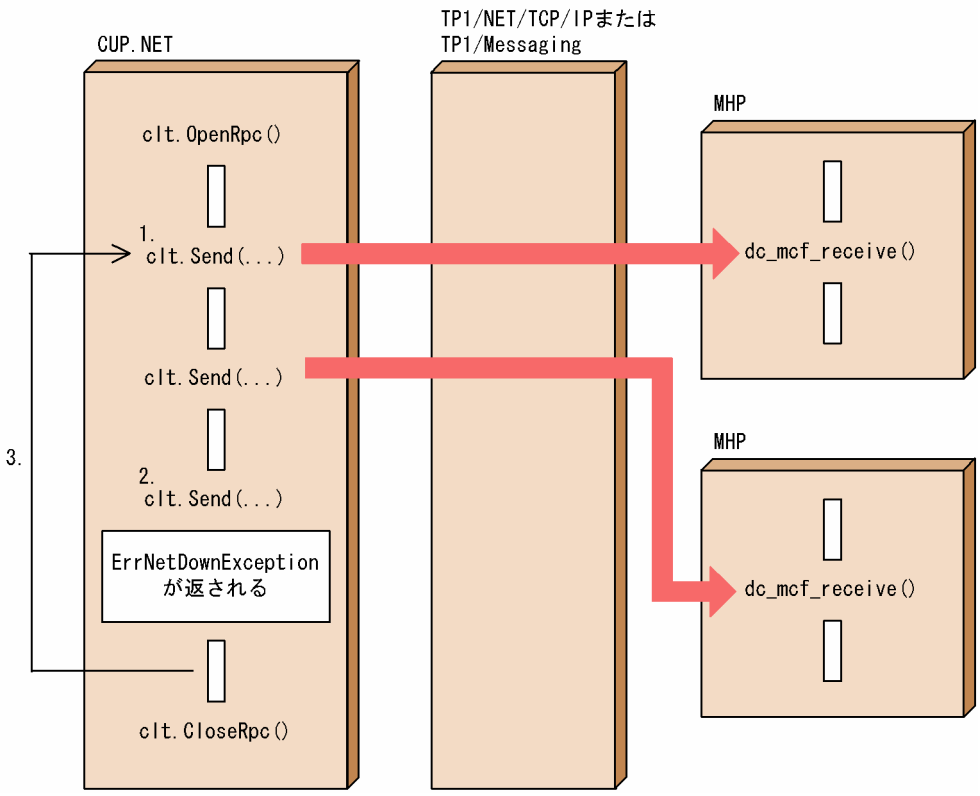
CUP.NET から Send メソッドを呼び出して，MHP へメッセージを送信します。メッセージの組み立て機能を使用する場合は，SendAssembledMessage メソッドを実行します。

メッセージを一方送信するには，次の指定をしておく必要があります。

- 接続先のノード名を次のどちらかの方法で指定
  (a) Send メソッド，または SendAssembledMessage メソッドの引数 hostname に指定
  (b) Client .NET 構成定義の<tcpip>要素の sendHost 属性に指定
- 接続先のポート番号（MCF 通信構成定義の定義コマンド mcftalccn の portno で指定したポート番号）を次のどちらかの方法で指定
  (a) Send メソッド，または SendAssembledMessage メソッドの引数 portnum に指定
  (b) Client .NET 構成定義の<tcpip>要素の sendPort 属性に指定
- Client .NET 構成定義の<tcpip>要素の type 属性に send を指定

メッセージの一方送信を次の図に示します。

図 2-10　メッセージの一方送信（CUP.NET）



1. MHP が起動されたあと，CUP.NET から Send メソッドを呼び出します。
2. Send メソッドの呼び出し時に，MHP からコネクションが解放されていたときは，CUP.NET に ErrNetDownException が返されます。
3. 再びメッセージを一方送信するには，Send メソッドを呼び出します。

(2)　メッセージの一方受信

MHP から CUP.NET に対して一方的に送信したメッセージを CUP.NET で受信できます。これを**メッセージの一方受信**といいます。

CUP.NET は，Receive メソッドを呼び出して，MHP からのメッセージを TCP/IP プロトコルを使用して受信します。メッセージの組み立て機能を使用する場合は，ReceiveAssembledMessage メソッドを実行します。

指定したメッセージ長よりも短いメッセージを受信した場合，Client .NET は，メッセージが分割されているものとみなし，指定した長さ分のメッセージを受信するまで，CUP.NET に制御を戻しません。タイムアウトやエラーが発生した場合に，指定した長さ分のメッセージに満たないときでも，その時点までのメッセージを受信できます。ただし，それ以降のメッセージの組み立ては，ユーザの責任で行ってください。

メッセージを一方受信するには，あらかじめ，Client .NET 構成定義で次の指定をしておく必要があります。

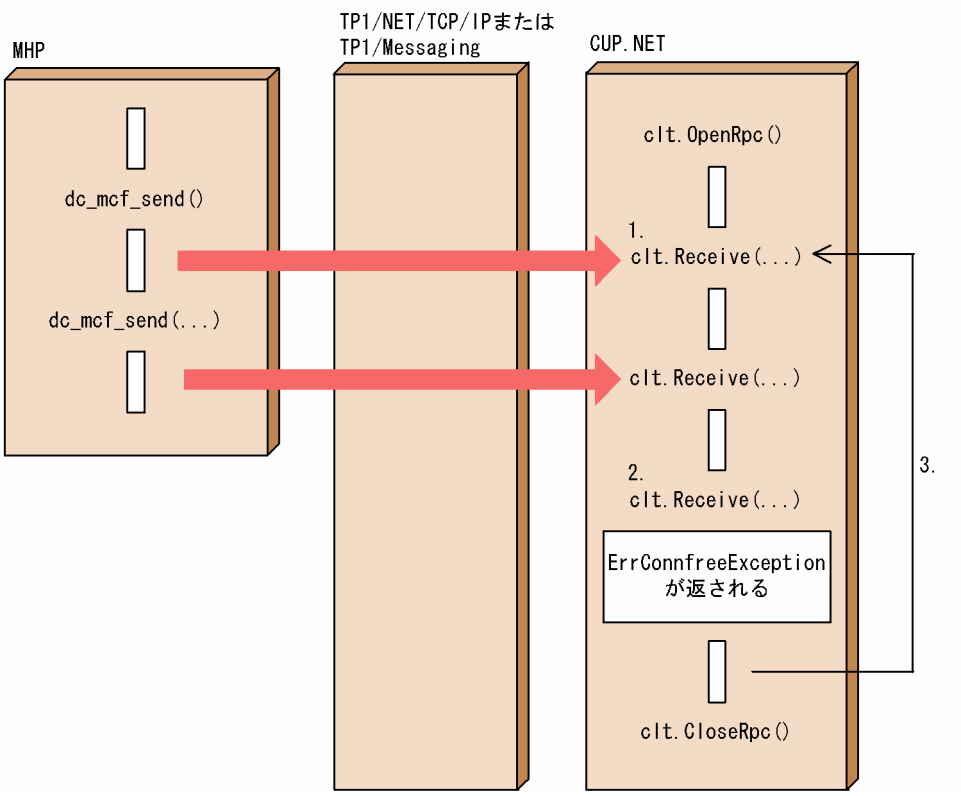
- <tcpip>要素の recvPort 属性に CUP.NET の受信用ポート番号（MCF 通信構成定義の定義コマンド mcftalccn の oportno オペランドで指定したポート番号）を指定

- <tcpip>要素の type 属性に recv を指定

マルチスレッドで動作する CUP.NET を含め，同一マシンで CUP.NET を複数実行する場合は，Client .NET 構成定義の<tcpip>要素の recvPort 属性に CUP ごと（スレッドごと）に異なる CUP.NET の受信用ポート番号を設定してください。CUP.NET ごとの Client .NET 構成定義の設定方法については，「3.　構成定義」を参照してください。受信用ソケットの開設は，OpenRpc メソッド実行時に行います。

メッセージの一方受信を次の図に示します。

図 2-11　メッセージの一方受信（CUP.NET）



1. MHP がコネクションの確立要求を再試行している間に，CUP.NET が Receive メソッドを呼び出します。再試行中にコネクションを確立できなかった場合，mcftactcn コマンドを入力してコネクションを確立します。
2. MHP からコネクションが解放された場合，CUP.NET に ErrConnfreeException が返されます。
3. 再びメッセージを一方受信するには，Receive メソッドを呼び出します。この場合，mcftactcn コマンドを入力してコネクションを確立してください。

(3)　メッセージの送受信

CUP.NET と MHP との間で，メッセージを送受信できます。

メッセージの送受信は，コネクションが解放されていなければ同じコネクションを使用します。

MHP がサーバ型の場合

CUP.NET から MHP に対してメッセージを送信した際に確立したコネクションを使用してメッセージの送受信を行います。このため，CUP.NET では受信用のソケットを使用しません。

CUP.NET から Send メソッドを実行して MHP へメッセージを送信し，Receive メソッドを実行して，MHP からのメッセージを受信します。メッセージの組み立て機能を使用する場合は，CUP.NET から SendAssembledMessage メソッドを実行して MHP へメッセージを送信し，ReceiveAssembledMessage メソッドを実行して MHP からのメッセージを受信します。

MHP がクライアント型の場合

MHP から送信されたメッセージを CUP.NET で受信した際に確立したコネクションを使用してメッセージの送受信を行います。

CUP.NET で Receive メソッドを実行して，MHP からのメッセージを受信し，Send メソッドを実行して MHP へメッセージを送信します。メッセージの組み立て機能，または送達確認機能を使用する場合は，CUP.NET で ReceiveAssembledMessage メソッドを実行して MHP からのメッセージを受信し，SendAssembledMessage メソッドを実行して MHP へメッセージを送信します。

メッセージを送受信するには，次の指定をしておく必要があります。Client .NET 構成定義の詳細については，「3.　構成定義」を参照してください。

MHP がサーバ型の場合

- 接続先のノード名を次のどちらかの方法で指定

  (a) Send メソッド，または SendAssembledMessage メソッドの引数 hostname に指定

  (b) Client .NET 構成定義の<tcpip>要素の sendHost 属性に指定

- 接続先のポート番号（MCF 通信構成定義の定義コマンド mcftalccn の portno で指定したポート番号）を次のどちらかの方法で指定

  (a) Send メソッド，または SendAssembledMessage メソッドの引数 portnum に指定

  (b) Client .NET 構成定義の<tcpip>要素の sendPort 属性に指定

- Client .NET 構成定義の<tcpip>要素の type 属性に sendrecv を指定
- Client .NET 構成定義の<tcpip>要素の openPortAtRecv 属性に true を指定

  CUP.NET では受信用ソケットを使用しないため，Client .NET 構成定義<tcpip>要素の openPortAtRecv 属性に true を指定し，受信用ソケットを開設しないようにしてください。

MHP がクライアント型の場合

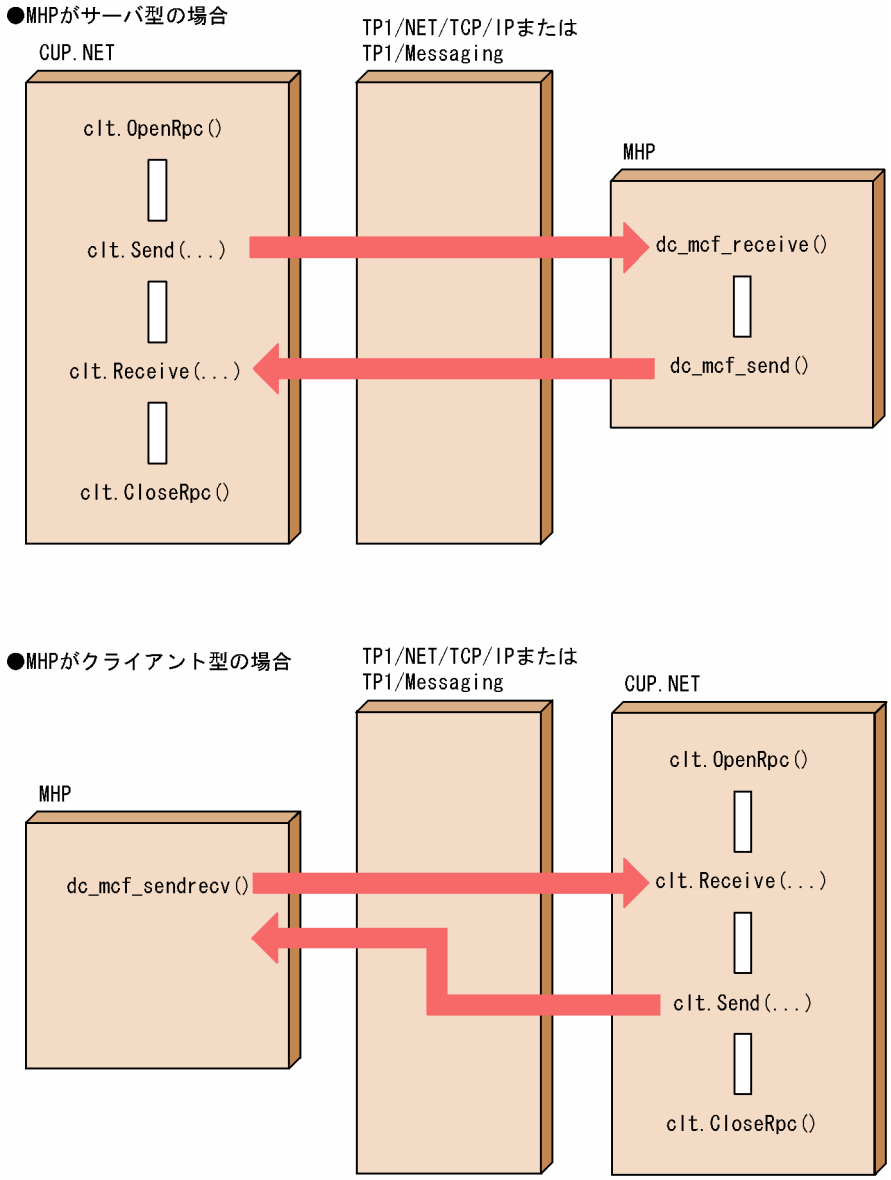
- Client .NET 構成定義の<tcpip>要素の recvPort 属性に CUP.NET の受信用ポート番号（MCF 通信構成定義の定義コマンド mcftalccn の oportno で指定したポート番号）を指定

  マルチスレッドで動作する CUP.NET を含め，同一マシンで CUP.NET を複数実行する場合は，Client .NET 構成定義の<tcpip>要素の recvPort 属性を CUP ごと（スレッドごと）に異なるポート番号となるように設定してください。CUP.NET ごとの Client .NET 構成定義の設定方法については，「3.　構成定義」を参照してください。

- Client .NET 構成定義の<tcpip>要素の type 属性に sendrecv を指定

メッセージの送受信を次の図に示します。

図 2-12　メッセージの送受信（CUP.NET）



2.4.3　受信メッセージの組み立て機能

Client .NET の**受信メッセージの組み立て機能**を使用すると，メッセージの送信時に，メッセージの先頭４バイトにメッセージ長エリアを付与します。また，メッセージの受信時には，メッセージの先頭４バイトをメッセージ長エリアとして扱います。このため，TP1/NET/TCP/IP の受信メッセージの組み立て機能を使用して送受信されるメッセージのメッセージ長エリアを，CUP.NET 側で意識する必要がなくなります。

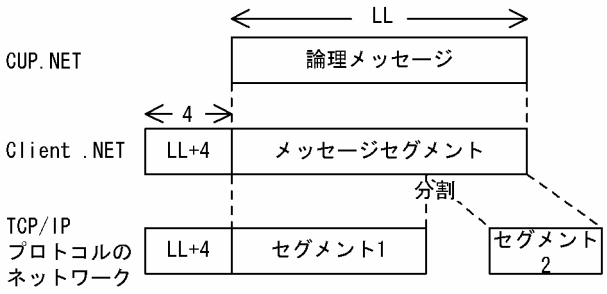
(1) メッセージ送信時の処理

CUP.NET から TP1Client クラスの SendAssembledMessage メソッドを呼び出すと，Client .NET がメッセージの先頭４バイトにメッセージ長エリアを付与して相手システムに送信します。Client .NET は，メッセージ長をネットワークバイトオーダーで設定します。

TP1Client クラスの SendAssembledMessage メソッドを使用するためには，Client .NET 構成定義で<tcpip>要素の type 属性に send または sendrecv を指定しておく必要があります。

メッセージ送信時の処理を次の図に示します。

図 2-13　メッセージ送信時の処理



（凡例）

LL：メッセージ長

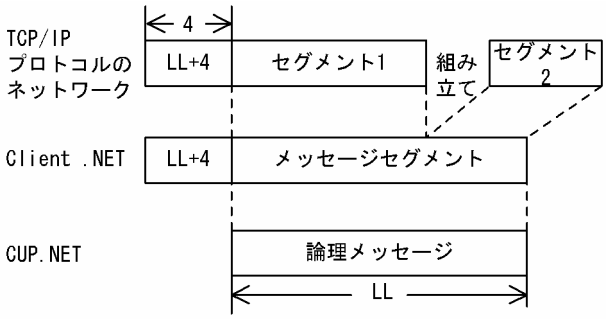
(2) メッセージ受信時の処理

CUP.NET から TP1Client クラスの ReceiveAssembledMessage メソッドを呼び出してメッセージの受信要求をすると，受信したメッセージの先頭４バイトをメッセージ長エリアとして扱います。Client .NET は，メッセージ長エリアを基にメッセージを組み立て，メッセージ長エリアを取り除いた状態で CUP.NET に通知します。Client .NET では，メッセージ長エリアに設定された値をネットワークバイトオーダーでチェックするため，相手システムでもメッセージ長をネットワークバイトオーダーで設定してください。

TP1Client クラスの ReceiveAssembledMessage メソッドを使用するためには，Client .NET 構成定義で<tcpip>要素の type 属性に send または sendrecv を指定しておく必要があります。

メッセージ受信時の処理を次の図に示します。

図 2-14　メッセージ受信時の処理



（凡例）

LL：メッセージ長

2.4.4 TCP/IP 通信機能を使用するときの注意事項

(1) 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

次に示す障害が発生すると，Client .NET および相手システムでは，メッセージが消失したことを検出できません。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。

- ソケットのバッファに Client .NET が送信したメッセージが書き込まれて送信が正常終了した直後に，通信障害が発生したり，コネクションが解放されたりした場合
- Client .NET が送信したメッセージが相手システムの受信バッファに書き込まれる直前に通信障害が発生したり，コネクションが解放されたりした場合

(b) コネクションの確立

Client .NET がクライアントとなり，相手システムにメッセージを送信します。そのため，Client .NET から相手システムに対して，コネクションを確立します。相手システムが TP1/NET/TCP/IP を使用している場合，コネクションはサーバ型となります。

(2) メッセージ受信時の注意事項

(a) 障害発生時のメッセージの消失

次に示す障害が発生すると，Client .NET および相手システムでは，メッセージが消失したことを検出できません。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。

- ソケットのバッファに相手システムが送信したメッセージが書き込まれて送信が正常終了した直後に，通信障害が発生したり，コネクションが解放されたりした場合
- 相手システムが送信したメッセージが Client .NET 側の受信バッファに書き込まれる直前に，通信障害が発生したり，コネクションが解放されたりした場合

(b) 受信するメッセージの確認

任意の相手システムからのメッセージを受信できます。そのため，コネクションの確立要求を受けると，その要求を無条件に受諾してメッセージを受信します。ユーザは，メッセージ識別子が含まれたヘッダをメッセージ中に含めるなどして，CUP.NET が受け取るメッセージかどうかを確認してください。

(c)　メッセージ長

TCP/IP プロトコルを使用してメッセージを受信します。

TCP/IP プロトコルでは，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，Client .NET は，メッセージが分割されているものと見なします。そのため，指定した長さ分のメッセージを受信するまで，CUP.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。

(d)　コネクションの確立

Client .NET がサーバとなり，相手システムからのメッセージを受信します。そのため，相手システムから Client .NET に対して，コネクションを確立します。相手システムが TP1/NET/TCP/IP を使用している場合，コネクションはクライアント型になります。

(3)　その他の注意事項

TP1/NET/TCP/IP の受信メッセージの組み立て機能を使用すると，MHP 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，MHP 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，MHP が受信するときには削除されます。CUP.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 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 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 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。

2.4.5　ユースケースごとの設定方法とポートの割り当て

TCP/IP 通信機能の主なユースケースを次の表に示し，それぞれのユースケースの設定方法とポートの割り当てについて説明します。ユースケースごとの設定方法と使用するポート番号を(5)に示します。

表 2-6　TCP/IP 通信機能のユースケース

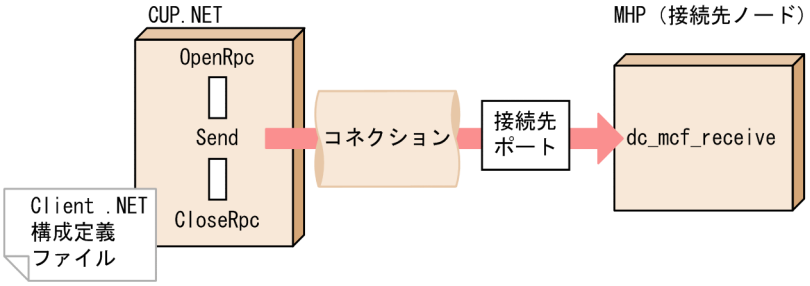
項番	ユースケース
1	MHP がサーバ型でメッセージの一方送信を行う場合

項番	ユースケース
2	MHP がクライアント型でメッセージの一方受信を行う場合
3	MHP がサーバ型でメッセージの送受信を 1 つのコネクションで行う場合
4	MHP がクライアント型でメッセージの送受信を 1 つのコネクションで行う場合

(1)　MHP がサーバ型でメッセージの一方送信を行う場合

メッセージの一方送信を次の図に示します。

図 2-15　MHP がサーバ型でメッセージの一方送信を行う場合



メッセージの一方送信は，CUP.NET から接続先のポートに対してコネクションを確立し，メッセージの送信を行います。メッセージの一方送信を行う場合は，次の設定をしてください。その他の設定は条件に応じて設定してください。

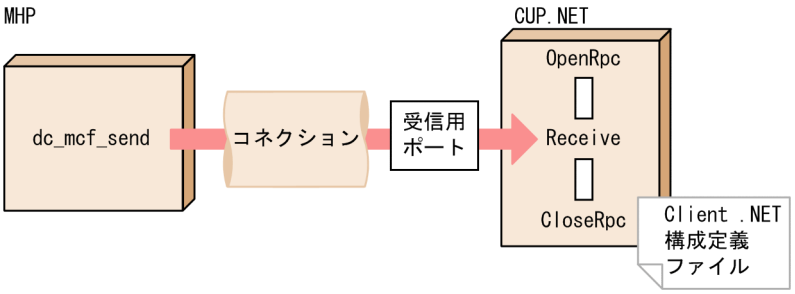
- 接続先のノード名を次のどちらかの方法で指定
  (a) Send メソッド，または SendAssembledMessage メソッドの引数 hostname に指定
  (b) Client .NET 構成定義の<tcpip>要素の sendHost 属性に MHP が存在するノードのホスト名を指定
- 接続先のポート番号を次のどちらかの方法で指定
  (a) Send メソッド，または SendAssembledMessage メソッドの引数 portnum に指定
  (b) Client .NET 構成定義の<tcpip>要素の sendPort 属性に指定
- Client .NET 構成定義の<tcpip>要素の type 属性に send を指定

接続先のノード名，またはポート番号が CUP.NET ごとに異なる場合は，CUP.NET ごとに設定を変更してください。CUP.NET ごとの構成定義の設定方法については，「3.　構成定義」を参照してください。

(2)　MHP がクライアント型でメッセージの一方受信を行う場合

メッセージの一方受信を次の図に示します。

図 2-16　MHP がクライアント型でメッセージの一方受信を行う場合



メッセージの一方受信は，MHP から CUP.NET に割り当てられた受信用ポートに対してコネクションを確立し，メッセージの送信を行います。

メッセージの一方受信を行う場合は，次の設定をしてください。その他の設定は条件に応じて設定してください。

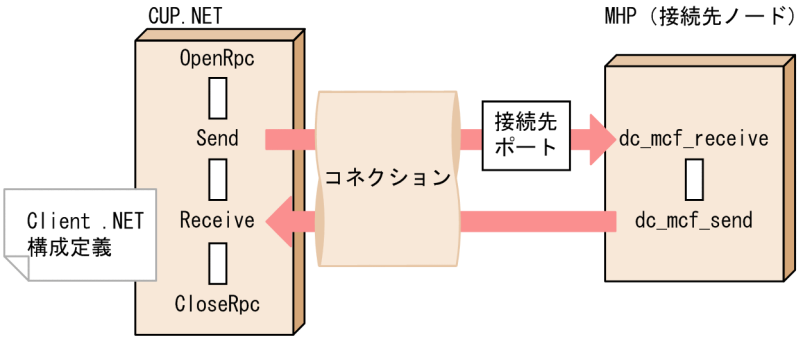
- Client .NET 構成定義の<tcpip>要素の recvPort 属性に CUP.NET の受信用ポート番号（MCF 通信構成定義の定義コマンド mcftalccn の oportno で指定したポート番号）を指定
- Client .NET 構成定義の<tcpip>要素の type 属性に recv を指定
- Client .NET 構成定義の<tcpip>要素の openPortAtRecv 属性に false を指定，または指定を省略

マルチスレッドで動作する CUP.NET を含め，同一マシンで CUP.NET を複数実行する場合は，Client .NET 構成定義の<tcpip>要素の recvPort 属性を CUP.NET ごと（スレッドごと）に異なるポート番号となるように設定してください。CUP.NET ごとの構成定義の設定方法については，「3.　構成定義」を参照してください。

(3)　MHP がサーバ型でメッセージの送受信を 1 つのコネクションで行う場合

MHP がサーバ型でメッセージの送受信を 1 つのコネクションで行う場合を次の図に示します。

図 2-17　MHP がサーバ型でメッセージの送受信を 1 つのコネクションで行う場合



MHP がサーバ型でメッセージの送受信を 1 つのコネクションで行う場合，CUP.NET から接続先のポートに対してコネクションを確立し，そのコネクションを使用してメッセージを送受信するため，CUP.NET では受信用のポートを使用しません。そのため，Client .NET 構成定義の<tcpip>要素の recvPort 属性を設定する必要はありません。

MHP がサーバ型でメッセージの送受信を 1 つのコネクションで行う場合は，次の設定をしてください。その他の設定は条件に応じて設定してください。

- 接続先のノード名を次のどちらかの方法で指定
  (a) Send メソッド，または SendAssembledMessage メソッドの引数 hostname に指定
  (b) Client .NET 構成定義の<tcpip>要素の sendHost 属性に MHP が存在するノードのホスト名を指定
- 接続先のポート番号を次のどちらかの方法で指定
  (a) Send メソッド，または SendAssembledMessage メソッドの引数 portnum に指定
  (b) Client .NET 構成定義の<tcpip>要素の sendPort 属性に指定
- Client .NET 構成定義の<tcpip>要素の openPortAtRecv 属性に true を指定
- Client .NET 構成定義の<tcpip>要素の type 属性に sendrecv を指定
- Send メソッド，または SendAssembledMessage メソッドの引数 flags に DCNOFLAGS を指定

接続先のノード名，またはポート番号が CUP.NET ごとに異なる場合は，CUP.NET ごとに設定を変更してください。CUP.NET ごとの構成定義の設定方法については，「3.　構成定義」を参照してください。

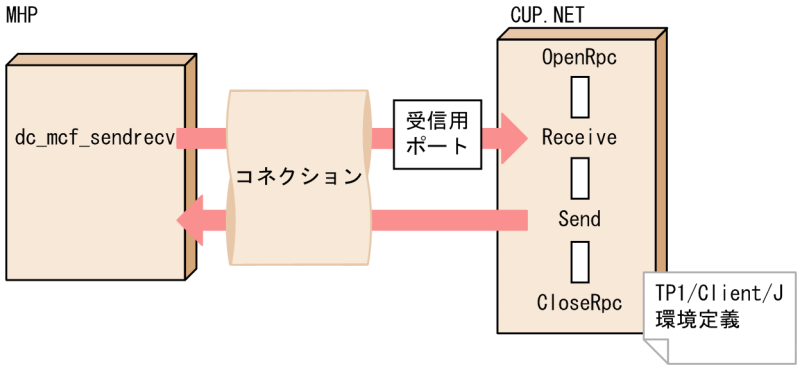
注意事項

何らかの障害が発生しコネクションが切断された場合，Receive メソッド，および ReceiveAssembledMessage メソッドを実行しないでください。コネクションの切断を検知した場合は，Send メソッド，または SendAssembledMessage メソッドを再実行し，コネクションを確立してからメッセージを受信してください。

(4)　MHP がクライアント型でメッセージの送受信を 1 つのコネクションで行う場合

MHP がクライアント型でメッセージの送受信を 1 つのコネクションで行う場合を次の図に示します。

図 2-18　MHP がクライアント型でメッセージの送受信を 1 つのコネクションで行う場合



MHP がクライアント型でメッセージの送受信を 1 つのコネクションで行う場合，MHP から CUP.NET に割り当てられた受信用ポートに対してコネクションを確立し，そのコネクションを使用してメッセージを送受信します。

MHP がクライアント型でメッセージの送受信を 1 つのコネクションで行う場合は，次の設定をしてください。その他の設定は条件に応じて設定してください。

- Client .NET 構成定義の<tcpip>要素の recvPort 属性に CUP.NET の受信用ポート番号（MCF 通信構成定義の定義コマンド mcftalccn の oportno で指定したポート番号）を指定
- Client .NET 構成定義の<tcpip>要素の openPortAtRecv 属性に false を指定，または指定を省略
- Client .NET 構成定義の<tcpip>要素の type 属性に sendrecv を指定
- Receive メソッド，または Receive AssembledMessage メソッドの引数 flags に DCNOFLAGS を指定

マルチスレッドで動作する CUP.NET を含め，同一マシンで CUP.NET を複数実行する場合は，Client .NET 構成定義の<tcpip>要素の recvPort 属性を CUP.NET ごと（スレッドごと）に異なるポート番号となるように設定してください。CUP.NET ごとの構成定義の設定方法については，「3. 構成定義」を参照してください。

(5) ユースケースごとの設定方法と使用するポート番号

項番	ユースケース	Client .NET 構成定義 <tcpip>要素					提供 API の発行順序※3	CUP.NET の使用ポート番号	
		openPortAtRecv 属性	sendHost 属性 ※1	sendPort 属性 ※2	recvPort 属性	type 属性		送信	受信
1	MHP がサーバ型でメッセージの一方送信を行う場合，詳細は 2.4.5 (1) を参照してください。	-	接続先のノード名	接続先のポート番号	-	send	Send	OS 自動割り当て	使用しない
2	MHP がクライアント型でメッセージの一方受信を行う場合，詳細は 2.4.5 (2) を参照してください。	false または省略	-	-	CUP.NET のポート番号	recv	Receive	使用しない	recvPort 属性の値
3	MHP がサーバ型で送受信を 1 本のコネクションで行う場合，詳細は 2.4.5 (3) を参照してください。	true※4	接続先のノード名	接続先のポート番号	-	sendrecv	Send※5→ Receive	OS 自動割り当て	使用しない

項番	ユースケース	Client .NET 構成定義　<tcpip>要素					提供 API の発行順序※3	CUP.NET の使用ポート番号	
		openPortAtRecv 属性	sendHost 属性 ※1	sendPort 属性 ※2	recvPort 属性	type 属性		送信	受信
4	MHP がクライアント型で送受信を 1 本のコネクションで行う場合，詳細は 2.4.5（4）を参照してください。	false または省略 ※6	-	-	CUP.NET のポート番号	sendrecv	Receive※7→Send	使用しない	recvPort 属性の値

（凡例）

－：該当しません。

注※1

接続先のノード名は，Send メソッド，および SendAssembledMessage メソッドの引数 hostname に指定することもできます。

注※2

接続先のポート番号は，Send メソッド，および SendAssembledMessage メソッドの引数 portnum に指定することもできます。

注※3

表中の Send メソッドは，SendAssembledMessage メソッドを含みます。また，Receive メソッド は，ReceiveAssembledMessage メソッドを含みます。

注※4

openPortAtRecv 属性に false を指定する，または省略する設定は，受信用のポートを余分に割り当てるため推奨しません。

注※5

Send メソッド，および SendAssembledMessage メソッドの引数 flags には DCNOFLAGS を指定してください。

注※6

openPortAtRecv 属性に true を指定する設定は，MHP とのコネクション確立の同期をユーザ処理で実装する必要があるため推奨しません。

注※7

Receive メソッド，および ReceiveAssembledMessage メソッドの引数 flags には DCNOFLAGS を指定してください。

2.5　サーバからの一方通知受信機能

サーバからクライアントへのメッセージの一方通知受信機能について説明します。

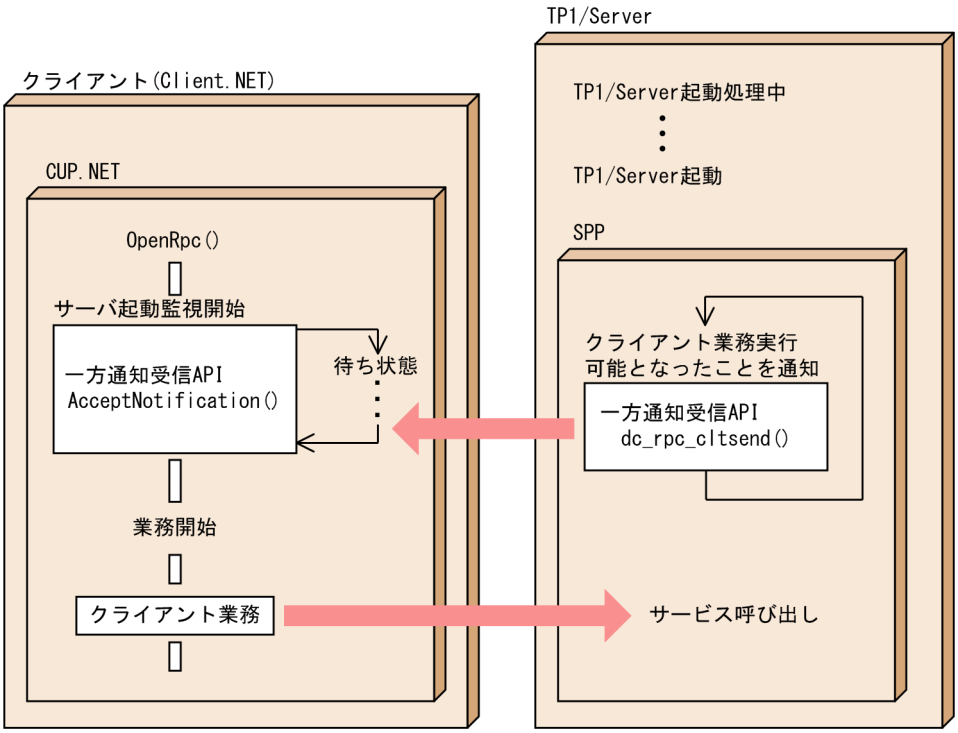
2.5.1　一方通知受信機能の処理の流れ

一方通知受信機能を使用すると，オンライン開始の合図をクライアント側に一斉配布する，従来メインフレームの OLTP での端末一斉起動と同様な運用ができるようになります。

一方通知受信機能を使用する場合，AcceptNotification メソッドを実行します。これによって，クライアント側ではサーバの状態（起動・未起動）に関係なくサーバ側からの送信メッセージをメソッドに指定した時間中待ち続けます。サーバ側で起動時にメッセージを送信することによって，クライアント側はサーバ起動を検出し，これを契機にユーザ業務（CUP.NET）を開始できます。

一方通知受信機能の処理の流れを次の図に示します。

図 2-19　一方通知受信機能の処理の流れ



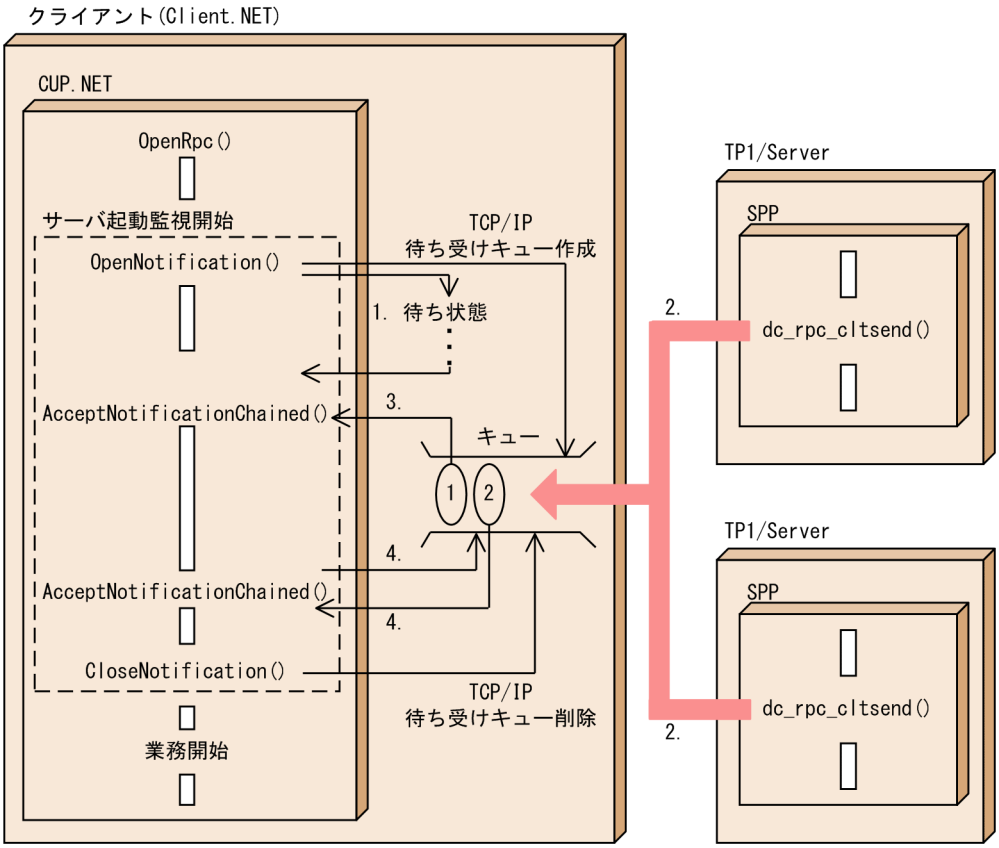
2.5.2　一方通知連続受信機能の処理の流れ

一方通知連続受信機能を使用すると，OpenNotification メソッドを実行してから CloseNotification メソッドを実行するまでの間は，サーバ側からの一方通知メッセージを連続して受信できます。この機能を利用すると，サーバ側から送信される一方通知メッセージを受信できる状態になっていない場合，サーバ側から一方通知メッセージを送信しても，クライアント側はエラーリターンしません。その場合は，クラ

イアント側が一方通知メッセージを受信する AcceptNotificationChained メソッドを実行した時点で，待ち受けキューからメッセージを取り出すことができます。

一方通知連続受信機能の処理の流れを次の図に示します。

図 2-20　一方通知連続受信機能の処理の流れ



1. 一方通知メッセージが送られてくるのを待ちます。
2. TP1/Server が起動して，クライアント業務が実行できる状態になったことを，一方通知メッセージを送信して CUP.NET に通知します。
3. TCP/IP の待ち受けキューに一方通知メッセージが届いたため，一方通知メッセージを取り出して，CUP.NET に制御を戻します。
4. AcceptNotificationChained メソッドの発行時に，TCP/IP の待ち受けキューにサーバからの一方通知メッセージが届いたため，一方通知メッセージを取り出して，CUP.NET に制御を戻します。

2.5.3　一方通知連続受信機能を使用するときの注意事項

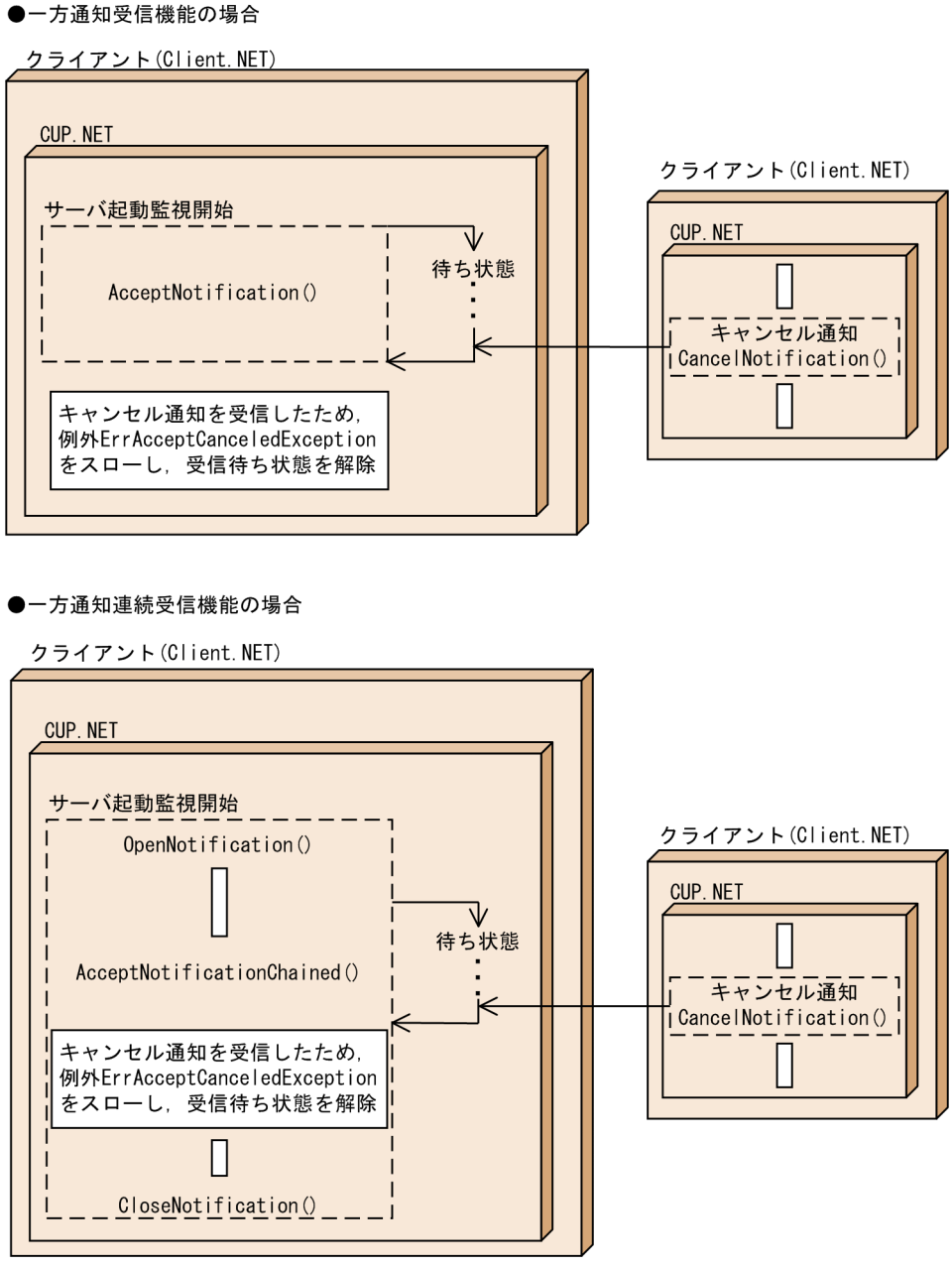
TCP/IP の待ち受けキューに保留できるメッセージの数には上限があります。上限値は 5 です。上限値を超えるメッセージが到着した場合，サーバ側で実行した dc_rpc_cltsend 関数は，DCRPCER_SERVICE_NOT_UP でエラーリターンします。

2.5.4　一方通知受信待ち状態の解除

AcceptNotification メソッド，または AcceptNotificationChained メソッドを発行し，サーバからの一方通知受信待ち状態となった CUP.NET が，別の CUP.NET からキャンセル通知を受信した場合，受信待ち状態を解除します。キャンセル通知は，CancelNotification メソッドを発行することで送信できます。

一方通知受信待ち状態を解除する処理の流れを次の図に示します。

図 2-21　一方通知受信待ち状態を解除する処理の流れ



2.6 動的定義変更機能

動的定義変更機能を使用すると，Client .NET の構成ファイルで指定した構成定義を CUP.NET の実行中に変更できます。動的定義変更用の SetXxxx メソッド（Xxxx は変更する構成定義項目に対応した名称）を呼び出して，構成定義を変更します。なお，動的定義変更用のメソッドを呼び出して変更した構成定義は，CUP.NET が終了するまで（CloseRpc メソッドが呼び出されるまで）の間は，同じ動的定義変更用のメソッドを呼び出して構成定義を変更しないかぎり，変更されません。

Client .NET 構成定義と，動的定義変更機能で設定した構成定義のどちらが有効になるかは，メソッドの呼び出し順序で決定します。

- **OpenRpc メソッドを呼び出したあとに動的定義変更用のメソッドを呼び出した場合**
  動的定義変更用のメソッドで変更した構成定義が有効になります。
- **動的定義変更用のメソッドを呼び出したあとに OpenRpc メソッドを呼び出した場合**
  Client .NET 構成定義が有効になります。

2.7　ノード間負荷バランス機能

OpenTP1 では，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があります。これを**ノード間負荷バランス機能**といいます。

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。

ここでは，OpenTP1 のノード間負荷バランス機能を使用する場合の Client .NET 側，TP1/Server 側の関連する定義，処理，および RPC の処理を説明します。

ノード間負荷バランス機能の詳細については，マニュアル「OpenTP1 解説」を参照してください。

2.7.1　サーバ側の判断で負荷分散を行う場合

TP1/Server のスケジュールサービスが，ノードのスケジュール状態に応じて，より効率的に処理できるノードへ負荷を分散させます。

(1)　Client .NET 側の定義

Client .NET 構成定義で必要な設定はありません。

(2)　TP1/Server 側の定義

TP1/Server の定義では，次の設定をする必要があります。

- スケジュールサービス定義に次のオペランドを指定または省略する。
  scd_this_node_first=N（デフォルト）
  scd_announce_server_status=Y（デフォルト）

2.7.2　サーバからの負荷情報からクライアント側で判断する場合

サーバから得たサーバの負荷レベルを基に，サービス要求を行うクライアントが OpenTP1 ノードを決めて RPC を実行します。

(1)　Client .NET 側の定義

Client .NET 構成定義で次の設定をする必要があります。

- <rpc>要素の use 属性に nam を指定する。
- <nameService>要素の loadBalance 属性に true を指定する。

(2) TP1/Server 側の定義

TP1/Server 側の定義では，次の設定をする必要があります。

- スケジュールサービス定義に次のオペランドを指定または省略する。
  scd_announce_server_status=Y（デフォルト）

2.7.3 RPC 種別による動作の違い

(1) リモート API 機能を組み合わせた場合

ノード間負荷バランス機能とリモート API 機能を使用した場合，サーバ側の判断で負荷分散を行います。

(2) スケジューラダイレクト機能を組み合わせた場合

スケジューラダイレクト機能を使用した RPC では，スケジュールを依頼する OpenTP1 ノードが複数ある場合（<tp1Server>要素を複数指定した場合），<tp1Server>要素を指定した順番にスケジュールを依頼します。

RPC ごとにサービス要求先スケジューラをラウンドロビン方式で分散させる場合は，Client .NET 構成定義<scheduleService>要素の hostChange 属性に true を指定します。

最初の RPC でのサービス要求時に，サービス要求先スケジューラをランダムに選択するには，Client .NET 構成定義<scheduleService>要素の randomSelect 属性に true を指定します。

(3) ネームサービスを使用した RPC を組み合わせた場合

ネームサービスを使用した RPC では，窓口となる TP1/Server のネームサービスにサービス情報を問い合わせ，ネームサービスからの応答メッセージを基にサービス要求先スケジューラを決定します。

RPC ごとにサービス要求先スケジューラを分散させる場合は，Client .NET 構成定義<nameService>要素の loadBalance 属性に true を指定します。これによって，Client .NET は，ネームサーバから複数のサービス要求先スケジューラの情報を取得し，負荷レベルが最も低いサービス要求先スケジューラの情報をキャッシュに格納します。

キャッシュに格納されるサービス要求先スケジューラの情報は，1 つだけではなく，複数の場合もあります。

キャッシュに格納されたサービス要求先スケジューラの情報が複数ある場合，最初のサービス要求先スケジューラは，ランダムに選択されます。RPC でのサービス要求が 2 度目以降の場合は，サービス要求先ス

ケジューラの情報を取得するためにキャッシュを参照し，RPC ごとにラウンドロビン方式でサービス要求先スケジューラを切り替え，分散させます。

クライアントからの RPC 実行時に，このキャッシュ領域中に該当するサービス情報が存在する場合には，窓口となる TP1/Server のネームサービスに対してサービス情報の問い合わせを行いません。クライアントでは LRU（Least Recently Used）方式でキャッシュを管理しているため，キャッシュ領域が不足した場合には参照されていないサービス情報から順に削除します。また，Client .NET 構成定義<nameService>要素の cacheTime 属性に指定した有効時間が過ぎたサービス情報は，RPC 実行時にキャッシュ領域から削除され，ネームサービスに対してサービス情報を問い合わせます。

注意事項

- Client .NET 構成定義<nameService>要素の cacheCapacity 属性の指定値を大きくすると，多くのサービス情報を格納でき，窓口となる TP1/Server のネームサービスとの通信回数を削減できます。ただし，多くのキャッシュ領域中からサービス情報を検索するのでオーバヘッドが掛かります。
- Client .NET 構成定義<nameService>要素の cacheTime 属性の指定値を小さくすると，古いサービス情報はただちに削除され，窓口となる TP1/Server のネームサービスに新しいサービス情報を問い合わせます。この場合，常に最新のサービス情報をキャッシュ領域に保持できるため，サーバの負荷に応じて RPC 要求を振り分けられます。ただし，ネームサービスとの通信回数が増え，また，キャッシュ領域の書き換え処理にもオーバヘッドが掛かります。
- Client .NET 構成定義<nameService>要素の cacheTime 属性の指定値を大きくすると，窓口となる TP1/Server のネームサービスとの通信回数を削減できます。ただし，SPP の状態変化への対応が遅れるため，SPP が停止したノードのスケジューラに対して RPC 要求を実行してしまうことがあります。

(4) 通信先を指定した RPC を組み合わせた場合

通信先を指定した RPC を使用した場合，ノード間負荷バランス機能は使用できません。

2.8　TCP/IP コネクションの確立の監視機能

TCP/IP コネクションの確立の監視機能では，データ送信時の TCP/IP コネクションの確立処理に対する最大監視時間を指定し，コネクションの確立処理を監視できます。

コネクションの確立処理には，Client .NET の内部 API（.NET Framework の API）である Socket.Connect メソッド（以降「connect メソッド」と呼びます）が使用されます。connect メソッドのタイムアウト監視時間は，レジストリの設定値によって決まります。Client .NET 構成定義でコネクション確立処理に対する最大監視時間を指定してコネクションの確立処理を監視すると，レジストリの設定値よりも短い時間で，Client .NET の API をエラーリターンさせることができます。connect メソッドでのコネクション確立処理に対する最大監視時間は，Client .NET 構成定義の<socket>要素の connectTimeout 属性で指定します。

2.8.1　内部で connect メソッドを使用する API

API が内部で connect メソッドを使用するかどうかは，Client .NET 構成定義の指定によって異なります。内部で connect メソッドを使用する API（メソッド）と，内部で connect メソッドを使用する Client .NET 構成定義の条件を次の表に示します。条件が二つ以上ある API（メソッド）の場合，内部で connect メソッドを使用するには，一つ以上の条件を満たしている必要があります。

表 2-7　内部で connect メソッドを使用する API（メソッド）

項番	内部で connect メソッドを使用する API（メソッド）	内部で connect メソッドを使用する Client .NET 構成定義の条件
1	SendAssembledMessage(System.Byte[], System.Int32, System.String, System.Int32, System.Int32, System.Int32)	• <tcpip>要素の type 属性="send" • <tcpip>要素の type 属性="sendrecv"
2	Send(System.Byte[], System.Int32, System.String, System.Int32, System.Int32)	
3	OpenConnection(System.String, System.Int32)	<rpc>要素の type 属性="rap"かつ<rapService>要素の autoConnect 属性="false"
4	OpenConnection()	
5	Call(System.String, System.String, System.Byte[], System.Int32, System.Byte[], System.Int32&, System.Int32)	• <rpc>要素の type 属性="scd" • <rpc>要素の type 属性="nam" または <rpc>要素の type 属性="rap"かつ<rapService>要素の autoConnect 属性="true"
6	CallTo(Hitachi.OpenTP1.Client.DCRpcBindTbl, System.String, System.String, System.Byte[], System.Int32, System.Byte[], System.Int32&, System.Int32)	• <rpc>要素の type 属性="scd" • <rpc>要素の type 属性="nam"

項番	内部で connect メソッドを使用する API（メソッド）	内部で connect メソッドを使用する Client .NET 構成定義の条件
7	Begin()	<rpc>要素の type 属性="rap"かつ<rapService>要素の autoConnect 属性="true"

2.8.2 connect メソッドがタイムアウトした場合の例外

OS による connect メソッドのタイムアウト監視時間が満了した場合と，Client .NET 構成定義の<socket>要素の connectTimeout 属性に指定した時間が満了した場合とでは，Client .NET の API から返る例外が異なります。これは，満了する監視時間の違いによって，発生する現象が異なるためです。

connectTimeout 属性に，OS によるタイムアウト監視時間よりも大きい値を指定した場合，OS によるタイムアウト監視時間が先に満了します。このため，connectTimeout 属性の指定は無効となります。connectTimeout 属性の指定を省略した場合，connect メソッドのタイムアウト監視時間は，レジストリの設定値となります。

OS による connect メソッドのタイムアウト監視時間が満了した場合と，connectTimeout 属性に指定した時間が満了した場合の，Client .NET の API が返す例外を次の表に示します。

表 2-8 connect メソッドがタイムアウトした場合に API（メソッド）が返す例外

項番	API（メソッド）	connectTimeout 属性に指定した時間が満了した場合※1	OS によるタイムアウト監視時間が満了した場合※2
1	SendAssembledMessage(System.Byte[], System.Int32, System.String, System.Int32, System.Int32, System.Int32)	ErrClientTimedOutException	ErrNetDownAtClientException
2	Send(System.Byte[], System.Int32, System.String, System.Int32, System.Int32)	ErrNetDownAtClientException	ErrNetDownAtClientException
3	OpenConnection(System.String, System.Int32)	ErrClientTimedOutException	ErrNetDownAtClientException
4	OpenConnection()	ErrClientTimedOutException	ErrNetDownAtClientException
5	Call(System.String, System.String, System.Byte[], System.Int32, System.Byte[], System.Int32&, System.Int32)	ErrClientTimedOutException	ErrNetDownAtClientException
6	CallTo(Hitachi.OpenTP1.Client.DCRpcBindTbl, System.String, System.String, System.Byte[], System.Int32, System.Byte[], System.Int32&, System.Int32)	ErrClientTimedOutException	ErrNetDownAtClientException

項番	API（メソッド）	connectTimeout 属性に指定した時間が満了した場合※1	OS によるタイムアウト監視時間が満了した場合※2
7	Begin()	ErrClientTimedOutException	ErrNetDownAtClientException

注※1

Client .NET 構成定義の<socket>要素の connectTimeout 属性に指定した時間が満了した場合に発生する例外です。

注※2

connect メソッドで OS によるコネクション確立処理の監視時間が満了した場合に発生する例外です。コネクション確立処理の監視時間（コネクション確立要求の再送回数，間隔）はレジストリの設定によって異なります。

2.9　トラブルシュート機能

Client .NET は，**トラブルシュート機能**として，次のトレース情報を取得できます。

- UAP トレース※1
- データトレース※1
- エラートレース※1
- メモリトレース※2
- メソッドトレース※1
- デバッグトレース※1，※3

注※1

ファイルに出力されます。

注※2

CUP.NET があらかじめ用意している System.String 型の配列に格納されます。

注※3

常に Client .NET のメモリ内に取得されます。また，Client .NET が提供するメソッドが例外を返した場合，標準出力に出力されることがあります。

2.9.1　トレースファイル

トレースファイルに指定できるオプションとその指定方法を次の表に示します。

表 2-9　トレースファイルに指定できるオプションとその指定方法

トレースファイル	オプション	指定方法
UAP トレース	ファイル出力先ディレクトリ	次のどちらかで指定します。 • SetUapTraceMode メソッドの TrcPath パラメタ • Client .NET 構成定義の<uapTrace>要素の path 属性
	トレースファイルのサイズ	次のどちらかで指定します。 • SetUapTraceMode メソッドの size パラメタ • Client .NET 構成定義の<uapTrace>要素の fileSize 属性
データトレース	ファイル出力先ディレクトリ	次のどちらかで指定します。 • SetDataTraceMode メソッドの TrcPath パラメタ • Client .NET 構成定義の<dataTrace>要素の path 属性
	トレースファイルのサイズ	次のどちらかで指定します。 • SetDataTraceMode メソッドの size パラメタ • Client .NET 構成定義の<dataTrace>要素の fileSize 属性

トレースファイル	オプション	指定方法
データトレース	データトレースの最大データ長	次のどちらかで指定します。 • SetDataTraceMode メソッドの DataSize パラメタ • Client .NET 構成定義の<dataTrace>要素の maxDataSize 属性
エラートレース	ファイル出力先ディレクトリ	次のどちらかで指定します。 • SetErrorTraceMode メソッドの TrcPath パラメタ • Client .NET 構成定義の<errTrace>要素の path 属性
	トレースファイルのサイズ	次のどちらかで指定します。 • SetErrorTraceMode メソッドの size パラメタ • Client .NET 構成定義の<errTrace>要素の fileSize 属性
メソッドトレース	ファイル出力先ディレクトリ	次のどちらかで指定します。 • SetMethodTraceMode メソッドの TrcPath パラメタ • Client .NET 構成定義の<methodTrace>要素の path 属性
	トレースファイルのサイズ	次のどちらかで指定します。 • SetMethodTraceMode メソッドの size パラメタ • Client .NET 構成定義の<methodTrace>要素の fileSize 属性
デバッグトレース	ファイル出力先ディレクトリ	Client .NET 構成定義の<debugTrace>要素の path 属性で指定します。
	デバッグトレースの最大ファイル数	Client .NET 構成定義の<debugTrace>要素の fileCount 属性で指定します。

トレース情報の出力ファイル名を次の表に示します。

表 2-10　トレース情報の出力ファイル名

トレースファイル	出力ファイル名
UAP トレース	dcuap1.trc dcuap2.trc
データトレース	dcdat1.trc dcdat2.trc
エラートレース	dcerr1.trc dcerr2.trc
メソッドトレース	dcmtd1.trc dcmtd2.trc
デバッグトレース	dcCltxxxxxxxxxxxxx.dmp （xxxxxxxxxxxxx：時刻から生成される数字）

指定したファイル出力先ディレクトリに出力ファイル名のファイルがない場合は，新規にトレースファイルを作成します。また，指定したディレクトリがない場合，および指定したディレクトリに書き込み権限がない場合は，トレースファイルを出力できません。

ファイルへの書き込みは追加モードで実行されます。二つのトレースファイルはラウンドロビン方式で使用されます（デバッグトレースのトレースファイルは除きます）。指定したトレースファイルのサイズを超えた書き込みが発生すると，ファイルの切り替えを行い，次のファイルの先頭から書き込みます。したがって，切り替えられる前のファイルは指定したサイズよりもファイルのサイズが大きくなる場合があります。

複数の CUP のトレースファイルの出力先に同じディレクトリを指定すると，次のような影響があるためプロセス単位，またはスレッド単位でトレースファイルの出力先を分けてください。

- RPC スループットが劣化するおそれがあります。
- 複数の CUP のトレース情報が混在し，トラブルシュートが困難になります。

2.9.2 UAP トレース

UAP トレースは，Client .NET が提供するメソッドの実行トレースとして，メソッドの開始時と終了時に次の情報を取得します。

- 日付
- 時刻
- 入り口と出口の種別
- 実行スレッドの情報
- 呼び出したメソッド名
- 発生した例外
- TP1/Server に伝播する性能検証用の識別情報※
- 指定した引数の情報
- 呼び出したメソッドに関係する情報
- 送受信したデータの内容

注※

一部のメソッドを実行したときに UAP トレースに出力される情報です。詳細については，「2.9.7 TP1/Server の性能検証用トレース」を参照してください。

送受信したデータの内容は，Call メソッドが受け渡したデータを取得します。なお，送受信データ長が 60 バイトを超える場合は，先頭の 60 バイトを取得します。クライアントスタブを使用している場合は，スタブが変換した送信データ，およびサーバから受信したデータが記録されます。

UAP トレースは，UAP および Client .NET のメソッド発行履歴を取得できるため，UAP のテストおよびデバッグに有効です。また，障害発生時の問題発生個所の特定や切り分けをする場合にも使用できます。

UAP トレース情報の出力ファイル名，および指定できるオプションとその指定方法については，「2.9.1 トレースファイル」を参照してください。

UAP トレースファイルの出力形式を次に示します。

```
 yyyy/mm/dd hh:mm:ss.uuu  Location = lll ThreadInfo = ttt
MethodName = nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
Exception  = eeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
PrfInfo    = xxxxxxxxxxxxxxxxxxxxxxxxxx
[Information]
iii
iii
iii
 :
 :
```

yyyy：UAP トレース取得年

mm：UAP トレース取得月

dd：UAP トレース取得日

hh:mm:ss.uuu：UAP トレース取得時間

lll：UAP トレース取得場所

入り口の場合は「In」，出口の場合は「Out」が出力されます。

ttt：UAP トレースを取得したスレッドの情報

次の形式で取得されます。

プロセス名[プロセス ID]，スレッド名[スレッド ID]

nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn：呼び出したメソッドの名前

eeeeeeeeeeeeeeeeeeeeeeeeeeeeeee：発生した例外

メソッドの入り口で取得されるトレースの場合，または例外が発生しなかった場合は出力されません。

xxxxxxxxxxxxxxxxxxxxxxxxxx：TP1/Server に伝播する性能検証用の識別情報

iii：メソッドの呼び出し情報

各メソッドの呼び出し情報を，以降の表に示します。

表 2-11　AcceptNotification メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	サーバからの通知メッセージを格納する領域の長さ	inf_len	inf_len 引数で指定したサーバからの通知メッセージを格納する領域の長さ
	サーバからの通知メッセージを受信するポート番号	port	port 引数で指定したサーバからの通知メッセージを受信するポート番号
	タイムアウト値	timeout	timeout 引数で指定したタイムアウト値
出口	サーバからの通知メッセージ	inf	inf 引数に格納したサーバからの通知メッセージ
	サーバからの通知メッセージの長さ	inf_len	inf_len 引数に格納したサーバからの通知メッセージの長さ
	サーバのホスト名	hostname	hostname 引数に格納したサーバのホスト名
	サーバのノード識別子	nodeid	nodeid 引数に格納したサーバのノード識別子

表 2-12　AcceptNotificationChained メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	サーバからの通知メッセージを格納する領域の長さ	inf_len	inf_len 引数で指定したサーバからの通知メッセージを格納する領域の長さ
	タイムアウト値	timeout	timeout 引数で指定したタイムアウト値
出口	サーバからの通知メッセージ	inf	inf 引数に格納したサーバからの通知メッセージ
	サーバからの通知メッセージの長さ	inf_len	inf_len 引数に格納したサーバからの通知メッセージの長さ
	サーバのホスト名	hostname	hostname 引数に格納したサーバのホスト名
	サーバのノード識別子	nodeid	nodeid 引数に格納したサーバのノード識別子

表 2-13　Begin メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	トランザクションブランチ限界経過時間	expireTime	Client .NET 構成定義の<transaction>要素の expireTime 属性で指定した値。 値を指定していない場合，-1 が設定されます。
	トランザクションブランチ CPU 監視時間	cpuTime	Client .NET 構成定義の<transaction>要素の cpuTime 属性で指定した値。 値を指定していない場合，-1 が設定されます。
	トランザクションブランチの監視時間の対象	expireSuspend	Client .NET 構成定義の<transaction>要素の expireSuspend 属性で指定した値。 値を指定していない場合，0 が設定されます。 • all：89 • async：78

取得場所	項目	文字列	内容
入り口	トランザクションブランチの監視時間の対象	expireSuspend	• none：70
	統計情報取得項目	statistics	Client .NET 構成定義の<transaction>要素の statistics 属性で指定した値。 値を指定していない場合，-1 が設定されます。 • base：0x80000000 • executiontime：0x40000000 • cputime：0x20000000 • nothing：0
	トランザクション最適化項目	optimize	Client .NET 構成定義の<transaction>要素の optimize 属性で指定した値。 値を指定していない場合，-1 が設定されます。 • base：0x00000003 • asyncprepare：0x00000004
	トランザクション同期点処理時の最大通信待ち時間	internalWatchTime	Client .NET 構成定義の<transaction>要素の internalWatchTime 属性で指定した値。 値を指定していない場合，-1 が設定されます。
	ロールバック情報取得の値	rollbackInfo	Client .NET 構成定義の<transaction>要素の rollbackInfo 属性で指定した値。 値を指定していない場合，-1 が設定されます。 • no：0 • self：0x00000001 • remote：0x00000002 • all：0x00000003
	トランザクションブランチ最大実行可能時間の値	limitTime	Client .NET 構成定義の<transaction>要素の limitTime 属性で指定した値。 値を指定していない場合，-1 が設定されます。
	ロールバック完了通知の値	rollbackResponse	Client .NET 構成定義の<transaction>要素の rollbackResponse 属性で指定した値。 値を指定していない場合，-1 が設定されます。 • true • false
	UAP 障害時の同期点処理方式の値	recoveryType	Client .NET 構成定義の<transaction>要素の recoveryType 属性で指定した値。 値を指定していない場合，-1 が設定されます。 • type1：0x00000001 • type2：0x00000002 • type3：0x00000004
出口	性能検証用トレースの識別情報	PrfInfo	性能検証用トレースの識別情報

表 2-14　Call メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	サービスグループ名	group	呼び出すサービスのサービスグループ名
	サービス名	service	呼び出すサービス名
	送信データ長	in_len	送信するデータの長さ
	送信データ	in_data	送信するデータ
	受信データ長	out_len	受信するデータの長さ
	オプションフラグ	flags	Call メソッドの flags パラメタに指定した値。 次のどれかの値が設定されます。 • DCNOFLAGS：0x00000000 • DCRPC_NOREPLY：0x00000001 • DCRPC_CHAINED：0x00000004 トランザクション中は次のフラグが設定される場合があります。 • DCRPC_TPNOTRAN：0x00000020
	最大応答待ち時間	watchTime	SetRpcWatchTime メソッドで指定された最大応答待ち時間
	rap サーバ通信遅延時間	delay	SetRapDelay メソッドで指定された rap サーバ通信遅延時間
出口※	受信データ長	out_len	受信した応答の長さ
	受信データ	out_data	受信したデータ
	性能検証用トレースの識別情報	PrfInfo	性能検証用トレースの識別情報

注※

非応答型サービス要求の場合，出口情報は性能検証用トレースの識別情報だけです。

表 2-15　CallTo メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	接続先ホスト名	ConnectionHost	DCRpcBindTbl クラスで指定されたホスト名
	接続先ポート番号	ConnectionPort	DCRpcBindTbl クラスで指定されたポート番号
	通信プロトコルフラグ	Protocol	CallTo メソッドの通信プロトコルフラグ • DCRPC_SCDPORT：0x00000002
	サービスグループ名	group	呼び出すサービスのサービスグループ名
	サービス名	service	呼び出すサービス名
	送信データ長	in_len	送信するデータの長さ

取得場所	項目	文字列	内容
入り口	送信データ	in_data	送信するデータ
	受信データ長	out_len	受信するデータの長さ
	オプションフラグ	flags	CallTo メソッドの flags パラメタに指定した値。 次のどちらかの値が設定されます。 • DCNOFLAGS：0x00000000 • DCRPC_NOREPLY：0x00000001
	最大応答待ち時間	watchTime	SetRpcWatchTime メソッドで指定された最大応答待ち時間
	rap サーバ通信遅延時間	delay	SetRapDelay メソッドで指定された rap サーバ通信遅延時間
出口※	受信データ長	out_len	受信した応答の長さ
	受信データ	out_data	受信したデータ
	性能検証用トレースの識別情報	PrfInfo	性能検証用トレースの識別情報

注※

非応答型サービス要求の場合，出口情報は性能検証用トレースの識別情報だけです。

表 2-16　CancelNotification メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	CUP.NET に通知するメッセージ	inf	inf 引数で指定した CUP.NET に通知するメッセージ
	CUP.NET に通知するメッセージの長さ	inf_len	inf_len 引数で指定した CUP.NET に通知するメッセージの長さ
	一方通知受信待ち状態の CUP.NET のホスト名	hostname	hostname 引数で指定した一方通知受信待ち状態の CUP.NET のホスト名
	一方通知受信待ち状態の CUP.NET のポート番号	port	port 引数で指定した一方通知受信待ち状態の CUP.NET のポート番号
出口	なし	－	－

（凡例）

－：該当しません。

表 2-17　CloseConnection メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	最大応答待ち時間	watchTime	SetRpcWatchTime メソッドで指定された最大応答待ち時間
出口	性能検証用トレースの識別情報	PrfInfo	性能検証用トレースの識別情報

表 2-18　Commit メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	なし	−	−
出口	性能検証用トレースの識別情報	PrfInfo	性能検証用トレースの識別情報

（凡例）

−：該当しません。

表 2-19　CommitChained メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	なし	−	−
出口	性能検証用トレースの識別情報	PrfInfo	性能検証用トレースの識別情報

（凡例）

−：該当しません。

表 2-20　GetTransactionID メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	なし	−	−
出口	トランザクショングローバル識別子	trngid	メソッドが正常終了の場合だけ取得されます。
	トランザクションブランチ識別子	trnbid	メソッドが正常終了の場合だけ取得されます。

（凡例）

−：該当しません。

表 2-21　GetTransactionInfo メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	なし	−	−
出口	トランザクション中フラグ	isTransaction	• トランザクション中である：True • トランザクション中ではない：False

（凡例）

−：該当しません。

表 2-22　OpenConnection メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	接続先ホスト名	host	host パラメタで指定された rap リスナーまたはファイアウォールのホスト名

取得場所	項目	文字列	内容
入り口	接続先ポート番号	port	port パラメタで指定された rap リスナーまたはファイアウォールのポート番号
	rap サーバ問い合わせ間隔最大時間	inquireTime	SetRapInquireTime メソッドで指定された rap サーバ問い合わせ間隔最大時間
	最大応答待ち時間	watchTime	SetRpcWatchTime メソッドで指定された最大応答待ち時間
出口	性能検証用トレースの識別情報	PrfInfo	性能検証用トレースの識別情報

表 2-23　OpenNotification メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	サーバからの通知メッセージを受信するポート番号	port	port 引数で指定したサーバからの通知メッセージを受信するポート番号
出口	なし	－	－

（凡例）

－：該当しません。

表 2-24　OpenRpc メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	プロファイル ID	profileId	profileId パラメタで指定された，使用するプロファイル ID
出口	プロファイル ID	profileId	profileId パラメタで指定された，使用するプロファイル ID

表 2-25　ReceiveAssembledMessage メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	受信メッセージの最大待ち時間	timeout	受信メッセージの最大待ち時間
	オプションフラグ	flags	ReceiveAssembledMessage メソッドの flags パラメタで指定した値。 次のどちらかの値が設定されます。 • DCNOFLAGS：0x00000000 • DCCLT_RCV_CLOSE：0x00000002
出口	受信データ長	recvleng	受信した応答の長さ
	受信データ	buff	受信したデータ

表 2-26　Rollback メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	なし	−	−
出口	性能検証用トレースの識別情報	PrfInfo	性能検証用トレースの識別情報

（凡例）

−：該当しません。

表 2-27　RollbackChained メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	なし	−	−
出口	性能検証用トレースの識別情報	PrfInfo	性能検証用トレースの識別情報

（凡例）

−：該当しません。

表 2-28　SendAssembledMessage メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	送信データ	buff	送信するデータ
	送信データ長	sendleng	送信データの長さ
	接続先ホスト名	hostname	hostname パラメタで指定した通信相手のホスト名
	接続先ポート番号	portnum	portnum パラメタで指定した通信相手のポート番号
	送信メッセージの最大待ち時間	timeout	送信メッセージの最大待ち時間
	オプションフラグ	flags	SendAssembledMessage メソッドの flags パラメタで指定した値。 次のどちらかの値が設定されます。 • DCNOFLAGS：0x00000000 • DCCLT_SND_CLOSE：0x00000001
出口	なし	−	−

（凡例）

−：該当しません。

表 2-29　SetConnectInformation メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	DCCM3 論理端末の論理端末名	inf	inf パラメタで指定された DCCM3 論理端末の論理端末名
	端末識別情報長	inf_len	inf_len パラメタで指定された端末識別情報長
出口	なし	−	−

（凡例）

－：該当しません。

表 2-30　SetDataTraceMode メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	データトレースを出力するディレクトリ	TrcPath	TrcPath 引数で指定したデータトレースを出力するディレクトリ
	出力するデータトレースファイルのサイズ	size	size 引数で指定した出力するデータトレースファイルのサイズ
	データトレースに出力するデータサイズ	DataSize	DataSize 引数で指定したデータトレースに出力するデータサイズ
	データトレースの取得可否フラグ	flag	flag 引数で指定したデータトレースの取得可否フラグ
出口	なし	－	－

（凡例）

－：該当しません。

表 2-31　SetErrorTraceMode メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	エラートレースを出力するディレクトリ	TrcPath	TrcPath 引数で指定したエラートレースを出力するディレクトリ
	出力するエラートレースファイルのサイズ	size	size 引数で指定した出力するエラートレースファイルのサイズ
	エラートレースの取得可否フラグ	flag	flag 引数で指定したエラートレースの取得可否フラグ
出口	なし	－	－

（凡例）

－：該当しません。

表 2-32　SetExtendLevel メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	拡張機能のレベル	flags	Client .NET の機能の拡張レベル
出口	なし	－	－

（凡例）

－：該当しません。

表 2-33　SetMethodTraceMode メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	メソッドトレースを出力するディレクトリ	TrcPath	TrcPath 引数で指定したメソッドトレースを出力するディレクトリ

取得場所	項目	文字列	内容
入り口	出力するメソッドトレースファイルのサイズ	size	size 引数で指定した出力するメソッドトレースファイルのサイズ
	メソッドトレースの取得可否フラグ	flag	flag 引数で指定したメソッドトレースの取得可否フラグ
出口	なし	−	−

(凡例)

−：該当しません。

表 2-34　SetRapDelay メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	rap サーバ通信遅延時間	sec	rap サーバ通信遅延時間
出口	なし	−	−

(凡例)

−：該当しません。

表 2-35　SetRapInquireTime メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	rap サーバ問い合わせ間隔最大時間	sec	rap サーバ問い合わせ間隔最大時間
出口	なし	−	−

(凡例)

−：該当しません。

表 2-36　SetRpcExtend メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	拡張機能のレベル	extendoption	RPC 機能の拡張レベル。 次のどれかの値が設定されます。 • DCRPC_SCD_LOAD_PRIORITY：0x00000008 • DCRPC_WATCHTIMINHERIT：0x00000010 • DCRPC_RAP_AUTOCONNECT：0x00000020 • DCRPC_WATCHTIMRPCINHERIT：0x00000040
出口	なし	−	−

(凡例)

−：該当しません。

表 2-37　SetRpcWatchTime メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	最大応答待ち時間	sec	最大応答待ち時間
出口	なし	−	−

（凡例）

−：該当しません。

表 2-38　SetTP1Server メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	接続先ホスト名	host	host パラメタで指定されたホスト名
	接続先ポート番号	port	port パラメタで指定されたポート番号
出口	なし	−	−

（凡例）

−：該当しません。

表 2-39　SetUapTraceMode メソッドの呼び出し情報

取得場所	項目	文字列	内容
入り口	UAP トレースを出力するディレクトリ	TrcPath	TrcPath 引数で指定した UAP トレースを出力するディレクトリ
	出力する UAP トレースファイルのサイズ	size	size 引数で指定した出力する UAP トレースファイルのサイズ
	UAP トレースの取得可否フラグ	flag	flag 引数で指定した UAP トレースの取得可否フラグ
出口	なし	−	−

（凡例）

−：該当しません。

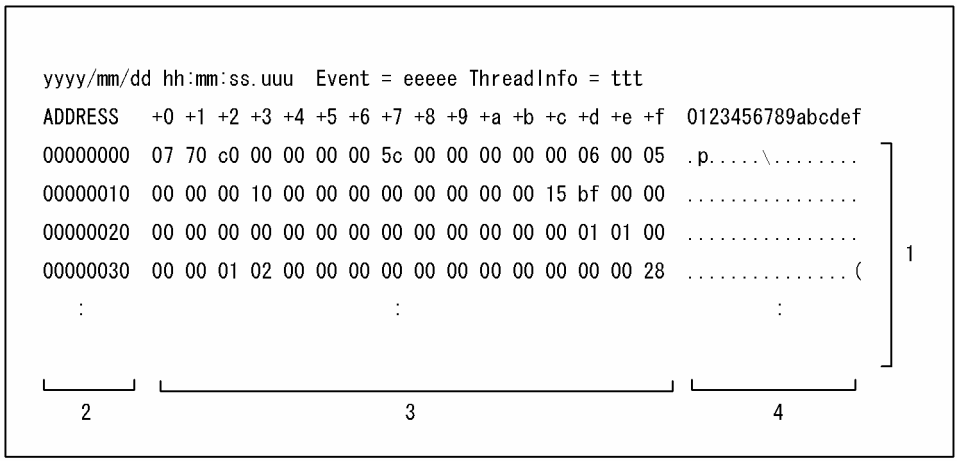
2.9.3　データトレース

データトレースは，CUP.NET と OpenTP1 間の送受信メッセージの内容を取得します。

データトレースは，UAP テスト時の RPC メッセージのデータ形式を確認する場合などに使用します。データ不正などの障害が発生した場合のメッセージ内容の検証にも使用できます。

データトレース情報の出力ファイル名，および指定できるオプションとその指定方法については，「2.9.1 トレースファイル」を参照してください。

データトレースファイルの出力形式を次に示します。



1. CUP.NET と OpenTP1 間の送受信メッセージのデータトレース
2. データトレースのロケーション
3. データトレースの 16 進表示のデータ
4. データトレースの文字表示のデータ

yyyy：データトレース取得年

mm：データトレース取得月

dd：データトレース取得日

hh:mm:ss.uuu：データトレース取得時間

eeeee：データトレース取得項目

送信の場合は「Send」，受信の場合は「Receive」が出力されます。

ttt：データトレースを取得したスレッドの情報

次の形式で取得されます。

```
プロセス名[プロセスID]，スレッド名[スレッドID]
```

2.9.4 エラートレース，メモリトレース

エラートレースは，Client .NET が検知した障害をメッセージ形式でファイルに記録します。メソッド実行中に障害が発生した場合，その原因は例外として報告されます。例外だけでは原因を特定できない場合，このファイルに出力された詳細なエラー情報を参照すると，通信障害などの障害発生時の原因を特定することが容易になります。エラートレースは，本番運用でも常に取得することをお勧めします。

メモリトレースは，エラートレースと同じ内容をメモリ上で System.String 型の配列に格納します。エラートレースのファイルに書き込みができない環境（ランタイムホスト）で，Client .NET が提供するクラスライブラリを動作させる場合は，メモリトレースを使用します。取得したメモリトレースの情報は，アプリケーション側で必要に応じて表示および記録ができるようにしておいてください。

エラートレース情報の出力ファイル名，および指定できるオプションとその指定方法については，「2.9.1 トレースファイル」を参照してください。

エラートレースファイルの出力形式，またはメモリトレースの System.String 型の配列への格納形式を次に示します。

```
(ttt) yyyy/mm/dd hh:mm:ss.uuu ee....ee
```

ttt：エラートレース，またはメモリトレースを取得したスレッドの情報

次の形式で取得されます。

プロセス名[プロセス ID]，スレッド名[スレッド ID]

yyyy：エラートレース，またはメモリトレース取得年

mm：エラートレース，またはメモリトレース取得月

dd：エラートレース，またはメモリトレース取得日

hh:mm:ss.uuu：エラートレース，またはメモリトレース取得時間

ee....ee：メッセージ

記録されるメッセージについては，マニュアル「OpenTP1 メッセージ」を参照してください。

2.9.5 メソッドトレース

メソッドトレースは，Client .NET の保守用のトレースで，内部動作が記録されます。

メソッドトレース情報の出力ファイル名，および指定できるオプションとその指定方法については，「2.9.1 トレースファイル」を参照してください。

障害発生時の原因を究明するために，メソッドトレースの取得が必要になる場合があります。保守員の指示に従ってください。

2.9.6 デバッグトレース

デバッグトレースは，Client .NET の保守用のトレースで，障害発生時の直前の内部動作が記録されます。

デバッグトレース情報は，常にメモリ内に取得し，Client .NET が提供するメソッドが例外を返した場合に，バイナリ形式のデータがファイルに出力されます。ファイルに保存できないときは，編集されたテキスト形式のデータが標準出力に出力されます。

デバッグトレースファイルの最大ファイルサイズは 1 メガバイトです。デバッグトレースの出力時にファイルサイズが 1 メガバイトを超える場合は，ファイルの情報を 0 バイトにクリアしてから出力されます。

障害発生時に調査を依頼する場合は，デバッグトレースを必ず送付してください。

2.9.7　TP1/Server の性能検証用トレース

TP1/Server の性能検証用トレース（prf トレース）とは，TP1/Server 上で動作する各種サービスの主なイベントのトレース情報です。この性能検証用トレースは，性能検証の効率，およびトラブルシュートの効率を向上させることが目的のトレース情報です。

Client .NET では，TP1/Server の性能検証用トレースに Client .NET が設定した性能検証用の識別情報を出力できます。この性能検証用の識別情報は，Client .NET のトレースにも出力されます。これによって，次の利点があります。

- Client .NET のメソッドの実行時間（UAP トレースによって取得）と，TP1/Server のサービスの実行時間（性能検証用トレースによって取得）との照合ができます。
- 障害が発生したときに，処理がどこまで到達したかを知ることができます。

(1)　TP1/Server への性能検証用の識別情報の伝播

TP1/Server へ性能検証用の識別情報を伝播する場合は，<tp1Server>要素の prfInfoSend 属性に true を指定します。TP1/Server のバージョンが 03-03 以降の場合に，性能検証用の識別情報を伝播できます。また，バージョン 07-02 以降の TP1/Server に対して，リモート API 機能を使用した RPC を行う場合は，スケジューラに送信する RPC メッセージ中への識別情報（IP アドレスなど）を付加できます。

注意事項

スケジューラダイレクト機能を使用した RPC でサービス要求先の TP1/Server のバージョンが 03-03 未満である場合，RPC を行ったときに，SPP のサービス関数に不正なデータを渡す可能性があります。したがって，この場合，TP1/Server 上へ性能検証用の識別情報を伝播しないでください（<tp1Server>要素の prfInfoSend 属性に false を指定してください）。

なお，ネームサービス機能を使用した RPC でサービス要求先の TP1/Server のバージョンが 03-03 未満である場合，TP1/Server 上へ性能検証用の識別情報を伝播するように設定しても，この設定は無視されるため性能検証用の識別情報は伝播できません。

(2)　TP1/Server と Client .NET とのトレースの照合

TP1/Server に対して，ネームサービスを使用した RPC，およびスケジューラダイレクト機能を使用した RPC を行う場合は，スケジューラに送信する RPC メッセージ中に，TP1Client クラスのインスタンスごとにほぼ一意となる識別情報（IP アドレスなど）を付加できます。付加した情報は，Client .NET のトレースに出力されます。また，これと同じ情報は TP1/Server の性能検証用トレースにも出力されます。

この Client .NET のトレースと，TP1/Server の性能検証用トレースとを照合することによって，Client .NET と TP1/Server との間の，一連の処理の流れを知ることができます。ただし，TP1/Server に対して，リモート API 機能を使用した RPC を行う場合は，スケジューラに送信する RPC メッセージ中への識別情報（IP アドレスなど）の付加はできません。

(3) 性能検証用の識別情報

Client .NET の性能検証用の識別情報は，Client .NET のトレース，および TP1/Server の性能検証用トレースに出力されます。出力例を次に示します。

(a) 性能検証用の識別情報として取得する情報

ノード ID：_N[bb]（4 バイトの英数字）

bb：ランダムな 2 文字の英数字（数字（0～9）またはアルファベット（A～Z，a～z））

ルート通信通番：[xxxxxxxx]（4 バイトの 16 進表示のデータ）

xxxxxxxx：IP アドレス

RPC 通信通番：[yyyy][zzzz]（4 バイトの 16 進表示のデータ）

yyyy：ランダムな 2 バイトの 16 進数字

zzzz：通信通番（該当する API の呼び出しごとにインクリメントされます）

(b) Client .NET のトレースファイルの出力例

UAP トレースファイル（取得ポイントは，TP1Client クラスの Call メソッドの出口）

```
2005/11/11 14:02:38.631  Location = Out  ThreadInfo = Test[3244],[2156]
MethodName = TP1Client.Call
Exception  =
PrfInfo    = _N[bb]/0x[xxxxxxxx]/0x[yyyy][zzzz]
[Information]
```

エラートレースファイル（取得ポイントは，TP1Client クラスの Call メソッドの出口）

```
(Test[2700],[2860]) 2005/11/11 15:06:00.108 KFCA32302-E 例外が発生しました。 保守情報=-76
32, 例外=ErrClientTimedOutException(PrfInfo=_N[bb]/0x[xxxxxxxx]/0x[yyyy][zzzz]), メソッド
=TP1Client.Call(TP1Client.Call)
```

(c) TP1/Server の性能検証用トレースファイルの出力例

```
PRF: Rec Node: smpl Run-ID: 0x43742533 Process: 1952       Trace: 1             Event: 0x200
2 Time: 2005/11/11 14:02:36 583.000.000 Server-name: _scd
Rc: ********** Client:  - 0x[yyyy][zzzz]     Server: **** Root: _N[bb] - 0x[xxxxxxxx]  Svc-
Grp: echo_svg                          Svc: echo1                              Trn: *
```

(4) トレース情報の取得ポイント

次に示すメソッドを実行するたびに，UAP トレースファイルの出口にトレース情報が取得されます。

- Begin
- Call
- CallTo
- CloseConnection
- Commit
- CommitChained
- OpenConnection
- Rollback
- RollbackChained

Call メソッド，および CallTo メソッドを実行した場合，<rpc>要素の use 属性が scd または nam で，かつ<rpc>要素の prfInfoSend 属性が true のときは，TP1/Server へ性能検証用の識別情報が伝播されます。

2.10　データ圧縮機能

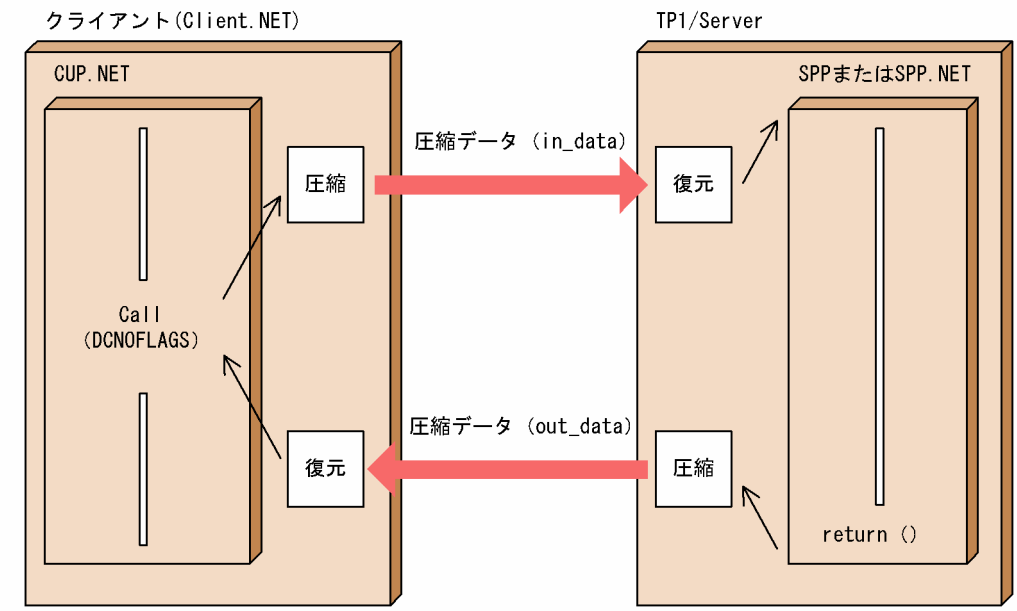
データ圧縮機能を使用すると，RPC によってネットワーク上に送り出されるユーザデータを圧縮できます。これによってネットワーク上に送り出されるパケット数を削減し，ネットワークの混雑を緩和できます。

データ圧縮機能を使用するかどうかは，Client .NET 構成定義の<rpc>要素の compressData 属性で指定します。

この機能を使用すると，Client .NET は CUP.NET から実行する Call メソッドで設定する入力パラメタ（in_data）の値を，圧縮効果がある場合は圧縮してネットワーク上に送り出します。その問い合わせに対し，SPP または SPP.NET から返される応答（out_data）の値も TP1/Server で圧縮されてネットワーク上に送り出されてきます。その応答を受け取った Client .NET は，圧縮データを復元して CUP.NET に渡します。

データ圧縮機能の概要を次の図に示します。

図 2-22　データ圧縮機能の概要



この機能は，サービス要求先の TP1/Server が 03-03 以降※の場合に使用できます。また，RPC の通信形態，およびすべての RPC インタフェースで使用できます。ただし，サービス要求先の TP1/Server のバージョンによって，次の点が異なります。

- TP1/Server のバージョンが 03-06 未満の場合

  入力パラメタの値が圧縮されていなかった場合，返される応答の値に圧縮効果があっても，応答の値は TP1/Server で圧縮されません。
- TP1/Server のバージョンが 03-06 以降の場合

入力パラメタの値が圧縮されていなかった場合でも，返される応答の値に圧縮効果があれば，応答の値は TP1/Server で圧縮されます。

注※

リモート API 機能を使用した RPC の場合は，TP1/Server のバージョンが 03-05 以降のときにデータ圧縮機能を使用できます。

2.10.1 データ圧縮機能の効果

データ圧縮機能の効果は，ユーザデータの内容に依存します。ユーザデータ中に連続した同じ文字が多く現れる場合には効果がありますが，ユーザデータの内容によっては，ほとんど効果がない場合もあります。

また，データ圧縮／復元のためのオーバヘッドが掛かるため，データ圧縮による効果とのバランスを考慮し，事前に性能評価をしてからデータ圧縮機能の使用を検討してください。

2.10.2 データ圧縮機能を使用するときの注意事項

データ圧縮機能を使用する場合の注意事項を次に示します。

- スケジューラダイレクト機能を使用した RPC でサービス要求先の TP1/Server のバージョンが 03-03 未満である場合，RPC を行ったときに，SPP のサービス関数に不正なデータを渡す可能性があります。したがって，この場合，TP1/Server に対して，データ圧縮機能を使用してサービス要求しないでください（<rpc>要素の compressData 属性に false を指定してください）。

  なお，ネームサービス機能を使用した RPC でサービス要求先の TP1/Server のバージョンが 03-03 未満である場合，データ圧縮機能を使用するように設定しても，この設定は無視されるためデータ圧縮機能は使用できません。
- 圧縮前より圧縮後のデータの方がデータが大きくなってしまう場合やサービス要求先の TP1/Server がデータ圧縮機能をサポートしていない場合は，Client .NET のエラートレースファイルに KFCA32306-W メッセージが出力され処理は続行されます。この場合，データは圧縮されません。
- DCCM3 の論理端末に常設コネクションを確立しサービスを要求する場合，データ圧縮機能は使用できません。
- Call メソッドがエラーを返した場合のサービスの応答（out_data）は保証しません。
- RPC メッセージの最大長（Call メソッドで送受信できるユーザメッセージの最大長）は，データ圧縮機能の使用に関係なく，常に<rpc>要素の maxMessageSize 属性で指定した値か，または省略時仮定値（1 メガバイト）です。

2.11 MSDTC 連携機能

MSDTC 連携機能を使用すると，MSDTC を利用するほかのリソースと OpenTP1 が使用するリソースとの間で分散トランザクション連携ができます。

Client .NET から MSDTC 連携機能を使用する場合は，Connector .NET が必要です。Client .NET 単体の場合は MSDTC 機能は使用できません。詳細については，マニュアル「TP1/Connector for .NET Framework 使用の手引」の MSDTC 連携機能に関する記述を参照してください。

MSDTC 連携機能を使用する場合に Client .NET で使用できる機能を次の表に示します。

表 2-40 MSDTC 連携機能使用時に Client .NET で使用できる機能

分類	Client .NET の機能	使用可否
Client .NET の通信方式	リモート API 機能	○
	スケジューラダイレクト機能	×
	ネームサービス	×
コネクトモード	オートコネクトモード	○
	非オートコネクトモード	×
RPC の呼び出し形態	同期応答型 RPC	○
	非応答型 RPC	○
	連鎖型 RPC	○
	トランザクションを引き継がない RPC	○

（凡例）

○：使用できます。

×：使用できません。

なお，Client .NET 構成定義の<tp1Server>要素に指定できる窓口となる TP1/Server は，プロファイル ID ごとに一つだけ指定してください。複数指定した場合，トランザクションを正常に制御できません。

2.12　環境設定

Client .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。

2.12.1　インストール後の環境設定

運用コマンドを使用する場合は，PATH 環境変数に〈インストールディレクトリ〉¥bin を追加してください。

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。

〈インストールディレクトリ〉¥bin¥Hitachi.OpenTP1.Client.dll

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。

2.12.2　セキュリティポリシーの設定

Client .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。

Client .NET の機能を利用するためには，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が実行できません。

(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）

- 〈インストールディレクトリ〉¥bin¥if2cstub.exe
- 〈インストールディレクトリ〉¥bin¥spp2cstub.exe
- 〈インストールディレクトリ〉¥bin¥Hitachi.OpenTP1.Client.dll
- CUP.NET のアセンブリ，および Client .NET のアセンブリを参照するすべてのアセンブリ

(2)　必要なアクセス許可

運用コマンドには完全信頼を指定する必要があります。Hitachi.OpenTP1.Client.dll には次の表に示すアクセス許可がすべて必要です。

表 2-41　Hitachi.OpenTP1.Client.dll に必要なアクセス許可

アクセス許可の種類	アクセス許可
セキュリティ	コードの実行を有効にする
リフレクション	無制限
ファイル IO	無制限
ソケットアクセス	無制限
DNS	無制限

2.12.3　メッセージの出力先

Client .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。クラスライブラリが出力するメッセージは，エラートレースに出力されます。

エラートレースの出力先は，構成ファイルのプロパティで指定するか，または SetErrorTraceMode メソッドで指定できます。

エラートレースについては，「2.9.4　エラートレース，メモリトレース」を参照してください。

2.13 送信元ホスト指定機能

Client .NET では，RPC や TCP/IP 通信機能でのコネクション確立要求時に，CUP.NET の送信元ホストを指定できます。この機能を，**送信元ホスト指定機能**といいます。

CUP.NET が動作するホスト上に，複数のネットワークアダプタが接続されている場合，CUP.NET がこれらのコネクション確立要求時に使用する送信元ホストは，TCP/IP の制御で任意に決まります。しかし，送信元ホスト指定機能を使用すると，次のコネクション確立要求時の送信元ホストを固定できます。

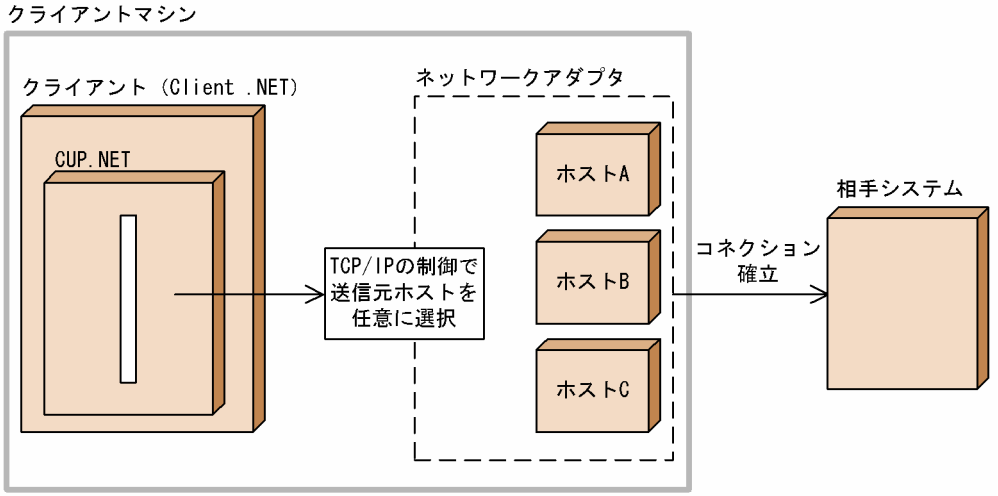
- スケジューラダイレクト機能を使用した RPC
- ネームサービスを使用した RPC
- リモート API 機能を使用した RPC
- 通信先を指定した RPC
- TCP/IP 通信機能

Client .NET 構成定義の<socket>要素の cupSendHost 属性を指定することで，CUP.NET の送信元ホストを指定できます。

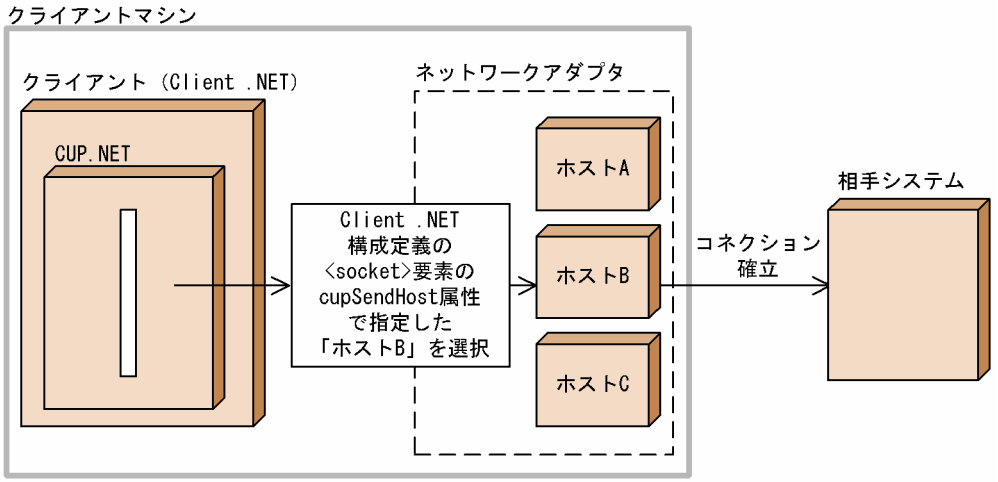
送信元ホスト指定機能を使用しない場合と使用する場合について，次の図に示します。

図 2-23　送信元ホスト指定機能を使用しない場合と使用する場合

●送信元ホスト指定機能を使用しない場合



●送信元ホスト指定機能を使用する場合



2.14　受信ポート固定機能

Client .NET では，スケジューラダイレクト機能を使用した RPC の受信ポート，およびネームサービスを使用した RPC の受信ポートを固定できます。この機能を，**受信ポート固定機能**といいます。

この機能は，TP1/Server から Client .NET への RPC の応答通信をする場合で，TP1/Server と Client .NET との間に設置したファイアウォールで Client .NET の受信ポートにだけ通知を許可するようにフィルタリングしたいときに使用します。

この機能を使用する場合は，Client .NET 構成定義の<rpc>要素の cupRecvPort 属性を指定します。

受信ポート固定機能を使用しない場合と使用する場合について説明します。

2.14.1　受信ポート固定機能を使用しない場合

RPC の応答通信で，受信ポートに対するフィルタリングはありません。OS が，Client .NET の RPC の受信ポートとして不定のポートを自動的に割り当てます。

受信ポート固定機能を使用しない場合について，次の図に示します。

図 2-24　受信ポート固定機能を使用しない場合（スケジューラダイレクト機能を使用した RPC）

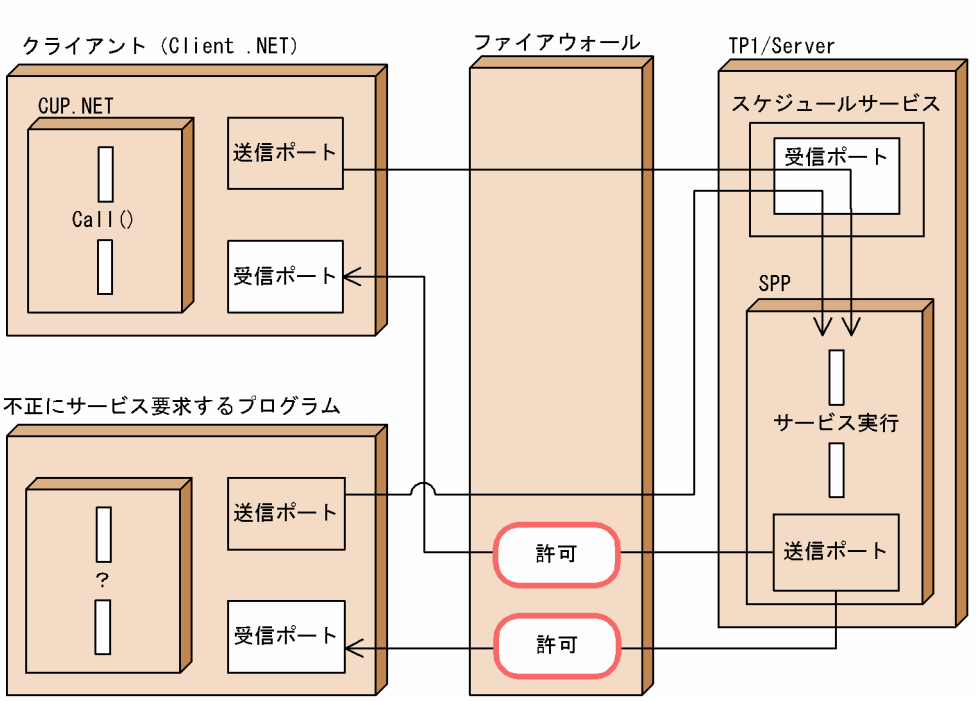
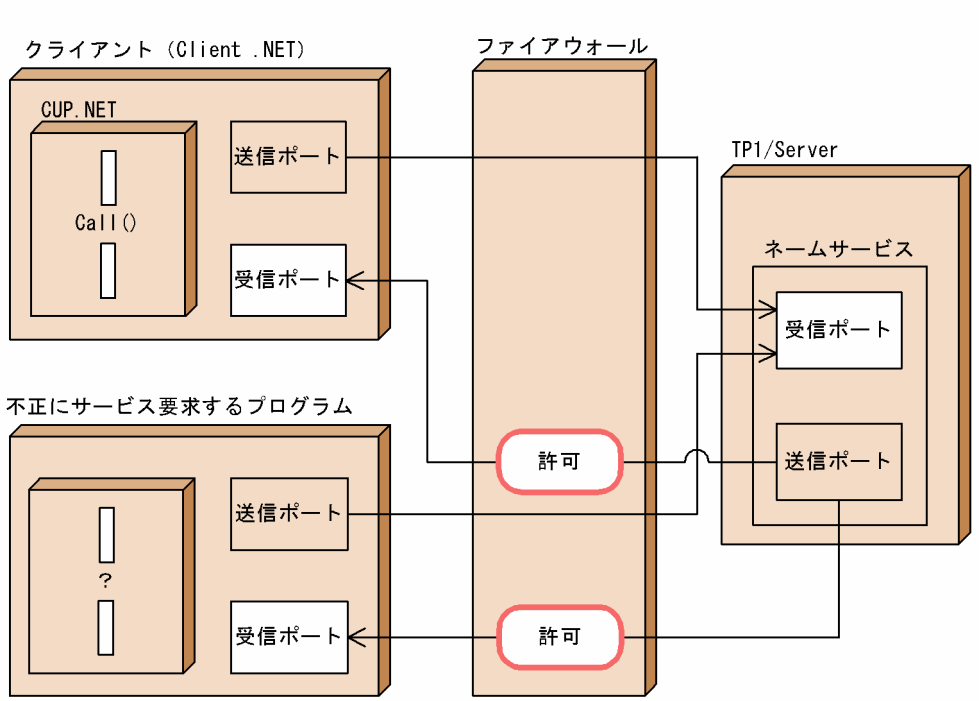


図 2-25　受信ポート固定機能を使用しない場合（ネームサービスを使用した RPC）



2.14.2　受信ポート固定機能を使用する場合

RPC への応答通信で許可する受信ポートを，Client .NET 構成定義の<rpc>要素の cupRecvPort 属性で指定したポートとし，それ以外はフィルタリング対象にします。このため，不正なサービス要求に対する TP1/Server からの応答通信を，ファイアウォールでフィルタリングできます。

受信ポート固定機能を使用する場合について，次の図に示します。

図 2-26　受信ポート固定機能を使用する場合（スケジューラダイレクト機能を使用した RPC）

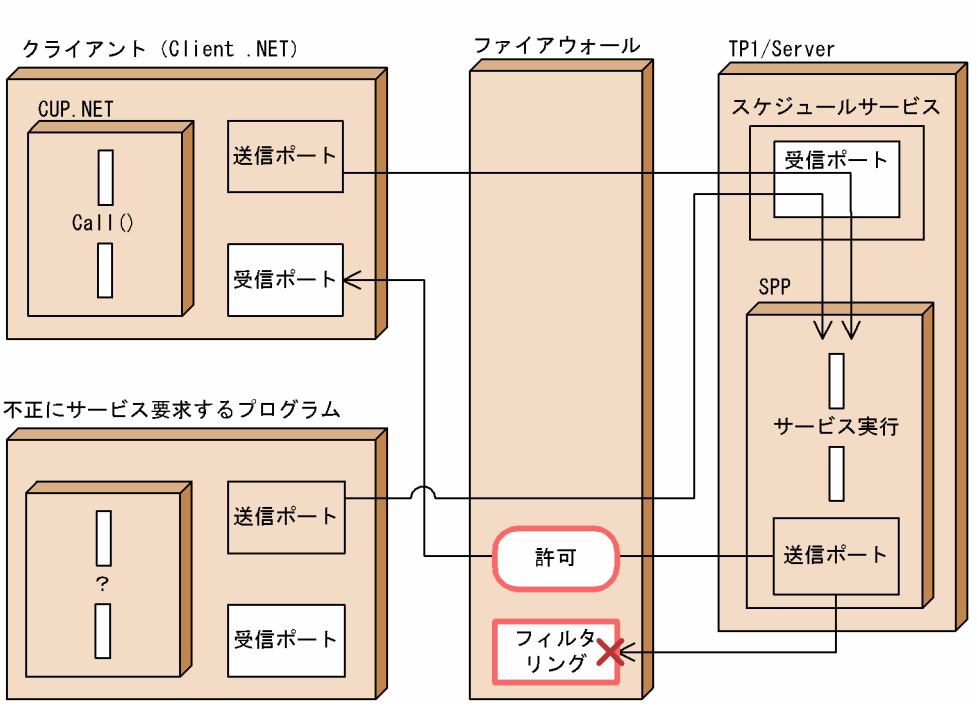
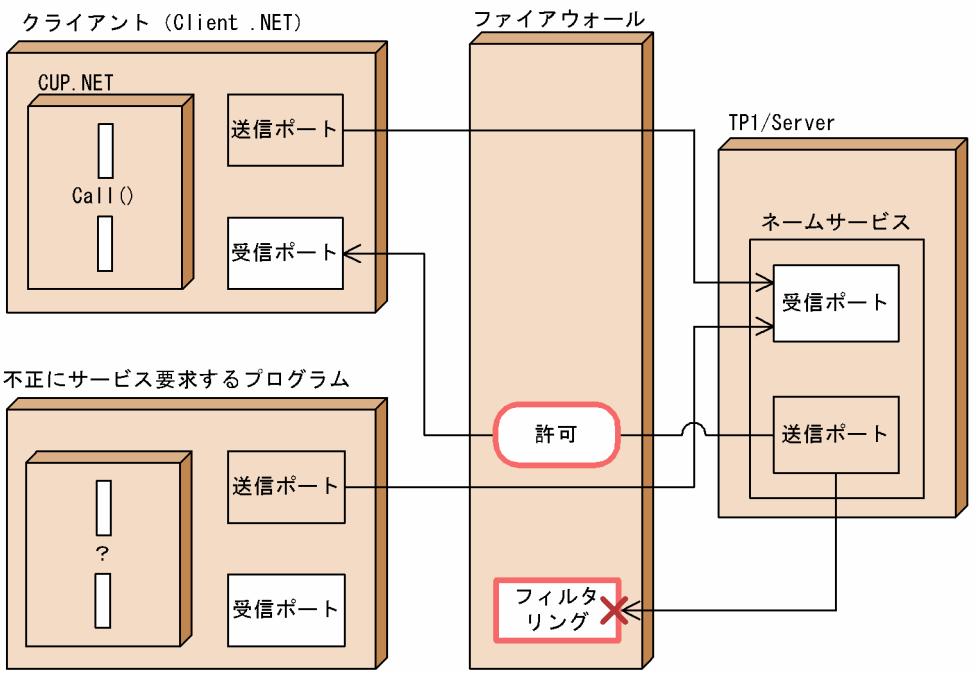


図 2-27　受信ポート固定機能を使用する場合（ネームサービスを使用した RPC）



2.15　ホスト切り替え機能

サーバダウンなどの理由で，指定したホストと一時的に通信ができなくなった場合，通信先ホストを 1 つしか指定していないと，Client .NET のメソッドはエラーリターンします。この場合，通信先ホストの指定を変更したあと，CUP.NET を再起動する必要があります。

通信先ホストを複数指定しておくと，Client .NET は，ホストを切り替えて，別のホストに通信を試みます。これによって，ユーザは通信先ホストの切り替えを意識することなく業務を継続させることができます。

なお，障害の内容によっては，ホストを切り替えない場合があります。

2.15.1　TP1/Server との通信時

(1)　通信先ホスト複数指定個所

Client .NET 構成定義の<tp1Server>要素を 2 個以上指定します。

(2)　通信先ホスト選択契機

OpenRpc メソッド実行時に選択します。

接続障害発生時は，Call メソッド実行時にホストを切り替えます。

表 2-42　ホスト切り替えの契機

実行メソッド	ホスト切り替えの契機
Call	Client .NET 構成定義の<rpc>要素が use="nam"の場合 　ネームサービスとの接続（サービス情報検索）に失敗したときにホストを切り替えます。 Client .NET 構成定義の<rpc>要素が use="scd"の場合 　スケジュールサービスとの接続に失敗したときにホストを切り替えます。

(3)　通信先ホスト選択方法

Client .NET 構成定義の<nameService>要素の randomSelect 属性の指定値によって，選択方法が異なります。randomSelect 属性の指定を省略，または false を指定した場合，<tp1Server>要素の host 属性に指定された順番に通信先ホストを選択します。

この指定の場合，Client .NET 構成定義の<tp1Server>要素の host 属性に同一の指定を行った複数の CUP.NET から一斉にサービスを要求した場合，1 つのホストに負荷が集中してしまいます。randomSelect 属性に true を指定すると，初回 Call()メソッド呼び出し時にランダムに通信先ホストを選択するため，サーバの負荷を分散させることができます。

表 2-43　通信先ホストの選択方法

randomSelect	通信先ホストの選択方法	
	初回選択時 (OpenRpc メソッド実行時)	接続障害発生時
false	先頭に指定されたホストを選択します。	障害となったホストの次に指定されたホストを選択します。
true	複数指定されたホストの中からランダムに選択します。	障害となったホストの次に指定されたホストを選択します。

ホスト切り替えは，通信が成功するまで繰り返し行います。指定されたすべてのホストとの接続に失敗した場合，メソッドはエラーリターンします。

次メソッド発行時，再度ホスト切り替えが発生した場合，前回障害となったホストも切り替えるホストの対象となります。

ホスト切り替えによって決定したホストは，次の契機で再びホストの選択，または切り替えが発生するまで，通信先ホストとなります。

- OpenRpc メソッドによるホスト選択
- SetTP1Server メソッドによるホスト選択
- Call メソッドの接続障害で再びホスト切り替えが発生

例 1) 次の指定の場合

```
<tp1Server host="hostA">
<tp1Server host="hostB">
<tp1Server host="hostC">
<nameService randomSelect="false">

1.  OpenRpcメソッド発行
      先頭に指定されたhostAを選択
        +-----+
        |hostA|  hostB   hostC
        +-----+
      hostAとの接続に成功

2.  Callメソッド発行
      hostAとの接続障害発生
      障害となったhostAの次に指定されたhostBを選択
              +-----+
          ×   |hostB|  hostC
              +-----+
      hostBとの接続障害発生
      障害となったhostBの次に指定されたhostCを選択
                    +-----+
          ×     ×   |hostC|
                    +-----+
      hostCとの接続に成功
```

```
3.  Callメソッド発行
      hostCとの接続障害発生
      障害となったhostCの次に指定されたhostAを選択
        +-----+
        |hostA|  hostB    ×
        +-----+
      hostAとの接続障害発生
      障害となったhostAの次に指定されたhostBを選択
                +-----+
            ×   |hostB|   ×
                +-----+
      hostBとの接続障害発生
      選択対象ホストがなくなりメソッドはエラーリターン

4.  Callメソッド発行
      3.で最後に接続障害が発生したホストhostBを選択
                +-----+
         hostA  |hostB|  hostC
                +-----+

5.  CloseRpcメソッド発行

6.  OpenRpcメソッド発行
      先頭に指定されたhostAを選択
```

例 2) 次の指定の場合

```
<tp1Server host="hostA">
<tp1Server host="hostB">
<tp1Server host="hostC">
<nameService randomSelect="true">

1.  OpenRpcメソッド発行
      hostA, hostB, hostCからランダム選択
                +-----+
         hostA  |hostB|  hostC
                +-----+
      hostBとの接続に成功

2.  Callメソッド発行
      hostBとの接続障害発生
      障害となったhostBの次に指定されたhostCを選択
                        +-----+
         hostA    ×     |hostC|
                        +-----+
      hostCとの接続障害発生
      障害となったhostCの次に指定されたhostAを選択
        +-----+
        |hostA|   ×      ×
        +-----+
      hostAとの接続に成功

3.  Callメソッド発行
      hostAとの接続障害発生
      障害となったhostAの次に指定されたhostBを選択
                +-----+
            ×   |hostB|  hostC
                +-----+
```

```
      hostBとの接続障害発生
      hostCを選択
                          +-----+
          ×       ×       |hostC|
                          +-----+
      hostCとの接続障害発生
      選択対象ホストがなくなりメソッドはエラーリターン

4.  Callメソッド発行
      3.で最後に接続障害が発生したホストhostCを選択
                          +-----+
         hostA    hostB   |hostC|
                          +-----+
5.  CloseRpcメソッド発行

6.  OpenRpc発行メソッド
      hostA, hostB, hostCからランダム選択
```

(4) 窓口ホストの切り替え契機の変更

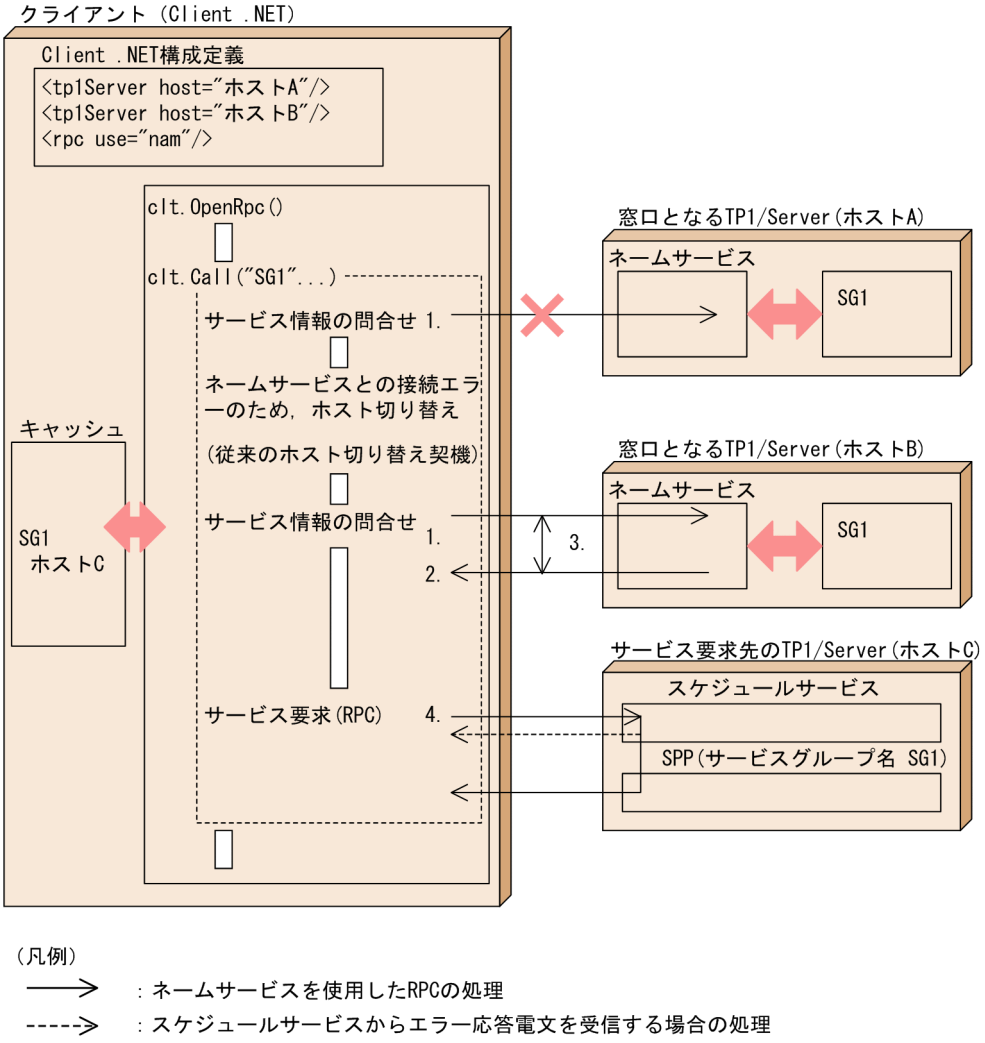
Client .NET は，Client .NET 構成定義に窓口となるホストを複数指定しておくと，指定したホストと一時的に通信できなくなった場合に，ホストを切り替えて別のホストに通信を試みます。

ネームサービスを使用した RPC の場合，従来の Client .NET は，ネームサービスとの接続エラーのときだけ窓口ホストを切り替えていました。

この機能を使用すると，下図の 1.で示す従来のホスト切り替え契機に加え，下図の 2.から 4.で示す範囲のエラー種別も，ホスト切り替え契機にできます。これによって，業務形態に応じて最適なホスト切り替え契機を選択できます。

窓口ホストの切り替え契機とするエラー種別は，Client .NET 構成定義 <nameService>要素の hostChangeMode 属性に指定します。

図 2-28　窓口ホストの切り替え契機であるエラー範囲



Call メソッドでは次の内部処理を実行しており，1.から 4.で示す範囲のエラーが発生したときを窓口ホストの切り替え契機にできます。

<サービス情報の問い合わせ>

1. ネームサービスとの接続エラー
2. ネームサービスからのエラー応答電文受信
3. ネームサービスからの応答電文受信タイムアウト※

注※

タイムアウトの監視時間を<nameService>要素の sysWatchTime 属性で指定します。

<サービス要求（RPC）>

4. スケジュールサービスからのエラー応答電文受信

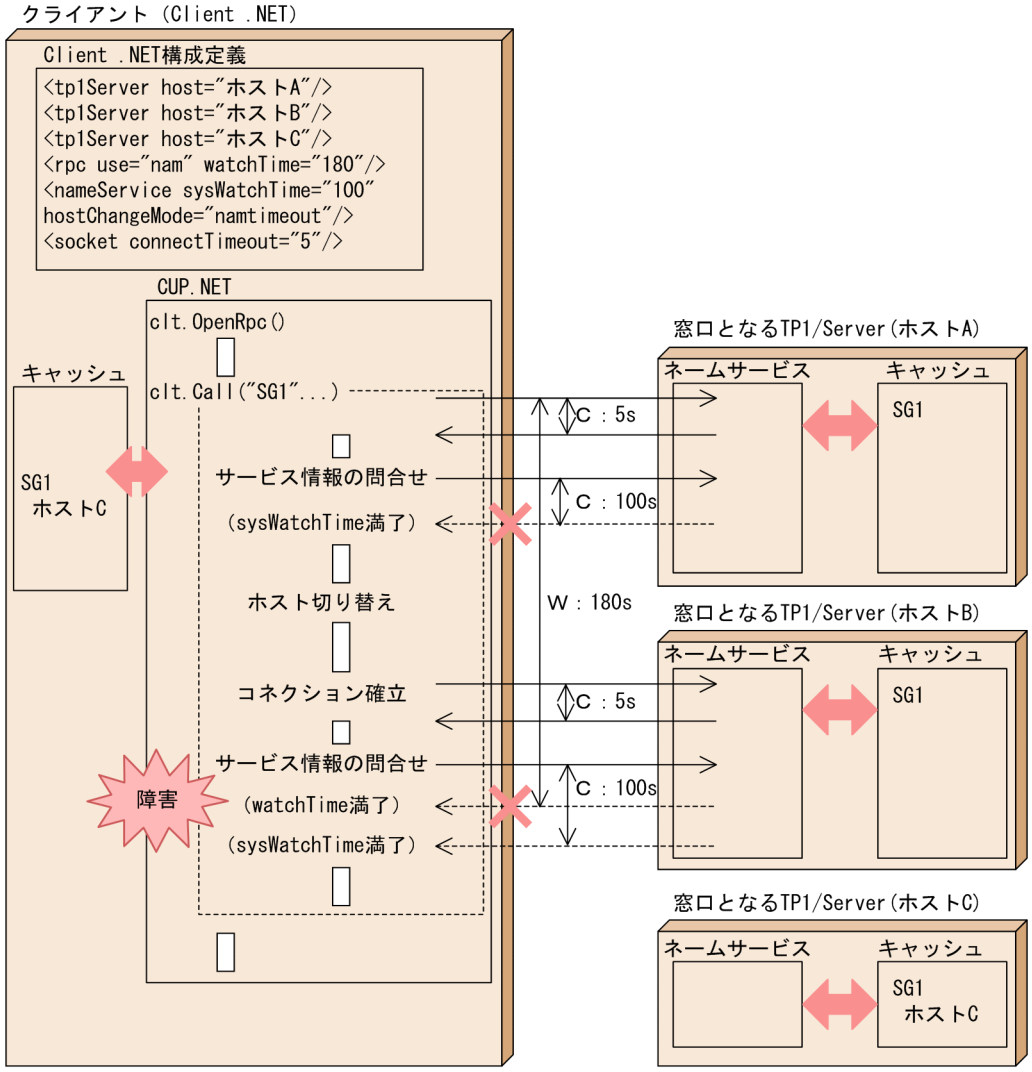
エラーの詳細を次の表に示します。

表 2-44　窓口となるホストの切り替え契機であるエラーの詳細

内部処理	エラー範囲	エラーの詳細	<nameService>要素の hostChangeMode 指定値	
サービス情報の問い合わせ	1. ネームサービスとの接続エラー	• ネームサービスとのコネクション確立に失敗した場合 • ネームサービスへサービス情報の問い合わせ電文送信に失敗した場合 • ネームサービスからのコネクション確立待ちで通信障害が発生した場合 • ネームサービスからのサービス情報電文受信で通信障害が発生した場合	省略	all
	2. ネームサービスからのエラー応答電文受信	• ネームサービスとの接続に成功したが，TP1/Server 側で障害が発生し，ネームサービスからエラー応答電文を受信した場合	namrcv	
	3. ネームサービスからの応答電文受信タイムアウト	• <nameService>要素の sysWatchTime 属性で指定した時間を過ぎても，ネームサービスから応答が返らない場合	namtimeout	
サービス要求（RPC）	4. スケジュールサービスからのエラー応答電文受信	• スケジュールサービスが開始処理中，終了処理中，または障害発生のため，スケジュールサービスからエラー応答電文を受信した場合	scdrcv	

<nameService>要素の hostChangeMode 属性に"namtimeout"を指定したときの，<rpc>要素の watchTime 属性，<nameService>要素の sysWatchTime 属性，および<socket>要素の connectTimeout 属性との関係性を次の図に示します。

図 2-29　各種タイマ監視機能の関係性



（凡例）

W：watchTimeの監視時間　S：sysWatchTimeの監視時間　C：connectTimeoutの監視時間

——→　：ネームサービスを使用したRPCの処理

----→　：スケジュールサービスからエラー応答電文を受信する場合の処理

1. "ホスト A"のネームサービスとのコネクション確立が成功し，サービス情報を問い合わせます。
2. 100 秒経過してもネームサービスから応答が返ってこなかった場合，"ホスト B"へ窓口ホストを切り替えます。
3. "ホスト B"のネームサービスとのコネクション確立が成功し，サービス情報を問い合わせます。
4. "ホスト B"のネームサービスからの応答待ちで 100 秒経過する前に，Call()メソッドの最大応答待ち時間である 180 秒が経過すると，"ホスト C"へホストを切り替えません。その時点で CUP.NET の Call()メソッドは ErrClientTimedOutException を返します。

(5)　応答電文受信障害時のキャッシュリフレッシュ

CUP.NET が同じサービスに対する 2 回目以降の RPC をする際は，Client .NET のキャッシュを参照します。キャッシュにサービス情報が格納されていれば，キャッシュ内のサービス情報を使用し，RPC を行

います。このとき，CUP.NET がサービス要求先の TP1/Server からの応答電文を受信する際に障害が発生すると，障害が発生したサービス情報がキャッシュに残るケースがあります。この状態になると，3 回目以降の RPC でも，キャッシュに残っているサービス情報を使うため，再び通信障害やタイムアウトが発生する可能性があります。

この現象は，次の 1.～3.のどれかを実施するまで発生し続けます。

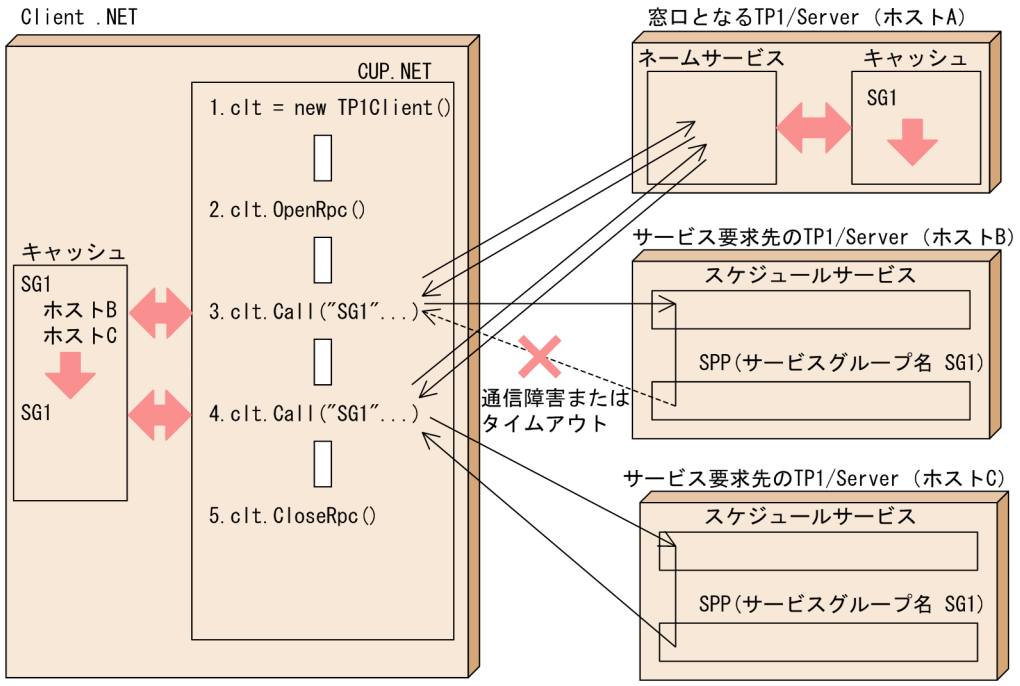
1. CloseRpc メソッドを呼び出して，RPC 環境を解放
2. キャッシュの有効期限満了によるキャッシュの更新
3. サーバ側のエラー要因の復旧

前述の 1.～3.のどれかを実施しなくても，応答電文受信障害時にキャッシュをリフレッシュすることで，上記の現象が回避できます。

この機能はネームサービスを使用した RPC で，Client .NET 構成定義<rpc>要素の extendOpt 属性に 00000001 を指定した場合に使用できます。

この機能を使用して 2 回目以降の RPC を正常に実行する例を次の図に示します。

図 2-30　ネームサービスを使用した RPC の障害発生時の流れ



1. TP1Client クラスのインスタンスを作成します。
2. OpenRpc メソッドを呼び出して，CUP.NET の RPC 環境を初期化します。
3. Call メソッドを呼び出して，該当する SPP にサービスを要求しますが，応答電文受信時に障害が発生します。Call メソッドでは，次の内部処理を実行します。
   - ホスト A へサービス情報の問い合わせを行い，Client .NET のキャッシュに格納します。
   - キャッシュに格納したサービス情報を使用して RPC を行います。

- RPC 応答受信時に通信障害または，タイムアウトが発生したため，Client .NET のキャッシュからサービス情報を削除します。

4. Call メソッドを呼び出して，該当する SPP にサービスを要求します。
   Call メソッドでは，次の内部処理を実行します。
   - ホスト A へサービス情報の問い合わせを行い，最新のサービス情報で Client .NET のキャッシュをリフレッシュします。
   - Client .NET のキャッシュからリフレッシュしたサービス情報を取得します。
   - 取得したサービス情報を使用して RPC を行います。
5. CloseRpc メソッドを呼び出して，RPC 環境を解放します。

2.15.2 rap リスナー，DCCM3 論理端末との通信時

(1) 通信先ホスト複数指定個所

Client .NET 構成定義の<tp1Server>要素の host 属性に指定します。

(2) 通信先ホスト選択契機

OpenRpc メソッド実行時に選択します。

接続障害が発生している場合は，メソッド実行時にホストを切り替えます。

表 2-45 ホスト切り替えの契機

実行メソッド	ホスト切り替えの契機
Call※	Client .NET 構成定義の<rapService>要素が autoConnect="true"，かつ<rpc>要素が use="rap"の場合 rap リスナー，DCCM3 論理端末との接続に失敗したときにホストを切り替えます。
Begin OpenConnection（引数なし）	Client .NET 構成定義の<rapService>要素が autoConnect="false"，かつ<rpc>要素が use="rap"の場合 rap リスナー，DCCM3 論理端末との接続に失敗したときにホストを切り替えます。

注※
トランザクションの範囲内で実行した場合は該当しません。

(3) 通信先ホスト選択方法

Client .NET 構成定義の<nameService>要素の randomSelect 属性の指定値によって，選択方法が異なります。randomSelect 属性の指定を省略，または false を指定した場合，<tp1Server>要素の host 属性に指定された順番に通信先ホストを選択します。

この指定の場合，Client .NET 構成定義の<tp1Server>要素の host 属性に同一の指定を行った複数の CUP.NET から一斉にサービスを要求した場合，1 つのホストに負荷が集中してしまいます。randomSelect 属性に true を指定すると，OpenRpc 実行時にランダムに通信先ホストを選択するため，サーバの負荷を分散させることができます。

表 2-46　通信先ホストの選択方法

randomSelect	通信先ホストの選択方法	
	初回選択時 (OpenRpc メソッド実行時)	接続障害発生時
false	先頭に指定されたホストを選択します。	障害となったホストの次に指定されたホストを選択します。
true	複数指定されたホストの中からランダムに選択します。	障害となったホストの次に指定されたホストを選択します。

ホスト切り替えは，通信が成功するまで繰り返し行います。指定されたすべてのホストとの接続に失敗した場合，メソッドはエラーリターンします。

次メソッド発行時，再度ホスト切り替えが発生した場合，前回障害となったホストも切り替えるホストの対象となります。

ホスト切り替えによって決定したホストは，次の契機で再びホストの選択，または切り替えが発生するまで，通信先ホストとなります。

- OpenRpc メソッドによるホスト選択
- SetTP1Server メソッドによるホスト選択
- OpenConnection，Begin，Call メソッドの接続障害で再びホスト切り替えが発生

例 1) 次の指定の場合

```
<tp1Server host="hostA">
<tp1Server host="hostB">
<tp1Server host="hostC">
<nameService randomSelect="false">

1.  OpenRpcメソッド発行
      先頭に指定されたhostAを選択
        +-----+
        |hostA|  hostB   hostC
        +-----+
      hostAとの接続に成功
2.  Callメソッド発行
      hostAとの接続障害発生
      障害となったhostAの次に指定されたhostBを選択
              +-----+
        ×     |hostB|  hostC
              +-----+
      hostBとの接続障害発生
      障害となったhostBの次に指定されたhostCを選択
```

```
                        +-----+
          ×      ×      |hostC|
                        +-----+
      hostCとの接続に成功
3.  Call発行メソッド
      hostCとの接続障害発生
      障害となったhostCの次に指定されたhostAを選択
        +-----+
        |hostA|  hostB    ×
        +-----+
      hostAとの接続障害発生
      障害となったhostAの次に指定されたhostBを選択
                +-----+
          ×     |hostB|   ×
                +-----+
      hostBとの接続障害発生
      選択対象ホストがなくなりメソッドはエラーリターン
4.  Callメソッド発行
      3.で最後に接続障害が発生したホストhostBを選択
                +-----+
        hostA   |hostB|  hostC
                +-----+
5.  CloseRpcメソッド発行
6.  OpenRpcメソッド発行
      先頭に指定されたhostAを選択
```

例 2) 次の指定の場合

```
<tp1Server host="hostA">
<tp1Server host="hostB">
<tp1Server host="hostC">
<nameService randomSelect="true">

1.  OpenRpcメソッド発行
      hostA, hostB, hostCからランダム選択
                +-----+
          hostA |hostB|  hostC
                +-----+
      hostBとの接続に成功
2.  Callメソッド発行
      hostBとの接続障害発生
      障害となったhostBの次に指定されたhostCを選択
                        +-----+
          hostA    ×    |hostC|
                        +-----+
      hostCとの接続障害発生
      障害となったhostCの次に指定されたhostAを選択
        +-----+
        |hostA|    ×     ×
        +-----+
      hostAとの接続に成功
3.  Callメソッド発行
      hostAとの接続障害発生
      障害となったhostAの次に指定されたhostBを選択
                +-----+
          ×     |hostB|  hostC
                +-----+
      hostBとの接続障害発生
```

```
      hostCを選択
                          +-----+
          ×       ×       |hostC|
                          +-----+
      hostCとの接続障害発生
      選択対象ホストがなくなりメソッドはエラーリターン
4.  Callメソッド発行
      3.で最後に接続障害が発生したホストhostCを選択
                          +-----+
         hostA    hostB   |hostC|
                          +-----+
5.  CloseRpcメソッド発行
6.  OpenRpcメソッド発行
      hostA, hostB, hostCからランダム選択
```

3 構成定義

この章では，Client .NET で使用する構成定義について説明します。

構成ファイルの形式

Client .NET 構成定義は，アプリケーション構成ファイル，またはマシン構成ファイルにカスタム構成セクションとして記述します。

カスタム構成セクションの宣言

形式

```
<configSections>
  <section
    name="hitachi.opentp1.client"
    type="Hitachi.OpenTP1.Common.Util.ProfileSectionHandler,
          Hitachi.OpenTP1.Client,Version=7.0.0.0,Culture=neutral,
          PublicKeyToken=2440cf5f0d80c91c,Custom=null"/>
</configSections>
```

説明

カスタム構成セクション宣言の<section>要素は，<configSections>要素の子要素として，各アプリケーション構成ファイルに記述するか，マシン構成ファイルに記述します。マシン構成ファイルに記述した場合，各アプリケーション構成ファイルに記述する必要はありません。

カスタム構成セクション宣言は，必ず<hitachi.opentp1.client>要素よりも前に記述してください。

構成定義の記述

形式

```
<hitachi.opentp1.client>
  <common>
     〔〔<構成定義要素名〔〔属性名="属性値"〕…〕/>〕…〕
  </common>
   〔〔<profile id="プロファイルID">
     〔〔<構成定義要素名〔〔属性名="属性値"…〕…〕/>〕…〕
      </profile>〕…〕
</hitachi.opentp1.client>
```

説明

<hitachi.opentp1.client>要素の子要素として，<common>要素と<profile>要素が記述できます。<profile>要素は複数記述できます。<common>要素または<profile>要素の子要素として，各構成定義要素が記述できます。

有効になる要素について，次の場合に分けて説明します。

- OpenRpc メソッドを呼び出さない場合
- 引数なしの OpenRpc メソッドを呼び出す場合
- OpenRpc メソッドの引数でプロファイル ID を指定する場合

OpenRpc メソッドを呼び出さない場合

すべての構成定義要素はデフォルト値が有効になります。

引数なしの OpenRpc メソッドを呼び出す場合

<common>要素内に記述された構成定義要素が有効になります。これは，OpenRpc メソッドの引数でプロファイル ID に null（Nothing）または空文字列（""）を指定した場合も同様です。

OpenRpc メソッドの引数でプロファイル ID を指定する場合

- <common>要素内および<profile>要素内に，同じ構成定義要素が記述されている場合，または<profile>要素内にだけ該当する構成定義要素が記述されている場合，<profile>要素内に記述された構成定義要素が有効になります。
- <profile>要素内に該当する構成定義要素が記述されていない場合，<common>要素内に記述された構成定義要素が有効になります。

なお，<common>要素内または 1 つの<profile>要素内に，複数記述できない構成定義要素が複数記述されている場合は，最後に記述した構成定義要素が有効になります。

また，メソッドの呼び出しに必要な構成定義要素が記述されていない場合は，該当するメソッドを呼び出すときにエラーが発生します。

相手先のサーバ別，通信方式別など，カテゴリ別に<profile>要素を記述しておくと便利です。

外部ファイルでの構成定義の記述

形式

```
<hitachi.opentp1.client import="絶対パスの外部XMLファイル名">
</hitachi.opentp1.client>
```

外部ファイルの形式

```
<hitachi.opentp1.client>
  <common>
     〔〔<構成定義要素名〔〔属性名="属性値"〕…〕/>〕…〕
  </common>
   〔〔<profile id="プロファイルID">
     〔〔<構成定義要素名〔〔属性名="属性値"…〕…〕/>〕…〕
      </profile>〕…〕
</hitachi.opentp1.client>
```

説明

<hitachi.opentp1.client>要素の import 属性を指定すると，外部ファイルで定義した構成定義を読み込ませることができます。複数のアプリケーションで同じ構成定義を使用したい場合に指定します。

外部ファイルは，カスタム構成セクションの<hitachi.opentp1.client>要素をルート要素とする XML ファイルで指定します。また，import 属性には外部ファイルのパスを絶対パスで指定してください。

【外部ファイルの指定例】

c:¥MyApp¥tp1config.xml

注意事項

- 構成定義要素の属性値として指定する true および false には，大文字と小文字の区別はありません。
- 同一マシンで CUP.NET を複数実行する場合，CUP.NET ごとに Client .NET 構成定義を設定してください。次に示す Client .NET 構成定義は，プロセス単位，またはスレッド単位で CUP ごとに異なる値を指定する必要があります。
  - <tcpip>要素の recvPort 属性
  - <rpc>要素の cupRecvPort 属性
  - <errTrace>要素の path 属性
  - <methodTrace>要素の path 属性
  - <uapTrace>要素の path 属性
  - <dataTrace>要素の path 属性

hitachi.opentp1.client

形式

```
<hitachi.opentp1.client>
  <common>
    <構成定義要素名 属性名="属性値"/>
  </common>
  <profile id="プロファイルID">
    <構成定義要素名 属性名="属性値"/>
  </profile>
</hitachi.opentp1.client>
```

説明

Client .NET の構成ファイルの構成セクション名です。

<hitachi.opentp1.client>要素の子要素として，<common>要素と<profile>要素が指定できます。

<common>要素は省略できません。必ず<hitachi.opentp1.client>要素内に<common>要素を一つだけ指定してください。

<profile>要素は省略できます。

common

形式

```
<common>
  <構成定義要素名 属性名=”属性値”/>
</common>
```

説明

構成定義要素を指定します。

この要素内では構成定義要素の指定を省略できます。

また，この要素内にだけ指定できる構成定義要素，属性名，および属性値もあります。

この要素は省略できません。

profile

形式

```
<profile id=”プロファイルID”>
  <構成定義要素名 属性名=”属性値”/>
</profile>
```

説明

構成定義要素を指定します。

この要素内では構成定義要素の指定を省略できます。

この要素内で指定していない構成定義要素，属性名，および属性値については，<common>要素内で指定した構成定義要素，属性名，および属性値が有効になります。

この要素は省略できます。

属性

●id="プロファイル ID"　～〈文字列〉

プロファイルを一意に識別するための ID を指定します。

<profile>要素を指定した場合，この属性は省略できません。

tp1Server

形式

```
<tp1Server host="窓口となるOpenTP1のホスト名"
           port="窓口となるOpenTP1のポート番号"/>
```

説明

窓口となる OpenTP1 のホスト名およびポート番号を指定します。

窓口となる OpenTP1 を複数指定する場合は，この要素を複数指定してください。

サービス呼び出し（RPC）を rap リスナー経由で行う場合は，rap リスナーのホスト名，またはファイアウォールのホスト名を指定してください。

この要素は省略できません。必ず<common>要素内または<profile>要素内にこの要素を一つ以上指定してください。

属性

●host="窓口となる OpenTP1 のホスト名"　～〈文字列〉

窓口となる OpenTP1 のホスト名を指定します。

この属性は省略できません。

●port="窓口となる OpenTP1 のポート番号"　～〈符号なし整数〉((5001～65535))

窓口となる OpenTP1 のポート番号を指定します。

この属性は省略できます。省略した場合，<nameService>要素，<rapService>要素，または<scheduleService>要素の port 属性の値が有効になります。

記述例

窓口となる OpenTP1 を 2 つ指定する場合

```
<tp1Server host="10.210.208.13"
           port="10000"/>
<tp1Server host="10.210.208.14"
           port="10000"/>
```

rpc

形式

```
<rpc use="通信形態"
     watchTime="最大応答待ち時間"
     watchTimeNotification="true|false"
     maxMessageSize="RPCメッセージの最大長"
     compressData="true|false"
     prfInfoSend="true|false"
     cupRecvPort="CUP.NETの受信で使用するポート番号"
     useMultiScheduler="true|false"
     extendOpt="RPCサービスの機能拡張オプション"/>
```

説明

RPC に関する指定をします。

この要素は省略できません。必ず<common>要素内または<profile>要素内にこの要素を一つ以上指定してください。

属性

●use="通信形態"

通信形態を指定します。

この属性は省略できません。次のどれかを指定してください。

rap：rap サービスを使用

nam：ネームサービスを使用

scd：スケジューラサービスを使用

●watchTime="最大応答待ち時間"　～〈符号なし整数〉((0～65535))《180》(単位：秒)

応答型 RPC の場合に，CUP.NET から SPP.NET または SPP へサービス要求を送ってからサービスの応答が返るまでの待ち時間の最大値を指定します。

指定時間を過ぎても応答が返らない場合は，CUP.NET にエラーリターンします。0 を指定した場合は，応答を受信するまで待ち続けます。

この属性は省略できます。

●watchTimeNotification="true | false"　～《false》

CUP.NET の最大応答待ち時間をサーバ側に引き継ぐかどうかを指定します。

CUP.NET の最大応答待ち時間を引き継ぐと，CUP.NET がタイムアウトしているのにサーバ側でサービスが実行されることを防止できます。

この属性は省略できます。

true：サーバ側に CUP.NET の最大応答待ち時間を引き継ぎます。

false：サーバ側に CUP.NET の最大応答待ち時間を引き継ぎません。

●maxMessageSize="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)

TP1Client クラスの Call メソッドで送受信できるユーザメッセージの最大長を指定します。

Call メソッドの引数に，この属性で指定した値より大きな値を送信メッセージ長および受信メッセージ長として指定できません。

送信メッセージ長にこの属性の指定値よりも大きい値を指定した場合，Call メソッドは，ErrMessageTooBigException を返します。

送信メッセージ長にこの属性の指定値よりも小さい値を指定して，受信メッセージ長にこの属性の指定値よりも大きい値を指定した場合，Call メソッドは ErrInvalidArgsException を返します。

この属性に 1（デフォルト値）または 8（最大に拡張した値）を指定した場合，RPC メッセージの最大長は次のようになります。

- 1（デフォルト値）の場合：1 メガバイト＝ 1048576 バイト
- 8（最大に拡張した値）の場合：8 メガバイト＝ 8388608 バイト

この属性に 2 以上を指定した場合，ネームサービスを使用した RPC でネームサーバへの問い合わせをしたとき，RPC メッセージの最大長が拡張されたサービス情報を，要求先として使用します。

この属性は以下のメソッドにも有効です。詳細は「6. クラスリファレンス」の各メソッドの説明を参照してください。

- AcceptNotification メソッド
- AcceptNotificationChained メソッド
- CancelNotification メソッド
- CallTo メソッド

システム共通定義の rpc_max_message_size オペランドをサポートしていない TP1/Server（バージョン 06-02 より前）に対して，maxMessageSize 属性を指定して，次の機能を使用しないでください。使用した場合，通信先の TP1/Server ノードでエラーが発生します。

- 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

この属性は省略できます。

●compressData="true | false"　～《false》

データ圧縮機能を使用するかどうかを指定します。

compressData に true を指定した場合でも，サービス要求を受信する SPP のメッセージ格納バッファプール長（message_store_buflen オペランド）は，圧縮前のユーザデータ長でサイズを計算してください。

true：データ圧縮機能を使用します。

false：データ圧縮機能を使用しません。

この属性は省略できます。

●prfInfoSend="true | false"　～《true》

TP1/Server に対して，ネームサービスを使用した RPC，およびスケジューラダイレクト機能を使用した RPC を行う場合に，OpenTP1 の性能検証用トレースに出力する情報として Client .NET 内部で識別情報を付加するかどうかを指定します。

スケジューラダイレクト機能を使用した RPC でサービス要求先の TP1/Server のバージョンが 03-03 未満である場合は，SPP のサービス関数に不正なデータを渡す可能性があるので必ず false を指定してください。

なお，ネームサービス機能を使用した RPC でサービス要求先の TP1/Server のバージョンが 03-03 未満である場合，true を指定しても，この指定は無視されるため TP1/Server 上へ性能検証用の識別情報は伝播できません。

true：Client .NET 内部で識別情報（IP アドレスなど）を付加します。

false：Client .NET 内部で識別情報（IP アドレスなど）を付加しません。

この属性は省略できます。

●cupRecvPort="CUP.NET の受信で使用するポート番号"　～〈符号なし整数〉((5001～65535))

サーバからのメッセージを受信する CUP.NET のポート番号を指定します。

この属性で指定したポート番号は，次の機能を使用する場合に有効です。

- スケジューラダイレクト機能を使用した RPC の受信時（受信ポート）
- ネームサービスを使用した RPC の受信時（受信ポート）

この属性を省略すると，システムが任意に割り当てたポート番号を使用します。

同一マシン内で，複数のプロセス，または複数のスレッドを同時に実行する場合は，それぞれ異なるポート番号を指定してください。

指定できるポート番号でも，OS またはほかのプログラムで使用するポート番号は指定しないでください。指定した場合，応答データを正しく受信できないことがあります。なお，OS が使用するポート番号は，OS ごとに異なります。OS が使用するポート番号については，ご利用の OS の関連ドキュメントを参照してください。

●useMultiScheduler="true | false"　～《false》

マルチスケジューラ機能を使用するかどうかを指定します。

true：マルチスケジューラ機能を使用します。

false：マルチスケジューラ機能を使用しません。

マルチスケジューラ機能を使用すると，複数起動されたマルチスケジューラデーモンの中から，一つをランダムに選択することで，スケジューリングの負荷を軽減できます。この属性に true を指定した場合，サービス要求を送信するスケジューラデーモンは Client .NET 構成定義の指定によって異なります。次の Client .NET 構成定義も併せて参照してください。

- <rpc>要素の use 属性
- <tp1Server>要素の port 属性
- <scheduleService>要素の port 属性
- <scheduleService>要素の multiSchedulerCount 属性

この属性は，CallTo メソッド実行時は無効です。

この属性は省略できます。

●extendOpt="RPC サービスの機能拡張オプション"　～〈16 進整数〉((00000000～00000001))《00000000》

RPC サービスの機能拡張オプションを指定します。

00000000：RPC サービスの機能を拡張しません。

00000001：応答電文受信障害時のキャッシュリフレッシュ機能を使用します。

ネームサービスを使用した RPC で Call メソッドを呼び出した場合，サービス要求先の TP1/Server からの応答電文受信時に ErrNetDownException，または ErrTimedOutException が発生したときに，例外が発生したサービス要求先を RPC の宛先から除外します。同じサービスに対する 2 回目以降の RPC は，キャッシュにサービス情報がないため，ネームサービスから新しいサービス要求先を取得し，キャッシュをリフレッシュします。これによって，通信障害，またはタイムアウトが発生し続ける現象を回避できます。

この指定は，<rpc>要素の use 属性で nam を指定したときだけ有効です。

この属性は省略できます。

ErrTimedOutException は，クライアント側，またはサーバ側の高負荷など環境によって発生することもあり，必ずしも通信障害やサーバ障害ではない場合があります。

なお，サービス要求先の TP1/Server でサーバ障害や通信障害が発生すると，窓口となる TP1/Server へサービス情報の通知ができなくなるため，窓口となる TP1/Server のキャッシュには障害が発生した TP1/Server のサービス情報が残ったままになることがあります。この状態では，CUP.NET がネームサービスへの問い合わせをしても，障害が発生した TP1/Server のサービス情報を再度取得するため，キャッシュをリフレッシュできず，Call メソッドで例外が発生することがあります。そのため，この機能を使用する場合，TP1/Server 側でノード監視機能の使用をお勧めします。

また，ノード監視機能を使用した場合でも，監視タイミングによってはネームサービスのキャッシュが更新されていないことがあります。そのため，ErrNetDownException，または ErrTimedOutException が発生した場合，ネームサービス定義の name_audit_interval オペランドに指定した監視間隔時間の間，CUP.NET の RPC 呼び出し処理でリトライすることをお勧めします。

記述例

```
<rpc use="nam"
     watchTime="180"
     watchTimeNotification="false"
     maxMessageSize="1"
     compressData="false"
     prfInfoSend="true"
     cupRecvPort="6000"
     useMultiScheduler="false"
     extendOpt="00000001"/>
```

rapService

形式

```
<rapService port="rapリスナーのポート番号"
            autoConnect="true|false"
            delay="最大通信遅延時間"
            watchTimeInheritance="true|false"
            inquireTime="問い合わせ間隔最大時間"
            randomSelect="true|false"
            connectInformation="端末識別情報"
            inquireTimeCheck="true|false"/>
```

説明

rap サービスを使用した RPC に関する指定をします。

<rpc>要素の use 属性で rap を指定した場合，この要素は省略できません。その場合，必ず<common>要素内または<profile>要素内にこの要素を一つ以上指定してください。

属性

●port="rap リスナーのポート番号"　～〈符号なし整数〉((5001～65535))

クライアントが属している OpenTP1 の rap リスナーのポート番号またはファイアウォールのポート番号を指定します。

<tp1Server>要素の port 属性が省略された場合，この属性の値が有効になります。

この属性は省略できます。省略する場合，必ず<tp1Server>要素の port 属性の値を指定してください。

●autoConnect="true | false"　～《false》

CUP.NET と OpenTP1 の rap サーバとの間の常設コネクションを自動的に確立させるかどうかを指定します。

この属性は省略できます。

true：次回メソッド実行時に，常設コネクションが確立されていない場合は，自動的に常設コネクションを確立します。常設コネクション確立要求先は，<tp1Server>要素の host 属性に指定した rap リスナーです。

false：常設コネクションを自動的に確立させません。

●delay="最大通信遅延時間"　～〈符号なし整数〉((0～65535))《0》(単位：秒)

CUP.NET と OpenTP1 の rap サーバとの間の通信オーバヘッドを考慮し，rap サーバ側の応答監視をクライアント側よりも早く終わらせる場合に指定します。この属性を指定すると，rap サーバ側の監視を指

定した時間分だけ早く終了させ，クライアント側の監視時間のタイムアウトによるメッセージのすれ違いを防ぎます。

Client .NET の<rpc>要素の watchTime 属性に 0 を指定した場合，または SetDcwatchtim メソッドで 0 を指定した場合は，<rapService>要素の delay 属性に指定した値は無視されます。<rpc>要素の watchTime 属性に指定した値から<rapService>要素の delay 属性に指定した値を引いたときに，0 または負の値になった場合も，<rapService>要素の delay 属性に指定した値は無視され，1 が仮定されます。

●watchTimeInheritance="true | false"　～《false》

rap サービスを使用した RPC を行う場合に，CUP.NET の最大応答待ち時間を rap サーバ側に引き継ぐかどうかを指定します。

この属性は省略できます。

true：rap サーバ側に CUP.NET の最大応答待ち時間を引き継ぎます。

false：rap サーバ側に CUP.NET の最大応答待ち時間を引き継ぎません。

●inquireTime="問い合わせ間隔最大時間"　～〈符号なし整数〉((0～1048575))（単位：秒）

CUP.NET 実行プロセスが rap サーバに対して問い合わせを行ってから，次回の問い合わせをするまでの間隔の最大時間を指定します。問い合わせ間隔最大時間は，rap サーバで監視するタイマです。rap サーバでは，問い合わせ間隔を監視し，問い合わせ間隔最大時間を超えても rap サーバに問い合わせがない場合，常設コネクションを解放します。また，トランザクション内で問い合わせ間隔最大時間に達したことを検知した場合は，該当するトランザクションを強制的にロールバックします。

常設コネクションを解放しないで CUP.NET 実行プロセスが異常終了した場合は，この属性を指定することで rap サーバを有効活用できます。

この属性の値に 0 を指定した場合は，間隔を監視しません。

この属性を省略した場合は，rap サーバで指定された問い合わせ間隔最大時間が有効になります。

TP1/Client for .NET Framework のバージョン 07-03-07 以降では，この属性に 0 以外を指定し，オートコネクトモードで RPC を実行した場合，CUP.NET 実行プロセスでも問い合わせ間隔最大時間をチェックします。

対象となる RPC は，次の 3 種類です。

- Call メソッドを使用した RPC
- .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

CUP.NET 実行プロセスで問い合わせ間隔最大時間をチェックするかどうかは，<rapService>要素の inquireTimeCheck 属性の指定値に従います。

この属性とあわせて，inquireTimeCheck 属性も参照してください。

なお，rap サーバが常設コネクションを解放するタイミングと，CUP.NET 実行プロセスが RPC を実行するタイミングが重なった場合，RPC の実行に失敗して ErrNetDownAtClientException 例外を返すことがあります。

この現象を回避するため，<rapService>要素の次の属性を指定し，CUP.NET 実行プロセスで「問い合わせ間隔最大時間」をチェックするように設定してください。

- inquireTimeCheck 属性に true を指定
- inquireTime 属性に 0 以外を指定
- autoConnect 属性に true を指定

属性の詳細は，<rapService>要素の inquireTimeCheck，および autoConnect 属性を参照してください。

●randomSelect="true | false"　～《false》

窓口となる TP1/Server をランダムに選択するかどうかを指定します。

この属性は，窓口となる TP1/Server を複数指定した場合にだけ有効です。

この属性は省略できます。

true：窓口となる TP1/Server をランダムに選択します。

false：窓口となる TP1/Server をランダムに選択しません。

●connectInformation="端末識別情報"

端末識別情報として，DCCM3 論理端末の論理端末名称を EBCDIK コードで指定します。端末識別情報は，先頭に 0x を付け，0x の後ろに 128 文字までの 16 進数表記で指定します。先頭の 0x は 128 文字に含まれません。2 文字で 1 バイトの値として，64 バイト（128 文字）まで指定できます。ただし，DCCM3 側では先頭 8 バイト目までに指定した値だけが有効になるため，9 バイト目以降に指定された値は無視されます。

この属性に指定した値は，SetConnectInformation メソッドを呼び出すことで，動的に変更できます。この属性に指定した値は，SetConnectInformation メソッドを呼び出したあと，再び OpenRpc メソッドを呼び出すまで無視されます。

この属性は，Client .NET 構成定義の<tp1Server>要素の host 属性に DCCM3 論理端末のホスト名を，<tp1Server>要素の port 属性または<rapService>要素の port 属性に DCCM3 論理端末のポート番号を指定し，次のどちらかの方法で DCCM3 論理端末との常設コネクションを確立した場合に有効となります。

- OpenConnection メソッドを呼び出します。引数有りの OpenConnection メソッドの場合，引数 host に DCCM3 論理端末のホスト名，引数 port に DCCM3 論理端末のポート番号を指定します。
- Client .NET 構成定義の<rapService>要素の autoConnect 属性に true を指定し，Call メソッドを呼び出します。

この属性は省略できます。省略した場合，DCCM3 論理端末に端末識別情報を通知しません。ただし，SetConnectInformation メソッドを呼び出した場合は，SetConnectInformation メソッドに指定した端末識別情報が，DCCM3 論理端末との常設コネクション確立時に DCCM3 論理端末に通知されます。

●inquireTimeCheck="true ｜ false"　～《true》

常設コネクションを使用し，かつオートコネクトモード※1 の場合に，CUP.NET 実行プロセスで「問い合わせ間隔最大時間」をチェックするかどうかを指定します。

rap サーバが常設コネクションを解放するタイミングと，CUP.NET 実行プロセスが RPC を実行するタイミングが重なった場合，RPC の実行に失敗して ErrNetDownAtClientException 例外を返すことがあります。

この現象を回避するためには，inquireTimeCheck 属性を指定します。

この属性を省略，またはこの属性に true を指定した場合，「問い合わせ間隔最大時間」には 0 以外を指定※2 してください。

なお，この属性は TP1/Client for .NET Framework のバージョン 07-03-07 以降で指定できます。

この属性は省略できます。

true：CUP.NET 実行プロセスで，「問い合わせ間隔最大時間」をチェックします。

false：CUP.NET 実行プロセスで，「問い合わせ間隔最大時間」をチェックしません。

省略した場合は，CUP.NET 実行プロセスでも「問い合わせ間隔最大時間」をチェックします。

TP1/Client for .NET Framework のバージョン 07-03-07 以降，CUP.NET 実行プロセスは，トランザクション処理の処理区間外※3，かつ連鎖型 RPC の処理区間外※4 で RPC を実行するタイミングで問い合わせ間隔をチェックします。

対象となる RPC は，次の 3 種類です。

- Call メソッドを使用した RPC
- .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

問い合わせ間隔のチェックは次の計算式で求めた時間を使用します。この時間を超過した場合は，常設コネクションを解放して再接続し RPC を実行します。

チェック時間＝問い合わせ間隔最大時間－T

T：問い合わせ間隔最大時間の 2%の値
　ただし，250 ミリ秒≦T≦3 秒となります。

注※1　次のどちらかの場合が，オートコネクトモードに該当します。

(a) Client .NET 構成定義の<rapService>要素の autoConnect 属性に true を指定している。

(b) CUP.NET 内で，SetRpcExtend メソッドの extendoption パラメタに DCRPC_RAP_AUTOCONNECT を指定して実行している。

注※2　<rapService>要素の inquireTime 属性に 0 以外を指定してください。

または，CUP.NET 内で，SetRapInquireTime メソッドの sec パラメタに 0 以外を指定して実行してください。

注※3　次の(a)，(b)の条件が重なった場合がトランザクション処理の処理区間外になります。

(a) MSDTC 連携機能を使用していない。

(CUP.NET が TP1/Connector for .NET Framework 上で動作していない，または CUP.NET が TP1/Connector for .NET Framework 上で動作しているが Connector .NET 構成定義の <distributedTransaction>要素の use 属性に true を指定していない。)

(b) Begin メソッド実行から Commit/Rollback メソッドを実行するまでの処理区間外である。

注※4　連鎖型 RPC でも，初回の Call メソッドでは問い合わせ間隔をチェックします。

記述例

```
<rapService port="10020"
            autoConnect="false"
            delay="0"
            watchTimeInheritance="false"
            inquireTime="100"
            connectInformation="0xC1C2F0F1"
            inquireTimeCheck="true"/>
```

nameService

形式

```
<nameService port="ネームサービスのポート番号"
             cacheCapacity="ネームキャッシュの最大エントリ数"
             randomSelect="true|false"
             loadBalance="true|false"
             cacheTime="キャッシュの有効期限"
             multiHomedHost="true|false"
             usePriority="true|false"
             hostChangeMode="ホスト切り替え契機"
             sysWatchTime="サービス情報問合せ最大応答待ち時間"/>
```

説明

ネームサービスを使用した RPC に関する指定をします。

<rpc>要素の use 属性で nam を指定した場合，この要素は省略できません。その場合，必ず<common>要素内または<profile>要素内にこの要素を一つ以上指定してください。

属性

●port="ネームサービスのポート番号"　～〈符号なし整数〉((5001～65535))《10000》

クライアントが属している OpenTP1 のシステム共通定義の name_port オペランドで指定したネームサービスのポート番号を指定します。

<tp1Server>要素の port 属性が省略された場合，この属性の値が有効になります。

この属性は省略できます。省略する場合，必ず<tp1Server>要素の port 属性の値を指定してください。

●cacheCapacity="ネームキャッシュの最大エントリ数"　～〈符号なし整数〉((2～256))《8》

Client .NET で使用するサービス情報をキャッシュするためのキャッシュの最大エントリ数を指定します。

キャッシュのエントリは LRU 方式で管理します。キャッシュのエントリ数が指定値を超える場合は，最も過去に呼び出されたサービス情報を削除します。

この属性は省略できます。

●randomSelect="true | false"　～《false》

窓口となる TP1/Server をランダムに選択するかどうかを指定します。

この属性は，窓口となる TP1/Server を複数指定した場合にだけ有効です。

この属性は省略できます。

true：窓口となる TP1/Server をランダムに選択します。

false：窓口となる TP1/Server をランダムに選択しません。

●loadBalance="true ｜ false"　～《false》

ネームサーバから得た複数のサービス要求先スケジューラの情報から，負荷レベルが最も低いサービス要求先スケジューラの情報をキャッシュに格納するかどうかを指定します。

true：負荷レベルが最も低いサービス要求先スケジューラの情報をキャッシュに格納します。

false：負荷レベルが最も低いサービス要求先スケジューラの情報をキャッシュに格納しません。

true を指定した場合の処理を説明します。

キャッシュに格納されたサービス要求先スケジューラが複数ある場合，最初のサービス要求先スケジューラは，ランダムに選択されます。RPC でのサービス要求が 2 度目以降の場合，キャッシュに格納されたサービス要求先スケジューラが，ラウンドロビン方式で選択されます。

負荷レベルの詳細については，マニュアル「OpenTP1 解説」を参照してください。

この属性は省略できます。

●cacheTime="キャッシュの有効期限"　～〈符号なし整数〉((0～65535))《30》（単位：秒）

サービス要求先スケジューラの情報をキャッシュに格納した時点からの，キャッシュの有効期限を指定します。キャッシュの有効期限を満了したサービス要求先スケジューラの情報は，キャッシュから削除されます。その後，サービス要求先スケジューラの情報を再度取得して，その情報をキャッシュに格納することで，キャッシュを更新します。0 を指定した場合は，キャッシュの有効期限は無しになります。

この属性は，<nameService>要素の loadBalance 属性で true を指定したときだけ有効です。

この属性は省略できます。

●multiHomedHost="true ｜ false"　～《false》

マルチホームドホスト形態の TP1/Server に対してネームサービスを使用した RPC を行うかどうかを指定します。

true：マルチホームドホスト形態の TP1/Server に対してネームサービスを使用した RPC を行います。RPC の要求先を複数定義している場合，またはマルチホームドホスト形態の TP1/Server が一つ以上ある場合に指定してください。

false：マルチホームドホスト形態の TP1/Server に対してネームサービスを使用した RPC を行いません。マルチホームドホスト形態の TP1/Server に対してネームサービスを使用した RPC を行うと，エラーが発生する場合があります。

詳細については，「2.1.4(2)(c)　マルチホームドホスト形態の TP1/Server に対して RPC を行う場合の定義」を参照してください。

この属性は省略できます。

●usePriority="true ｜ false"　～《false》

ネームサービスを使用した RPC で，そのサービス要求を受け付けた TP1/Server が優先的にサービス処理を行うかどうかを指定します。

true：サービス要求を受け付けた TP1/Server が優先的にサービス処理を行います。

false：サービス要求を受け付けた TP1/Server が優先的にサービス処理を行いません。サービス要求を受け付けた TP1/Server はノード間で負荷分散します。

この属性は省略できます。

この属性をサポートしていない TP1/Client. NET は，この属性に false を指定した場合と同じ動作となります。

TP1/Client/P および TP1/Client/W のネームサービスを使用した RPC では，サービス要求を受け付けた TP1/Server が優先的にサービス処理を行います。TP1/Client/P あるいは TP1/Client/W でネームサービスを使用した RPC を使用していたユーザが TP1/Client. NET に移行する場合，この属性は true を指定してください。

●hostChangeMode="ホスト切り替え契機〔,ホスト切り替え契機〕…"

ネームサービスを使用した RPC の場合に，窓口ホストの切り替え契機であるエラー種別を変更します。

この属性は，<tp1Server>要素を複数指定し，<rpc>要素の use 属性に nam を指定した場合に有効です。

この属性は省略できます。この属性を省略すると，次に示した場合に窓口ホストを切り替えます。

- ネームサービスとのコネクション確立に失敗した場合
- ネームサービスへサービス情報の問合せ電文送信に失敗した場合
- ネームサービスからのコネクション確立待ちで通信障害が発生した場合
- ネームサービスからのサービス情報電文受信で通信障害が発生した場合

この属性を指定すると，省略した場合のホスト切り替え契機に加えて，次に示すそれぞれのホスト切り替え契機にも，窓口ホストを切り替えます。

ホスト切り替え契機を複数指定する場合は，それぞれの指定値をコンマ区切りで指定してください。

namrcv：ネームサービスからエラー応答電文を受信した場合にも，窓口ホストを切り替えます。

namtimeout：<nameService>要素の sysWatchTime 属性に指定した時間を満了してもネームサービスから応答が返ってこなかった場合にも，窓口ホストを切り替えます。

namtimeout は<nameService>要素の sysWatchTime 属性を指定したときだけ有効です。

ネームサービスから応答が返ってこなかった場合に，<tp1Server>要素に指定したすべての窓口ホストを確実に切り替え対象とするには，<rpc>要素の watchTime 属性に次の計算式より大きい値を指定します。

SPP のサービス実行時間＋<tp1Server>要素の数×（<nameService>要素の sysWatchTime 属性指定値＋<socket>要素の connectTimeout 属性指定値）

上記計算式以下の値を<rpc>要素の watchTime 属性に指定した場合，CUP.NET はすべての窓口ホストを切り替え対象とする前に，<rpc>要素の watchTime 属性で指定した時間が満了し，ErrClientTimedOutException を返すことがあります。

各種タイマ監視機能の関係性についての詳細は，「図 2-29　各種タイマ監視機能の関係性」を参照してください。

scdrcv：スケジュールサービスが開始処理中，終了処理中，または障害発生のため，スケジュールサービスからエラー応答電文を受信した場合にも，窓口ホストを切り替えます。

all：上記すべての場合に，窓口となるホストを切り替えます。

all と all 以外の値を同時に指定した場合，all が有効になります。

この属性に上記に示す値以外を指定した場合は，CUP.NET 実行時にエラーになります。ただし，コンマの前後に何も指定していない場合，コンマは無視されます。

例：hostChangeMode=",namrcv"または"namrcv,"

ホスト切り替え契機は namrcv になります。

また，コンマだけを指定した場合および空文字を指定した場合は省略値と解釈されます。

例：hostChangeMode=","または""

hostChangeMode 属性を省略したときのホスト切り替え契機になります。

●sysWatchTime="サービス情報問合せ最大応答待ち時間"　～〈符号なし整数〉((0～65535))《0》(単位：秒)

窓口となる TP1/Server のネームサービスにサービス情報を問い合わせてから，応答が返るまでの待ち時間の最大値を指定します。指定時間を過ぎても応答が返らない場合，Call メソッドは ErrClientTimedOutException を返します。ただし，<tp1Server>要素を複数指定し，かつ<nameService>要素の hostChangemode 属性に namtimeout を指定していた場合は，窓口ホストを切り替えます。

この属性は，<rpc>要素の use 属性に nam を指定した場合に有効です。

この属性に<rpc>要素の watchTime 属性より大きい値または 0 を指定した場合は，watchTime 属性の指定に従います。この属性と watchTime 属性との両方に 0 を指定した場合は，応答を受信するまで無限に待ち続けます。

この属性の指定が適用される API（メソッド）を次に示します。

- Call メソッド

この属性の監視時間には，ネームサービスとのコネクション確立時間を含みません。ネームサービスとのコネクション確立時間を監視するためには，<socket>要素の connectTimeout 属性の設定が必要です。

記述例

```
<nameService port="10000"
             cacheCapacity="8"
             randomSelect="false"
             loadBalance="true"
             cacheTime="30"
             multiHomedHost="true"
             usePriority="true"
             hostChangeMode="all"
             sysWatchTime="60"/>
```

scheduleService

形式

```
<scheduleService port="スケジュールサービスのポート番号"
                 usePriority="true|false"
                 randomSelect="true|false"
                 hostChange="true|false"
                 multiSchedulerCount
                 ="スケジューラデーモンのプロセス数"/>
```

説明

スケジュールサービスを使用した RPC に関する指定をします。

<rpc>要素の use 属性で scd を指定した場合，この要素は省略できません。その場合，必ず<common>要素内または<profile>要素内にこの要素を一つ以上指定してください。

属性

●port="スケジュールサービスのポート番号"　～〈符号なし整数〉((5001～65535))

スケジュールサービスのポート番号を指定します。<tp1Server>要素の port 属性が省略された場合，この属性の値が有効になります。

Client.NET 構成定義の<rpc>要素 use 属性に scd を指定し，かつ<rpc>要素 useMultiScheduler 属性に true を指定した場合は，マルチスケジューラデーモンのポート番号を指定します。マスタスケジューラデーモンとマルチスケジューラデーモンのベースとなるポート番号が連続している場合は，マスタスケジューラデーモン（スケジュールサービス）のポート番号を指定することもできます。

この属性は省略できます。省略する場合，必ず<tp1Server>要素の port 属性の値を指定してください。

なお，通信相手の TP1/Server は，スケジュールサービスまたはマルチスケジューラデーモンを，この定義で指定したポート番号で起動する必要があります。TP1/Server のスケジュールサービスまたはマルチスケジューラデーモンの指定については，マニュアル「OpenTP1 システム定義」を参照してください。

●usePriority="true | false"　～《false》

サービス要求を受け付けた窓口となる TP1/Server を優先して負荷分散するかどうか指定します。

この属性は省略できます。

true：サービス要求を受け付けた窓口となる TP1/Server を優先して負荷分散します。

false：サービス要求を受け付けた窓口となる TP1/Server を優先しません。TP1/Server は，ノード間の負荷を分散します。

●randomSelect="true | false"　～《false》

窓口となる TP1/Server をランダムに選択するかどうかを指定します。

この属性は，窓口となる TP1/Server を複数指定した場合にだけ有効です。

この属性は省略できます。

true：窓口となる TP1/Server をランダムに選択します。

false：窓口となる TP1/Server をランダムに選択しません。

●hostChange="true | false"　～《false》

<tp1Server>要素に指定したサービス要求先スケジューラを RPC ごとに分散させるかどうかを指定します。

RPC でのサービス要求が 1 度目の場合，<scheduleService>要素の randomSelect 属性に false を指定したとき，<tp1Server>要素で指定した先頭のサービス要求先スケジューラが選択されます。<scheduleService>要素の randomSelect 属性に true を指定したとき，<tp1Server>要素で指定したサービス要求先スケジューラがランダムに選択されます。

RPC でのサービス要求が 2 度目以降の場合，前回のサービス要求先スケジューラから<tp1Server>要素で指定した順にラウンドロビン方式でサービス要求先スケジューラが選択されます。

この属性は省略できます。

true：サービス要求先スケジューラを RPC ごとに分散させます。

false：サービス要求先スケジューラを RPC ごとに分散させません。

●multiSchedulerCount="スケジューラデーモンのプロセス数"　～〈符号なし整数〉((1～4096))《1》

スケジューラデーモンのプロセス数を指定します。スケジュールサービス定義の scdmulti 定義コマンドの-m オプションに指定したプロセス数か，またはそれ以下の値を指定します。

この属性に指定するプロセス数は，<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性に指定するポート番号によって，次のように異なります。

表 3-1　Client .NET 構成定義の<scheduleService>要素の multiSchedulerCount 属性に指定するプロセス数の違い

<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性の指定値	<scheduleService>要素の multiSchedulerCount 属性の指定値
マルチスケジューラデーモンのベースとなるポート番号	マルチスケジューラデーモンのプロセス数と同じ値，またはそれ以下の値を指定します。
マルチスケジューラデーモンの任意のポート番号	「マルチスケジューラデーモンのポート番号の最大 - <tp1Server>要素の port 属性，または<scheduleService>要素の port 属性に指定したポート番号 ＋ 1」の値，またはそれ以下の値を指定します。

<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性の指定値	<scheduleService>要素の multiSchedulerCount 属性の指定値	
マスタスケジューラデーモンのポート番号	マスタスケジューラデーモンとマルチスケジューラデーモンのベースとなるポート番号が連続している場合	「マルチスケジューラデーモンのプロセス数＋1」の値，またはそれ以下の値を指定します。
	上記以外	1 を指定するか，または指定を省略します（マルチスケジューラデーモンは使用できません）。

この定義は，Client.NET 構成定義の<rpc>要素の use 属性に scd を指定し，かつ<rpc>要素の useMultiScheduler 属性に true を指定した場合に有効です。この場合，ポート番号は，次に示す範囲の値からランダムに選択します。

- 下限値：Client .NET 構成定義の<tp1Server>要素の port 属性，または<scheduleService>要素の port 属性に指定したポート番号の値
- 上限値：下限値 ＋ Client .NET 構成定義の<scheduleService>要素の multiSchedulerCount 属性に指定したプロセス数 - 1

この属性は省略できます。

記述例

```
<scheduleService port="10010"
                 usePriority="false"
                 hostChange="false"
                 multiSchedulerCount="10"/>
```

extendLevel

形式

```
<extendLevel value="00000000|00000001"/>
```

説明

Client .NET の機能拡張レベルの設定を指定します。

この要素は省略できます。

属性

●value="00000000 | 00000001"　～《00000000》

Client .NET の機能拡張レベルの設定を指定します。

この属性は省略できます。

00000000：Client .NET の機能を拡張しません。

00000001：Call メソッドを呼び出したとき，自 CUP.NET の IP アドレスをサービスに連絡します。呼び出したサービスで Rpc クラスの GetCallersAddress メソッドまたは dc_rpc_get_callers_address メソッドを実行し，CUP.NET のアドレスを求める必要がある場合に指定します。

記述例

```
<extendLevel value="00000000"/>
```

errTrace

形式

```
<errTrace use=”true|false”
          path=”エラートレースファイル作成ディレクトリ”
          fileSize=”エラートレースファイルサイズ”/>
```

説明

CUP.NET でのエラートレース取得に関する指定をします。

この要素は省略できます。

属性

●use="true | false"　～《false》

CUP.NET でエラートレースを取得するかどうかを指定します。

この属性は省略できます。

true：エラートレースを取得します。

false：エラートレースを取得しません。

●path="エラートレースファイル作成ディレクトリ"　～〈文字列〉

エラートレースファイルを作成するディレクトリのパス名を指定します。

<errTrace>要素の use 属性に true を指定した場合，必ず指定してください。

●fileSize="エラートレースファイルサイズ"　～〈符号なし整数〉((4096～1048576))《4096》(単位：バイト)

エラートレースファイルのサイズを指定します。

この属性は省略できます。

記述例

```
<errTrace use=”true”
          path=”c:¥temp¥clientn”
          fileSize=”10000”/>
```

methodTrace

形式

```
<methodTrace use=”true|false”
             path=”メソッドトレースファイル作成ディレクトリ”
             fileSize=”メソッドトレースファイルサイズ”/>
```

説明

CUP.NET でのメソッドトレース取得に関する指定をします。

この要素は省略できます。

属性

●use="true | false"　～《false》

CUP.NET でメソッドトレースを取得するかどうかを指定します。

この属性は省略できます。

true：メソッドトレースを取得します。

false：メソッドトレースを取得しません。

●path="メソッドトレースファイル作成ディレクトリ"　～〈文字列〉

メソッドトレースファイルを作成するディレクトリのパス名を指定します。

<methodTrace>要素の use 属性に true を指定した場合，必ず指定してください。

●fileSize="メソッドトレースファイルサイズ"　～〈符号なし整数〉((4096～1048576))《4096》(単位：バイト)

メソッドトレースファイルのサイズを指定します。

この属性は省略できます。

記述例

```
<methodTrace use=”true”
             path=”c:¥temp¥clientn”
             fileSize=”10000”/>
```

uapTrace

形式

```
<uapTrace use=”true|false”
          path=”UAPトレースファイル作成ディレクトリ”
          fileSize=”UAPトレースファイルサイズ”/>
```

説明

CUP.NET での UAP トレース取得に関する指定をします。

この要素は省略できます。

属性

●use="true | false"　～《false》

CUP.NET で UAP トレースを取得するかどうかを指定します。

この属性は省略できます。

true：UAP トレースを取得します。

false：UAP トレースを取得しません。

●path="UAP トレースファイル作成ディレクトリ"　～〈文字列〉

UAP トレースファイルを作成するディレクトリのパス名を指定します。

<uapTrace>要素の use 属性に true を指定した場合，必ず指定してください。

●fileSize="UAP トレースファイルサイズ"　～〈符号なし整数〉((4096～1048576))《4096》(単位：バイト)

UAP トレースファイルのサイズを指定します。

この属性は省略できます。

記述例

```
<uapTrace use=”true”
          path=”c:¥temp¥clientn”
          fileSize=”10000”/>
```

dataTrace

形式

```
<dataTrace use="true|false"
           path="データトレースファイル作成ディレクトリ
           fileSize="データトレースファイルサイズ"
           maxDataSize="データトレースの最大データ長"/>
```

説明

CUP.NET でのデータトレース取得に関する指定をします。

この要素は省略できます。

属性

●use="true | false"　～《false》

CUP.NET でデータトレースを取得するかどうかを指定します。

この属性は省略できます。

true：データトレースを取得します。

false：データトレースを取得しません。

●path="データトレースファイル作成ディレクトリ"　～〈文字列〉

データトレースファイルを作成するディレクトリのパス名を指定します。

<dataTrace>要素の use 属性に true を指定した場合，必ず指定してください。

●fileSize="データトレースファイルサイズ"　～〈符号なし整数〉((4096～1048576))《4096》(単位：バイト)

データトレースファイルのサイズを指定します。

この属性は省略できます。

●maxDataSize="データトレースの最大データ長"　～〈符号なし整数〉((16～1048576))《128》(単位：バイト)

一つのデータトレースのデータ長の最大値を指定します。

この属性は省略できます。

記述例

```
<dataTrace use="true"
           path="c:¥temp¥clientn"
           fileSize="10000"
           maxDataSize="128"/>
```

debugTrace

形式

```
<debugTrace path="デバッグトレースファイル出力ディレクトリ作成パス"
            fileCount="デバッグトレース最大ファイル数"/>
```

説明

CUP.NET でのデバッグトレース取得に関する指定をします。

この要素は省略できます。

また，この要素は<common>要素内でだけ有効です。

属性

●path="デバッグトレースファイル出力ディレクトリ作成パス"

デバッグトレースファイル出力ディレクトリ作成用のパス名を指定します。

実際のデバッグトレースファイルは，次のディレクトリに作成されます。

〈この属性で指定したパス〉¥Hitachi¥OpenTP1¥TP1ClientNET

この属性は省略できます。省略した場合，デバッグトレースファイルは次のディレクトリに作成されます。

〈アプリケーション実行ユーザのアプリケーションデータディレクトリ〉
¥Hitachi¥OpenTP1¥TP1ClientNET

【アプリケーションデータディレクトリの例】

C:¥Document and Settings¥〈ユーザ名〉¥Application Data

【アプリケーションデータディレクトリの例（Windows サービスの場合）】

C:¥Document and Settings¥LocalService¥Application Data

●fileCount="デバッグトレース最大ファイル数"　～〈符号なし整数〉((0～256))《32》

デバッグトレースの最大ファイル数を指定します。0 を指定した場合，最大ファイル数の制限はなくなります。最大ファイル数を超えた場合，ファイル作成日付の最も古いファイルが削除されます。

同じデバッグトレース作成ディレクトリに対して異なる最大ファイル数を指定した場合，最大ファイル数は保証されません。

この属性の指定が不正の場合，エラーは通知されません。その場合，属性の値はデフォルト値が有効になります。

この属性は省略できます。

記述例

```
<debugTrace path="c:¥temp¥clientn"
            fileCount="10"/>
```

transaction

形式

```
<transaction internalWatchTime
             =”トランザクション同期点処理時の最大通信待ち時間”
             rollbackInfo=”no|self|remote|all”
             limitTime=”トランザクションブランチ最大実行可能時間”
             recoveryType=”type1|type2|type3”
             expireTime=”トランザクションブランチ限界経過時間”
             cpuTime=”トランザクションブランチCPU監視時間”
             expireSuspend=”all|async|none”
             statistics=”統計情報項目”
             optimize=”トランザクション最適化項目”
             rollbackResponse=”true|false”
             completionLimitTime=”トランザクション完了限界時間”/>
```

説明

CUP.NET からトランザクションを開始する場合に指定します。

この要素は省略できます。

属性

●internalWatchTime="トランザクション同期点処理時の最大通信待ち時間"　～〈符号なし整数〉((1～65535))（単位：秒）

トランザクションの同期点処理で，トランザクションブランチ間で行う通信（プリペア，コミット，ロールバック指示，応答など）の受信待ち時間の最大値を指定します。

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_watch_time オペランドの指定に従います。

●rollbackInfo="no | self | remote | all"

トランザクションブランチがロールバックした場合に，ロールバック要因に関する情報をログに取得するかどうかを指定します。

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_rollback_information_put オペランドの指定に従います。

no：ロールバック情報を取得しません。

self：ロールバック要因が発生したトランザクションブランチでだけ，ログにロールバック情報を取得します。

remote：self に加え，他ノードのトランザクションブランチからロールバック要求されたトランザクションブランチでも，ログにロールバック情報を取得します。

all：remote に加え，自ノードのトランザクションブランチからロールバック要求されたトランザクションブランチでも，ログにロールバック情報を取得します。

●limitTime="トランザクションブランチ最大実行可能時間"　～〈符号なし整数〉((0～65535))（単位：秒）

トランザクションブランチの最大実行可能時間を指定します。

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_limit_time オペランドの指定に従います。

●recoveryType="type1 | type2 | type3"

RPC がタイムアウトし，RPC 発行先プロセスのアドレスが未解決の場合やトランザクション実行中の UAP がダウンした場合に，トランザクションブランチ間の連絡がスムーズにできないで，トランザクションの決着に時間が掛かることがあります。

この属性では，次に示す障害が発生した場合のトランザクション同期点処理方式を，指定値に示す三つの方式から選択して指定します。

（障害 1）RPC がタイムアウトした場合

この場合，RPC 発行元トランザクションブランチは，サービス要求がどのプロセスで実行されているかがわからないため，RPC 発行先トランザクションブランチにトランザクション同期点メッセージを送信できないで，RPC 発行元トランザクションブランチおよび RPC 発行先トランザクションブランチがトランザクション同期点メッセージ待ちとなり，トランザクションの決着に時間が掛かります。

（障害 2）RPC 発行元 UAP が RPC の応答受信前にダウンした場合

この場合，RPC 発行元トランザクションブランチは，サービス要求がどのプロセスで実行されているかがわからないため，RPC 発行先トランザクションブランチにトランザクション同期点メッセージを送信できないで，RPC 発行先トランザクションブランチはトランザクション同期点メッセージ待ちとなり，トランザクションの決着に時間が掛かります。

（障害 3）RPC 発行先 UAP からの応答受信後に RPC 発行元 UAP と RPC 発行先 UAP がほぼ同時にダウンした場合

この場合，それぞれのトランザクションブランチを引き継いだトランザクション回復プロセスは，相手 UAP プロセスのダウンを知らないため，すでに存在しない UAP プロセスにトランザクション同期点メッセージを送信してしまい，トランザクションの決着に時間が掛かることがあります。

type1

（障害 1）が発生した場合，RPC 発行元トランザクションブランチおよび RPC 発行先トランザクションブランチは，トランザクション同期点メッセージ受信処理がタイムアウトすることによって，トランザクションを決着します。

（障害 2）が発生した場合，RPC 発行元トランザクションブランチは，RPC 発行先トランザクションブランチにトランザクション同期点メッセージを送信しないでトランザクションを決着します。RPC 発行先トランザクションブランチは，トランザクション同期点メッセージ受信処理がタイムアウトすることによって，トランザクションを決着します。

（障害 3）が発生した場合，RPC 発行元トランザクションブランチおよび RPC 発行先トランザクションブランチは，トランザクション同期点メッセージ受信処理がタイムアウトすることによって，トランザクションを決着します。

type2

（障害 1）が発生してトランザクションをコミットする場合は type1 と同じです。

（障害 1）が発生してトランザクションをロールバックする場合，または（障害 2）が発生した場合は，RPC 発行元トランザクションブランチは，RPC 発行先トランザクションブランチが存在するノードのトランザクションサービスプロセスにトランザクション同期点メッセージを送信後，トランザクションを決着します。トランザクション同期点メッセージを受信したトランザクションサービスプロセスは，該当するトランザクションブランチを処理中のプロセスに，トランザクション同期点指示を送信します。

（障害 3）が発生した場合，RPC 発行元トランザクションブランチおよび RPC 発行先トランザクションブランチは，トランザクション同期点メッセージ受信処理がタイムアウトすることによって，トランザクションを決着します。

type3

（障害 1）が発生してトランザクションをコミットする場合は，type1 と同じです。

（障害 1）が発生してトランザクションをロールバックする場合，（障害 2）が発生した場合，または（障害 3）が発生した場合，相手トランザクションブランチが存在するノードのトランザクションサービスプロセスに，トランザクション同期点メッセージを送信後，トランザクションを決着します。トランザクション同期点メッセージを受信したトランザクションサービスプロセスは，該当するトランザクションブランチを処理中のプロセスに，トランザクション同期点指示を送信します。

次に示す場合，このオペランドに type2 または type3 を指定しても，トランザクションの決着に時間が掛かることがあります。

1. RPC 実行中に，RPC 発行先 UAP の状態が変更となり(負荷増加，UAP 終了，UAP 閉塞など)，ほかのノードの同一 UAP にサービス要求が再転送された場合
2. 相手先の OpenTP1 がこのオプションをサポートしていないバージョンの場合
3. 相手先トランザクションブランチがトランザクション同期点メッセージ受信処理以外で時間が掛かっている場合

なお，このオペランドは，ユーザサービス定義または rap リスナーサービス定義，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。

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_partial_recovery_type オペランドの指定に従います。

●expireTime="トランザクションブランチ限界経過時間"　～〈符号なし整数〉((0～65535))（単位：秒)

トランザクションブランチの処理時間の最大値を指定します。指定時間を超えてもトランザクションブランチが完了しない場合は，そのトランザクションブランチのプロセスを異常終了させてロールバックします。

0 を指定した場合，時間監視をしません。

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_expiration_time オペランドの指定に従います。

●cpuTime="トランザクションブランチ CPU 監視時間"　～〈符号なし整数〉((0～65535))（単位：秒）

トランザクションブランチが同期点処理までに使用できる CPU 時間を指定します。

0 を指定した場合，CPU 時間を監視しません。

指定時間を超えた場合は，そのトランザクションブランチのプロセスを異常終了させてロールバックします。

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_cpu_time オペランドの指定に従います。

●expireSuspend="all | async | none"

トランザクションブランチの処理を監視するとき，次の処理時間も監視時間に含むかどうかを指定します。

1. 監視対象のトランザクションブランチが，RPC 機能を使用してほかのトランザクションブランチを呼び出し，その処理が終わるのを待つ時間
2. 連鎖 RPC で呼び出されたサーバ UAP が次のサービス要求を待つ時間
3. 監視対象のトランザクションブランチが，非同期型 RPC を使用してほかのトランザクションブランチを呼び出したあと，処理結果受信処理をしている時間

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_expiration_time_suspend オペランドの指定に従います。

all：1，2，3 すべてを監視時間に含みます。

async：3 だけを監視時間に含みます。

none：1，2，3 のどれも監視時間に含みません。

●statistics="統計情報項目"

トランザクションブランチの統計情報を取得する項目を，次の文字列で指定します。

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_statistics_item オペランドの指定に従います。

また，統計情報項目は複数指定できます。複数指定するときは，コンマ（,）で区切って指定します。

nothing：統計情報を取得しません。

base：基本情報として，次の情報を取得します。

- トランザクションブランチの識別子
- トランザクションブランチの決着結果
- トランザクションブランチの実行プロセス種別

- トランザクションブランチの実行サーバ名
- トランザクションブランチの実行サービス名

executiontime

基本情報とトランザクションブランチの実行時間情報を取得します。

cputime

基本情報とトランザクションブランチの CPU 時間情報を取得します。

●optimize="トランザクション最適化項目"

複数のユーザサーバで構成されるグローバルトランザクションの性能を向上させるための最適化項目を，次の文字列で指定します。CUP.NET からトランザクションを開始する場合にだけ有効です。

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_optimum_item オペランドの指定に従います。

また，最適化項目は複数指定できます。複数指定するときは，コンマ（,）で区切って指定します。

base

同期点取得処理全体（プリペア処理，コミット処理，およびロールバック処理）を最適化します。
OpenTP1 のトランザクション制御は 2 相コミット方式で実行しているため，二つのトランザクションブランチ間のコミット制御には，4 回のプロセス間通信が必要となります。
次の条件をすべて満たす場合，親トランザクションブランチが子トランザクションブランチのコミット処理を代わりに実行することで，コミット制御で必要な 4 回のプロセス間通信を削減します。

- 親トランザクションブランチと子トランザクションブランチが同一 OpenTP1 下にあること。
- 親トランザクションブランチが子トランザクションブランチを同期応答型 RPC で呼び出していること。
- 子トランザクションブランチでアクセスしたリソースマネジャの XA インタフェース用オブジェクトが，親トランザクションブランチにもリンケージされていること。

asyncprepare

base の指定条件を満たしていないため同期点取得処理全体の最適化ができない場合に，プリペア処理を最適化します。
次の条件をすべて満たす場合，親トランザクションブランチから発行された RPC によって子トランザクションブランチがサービス要求を実行したときに，RPC が返される前にプリペア処理を実行することで，2 回のプロセス間通信を削減します。

- base を指定した最適化ができないこと。
- 親トランザクションブランチが，子トランザクションブランチを同期応答型 RPC で呼び出していること。

●rollbackResponse="true | false"

RPC 発行先トランザクションブランチにロールバック指示を送信したあと，ロールバック完了通知を受信するかどうかを指定します。CUP.NET からトランザクションを開始する場合にだけ有効です。

この属性は省略できます。省略した場合，クライアントが属している OpenTP1 の rap リスナーサービス定義の trn_rollback_response_receive オペランドの指定に従います。

true：ロールバック完了通知を受信します。

false：ロールバック完了通知を受信しません。

false を指定した場合，RPC 発行先トランザクションブランチからのロールバック完了通知を受信しないで（RPC 発行先トランザクションブランチのロールバック処理の完了を待たないで）自トランザクションブランチを終了します。

●completionLimitTime="トランザクション完了限界時間"　～〈符号なし整数〉((0～65535))（単位：秒）

rap サーバで代理実行するトランザクションブランチの開始から終了までの最大実行時間を指定します。指定時間を超えた場合，rap サーバのプロセスが異常終了したあとに，トランザクションブランチが回復プロセスによってコミット，またはロールバックのどちらかに決着して終了します。

0 を指定した場合は，トランザクションブランチの最大実行時間を監視しません。

この指定を省略した場合は，rap リスナーサービス定義の trn_completion_limit_time オペランドの指定に従います。

この属性は省略できます。

記述例

```
<transaction internalWatchTime="180"
             rollbackInfo="all"
             limitTime="0"
             recoveryType="type1"
             expireTime="0"
             cpuTime="0"
             expireSuspend="all"
             statistics="base"
             optimize="base"
             rollbackResponse="false"
             completionLimitTime="0"/>
```

tcpip

形式

```
<tcpip use="true|false"
       type="send|recv|sendrecv"
       sendHost="ノードのホスト名"
       sendPort="MHPのポート番号"
       recvPort="CUP.NETのポート番号"
       openPortAtRecv="true|false"/>
```

説明

TCP/IP 通信機能に関する指定をします。

この要素は省略できます。

属性

●use="true | false"　～《false》

TCP/IP 通信機能を使用するかどうかを指定します。

この属性は省略できます。

true：TCP/IP 通信機能を使用します。

false：TCP/IP 通信機能を使用しません。

●type="send | recv | sendrecv"

TCP/IP 通信機能を使用する場合に，初期化する環境を指定します。

<tcpip>要素の use 属性に true を指定した場合，この属性は省略できません。その場合，次のどれかを指定してください。

send：メッセージを一方送信するための環境

recv：メッセージを一方受信するための環境

sendrecv：メッセージを送受信するための環境

●sendHost="ノードのホスト名"　～〈文字列〉

CUP.NET からメッセージを送信する場合，コネクションを確立して接続する MHP が存在するノードのホスト名を指定します。また，ホスト名として 10 進ドット記法の IP アドレスも指定できます。

この属性は省略できます。

●sendPort="MHP のポート番号"　～〈符号なし整数〉((1～65535))《12000》

CUP.NET からメッセージを送信する場合，コネクションを確立して接続する MHP のポート番号を指定します。<nameService>要素の port 属性で指定するポート番号と重複しないように指定してください。

この属性は省略できます。

●recvPort="CUP.NET のポート番号"　～〈符号なし整数〉((1～65535))《11000》

TCP/IP 通信機能を使用してメッセージを受信する場合，メッセージを受信する CUP.NET のポート番号を指定します。メッセージを送信する側では，このポート番号を指定して送信してください。

同一マシン内で，複数のプロセス，またはスレッドを同時に実行する場合は，それぞれ異なるポート番号を指定してください。

指定できるポート番号でも，OS またはほかのプログラムで使用するポート番号は指定しないでください。指定した場合，応答データを正しく受信できないことがあります。

なお，OS が使用するポート番号は，OS ごとに異なります。OS のマニュアルなどを参照してください。ほかのアプリケーションとの重複を避けるため，OS が任意に割り当てるポート番号（動的ポートまたは短命ポートと呼ばれるポート番号）を使用しないでください。

メッセージの一方送信機能（<tcpip>要素の type 属性に send を指定）の場合，デフォルト値を含めこの定義で指定したポートは使用されません。

この属性は省略できます。

●openPortAtRecv="true|false"　～《false》

TCP/IP 通信機能使用時に，メッセージの送受信をする場合の受信用ソケットの開設契機（送信相手からの接続を待ち受け始める契機）を指定します。

この属性は省略できます。省略した場合，OpenRpc メソッド実行時に受信用ソケットを開設します。

MHP がサーバ型でメッセージの送受信を 1 つのコネクションで行う場合，CUP.NET から MHP にメッセージを送信した際に確立したコネクションを使用して送受信を行うため，CUP.NET は受信用のソケットは使用しません。そのため，この定義に true を指定し，受信用ソケットを開設しないようにしてください。

MHP がクライアント型の場合，MHP からのメッセージを CUP.NET が受信する際に受信用のソケットが必要なため，この定義に false を指定するか定義を省略してください。

この定義の具体的なユースケースは「2.4.5　ユースケースごとの設定方法とポートの割り当て」を参照してください。

true：OpenRpc メソッド実行時に受信ソケットを開設しません。ただし，次のメソッドを実行したときに，コネクションが確立されていない場合，受信用ソケットを開設します。

- Receive

- ReceiveAssembledMessage

false：OpenRpc メソッド実行時に受信用ソケットを開設します。

記述例

```
<tcpip use="true"
       type="sendrecv"
       sendHost="10.210.208.13"
       sendPort="12000"
       recvPort="22000"
       openPortAtRecv="false"/>
```

socket

形式

```
<socket connectTimeout=”コネクション確立最大監視時間”
        cupSendHost=”送信元ホスト”
        backlogCount=”コネクション確立要求を格納するキューのサイズ”
        sendBufferSize=”TCP/IP の送信バッファサイズ”
        receiveBufferSize=”TCP/IP の受信バッファサイズ”/>
```

説明

ソケットに関する指定をします。

属性

●connectTimeout="コネクション確立最大監視時間"　～〈符号なし整数〉((0～65535))《0》(単位：秒)

コネクション確立に対する最大監視時間を指定します。

この属性で指定する値は，Client .NET が提供するメソッドで実行されるコネクションの確立処理に掛かる最大監視時間です。なお，Client .NET 構成定義の<rpc>要素の watchTime 属性やメソッド引数などで監視するメソッドの処理時間は，この属性で指定する値よりも大きな値を指定してください。この属性で指定した値より小さな値を指定した場合，コネクション確立でタイムアウトが発生したときにメソッドがリターンするまでに掛かる時間は，この属性で指定した時間となるので注意してください。

相手システムが未起動であるなどの要因でコネクションを確立できない場合，この属性で指定した監視時間が経過する前に CUP.NET から発行したメソッドがエラーリターンすることがあります。これは，この属性で指定した最大監視時間よりも，OS によるコネクション確立処理の監視時間が優先されるためです。OS によるコネクション確立処理の監視時間，コネクション確立要求の再送回数および再送処理の間隔は，OS によって異なります。

この属性は省略できます。この属性を省略した場合，または 0 を指定した場合，コネクション確立の監視処理は OS によって行われます。

●cupSendHost="送信元ホスト"　～〈文字列〉

次のコネクション確立要求時の送信元ホストを指定します。

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

ホスト名として，10 進ドット記法の IP アドレスを指定することもできます。

なお，次の場合は，発行したメソッドで例外が発生します。

- ホスト名として，localhost を指定した場合
- IP アドレスが 127 で始まるホストを指定した場合
- CUP.NET を実行するマシン上に存在しないホスト名を指定した場合

この属性を省略すると，送信元ホストは任意に割り当てられます。

この属性は，CUP.NET のコネクション確立要求時の送信元ホストを指定できますが，CUP.NET が相手システムからコネクション確立を受け付ける CUP.NET の受信元ホストには適用されません。

●backlogCount="コネクション確立要求を格納するキューのサイズ"　～〈符号なし整数〉((1～2147483647))《5》

次の機能で使用されるコネクション確立要求を格納するキューのサイズ（listen システムコールのバックログ数）を指定します。

- スケジューラダイレクト機能を使用した RPC
- ネームサービスを使用した RPC
- 通信先を指定した RPC
- TCP/IP 通信機能
- 一方通知受信機能
- 一方通知連続受信機能

この属性は省略できます。

なお，キューのサイズの上限値および下限値は，OS によって異なります。OS によってキューのサイズの上限値および下限値が制限されている場合，指定した値が有効にならないことがあります。

コネクション確立要求を格納するキューについての詳細は，OS のマニュアルまたは TCP/IP のドキュメントを参照してください。

●sendBufferSize="TCP/IP の送信バッファサイズ" ～〈符号なし整数〉((8192～2147483647))（単位：バイト）

CUP.NET とサーバ間のコネクションで確保される TCP/IP の送信バッファサイズを指定します。高速な通信媒体や MTU の大きな通信媒体を使用している場合，この値を大きくすることによって通信の性能向上を図れます。

このオペランドを省略した場合は，.NET Framework のデフォルトのバッファサイズを適用します。

このオペランドの値は，.NET Framework で使用できる TCP/IP 送信バッファの上限値以下を指定してください。

CUP.NET と通信するすべてのノードで同じ値を指定してください。同じ値を指定しない場合，通信するノードとバッファサイズに差異が生じ，通信性能が劣化するおそれがあります。

●receiveBufferSize="TCP/IP の受信バッファサイズ" ～〈符号なし整数〉((8192～2147483647))（単位：バイト）

CUP.NET とサーバ間のコネクションで確保される TCP/IP の受信バッファのサイズを指定します。高速な通信媒体や MTU の大きな通信媒体を使用している場合，この値を大きくすることによって通信の性能向上を図れます。

このオペランドを省略した場合は，.NET Framework のデフォルトのバッファサイズを適用します。

このオペランドの値は，.NET Framework で使用できる TCP/IP 受信バッファの上限値以下を指定してください。

CUP.NET と通信するすべてのノードで同じ値を指定してください。同じ値を指定しない場合，通信するノードとバッファサイズに差異が生じ，通信性能が劣化するおそれがあります。

記述例

```
<socket connectTimeout="10"
        cupSendHost="HOSTA"
        backlogCount="20"
        sendBufferSize="8192"
        receiveBufferSize="8192"/>
```

xarTransaction

形式

```
<xarTransaction expireTime="トランザクションブランチ限界経過時間"/>
```

説明

CUP.NET から MSDTC 連携機能を使用したトランザクションを開始する場合に指定します。

この要素は省略できます。

属性

●expireTime="トランザクションブランチ限界経過時間"　～〈符号なし整数〉((0～65535))（単位：秒）

トランザクションブランチ限界経過時間を指定します。指定時間を超えてもトランザクションブランチが完了しない場合，TP1/Server はそのトランザクションブランチのプロセスを異常終了させてロールバックします。

0 を指定した場合，時間監視をしません。

この属性は省略できます。省略した場合，クライアントが属している TP1/Server の rap リスナーサービス定義の trn_expiration_time オペランドに指定された値に従います。

記述例

```
<xarTransaction expireTime="0"/>
```

定義例

ネームサービスを使用する場合

```
<configuration>
  <configSections>
    <section
      name="hitachi.opentp1.client"
      type="Hitachi.OpenTP1.Common.Util.ProfileSectionHandler,
            Hitachi.OpenTP1.Client,Version=7.0.0.0,
            Culture=neutral,PublicKeyToken=2440cf5f0d80c91c,
            Custom=null"/>
  </configSections>

  <hitachi.opentp1.client>
    <common>
      <tp1Server host="10.210.208.13"/>
      <rpc use="nam" watchTime="0"/>
      <nameService port="10000"/>
    </common>
    <profile id="traceMode">
      <errTrace use="true" path="c:¥temp¥clientn"
                fileSize="100000"/>
      <methodTrace use="true" path="c:¥temp¥clientn"
                   fileSize="100000"/>
      <uapTrace use="true" path="c:¥temp¥clientn"
                fileSize="100000"/>
      <dataTrace use="true" path="c:¥temp¥clientn"
                 fileSize="100000"/>
    </profile>
  </hitachi.opentp1.client>

</configuration>
```

スケジューラダイレクト機能を使用する場合

```
<configuration>
  <configSections>
    <section
      name="hitachi.opentp1.client"
      type="Hitachi.OpenTP1.Common.Util.ProfileSectionHandler,
            Hitachi.OpenTP1.Client,Version=7.0.0.0,
            Culture=neutral,PublicKeyToken=2440cf5f0d80c91c,
            Custom=null"/>
  </configSections>

  <hitachi.opentp1.client>
    <common>
      <tp1Server host="10.210.208.13"/>
      <rpc use="scd" watchTime="0"/>
      <scheduleService port="10010"/>
    </common>
    <profile id="traceMode">
      <errTrace use="true" path="c:¥temp¥clientn"
                fileSize="100000"/>
```

```
      <methodTrace use="true" path="c:¥temp¥clientn"
                   fileSize="100000"/>
      <uapTrace use="true" path="c:¥temp¥clientn"
                fileSize="100000"/>
      <dataTrace use="true" path="c:¥temp¥clientn"
                 fileSize="100000"/>
    </profile>
  </hitachi.opentp1.client>

</configuration>
```

rap サービスを使用する場合

```
<configuration>
  <configSections>
    <section
      name="hitachi.opentp1.client"
      type="Hitachi.OpenTP1.Common.Util.ProfileSectionHandler,
            Hitachi.OpenTP1.Client,Version=7.0.0.0,
            Culture=neutral,PublicKeyToken=2440cf5f0d80c91c,
            Custom=null"/>
  </configSections>

  <hitachi.opentp1.client>
    <common>
      <tp1Server host="10.210.208.13"/>
      <rpc use="rap" watchTime="0"/>
      <rapService port="10020"/>
    </common>
    <profile id="traceMode">
      <errTrace use="true" path="c:¥temp¥clientn"
                fileSize="100000"/>
      <methodTrace use="true" path="c:¥temp¥clientn"
                   fileSize="100000"/>
      <uapTrace use="true" path="c:¥temp¥clientn"
                fileSize="100000"/>
      <dataTrace use="true" path="c:¥temp¥clientn"
                 fileSize="100000"/>
    </profile>
  </hitachi.opentp1.client>

</configuration>
```

サーバおよび通信方式別にプロファイルを宣言する場合

```
<configuration>
  <configSections>
    <section
      name="hitachi.opentp1.client"
      type="Hitachi.OpenTP1.Common.Util.ProfileSectionHandler,
            Hitachi.OpenTP1.Client,Version=7.0.0.0,
            Culture=neutral,PublicKeyToken=2440cf5f0d80c91c,
            Custom=null"/>
  </configSections>

  <hitachi.opentp1.client>
```

```
    <common>
      <tp1Server host="10.210.208.1"/>
      <rpc use="rap" watchTime="0"/>
      <rapService port="10020"/>
      <errTrace use="true" path="c:¥temp¥clientn"
                fileSize="100000"/>
      <dataTrace use="true" path="c:¥temp¥clientn"
                 fileSize="100000"/>
    </common>
    <profile id="nam">
      <tp1Server host="10.210.208.1"/>
      <rpc use="nam" watchTime="0"/>
      <nameService port="10000"/>
    </profile>
    <profile id="server2Rap">
      <tp1Server host="10.210.208.2"/>
      <rpc use="rap" watchTime="0"/>
      <rapService port="10020"/>
    </profile>
    <profile id="server2Nam">
      <tp1Server host="10.210.208.2"/>
      <rpc use="nam" watchTime="0"/>
      <nameService port="10000"/>
    </profile>
  </hitachi.opentp1.client>

</configuration>
```

4 UAP の作成と実行

この章では，Client .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（Connector .NET 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)　.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#
- 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#	Visual Basic
System.Void※	void	−
System.Byte	byte	Byte
System.Int16	short	Short
System.Int32	int	Integer
System.Int64	long	Long
System.String	string	String
System.Byte[]	byte[]	Byte()
System.Int16[]	short[]	Short()
System.Int32[]	int[]	Integer()
System.Int64[]	long[]	Long()
System.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#	Visual Basic
値渡し	var（省略可）	ByVal（省略可）
出力渡し	out	指定不可※
参照渡し	ref	ByRef

注※

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。

- 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。

(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利用するクラスは，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#	Visual Basic
System.Byte	byte	Byte
System.Int16	short	Short
System.Int32	int	Integer
System.Int64	long	Long
System.String	string	String
System.Byte[]	byte[]	Byte()
System.Int16[]	short[]	Short()
System.Int32[]	int[]	Integer()

共通型システム	プログラム言語	
	C#	Visual Basic
System.Int64[]	long[]	Long()
System.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。

(3) .NET インタフェース定義の定義例

.NET インタフェース定義の定義例を，開発言語ごとに示します。

(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) 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（Connector .NET 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(2)　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

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

CUP.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。

Extension .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/Extension for .NET Framework 使用の手引」，マニュアル「TP1/Connector for .NET Framework 使用の手引」を参照してください。なお，次のプログラム言語が使用できます。

- C#
- Visual Basic

4.2.1　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（CUP.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表に従って対応づけられます。

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#	Visual Basic
C#	なし	なし	ByVal
	out	out	ByRef※
	ref	ref	ByRef
Visual Basic	なし	なし	ByVal
	ByVal	なし	ByVal
	ByRef	ref	ByRef

注※

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。

4.2.2　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 TP1Client クラスのインスタンスごと，およびサービスグループごとにインスタンス化します。
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。

4.2.3　クライアントスタブ使用時のデータ長の計算方法

クライアントスタブを使用する場合は，入出力パラメタやカスタムレコードクラスと，入出力メッセージの間の変換を Client .NET が行います。ただし，送受信するユーザメッセージが RPC メッセージの最大長を超えないようにするためには，入力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。

(1)　データ型ごとの合計値を計算する

次に示す計算式を用いて，入力および出力で使用するデータ型ごとの合計値を計算してください。

入力メッセージ長＝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＋ 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＋ 512

計算式に代入する値を，次の表に示します。

表 4-6　計算式に代入する値（.NET インタフェース定義）

パラメタのデータ型	メッセージ上のサイズの最大値（単位：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[]	Σ（格納された文字数×2 ＋ 13）＋ 11

パラメタのデータ型	メッセージ上のサイズの最大値（単位：バイト）
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	Σ（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調整されるサイズを含みます。

(2) データトレースを使用する

Client .NET のトラブルシュート機能であるデータトレースを使用して，実際に送受信されたメッセージ長を確認できます。データトレースの詳細については，「2.9.3 データトレース」を参照してください。

4.2.4 .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
- リモート API 機能：使用（非オートコネクトモード）
- 構成ファイル：指定あり（プロファイル ID ＝"TP1Host1"）

なお，コメント中の(1)，(2)などは「4.2.2 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（CUP.NET，C#の場合（リモート API 機能使用時））

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Client;

namespace MyCompany
{
  public class CallerSample
  {
    public static void Main(string[] args)
    {
      try {
        TP1Client clt = new TP1Client();  // TP1Clientの生成
        IGyoumuAStub server = null;
        // (1) クライアントスタブの生成
        server = new IGyoumuAStub(clt, "GRP1");
        clt.OpenRpc("TP1Host1");      // RPCオープン
        clt.OpenConnection();         // 常設コネクションの確立
        // (2) 入力データの設定
        string[] ids = {"data1", "data2", "data3"};
        // (3) 同期応答型RPCに設定
        server.Flags = TP1ClientFlags.DCNOFLAGS;
        // (4) service3を呼び出す
        int ret = server.Service3(3, ref ids);
        clt.CloseConnection();        // 常設コネクションの解放
        clt.CloseRpc();               // 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 (TP1ClientException exp) {
        // Client .NETが検知したエラー
      } catch (TP1Exception exp) {
        // その他スタブなど検知したエラー
      } catch (Exception exp) {
        // 予期しない例外
      }
    }
  }
}
```

(3) クライアントスタブの使用例（CUP.NET，Visual Basic の場合（リモート API 機能使用時））

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Client

Namespace MyCompany
  Public Class CallerSample
    Public Shared Sub Main(ByVal args() As String)
      Dim clt As TP1Client
```

```
      Dim server As IGyoumuAStub
      Dim ret As Integer
      Dim ids() As String
      Try
        clt = New TP1Client()  ' TP1Clientの生成
        '(1) クライアントスタブの生成
        server = New IGyoumuAStub(clt, "GRP1")
        clt.OpenRpc("TP1Host1")       ' RPCオープン
        clt.OpenConnection()          ' 常設コネクションの確立
        '(2) 入力データの設定
        ids = New String() {"data1", "data2", "data3"}
        '(3) 同期応答型RPCに設定
        server.Flags = TP1ClientFlags.DCNOFLAGS
        '(4) service3を呼び出す
        ret = server.Service3(3, ids)
        clt.CloseConnection()         ' 常設コネクションの解放
        clt.CloseRpc()                ' 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 TP1ClientException
        ' Client .NETが検知したエラー
      Catch exp As TP1Exception
        ' その他スタブなどが検知したエラー
      Catch exp As Exception
        ' 予期しない例外
      End Try
    End Function
  End Class
End Namespace
```

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

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

Extension .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/Extension for .NET Framework 使用の手引」，マニュアル「TP1/Connector for .NET Framework 使用の手引」を参照してください。なお，次のプログラム言語が使用できます。

- C#
- Visual Basic

4.3.1　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（CUP.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。

4.3.2　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 TP1Client クラスのインスタンスごと，およびサービスグループごとにインスタンス化します。
2. 入力用，および出力用の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。

4.3.3　クライアントスタブ使用時のデータ長の計算方法

クライアントスタブを使用する場合は，入出力パラメタやカスタムレコードクラスと，入出力メッセージの間の変換を Client .NET が行います。ただし，送受信するユーザメッセージが RPC メッセージの最大長を超えないようにするためには，入力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。

(1)　データ型ごとの合計値を計算する

次に示す計算式を用いて，入力および出力で使用するデータ型ごとの合計値を計算してください。

入力メッセージ長＝データ型定義の各メンバのデータ長の合計

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

計算式に代入する値を，次の表に示します。

表 4-7　計算式に代入する値（サービス定義）

データ型定義のメンバのデータ型	メッセージ上のサイズ（単位：バイト）
char	1
short	2
int	4
long	4
char[a][b]	a×b
short[a]	2×a
byte[a]	1×a
struct	構造体の各メンバのサイズの合計
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

注

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。

(2)　カスタムレコードクラスのソースコードを確認する

カスタムレコードクラスのソースコードの private const フィールド_length を参照して，計算後のメッセージ上のサイズが確認できます。

(3) データトレースを使用する

Client .NET のトラブルシュート機能であるデータトレースを使用して，実際に送受信されたメッセージ長を確認できます。データトレースの詳細については，「2.9.3 データトレース」を参照してください。

4.3.4 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 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
- サービス定義名称：GYOUMU1
- 呼び出すサービス名（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
- リモート API 機能：未使用

なお，コメント中の(1)，(2)などは「4.3.2 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（CUP.NET，C#の場合（リモート API 機能未使用時））

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Client;

namespace MyCompany
{
  public class Caller2Sample
  {
    public static void Main(string[] args)
    {
      try {
        TP1Client clt = new TP1Client();  // TP1Clientの生成
        GYOUMU1Stub server = null;
        // (1) クライアントスタブの生成
        server = new GYOUMU1Stub(clt, "SVGRP1");
        clt.OpenRpc();              // RPCオープン
        // (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 = TP1ClientFlags.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();
        clt.CloseRpc();             // RPCクローズ
      } catch (TP1ClientException exp) {
        // Client .NETが検知したエラー
      } catch (TP1Exception exp) {
        // その他スタブなどが検知したエラー
      } catch (Exception exp) {
        // 予期しない例外
      }
    }
  }
}
```

(4) クライアントスタブの使用例（CUP.NET，Visual Basic の場合（リモート API 機能未使用時））

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Client

Namespace MyCompany
  Public Class Caller2Sample
    Public Shared Sub Main(ByVal args() As String)
```

```
      Dim clt As TP1Client
      Dim server As GYOUMU1Stub
      Dim name As String
      Dim inRecord As in_data
      Dim outRecord As out_data
      Try
        clt = New TP1Client()  ' TP1Clientの生成
        '(1) クライアントスタブの生成
        server = New GYOUMU1Stub(clt, "SVGRP1")
        clt.OpenRpc()             ' RPCオープン
        '(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 = TP1ClientFlags.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()
        clt.CloseRpc()            ' RPCクローズ
      Catch exp As TP1ClientException
        ' Client .NETが検知したエラー
      Catch exp As TP1Exception
        ' その他スタブなどが検知したエラー
      Catch exp As Exception
        ' 予期しない例外
      End Try
    End Sub
  End Class
End Namespace
```

4.4　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

CUP.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。

Extension .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/Extension for .NET Framework 使用の手引」，マニュアル「TP1/Connector for .NET Framework 使用の手引」を参照してください。なお，次のプログラム言語が使用できます。

- C#
- Visual Basic

4.4.1　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，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 CUP.NET が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 CUP.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，CUP.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。

4.4.2　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
- リモート API 機能：未使用

- 構成ファイル：指定あり（プロファイル ID="TP1Host1"）

(1) Call メソッドの使用例（CUP.NET，C#の場合（リモート API 機能未使用時））

```
using System;
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Client;

namespace MyCompany
{
  public class MyApplication1
  {
    ...
    public static void Main(string[] args)
    {
      try {
        TP1Client clt = new TP1Client();  // TP1Clientの生成
        clt.OpenRpc("TP1Host1");          // RPCオープン

        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
        clt.Call("SVGRP1", "SERV1", inData, inLen,
            outData, ref outLen, TP1ClientFlags.DCNOFLAGS);
        outStr = enc.GetString(outData, 0, outLen);

        clt.CloseRpc();                   // RPCクローズ
      } catch (TP1ClientException exp) {
        // Client .NETが検知したエラー
      } catch (TP1Exception exp) {
        // Client .NET（OpenTP1共通クラス）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（CUP.NET，Visual Basic の場合（リモート API 機能未使用時））

```
Imports System
Imports Hitachi.OpenTP1
Imports Hitachi.OpenTP1.Client

Namespace MyCompany
  Public Class MyApplication1
    ...
    Public Shared Sub Main(ByVal args() As String)
      Dim clt As TP1Client
      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
        clt = New TP1Client()             ’TP1Clientの生成
        clt.OpenRpc(”TP1Host1”)           ’RPCオープン

        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
        clt.Call(”SVGRP1”, ”SERV1”, inData, inLen,
            outData, outLen, TP1ClientFlags.DCNOFLAGS)
        outStr = enc.GetString(outData, 0, outLen)

        clt.CloseRpc()                    ’RPCクローズ
      Catch exp As TP1ClientException
        ’Client .NETが検知したエラー
      Catch exp As TP1Exception
        ’Client .NET（OpenTP1共通クラス）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　通信形態によるメソッドの発行手順

使用する通信形態によって，メソッドの発行順序が異なります。通信形態別のメソッド発行順序について，次に説明します。

4.5.1　非オートコネクトモードでリモート API 機能を使用する場合

1. TP1Client クラスのインスタンスを生成します。
2. 構成定義を読み込む場合は，OpenRpc メソッドを発行します。
   OpenRpc メソッドを発行しない場合，すべての構成定義要素はデフォルト値が有効になります。
3. コネクション確立のため OpenConnection メソッドを発行します。
4. RPC 要求をします（何度でも発行できます）。
5. CloseConnection メソッドを発行します。
   再度 RPC 要求をする場合は，3.から繰り返します。
6. 構成定義をリセットする場合は，CloseRpc メソッドを発行します。
   再度 RPC 要求をする場合は，2.または 3.から繰り返します。

4.5.2　オートコネクトモードでリモート API 機能を使用する場合

1. TP1Client クラスのインスタンスを生成します。
2. OpenRpc メソッドを発行します。
3. RPC 要求をします（何度でも発行できます）。
4. CloseConnection メソッドを発行します。
   再度 RPC 要求をする場合は，3.から繰り返します。
5. 構成定義をリセットする場合は，CloseRpc メソッドを発行します。
   再度 RPC 要求をする場合は，2.または 3.から繰り返します。

4.5.3　ネームサービスを使用した RPC の場合

1. TP1Client クラスのインスタンスを生成します。
2. OpenRpc メソッドを発行します。

3. RPC 要求をします（何度でも発行できます）。

4. CloseRpc メソッドを発行します。
   再度 RPC 要求をする場合は，2.から繰り返します。

4.5.4　スケジューラダイレクト機能を使用した RPC の場合

1. TP1Client クラスのインスタンスを生成します。

2. OpenRpc メソッドを発行します。

3. RPC 要求をします（何度でも発行できます）。

4. CloseRpc メソッドを発行します。
   再度 RPC 要求をする場合は，2.から繰り返します。

4.5.5　TCP/IP 通信機能（一方送信）を使用する場合

1. TP1Client クラスのインスタンスを生成します。

2. OpenRpc メソッドを発行します。

3. Send メソッドを発行します（何度でも発行できます）。

4. 構成定義をリセットする場合は，CloseRpc メソッドを発行します。
   再度 TCP/IP 通信をする場合は，2.から繰り返します。

4.5.6　TCP/IP 通信機能（一方受信）を使用する場合

1. TP1Client クラスのインスタンスを生成します。

2. OpenRpc メソッドを発行します。

3. Receive メソッドを発行します（何度でも発行できます）。

4. 構成定義をリセットする場合は，CloseRpc メソッドを発行します。
   再度 TCP/IP 通信をする場合は，2.から繰り返します。

4.5.7 TCP/IP 通信機能（送受信，CUP.NET がクライアント）を使用する場合

1. TP1Client クラスのインスタンスを生成します。
2. OpenRpc メソッドを発行します。
3. Send メソッドを発行します。
4. Receive メソッドを発行します。
   3.，4.を繰り返すことができます。
5. 構成定義をリセットする場合は，CloseRpc メソッドを発行します。
   再度 TCP/IP 通信をする場合は，2.から繰り返します。

4.5.8 TCP/IP 通信機能（送受信，CUP.NET がサーバ）を使用する場合

1. TP1Client クラスのインスタンスを生成します。
2. OpenRpc メソッドを発行します。
3. Receive メソッドを発行します。
4. Send メソッドを発行します。
   3.，4.を繰り返すことができます。
5. 構成定義をリセットする場合は，CloseRpc メソッドを発行します。
   再度 TCP/IP 通信をする場合は，2.から繰り返します。

4.5.9 動的定義変更機能を使用する場合

1. TP1Client クラスのインスタンスを生成します。
2. 構成定義を読み込む場合は，OpenRpc メソッドを発行します。
   OpenRpc メソッドを発行しない場合，すべての構成定義要素はデフォルト値が有効になります。
3. 動的定義変更用の SetXxxx メソッドを発行します。
   Xxxx は変更する構成定義項目に対応した名称になります。
4. 通信形態に従って通信します。
   3.，4.を繰り返すことができます。

5. 構成定義をリセットする場合は，CloseRpc メソッドを発行します。

再度通信をする場合は，2.または 3.から繰り返します。

4.6　UAP 作成時の注意事項

4.6.1　CUP.NET の実行環境と留意点

Client .NET は CUP.NET で利用するクラスライブラリをアセンブリ（DLL 形式）として提供します。CUP.NET は，アプリケーションの形態によってさまざまなランタイムホストで実行されます。

使用するランタイムホストごとに，アプリケーションドメインの構成，アセンブリの配置と検索方法，セキュリティポリシー，スレッド構成，およびアプリケーション構成ファイルの配置が異なるため，CUP.NET を実行する前に，使用するランタイムホストの仕様を確認してください。

なお，複数のスレッドが同時に実行される環境で使用する場合には，TP1Client クラスのインスタンスを複数のスレッドで同時に使用しないよう適切な排他制御が必要です。

4.6.2　例外の捕捉とエラーの判定

CUP.NET では，Client .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。

Client .NET からは，次の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
	TP1MarshalException	スタブでのデータ変換エラー
Hitachi.OpenTP1.Client	TP1ClientException	クラスライブラリでのエラー検知
	ErrXxxxException （TP1ClientException のサブクラス）	クラスライブラリでの各種エラー検知※

注※

詳細については，「6.　クラスリファレンス」を参照してください。

(1)　TP1Exception

OpenTP1 のすべての例外の基底クラスです。個別の例外を補捉しないで，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 RPC 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。

(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，RPC 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，クライアント側に RPC 要求のエラーが通知されます。

(4) TP1MarshalException

クライアントスタブを使用して RPC 要求をした場合に，データ変換エラーが発生したことを示す例外です。クライアントとサーバで.NET インタフェース定義が一致していない場合や，サービス定義（データ型定義）に誤りがあった場合などに発生します。

(5) TP1ClientException

Client .NET が提供するクラスライブラリの各クラスおよびメソッドでエラーを検知した場合に発生する，すべての例外の基底クラスです。

各例外のサブクラス，および各メソッドでの発生条件や内容の詳細については，「6. クラスリファレンス」を参照してください。

(6) ErrXxxxException（TP1ClientException のサブクラス）

ErrXxxxException 例外については，「6. クラスリファレンス」を参照してください（Xxxx は，例外クラス名によって異なります）。

(7) その他の例外

TP1Client クラスのインスタンスを使用するスレッドが Thread.Abort メソッドを受けた場合，TP1Client クラスのメソッドは ThreadAbortException 例外が発生します。この例外が発生した場合，次の点に注意してください。

- ThreadAbortException 例外が発生したとき，Client .NET は製品が保持するリソースの初期化を行います。そのため，例外が発生したインスタンスを再使用する場合は，必ず OpenRpc メソッドから再実行してください。

なお，ThreadAbortException 例外の影響で，Client .NET 以外のリソースで問題が発生するおそれがあります。その場合インスタンスの再生成，またはプロセス再起動をしてください。

- ThreadAbortException 例外を Thread.ResetAbort メソッドでキャンセルする場合，そのスレッド内で TP1Client クラスのインスタンスを再使用しないでください。再使用した場合，予期しない例外を返すことがあります。
- トランザクション制御機能を使用するスレッドで ThreadAbortException 例外が発生した場合，この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，Client .NET から OpenTP1 のトランザクションを制御できなくなります。

  ThreadAbortException 例外の発生したタイミングによってトランザクションの決着結果は次のようになります。

  Commit メソッド発行前

  グローバルトランザクションはロールバックされます。

  Commit メソッド発行中

  CUP.NET では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。

  同期点の結果は，TP1/Server のメッセージログファイルを参照してください。詳細は「2.3.5　障害発生時のトランザクションの同期点を検証する方法」を参照してください。

  Commit メソッド発行後

  グローバルトランザクションはコミット済みです。

4.6.3　クラスライブラリを使用する場合の注意事項

クラスライブラリを使用する場合，次の点に注意してください。

- 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。
- Client .NET が提供するクラスライブラリはマルチスレッド環境で使用できますが，スレッドセーフではありません。マルチスレッド環境で使用する場合は，TP1Client クラスのインスタンスをスレッド間で同時に使用しないよう，適切に排他をしてください。

  マルチスレッド環境で同時に使用する場合は，Connector .NET の使用を推奨します。
- Client .NET が提供するクラスライブラリは SPP.NET，SUP.NET 上では使用しないでください。これらの UAP 上では，Extension .NET が提供するクラスライブラリを使用してください。

4.7　UAP の実行

サンプルプログラムの使用方法とノータッチデプロイメントによる CUP.NET の配布の手順について説明します。

4.7.1　サンプルプログラムの使用方法

Client .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，%DCCLNDIR%は Client .NET のインストールディレクトリを示します。

(1)　ディレクトリ構成

Client .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
%DCCLNDIR%¥examples	Client .NET のサンプルプログラム格納ディレクトリ
%DCCLNDIR%¥examples¥C#	C#のサンプルプログラム格納ディレクトリ
%DCCLNDIR%¥examples¥VB.NET	Visual Basic のサンプルプログラム格納ディレクトリ

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の名称と，格納されているサンプルプログラムの種類について，開発言語別に次の表に示します。

表 4-10　サンプルプログラムの種類（C#，および Visual Basic の場合）

名称	説明
CUPBIN	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 CUP.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。
CUPCR	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 CUP.NET です。
CUPIF	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 CUP.NET です。

(2)　サンプルプログラムのビルド方法

Client .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。

(a)　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#，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

2. 環境変数 DCCLNDIR に，Client .NET のインストールディレクトリを設定します。

【例】

```
set DCCLNDIR=C:¥Program Files¥HITACHI¥TP1Client for .NET Framework
```

(b) サンプルプログラムをまとめてビルドする方法

サンプルプログラムをまとめてビルドする方法は，どの開発言語のサンプルプログラムをビルドするかによって異なります。それぞれの場合に分けて方法を示します。

- C#，および Visual Basic のサンプルプログラムをまとめてビルドする場合
  %DCCLNDIR%¥examples¥build_all.bat を実行します。
- 言語別にサンプルプログラムをまとめてビルドする場合
  - C#の場合
    %DCCLNDIR%¥examples¥C#¥build_cs.bat を実行します。
  - Visual Basic の場合
    %DCCLNDIR%¥examples¥VB.NET¥build_vb.bat を実行します。

(c) 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】Visual Basic の CUPIF サンプルの場合

```
%DCCLNDIR%¥examples¥VB.NET¥CUPIF¥build.bat
```

(3) サンプルプログラムの実行手順

Client .NET のサンプルプログラムの実行手順を説明します。

(a) SPP.NET のサンプルプログラムの起動

各サンプルプログラムを実行するためには，Extension .NET のサンプルプログラムのうち，各サンプルプログラムが利用する SPP.NET を起動します。各サンプルプログラムが利用する SPP.NET を次の表に示します。

表 4-11 サンプルプログラムが利用する SPP.NET

サンプルプログラムの種類	利用する SPP.NET	ユーザサーバ名
C#¥CUPBIN	C#¥SPPBIN	CSSPPBIN
C#¥CUPCR	C#¥SPPBIN	CSSPPBIN

サンプルプログラムの種類	利用する SPP.NET	ユーザサーバ名
C#¥CUPIF	C#¥SPPIF	CSSPPIF
VB.NET¥CUPBIN	VB.NET¥SPPBIN	VBSPPBIN
VB.NET¥CUPCR	VB.NET¥SPPBIN	VBSPPBIN
VB.NET¥CUPIF	VB.NET¥SPPIF	VBSPPIF

SPP.NET の起動方法については，マニュアル「TP1/Extension for .NET Framework 使用の手引」を参照してください。

(b) 構成ファイルの変更

利用する OpenTP1 サーバ環境に応じて，構成ファイルの Client .NET 構成定義を変更します。例えば，Visual Basic の CUPIF サンプルで，OpenTP1 のホスト名やネームサービスのポート番号などを変更する場合は，%DCCLNDIR%¥examples¥VB.NET¥CUPIF¥VBCUPIF.exe.config ファイルの tp1Server 要素，rpc 要素および nameService 要素を変更します。

(c) CUP.NET の実行

ビルドされた CUP.NET のアセンブリを実行します。

【例】Visual Basic の CUPIF サンプルの場合
%DCCLNDIR%¥examples¥VB.NET¥CUPIF¥VBCUPIF.exe

4.7.2 ノータッチデプロイメントによる CUP.NET の配布

ノータッチデプロイメントを利用して CUP.NET を配布する方法を説明します。

(1) CUP.NET および Client .NET を配布する場合

CUP.NET および Client .NET を配布する場合は，配布先のクライアントマシンに Client .NET をインストールしないでください。配布先のクライアントマシンに Client .NET をインストールすると，ノータッチデプロイメントによって起動した CUP.NET はインストール済みの Client .NET を参照します。そのため，CUP.NET と一緒に配置した Web サーバ上の Client .NET が参照されません。

CUP.NET および Client .NET を配布する手順を次に示します。なお，次に示す手順では，CUP.NET および Client .NET を更新するたびに配布 URL が変更されるため，一つのクライアントマシンで複数バージョンの Client .NET を使用することができます。

(a) 配布環境の構築手順

1. Web サーバの公開ディレクトリ下（例：http://localhost/gyoumu）に，次に示すファイルを配置します。

CUP.NET

- 実行アセンブリ（*.exe）
- ユーザ作成のクラスライブラリ（*.dll）
- Client .NET 構成定義を記述したアプリケーション構成ファイル（*.exe.config）

Client .NET

- Client .NET のクラスライブラリ（Hitachi.OpenTP1.Client.dll）
  Hitachi.OpenTP1.Client.dll は，Web サーバにインストールした Client .NET のインストールディレクトリ下の bin ディレクトリからコピーしてください。

2. 配布先であるクライアントマシンのランタイムセキュリティポリシーに対して，CUP.NET の実行時に必要となるアクセス許可を設定します。
   必要なアクセス許可については，「2.12.2　セキュリティポリシーの設定」を参照してください。

3. 配布先であるクライアントマシンで，Microsoft Internet Explorer 5.01 以上を起動します。

4. 起動した Microsoft Internet Explorer から，Web サーバの公開ディレクトリ下に配置した CUP.NET の実行アセンブリの URL（例：http://localhost/gyoumu/cup.exe）にアクセスしてノータッチデプロイメントを行います。

5. クライアントマシンに CUP.NET および Client .NET がキャッシュ（配布）されて，CUP.NET が起動します。

6. クライアントマシンで再び CUP.NET を起動する場合は，手順 3.と 4.を繰り返します。

(b)　配布環境の CUP.NET および Client .NET の更新手順

Client .NET は，Client .NET がバージョンアップしても，既存の CUP.NET に対するリコンパイルを不要とし，下位互換性を保証することを目的として，Client .NET のクラスライブラリ（Hitachi.OpenTP1.Client.dll）のアセンブリバージョンを「7.0.0.0」固定としています。したがって，Web サーバの公開ディレクトリ下に配置した Hitachi.OpenTP1.Client.dll を更新して，再びクライアントマシンで CUP.NET をノータッチデプロイメントしても，起動した CUP.NET はすでにキャッシュ（配布）されているバージョンアップ前の Hitachi.OpenTP1.Client.dll を参照します。

クライアントマシンに，更新後の Hitachi.OpenTP1.Client.dll を反映させるために，CUP.NET および Client .NET のファイルを更新する手順を次に示します。

1. Web サーバに新しい公開ディレクトリ（URL）を用意します。

   【例】

   既存の公開ディレクトリ：http://localhost/gyoumu
   新しく用意する公開ディレクトリ：http://localhost/gyoumu_ver2

2. 新しい公開ディレクトリ下（例：http://localhost/gyoumu_ver2）に，更新したファイルを含む CUP.NET および Client .NET のファイル一式を配置します。

3. 配布先であるクライアントマシンで， Microsoft Internet Explorer 5.01 以上を起動します。

4. 起動した Microsoft Internet Explorer から，新しい公開ディレクトリ下に配置した CUP.NET の実行アセンブリの URL（例：http://localhost/gyoumu_ver2/cup.exe）にアクセスしてノータッチデプロイメントを行います。

5. クライアントマシンに，更新後の CUP.NET および Client .NET がキャッシュ（配布）されて，CUP.NET が起動します。

6. クライアントマシンで更新前の CUP.NET を起動したい場合は，既存の公開ディレクトリ下に配置した CUP.NET の実行アセンブリの URL（例：http://localhost/gyoumu/cup.exe）にアクセスしてノータッチデプロイメントを行います。

(2) CUP.NET だけを配布する場合

CUP.NET だけを配布する場合は，配布先であるクライアントマシンに Client .NET をインストールしておく必要があります。

Client .NET は，1 台のマシンに複数をインストールできないため，side-by-side の実行によって複数バージョンを同時に使用することはできません。

(a) 配布環境の構築手順

1. Web サーバの公開ディレクトリ下（例：http://localhost/gyoumu）に，次に示すファイルを配置します。

   CUP.NET

   - 実行アセンブリ（*.exe）
   - ユーザ作成のクラスライブラリ（*.dll）
   - Client .NET 構成定義を記述したアプリケーション構成ファイル（*.exe.config）※

2. 「(1) CUP.NET および Client .NET を配布する場合」に示した配布環境の構築手順の 2.～6.と同様に操作します。

   このとき，手順 5.でクライアントマシンにキャッシュ（配布）されるのは，CUP.NET だけです。

注※

次に示すように，カスタム構成セクション宣言の<section>要素に requirePermission 属性を追加し，その値を false に設定してください。

```
<configSections>
  <section
     name="hitachi.opentp1.client"
     type="Hitachi.OpenTP1.Common.Util.ProfileSectionHandler,
     Hitachi.OpenTP1.Client,Version=7.0.0.0,Culture=neutral,
     PublicKeyToken=2440cf5f0d80c91c,Custom=null"
     requirePermission="false"/>
</configSections>
```

(b)　配布環境の CUP.NET の更新手順

配布環境の CUP.NET を更新する手順については，Microsoft が提供する.NET Framework に関するドキュメントを参照してください。

(3)　配布した CUP.NET および Client .NET の削除方法

ノータッチデプロイメントによってクライアントマシンにキャッシュ（配布）されたファイルを削除する手順を説明します。次に示す操作を行うと，ノータッチデプロイメントによってキャッシュ（配布）されたすべてのファイルが削除されます。

1. コマンドプロンプトから，.NET Framework SDK に含まれている Gacutil.exe に/cdl オプションを付加して実行します。

2. Microsoft Internet Explorer でインターネット一時ファイルを削除します。

   Microsoft Internet Explorer を起動し，「ツール」メニューの「インターネット オプション」を選択すると表示されるウィンドウの「全般」タブで，「インターネット一時ファイル」エリアの「ファイルの削除」ボタンをクリックしてください。

5 運用コマンド

この章では，Client .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

Client .NET で使用できる運用コマンドを，次の表に示します。

表 5-1　運用コマンドの種類（Client .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。
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#
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。

また，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 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。

●-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。

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クライアントスタブクラス名称〉.〈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〉

TP1/Server への送信データを，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〉

TP1/Server から受け取った受信データを，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》

TP1/Server への送信データを，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》

TP1/Server から受け取った受信データを，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。

6 クラスリファレンス

この章では，Client .NET で使用するクラスについて説明します。

Client .NET で利用できるクラス

Client .NET で利用できるクラスの一覧を次に示します。

表 6-1　Client .NET で利用できるクラスの一覧

名前空間	クラス名	説明
Hitachi.OpenTP1.Client	DCRpcBindTbl	通信先情報を管理するクラスです。CallTo メソッドで使用します。
	ErrAcceptCanceledException	Client .NET の各種例外クラスです。
	ErrBufferOverflowException	
	ErrClientTimedOutException	
	ErrCollisionMessageException	
	ErrConnfreeException	
	ErrConnRefusedException	
	ErrFatalException	
	ErrHazardException	
	ErrHazardNoBeginException	
	ErrHeuristicException	
	ErrHeuristicNoBeginException	
	ErrHostUndefException	
	ErrInitializingException	
	ErrInvalidArgsException	
	ErrInvalidMessageException	
	ErrInvalidPortException	
	ErrInvalidReplyException	
	ErrIOErrException	
	ErrMarshalException	
	ErrMessageTooBigException	
	ErrNetDownAtClientException	
	ErrNetDownAtServerException	
	ErrNetDownException	
	ErrNoBeginException	
	ErrNoBufsAtServerException	

名前空間	クラス名	説明
Hitachi.OpenTP1.Client	ErrNoBufsException	Client .NET の各種例外クラスです。
	ErrNoSuchServiceException	
	ErrNoSuchServiceGroupException	
	ErrNotTrnExtendException	
	ErrNotUpException	
	ErrProtoException	
	ErrReplyTooBigException	
	ErrRMException	
	ErrRollbackException	
	ErrRollbackNoBeginException	
	ErrSecchkException	
	ErrServerBusyException	
	ErrServerTimedOutException	
	ErrServiceClosedException	
	ErrServiceNotUpException	
	ErrServiceTerminatedException	
	ErrServiceTerminatingException	
	ErrSyserrAtServerException	
	ErrSyserrException	
	ErrTestmodeException	
	ErrTimedOutException	
	ErrTMException	
	ErrTrnchkException	
	ErrTrnchkExtendException	
	ErrVersionException	
	TP1Client	TP1/Server のサーバに対してサービス要求を行うクラスです。
	TP1ClientError	Client .NET で使用するエラーコードを定義するクラスです。
	TP1ClientException	Client .NET が返すすべての例外のスーパークラスです。

名前空間	クラス名	説明
Hitachi.OpenTP1.Client	TP1ClientFlags	Client .NET で使用するメソッドのパラメタで用いるフラグを定義するクラスです。
Hitachi.OpenTP1	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。
	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。
	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パラメタです。
	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。
	UByteArrayHolder	System.Byte 配列を保持するホルダークラスです。

名前空間	クラス名	説明
Hitachi.OpenTP1	UByteHolder	System.Byte 値を保持するホルダークラスです。

DCRpcBindTbl

DCRpcBindTbl の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Client.DCRpcBindTbl
```

説明

通信先情報を管理するクラスです。CallTo メソッドで使用します。

コンストラクタの一覧

名称	説明
DCRpcBindTbl()	通信先情報を管理するクラスのインスタンスを作成します。

コンストラクタの詳細

●DCRpcBindTbl

説明

通信先情報を管理するクラスのインスタンスを作成します。

宣言

【C#の場合】

```
public DCRpcBindTbl(
);
```

【Visual Basic の場合】

```
Public New( _
)
```

パラメタ

なし

例外

なし

ErrAcceptCanceledException

ErrAcceptCanceledException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrAcceptCanceledException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

一方通知受信待ち状態が解除されました。

ErrBufferOverflowException

ErrBufferOverflowException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrBufferOverflowException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

不正なメッセージを受信しました。メッセージ長またはセグメント情報が不正です。

ErrClientTimedOutException

ErrClientTimedOutException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrTimedOutException
          +- Hitachi.OpenTP1.Client.ErrClientTimedOutException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

Client .NET 側でタイムアウトが発生しました。

ErrCollisionMessageException

ErrCollisionMessageException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrCollisionMessageException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

送受信メッセージの衝突が発生しました。

ErrConnfreeException

ErrConnfreeException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrConnfreeException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

- Call メソッドがこの例外を返した場合
  rap サーバとの常設コネクションが切断されました。
- Begin メソッド，CommitChained メソッド，RollbackChained メソッド，Commit メソッド，または Rollback メソッドがこの例外を返した場合
  CUP.NET 実行プロセス側から常設コネクションが解放されました。
- Receive メソッドがこの例外を返した場合
  MHP からコネクションが解放されました。

ErrConnRefusedException

ErrConnRefusedException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrConnRefusedException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

リモートのアドレスおよびポートに対してソケットの接続を試行しているときに，エラーが発生したことを通知します。一般的には，接続がリモートで拒否された（リモートのアドレスとポートで待機中のプロセスがない）ことが原因です。

ErrFatalException

ErrFatalException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrFatalException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

- Client .NET 構成定義の指定に誤りがあります。
- TP1/Server との通信環境の初期化に失敗しました。
- Client .NET 構成定義の次の属性に指定したポートが他プロセスで使用中です。
  <rpc>要素の cupRecvPort 属性
  <tcpip>要素の recvPort 属性

ErrHazardException

ErrHazardException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrHazardException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

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。この例外が返される原因になった 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，TP1/Server のメッセージログファイルを参照してください。この例外が返されたあとも，このプロセスはトランザクション下にあり，グローバルトランザクションの範囲内です。

ErrHazardNoBeginException

ErrHazardNoBeginException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrHazardNoBeginException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

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。この例外が返される原因になった 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，TP1/Server の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。この例外が返されたあと，このプロセスはトランザクション下にありません。

ErrHeuristicException

ErrHeuristicException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrHeuristicException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

グローバルトランザクションがヒューリスティック決定であるため，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。

この例外は，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い場合に返されます。この例外が返される原因になった 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，TP1/Server のメッセージログファイルを参照してください。この例外が返されたあとも，このプロセスはトランザクション下にあり，グローバルトランザクションの範囲内です。

ErrHeuristicNoBeginException

ErrHeuristicNoBeginException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrHeuristicNoBeginException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

グローバルトランザクションがヒューリスティック決定であるため，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。

この例外は，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い場合に返されます。この例外が返される原因になった 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，TP1/Server の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。この例外が返されたあと，このプロセスはトランザクション下にありません。

ErrHostUndefException

ErrHostUndefException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrHostUndefException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

- SetTP1Server メソッドがこの例外を返した場合
  host パラメタの指定に誤りがあります。
- OpenConnection メソッドがこの例外を返した場合
  rap リスナーのホスト名が，Client .NET 構成定義の<TP1Server>要素に指定されていません。または，host パラメタの指定に誤りがあります。
- Call メソッドがこの例外を返した場合
  通信先となる TP1/Server のホスト名が，Client .NET 構成定義の<TP1Server>要素に指定されていないか，または指定に誤りがあります。
- Begin メソッドがこの例外を返した場合
  通信先となる TP1/Server のホスト名が，Client .NET 構成定義の<TP1Server>要素に指定されていないか，または指定に誤りがあります。
- Send メソッドがこの例外を返した場合
  hostname パラメタの指定に誤りがあるか，または hostname パラメタと Client .NET 構成定義の<tcpip>要素の sendHost 属性の両方に，ホスト名が指定されていません。

ErrInitializingException

ErrInitializingException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrInitializingException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

サービス要求したノードにある TP1/Server は開始処理中です。

ErrInvalidArgsException

ErrInvalidArgsException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrInvalidArgsException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

メソッドのパラメタの指定に誤りがあります。

ErrInvalidMessageException

ErrInvalidMessageException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrInvalidMessageException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

不正なメッセージを受信しました。メッセージ長またはセグメント情報が不正です。

ErrInvalidPortException

ErrInvalidPortException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrInvalidPortException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

- SetTP1Server メソッド，OpenConnection メソッド，または Begin メソッドがこの例外を返した場合
  port パラメタの指定に誤りがあります。
- Call メソッドがこの例外を返した場合
  - リモート API 機能を使用した RPC を行う場合，Client .NET 構成定義の<rapService>要素の port 属性が指定されていません。
  - スケジューラダイレクト機能を使用した RPC を行う場合，Client .NET 構成定義の <scheduleService>要素の port 属性が指定されていません。
- Receive メソッドまたは Send メソッドがこの例外を返した場合
  portnum パラメタの指定に誤りがあります。

ErrInvalidReplyException

ErrInvalidReplyException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrInvalidReplyException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

サービスメソッドまたはサービス関数が返した応答の長さが，1 から DCRPC_MAX_MESSAGE_SIZE で指定した値までの範囲にありません。

ErrIOErrException

ErrIOErrException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrIOErrException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

何らかの入出力例外が発生しました。詳細は各メソッドの例外および注意事項を参照してください。

ErrMarshalException

ErrMarshalException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrMarshalException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

カスタムレコードのマーシャリング/アンマーシャリング処理に失敗しました。

ErrMessageTooBigException

ErrMessageTooBigException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrMessageTooBigException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

Call メソッドの in_length パラメタに指定した入力パラメタ長が，最大値を超えています。

ErrNetDownAtClientException

ErrNetDownAtClientException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrNetDownException
          +- Hitachi.OpenTP1.Client.ErrNetDownAtClientException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

TP1/Server と CUP.NET の間でネットワーク障害が発生しました。

ErrNetDownAtServerException

ErrNetDownAtServerException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrNetDownException
          +- Hitachi.OpenTP1.Client.ErrNetDownAtServerException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

TP1/Server と，SPP.NET または SPP との間でネットワーク障害が発生しました。

ErrNetDownException

ErrNetDownException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrNetDownException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

- ネットワーク障害が発生しました。
- 通信先の TP1/Server が稼働していません。

ErrNoBeginException

ErrNoBeginException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrNoBeginException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

コミットまたは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。この例外が返されたあと，このプロセスはトランザクション下にありません。

ErrNoBufsAtServerException

ErrNoBufsAtServerException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrNoBufsAtServerException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

指定したサービスでメモリ不足が発生しました。

ErrNoBufsException

ErrNoBufsException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrNoBufsException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

メモリ不足が発生しました。

ErrNoSuchServiceException

ErrNoSuchServiceException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrNoSuchServiceException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

service パラメタに指定したサービス名は定義されていません。

ErrNoSuchServiceGroupException

ErrNoSuchServiceGroupException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrNoSuchServiceGroupException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

group パラメタに指定したサービスグループ名は定義されていません。

ErrNotTrnExtendException

ErrNotTrnExtendException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrSyserrException
          +- Hitachi.OpenTP1.Client.ErrNotTrnExtendException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

トランザクション処理の連鎖 RPC を使用したあとで，flags パラメタに DCRPC_TPNOTRAN を指定したサービスを要求しています。

ErrNotUpException

ErrNotUpException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrNotUpException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

指定したサービスが存在するノードの TP1/Server が稼働していません。

ErrProtoException

ErrProtoException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrProtoException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

メソッドの発行順序に誤りがあります。

ErrReplyTooBigException

ErrReplyTooBigException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrReplyTooBigException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

サーバから返された応答の長さが，CUP.NET で用意した領域（out_data パラメタの指定値）の長さを超えています。

ErrRMException

ErrRMException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrRMException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

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。

ErrRollbackException

ErrRollbackException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrRollbackException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

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。

- CommitChained メソッドでこの例外が返された場合
  このプロセスはトランザクション下にあります。
- Commit メソッドでこの例外が返された場合
  このプロセスはトランザクション下にありません。

ErrRollbackNoBeginException

ErrRollbackNoBeginException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrRollbackNoBeginException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

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。新しいトランザクションは開始できませんでした。この例外が返されたあと，このプロセスはトランザクション下にありません。

ErrSecchkException

ErrSecchkException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrSecchkException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

サービス要求先の 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。サービス要求した CUP.NET には，SPP.NET または SPP へのアクセス権限がありません。

ErrServerBusyException

ErrServerBusyException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrServerBusyException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

サービス要求先のソケット受信型サーバが，サービス要求を受信できません。

ErrServerTimedOutException

ErrServerTimedOutException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrTimedOutException
          +- Hitachi.OpenTP1.Client.ErrServerTimedOutException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

TP1/Server 側でサービスの実行中にタイムアウトが発生しました。または，サービス要求先 SPP が処理を終了する前に異常終了しました。

ErrServiceClosedException

ErrServiceClosedException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrServiceClosedException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

指定されたサービスが存在するサービスグループは閉塞されています。

ErrServiceNotUpException

ErrServiceNotUpException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrServiceNotUpException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

- サービス要求した SPP.NET または SPP は稼働していません。
- サービス要求した SPP.NET または SPP が処理を完了する前に異常終了しました。

ErrServiceTerminatedException

ErrServiceTerminatedException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrServiceTerminatedException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

サービス要求した SPP.NET または SPP が処理を完了する前に異常終了しました。

ErrServiceTerminatingException

ErrServiceTerminatingException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrServiceTerminatingException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

指定されたサービスは終了処理中です。

ErrSyserrAtServerException

ErrSyserrAtServerException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrSyserrAtServerException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

指定されたサービスでシステムエラーが発生しました。

ErrSyserrException

ErrSyserrException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrSyserrException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

システムエラーが発生しました。詳細については，「2.9.4　エラートレース，メモリトレース」を参照してください。

ErrTestmodeException

ErrTestmodeException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrTestmodeException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

テストモードの SPP に対してサービス要求を行いました。

ErrTimedOutException

ErrTimedOutException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrTimedOutException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

- OpenConnection メソッドがこの例外を返した場合
  rap リスナーとのコネクションの確立中に，タイムアウトが発生しました。
- CloseConnection メソッドがこの例外を返した場合
  rap リスナー，rap サーバとのコネクション切断中に，タイムアウトが発生しました。
- Call メソッドがこの例外を返した場合
  タイムアウトが発生しました。または，サービス要求先 SPP.NET もしくは SPP が処理を終了する前に異常終了しました。
- CommitChained メソッド，RollbackChained メソッド，Commit メソッド，Rollback メソッド，または Receive メソッドがこの例外を返した場合
  タイムアウトが発生しました。

ErrTMException

ErrTMException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrTMException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

トランザクションサービスでエラーが発生したため，トランザクションを開始できませんでした。なお，この例外が返されたあと，適当な時間を置いてから再度実行すると，トランザクションを開始できることがあります。

ErrTrnchkException

ErrTrnchkException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrTrnchkException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

ノード間負荷バランス機能を使用している環境で，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 TP1/Server のバージョンが，Client .NET 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を実行できません。

この例外は，ノード間負荷バランス機能を使っている 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。

ErrTrnchkExtendException

ErrTrnchkExtendException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrSyserrException
          +- Hitachi.OpenTP1.Client.ErrTrnchkExtendException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次のどれかの要因が考えられます。

- 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 DCRPC_TPNOTRAN を設定していません。

ErrVersionException

ErrVersionException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
        +- Hitachi.OpenTP1.Client.ErrVersionException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

Client .NET が返す例外のクラスです。次の事象を表します。

通知元サーバのバージョンが不正です。

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()
```

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
```

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

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)	レコード名の設定を行います。
SetRecordShortDescription(System.String)	レコードの簡易説明の設定を行います。

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
```

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
```

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

レコード名の設定を行います。

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 _
)
```

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の設定を行います。

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 _
)
```

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

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()
```

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
```

例外

なし

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()
```

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
```

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()
```

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 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 string Value {get; set;}
```

【Visual Basic の場合】

```
Public Property Value As String
```

例外

なし

TP1Client

TP1Client の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Client.TP1Client
```

説明

TP1/Server のサーバに対してサービス要求を行うクラスです。

コンストラクタの一覧

名称	説明
TP1Client()	TP1/Server のサーバに RPC を行う TP1Client クラスのインスタンスを作成します。

フィールドの一覧

名称	説明
DCRPC_MAX_MESSAGE_SIZE	RPC 送受信メッセージの最大長（1048576 バイト）。 Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に 2 以上を指定した場合，RPC 送受信メッセージの最大長は，DCRPC_MAX_MESSAGE_SIZE ではなく，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に指定した値になります。

メソッドの一覧

名称	説明
AcceptNotification(System.Byte[], System.Int32&, System.Int32, System.Int32, System.Byte[], System.Byte[])	サーバ側の関数（dc_rpc_cltsend 関数）で通知されるメッセージを，timeout 引数で指定した時間まで待ち続けます。
AcceptNotificationChained(System.Byte[], System.Int32&, System.Int32, System.Byte[], System.Byte[])	サーバ側の関数（dc_rpc_cltsend 関数）で通知されるメッセージを，timeout 引数で指定した時間まで待ち続けます。
Begin()	グローバルトランザクションを，Begin メソッドを呼び出す TP1Client オブジェクトから開始します。
Call(System.String, System.String, System.Byte[], System.Int32, System.Byte[], System.Int32&, System.Int32)	SPP.NET または SPP のサービスを要求します。

名称	説明
CallTo(Hitachi.OpenTP1.Client.DCRpcBindTbl, System.String, System.String, System.Byte[], System.Int32, System.Byte[], System.Int32&, System.Int32)	Call メソッドと同様に，SPP.NET または SPP のサービスを要求します。 CallTo メソッドでは，サービスグループ名，サービス名，およびホスト名を検索のキーにして，該当するサービスメソッドまたはサービス関数をサービスの要求先に限定します。
CancelNotification(System.Byte[], System.Int32, System.String, System.Int32)	サーバからの一方通知受信待ち状態（AcceptNotification メソッド，または AcceptNotificationChained メソッドの発行）を解除します。
CloseConnection()	CUP.NET と rap リスナーおよび rap サーバとの間で確立されている常設コネクションを切断します。
CloseNotification()	一方通知連続受信機能を使用するための環境を削除します。
CloseRpc()	TP1/Server の SPP.NET または SPP を呼び出すための環境を解放します。
Commit()	トランザクションの同期点を取得します。 Commit メソッドが正常に終了すると，グローバルトランザクションは終了します。 グローバルトランザクションの範囲外からは，SPP.NET または SPP をトランザクションとして実行できません。
CommitChained()	トランザクションの同期点を取得します。 CommitChained メソッドが正常終了すると新しいグローバルトランザクションが発生し，以降実行するメソッドは新しいグローバルトランザクションの範囲になります。
CreateScdDirectObject(System.String, System.Int32, System.Int32)	通信先スケジューラのホスト名およびポート番号を設定した DCRpcBindTbl を作成します。
GetTransactionID(System.Byte[], System.Byte[])	現在のトランザクショングローバル識別子およびトランザクションブランチ識別子を取得します。
GetTransactionInfo()	GetTransactionInfo メソッドを呼び出した TP1Client オブジェクトが，現在トランザクションとして稼働しているかどうかを報告します。
OpenConnection()	Client .NET 構成定義の<TP1Server>要素および<rapService>要素の port 属性で指定された rap サーバとの間に常設コネクションを確立します。
OpenConnection(System.String, System.Int32)	リモート API 機能を使用した RPC を行うために，CUP.NET と rap リスナーおよび rap サーバとの間に常設コネクションを確立します。
OpenNotification(System.Int32)	一方通知連続受信機能を使用するための環境を作成します。
OpenRpc()	TP1/Server の SPP.NET または SPP を呼び出すための環境を初期化します。
OpenRpc(System.String)	TP1/Server の SPP.NET または SPP を呼び出すための環境を初期化します。
Receive(System.Byte[], System.Int32&, System.Int32, System.Int32)	MHP が送信したメッセージを受信します。
ReceiveAssembledMessage(System.Byte[], System.Int32&, System.Int32, System.Int32)	受信メッセージの組み立て機能を使用してメッセージを受信します。

名称	説明
Rollback()	トランザクションをロールバックします。 Rollback メソッドを呼び出すと，グローバルトランザクションは終了します。 グローバルトランザクションの範囲外からは，SPP.NET または SPP をトランザクションとして実行できません。
RollbackChained()	トランザクションをロールバックします。 RollbackChained メソッドが正常終了すると，新しいグローバルトランザクションが発生し，以降呼び出すメソッドは新しいグローバルトランザクションの範囲になります。
Send(System.Byte[], System.Int32, System.String, System.Int32, System.Int32)	MHP にメッセージを送信します。
SendAssembledMessage(System.Byte[], System.Int32, System.String, System.Int32, System.Int32, System.Int32)	受信メッセージの組み立て機能を使用してメッセージを送信します。
SetConnectInformation(System.Byte[] inf, System.Int16 inf_len)	端末識別情報を設定します。
SetDataTraceMode(System.String, System.Int32, System.Int32, System.Boolean)	データトレースを取得するかどうかを指定します。
SetErrorTraceMode(System.String, System.Int32, System.Boolean)	エラートレースを取得するかどうかを指定します。
SetExtendLevel(System.Int32)	Client .NET の機能の拡張レベルを指定します。
SetMethodTraceMode(System.String, System.Int32, System.Boolean)	メソッドトレースを取得するかどうかを指定します。
SetRapDelay(System.Int32)	rap サーバと CUP.NET 間の通信遅延時間を設定します。
SetRapInquireTime(System.Int32)	CUP.NET がサーバに対して問い合わせを行ってから，次の問い合わせをするまでの間隔の最大時間を設定します。
SetRpcExtend(System.Int32)	Client .NET から発行する RPC の機能拡張オプションを指定します。
SetRpcWatchTime(System.Int32)	同期応答型 RPC の場合に，CUP.NET から SPP.NET または SPP へサービス要求を送ってからサービスの応答が返るまでの最大応答待ち時間を設定します。
SetTP1Server(System.String, System.Int32)	窓口となる TP1/Server のホスト名とポート番号を設定します。
SetTraceArray(System.String[])	パラメタに指定された配列にエラートレースを取得するかどうかを指定します。
SetUapTraceMode(System.String, System.Int32, System.Boolean)	UAP トレースを取得するかどうかを指定します。

コンストラクタの詳細

●TP1Client

説明

TP1Client クラスのインスタンスを作成します。

宣言

【C#の場合】

```
public TP1Client(
);
```

【Visual Basic の場合】

```
Public New( _
)
```

パラメタ

なし

例外

なし

フィールドの詳細

●DCRPC_MAX_MESSAGE_SIZE

説明

RPC 送受信メッセージの最大長（1048576 バイトです）。

Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に 2 以上を指定した場合，RPC 送受信メッセージの最大長は，DCRPC_MAX_MESSAGE_SIZE ではなく，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に指定した値になります。

宣言

【C#の場合】

```
public const int DCRPC_MAX_MESSAGE_SIZE
```

【Visual Basic の場合】

```
Public Const DCRPC_MAX_MESSAGE_SIZE As Integer
```

メソッドの詳細

●AcceptNotification

説明

サーバ側の関数（dc_rpc_cltsend 関数）によって通知されるメッセージを，timeout 引数に指定した時間まで待ち続けます。メッセージを受信した時点で CUP.NET に制御を戻し，通知メッセージ，通知メッセージ長，通知元サーバのホスト名，および通知元サーバのノード識別子を返します。

宣言

【C#の場合】

```
public void AcceptNotification(
    byte[] inf,
    ref int inf_len,
    int port,
    int timeout,
    byte[] hostname,
    byte[] nodeid
);
```

【Visual Basic の場合】

```
Public Sub AcceptNotification( _
    ByVal inf() As Byte, _
    ByRef inf_len As Integer, _
    ByVal port As Integer, _
    ByVal timeout As Integer, _
    ByVal hostname() As Byte, _
    ByVal nodeid() As Byte _
)
```

パラメタ

inf

サーバからの通知メッセージを格納する領域を指定します。

メソッドが正常終了した場合，サーバからの通知メッセージが格納されます。

inf_len

サーバからの通知メッセージを格納する領域の長さ（inf 引数の長さ）を指定します。0 から DCRPC_MAX_MESSAGE_SIZE の範囲で指定します。

ただし，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に 2 以上を指定した場合，指定できる最大値は，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に指定した値になります。

メソッドが正常終了した場合，サーバからの通知メッセージ長が格納されます。

port

サーバからの通知メッセージを受信するポート番号を，5001 から 65535 の範囲で指定します。

同一マシン内で，複数のプロセス，または複数のスレッドを同時に実行する場合，port 引数にはそれぞれ異なるポート番号を指定してください。また，port 引数に指定できるポート番号でも，OS,

またはほかのプログラムが使用するポート番号は指定しないでください。指定した場合，応答データを正しく受信できないことがあります。なお，OS が使用するポート番号は，OS ごとに異なります。OS が使用するポート番号については，ご利用の OS の関連ドキュメントを参照してください。

timeout

タイムアウト値を，0 から 65535（単位：秒）の範囲で指定します。0 を指定した場合，無限に待ち続けます。

hostname

通知してきたサーバのホスト名を格納する領域を指定します。256 バイト以上を指定してください。

メソッドが正常終了した場合，通知元サーバのホスト名が格納されます。Client .NET は，通知元サーバの IP アドレスから System.Net.IPHostEntry クラスのオブジェクトを作成し，そのオブジェクトの HostName プロパティからホスト名を取得します。取得したホスト名は，プラットフォームのデフォルト文字セットを使用して，バイト配列に変換されます。この結果が，hostname に格納されます。通知元サーバの IP アドレスからホスト名への変換に失敗した場合，10 進ドット記法（例：10.209.15.124）の IP アドレスを hostname に格納します。

null を指定した場合は，通知してきたサーバのホスト名を格納しません。

nodeid

通知してきたサーバのノード識別子を格納する領域を指定します。8 バイト以上を指定してください。

メソッドが正常終了した場合，通知元サーバのノード識別子が格納されます。ノード識別子のフォーマットは，次のとおりです。

ノード識別子（4バイト）	NULL文字（=0）（4バイト）

戻り値

なし

例外

ErrInvalidArgsException

メソッドの引数の指定に誤りがあります。

ErrProtoException

OpenRpc メソッドが実行されていません。

ErrIOErrException

何らかの入出力例外が発生しました。

ErrInvalidPortException

port 引数で指定したポート番号は使用中です。

ErrClientTimedOutException

Client .NET でタイムアウトが発生しました。

ErrNetDownAtClientException

TP1/Server と CUP.NET の間でネットワーク障害が発生しました。

ErrInvalidMessageException

不正なメッセージを受信しました。

ErrAcceptCanceledException

一方通知受信待ち状態が CancelNotification によって解除されました。このとき inf 引数，inf_len 引数，および hostname 引数にはすでに値が設定されています。nodeid 引数には，先頭から 8 バイト目まで 0 で初期化された値が設定されます。

ErrReplyTooBigException

受信したメッセージが，CUP.NET で用意した領域に収まりません。収まらないメッセージは切り捨てました。このとき，hostname 引数および nodeid 引数には，すでに値が設定されています。

ErrVersionException

通知元サーバのバージョンが不正です。

ErrSyserrException

システムエラーが発生しました。

●AcceptNotificationChained

説明

サーバ側の関数（dc_rpc_cltsend 関数）によって通知されるメッセージを，timeout 引数で指定した時間まで待ち続けます。メッセージを受信した時点で CUP.NET に制御を戻し，通知メッセージ，通知メッセージ長，通知元サーバのホスト名，および通知元サーバのノード識別子を返します。このメソッドを発行するときは，あらかじめ OpenNotification メソッドを発行しておく必要があります。このメソッドは，OpenNotification メソッドから CloseNotification メソッドまでの間で発行できます。

宣言

【C#の場合】

```
public void AcceptNotificationChained(
    byte[] inf,
    ref int inf_len,
    int timeout,
    byte[] hostname,
    byte[] nodeid
);
```

【Visual Basic の場合】

```
Public Sub AcceptNotificationChained( _
    ByVal inf() As Byte, _
    ByRef inf_len As Integer, _
    ByVal timeout As Integer, _
    ByVal hostname() As Byte, _
    ByVal nodeid() As Byte _
)
```

パラメタ

inf

サーバからの通知メッセージを格納する領域を指定します。

メソッドが正常終了した場合，サーバからの通知メッセージが格納されます。

inf_len

サーバからの通知メッセージを格納する領域の長さ（inf 引数の長さ）を指定します。0 から DCRPC_MAX_MESSAGE_SIZE の範囲で指定します。

ただし，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に 2 以上を指定した場合，指定できる最大値は DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に指定した値になります。

メソッドが正常終了した場合，サーバからの通知メッセージ長が格納されます。

timeout

タイムアウト値を，0 から 65535（単位：秒）の範囲で指定します。0 を指定した場合，無限に待ち続けます。

hostname

通知してきたサーバのホスト名を格納する領域を指定します。256 バイト以上を指定してください。

メソッドが正常終了した場合，通知元サーバのホスト名が格納されます。Client .NET は，通知元サーバの IP アドレスから System.Net.IPHostEntry クラスのオブジェクトを作成し，そのオブジェクトの HostName プロパティからホスト名を取得します。取得したホスト名は，プラットフォームのデフォルト文字セットを使用して，バイト配列に変換されます。この結果が，hostname に格納されます。通知元サーバの IP アドレスからホスト名への変換に失敗した場合，10 進ドット記法（例：10.209.15.124）の IP アドレスを hostname に格納します。

null を指定した場合，通知してきたサーバのホスト名を格納しません。

nodeid

通知してきたサーバのノード識別子を格納する領域を指定します。8 バイト以上を指定してください。

メソッドが正常終了した場合，通知元サーバのノード識別子が格納されます。

ノード識別子のフォーマットは，次のとおりです。

ノード識別子（4バイト）	NULL文字（=0）（4バイト）

戻り値

なし

例外

ErrInvalidArgsException

メソッドの引数の指定に誤りがあります。

ErrProtoException

OpenNotification メソッドが実行されていません。

ErrIOErrException

何らかの入出力例外が発生しました。

ErrClientTimedOutException

Client .NET でタイムアウトが発生しました。

ErrNetDownAtClientException

TP1/Server と CUP.NET の間でネットワーク障害が発生しました。

ErrInvalidMessageException

不正なメッセージを受信しました。

ErrAcceptCanceledException

一方通知受信待ち状態が CancelNotification メソッドによって解除されました。このとき，inf 引数，inf_len 引数，および hostname 引数には，すでに値が設定されています。nodeid 引数は，先頭から 8 バイト目まで 0 で初期化された値が設定されます。

ErrReplyTooBigException

受信したメッセージが，CUP.NET で用意した領域に収まりません。収まらないメッセージは切り捨てました。このとき，hostname 引数および nodeid 引数には，すでに値が設定されています。

ErrVersionException

通知元サーバのバージョンが不正です。

ErrSyserrException

システムエラーが発生しました。

●Begin

説明

グローバルトランザクションを，Begin メソッドを呼び出す TP1Client オブジェクトから開始します。

このメソッドは，リモート API 機能を使用する場合で，要求先 TP1/Server Base のバージョンが 05-00 以降のときだけ使用できます。

宣言

【C#の場合】

```
public void Begin(
);
```

【Visual Basic の場合】

```
Public Sub Begin( _
)
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrProtoException

Begin メソッドを不正なコンテキストから呼び出しています。

Hitachi.OpenTP1.Client.ErrTMException

トランザクションサービスでエラーが発生したため，トランザクションは開始できませんでした。

この例外が返されたあと，適当な時間を置いてから再度このメソッドを呼び出すと，トランザクションを開始できることがあります。

Hitachi.OpenTP1.Client.ErrRMException

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。

Hitachi.OpenTP1.Client.ErrNoBufsException

メモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrNotUpException

TP1/Server が稼働していません。

Hitachi.OpenTP1.Client.ErrTimedOutException

タイムアウトが発生しました。

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrConnfreeException

CUP.NET 実行プロセス側から常設コネクションが解放されました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

Hitachi.OpenTP1.Client.ErrHostUndefException

通信先となる TP1/Server のホスト名が，Client .NET 構成定義の<TP1Server>要素に指定されていないか，または指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrInvalidPortException

通信先ポートが不正か，または設定されていません。

●Call

説明

SPP.NET または SPP のサービスを要求します。

サービスグループ名とサービス名の組み合わせで識別されるサービスメソッドまたはサービス関数を呼び出し，その応答を受け取ります。このメソッドを呼び出したときに，目的のサービスグループが閉塞されていると ErrServiceClosedException が返されます。

flags パラメタに DCRPC_TPNOTRAN を指定すると，トランザクションの処理からの RPC をトランザクションの処理ではないサービス要求にできます。

このメソッドを呼び出したときに，目的のサービスグループが dcsvstop コマンドなどで終了処理中，または終了している場合は，ErrServiceTerminatingException，ErrServiceClosedException，または ErrNoSuchServiceGroupException のどれかが返されます。どの例外が返されるかは，このメソッドを呼び出したタイミングによって異なります。

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 および max_socket_msglen オペランドの指定でメッセージの輻輳制御を行っています。そのためサーバがサービス要求を受信できない場合があります。この場合，メソッドを呼び出すと ErrServerBusyException が返されます。

この例外が返されたあと，適当な時間を置いてから再度このメソッドを呼び出すと，サービスを要求できることがあります。

宣言

【C#の場合】

```
public void Call(
  string group,
  string service,
  byte[] in_data,
  int in_length,
  byte[] out_data,
  ref int out_length,
  int flags
);
```

【Visual Basic の場合】

```
Public Sub Call( _
  ByVal group As String, _
  ByVal service As String, _
  ByVal in_data() As Byte, _
  ByVal in_length As Integer, _
  ByVal out_data() As Byte, _
  ByRef out_length As Integer, _
  ByVal flags As Integer _
)
```

パラメタ

group

サービスグループ名を，1 から 31 文字の識別子で指定します。

service

サービス名を，1 から 31 文字の識別子で指定します。

in_data

サービスの入力パラメタを指定します。

in_length

サービスの入力パラメタの応答領域長を，1 から DCRPC_MAX_MESSAGE_SIZE の指定値までの範囲で指定します。ただし，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に 2 以上を指定した場合，指定できる最大値は DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に指定した値になります。

out_data

サービスメソッドまたはサービス関数で指定した応答が返される領域を指定します。

非応答型 RPC の場合は null を指定します。非応答型 RPC の場合，null 以外の値を指定しても何も格納されません。

out_length

サービスの応答の長さを，1 から DCRPC_MAX_MESSAGE_SIZE の指定値までの範囲で指定します。ただし，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に 2 以上を指定した場合，指定できる最大値は DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に指定した値になります。

サービス要求終了時，SPP.NET のサービスメソッド，または SPP のサービス関数で指定した応答の長さが out_length パラメタに設定されます。

非応答型 RPC の場合，応答の長さを指定しても無視され，サービス要求終了時，out_length パラメタには何も設定されません。

flags

RPC の形態を指定します。

- DCNOFLAGS：同期応答型 RPC
- DCRPC_NOREPLY：非応答型 RPC
- DCRPC_CHAINED：連鎖 RPC
- DCRPC_TPNOTRAN：トランザクションを引き継がない RPC

  このフラグを指定すると，トランザクションの処理からの RPC は，トランザクションの処理にしないサービス要求になります。このフラグは DCNOFLAGS，DCRPC_NOREPLY，または DCRPC_CHAINED と組み合わせて指定できます。

  【指定例】

```
flags = TP1ClientFlags.DCNOFLAGS
        |TP1ClientFlags.DCRPC_TPNOTRAN;
```

DCNOFLAGS または DCRPC_CHAINED を指定すると，サーバから応答が返されるか，Client .NET 構成定義の<rpc>要素の WatchTime 属性で指定した最大応答待ち時間が経過してタイムアウトするまで，このメソッドは制御を戻しません。ただし，サービス要求先の SPP.NET または SPP が異常終了した場合は，すぐに例外を返します。この場合，Client .NET 構成定義の<rpc>要素の WatchTime 属性で指定した値によって，次の例外を返します。

- 1～65535 の値を指定した場合：ErrTimedOutException
- 0 を指定した場合：ErrServiceNotUpException

DCRPC_NOREPLY を指定すると，要求したサービスは応答を返さないサービスと見なされます。この場合，このメソッドはサービスの実行終了を待たないですぐに制御を戻します。このフラグを指定した場合，応答（out_data パラメタの指定値）と応答の長さ（out_length パラメタの指定値）は参照できません。さらに，サービスメソッドまたはサービス関数が実行されたかどうかは CUP.NET 側からはわかりません。

DCRPC_CHAINED を指定すると，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，最初の要求時と同一のプロセスで実行されます。

リモート API 機能の使用時に限り，DCRPC_CHAINED を指定できます。リモート API 機能を使用しない場合，flags 引数に DCRPC_CHAINED を指定すると ErrInvalidArgsException が返されます。

連鎖 RPC を使用する場合，次に示す制限があります。

- 2 回目以降のこのメソッドの呼び出しの場合

  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 回目以降のこのメソッドの呼び出しで，サービスメソッドまたはサービス関数の処理に異常が発生した場合

  サービス単位ではなく，ユーザサーバ全体が閉塞します。

- flags パラメタに DCRPC_CHAINED を指定して呼び出したサービスの場合

  最後のサービス要求では DCNOFLAGS を指定して RPC を実行する必要があります。最後のサービス要求で DCNOFLAGS を指定しないで CloseConnection メソッドを呼び出すと，ErrTimedOutException，またはほかの例外を返します。また，サービスを実行していたプロセスは，連鎖 RPC がタイムアウトするまでプロセスを占有されます。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。この場合，詳細メッセージに誤っているパラメタ名が設定されます。

Hitachi.OpenTP1.Client.ErrProtoException

メソッドの発行順序に誤りがあります。OpenConnection メソッドが呼び出されていません。

Hitachi.OpenTP1.Client.ErrNoBufsException

メモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。または，通信先の TP1/Server が稼働していません。

Hitachi.OpenTP1.Client.ErrTimedOutException

このメソッドの処理でタイムアウトが発生しました。または，サービス要求先の SPP.NET もしくは SPP が処理を完了する前に異常終了しました。

Hitachi.OpenTP1.Client.ErrMessageTooBigException

in_length パラメタに指定した入力パラメタ長が最大値を超えています。

Hitachi.OpenTP1.Client.ErrReplyTooBigException

サーバから返された応答の長さが，CUP.NET で用意した領域（out_data パラメタの指定値）の長さを超えています。

Hitachi.OpenTP1.Client.ErrNoSuchServiceGroupException

group パラメタに指定したサービスグループ名は定義されていません。

Hitachi.OpenTP1.Client.ErrNoSuchServiceException

service パラメタに指定したサービス名は定義されていません。

Hitachi.OpenTP1.Client.ErrServiceClosedException

service パラメタに指定したサービス名が存在するサービスグループは閉塞されています。

Hitachi.OpenTP1.Client.ErrServiceTerminatingException

service パラメタに指定したサービスは終了処理中です。

Hitachi.OpenTP1.Client.ErrServiceNotUpException

サービス要求した SPP.NET もしくは SPP は稼働していません。または，サービス要求した SPP.NET もしくは SPP が処理を完了する前に異常終了しました。

この例外は，Client .NET 構成定義の<rpc>要素の WatchTime 属性に 0 を指定した（応答を無限に待つ）場合に返されます。

Hitachi.OpenTP1.Client.ErrNotUpException

指定したサービスが存在するノードの TP1/Server が稼働していません。

この場合，異常終了，停止中，終了処理中，またはネットワーク障害の発生が考えられます。

Hitachi.OpenTP1.Client.ErrSyserrAtServerException

指定したサービスでシステムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

次のどちらかの要因が考えられます。

- 通信エラーが発生しました。

  Client .NET 構成定義の<tp1Server>要素，<nameService>要素，<scheduleService>要素に指定した窓口となる OpenTP1 のホスト名，またはポート番号に誤りがあります。

  OpenTP1 側のホスト名，ポート番号を確認し，適切な値を設定してください。

- システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrNoBufsAtServerException

指定したサービスでメモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrInvalidReplyException

サービスメソッドまたはサービス関数が返した応答の長さが，1 から DCRPC_MAX_MESSAGE_SIZE で指定した値までの範囲にありません。

Hitachi.OpenTP1.Client.ErrInitializingException

サービス要求したノードにある TP1/Server は開始処理中です。

Hitachi.OpenTP1.Client.ErrTrnchkException

ノード間負荷バランス機能を使用している環境で，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 TP1/Server のバージョンが，Client .NET 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を実行できません。

この例外は，ノード間負荷バランス機能を使用している SPP.NET または SPP にサービス要求した場合だけ返されます。

Hitachi.OpenTP1.Client.ErrNotTrnExtendException

トランザクション処理の連鎖 RPC を使用したあとで，flags パラメタに DCRPC_TPNOTRAN を指定したサービスを要求しています。

Hitachi.OpenTP1.Client.ErrTrnchkExtendException

次のどれかの要因が考えられます。

- 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ない。
- 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ない。
- 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 DCRPC_TPNOTRAN を設定していない。

Hitachi.OpenTP1.Client.ErrServerBusyException

サービス要求先のソケット受信型サーバが，サービス要求を受信できません。

Hitachi.OpenTP1.Client.ErrSecchkException

サービス要求先の SPP.NET または SPP は，セキュリティ機能で保護されています。Call メソッドを呼び出した CUP.NET には，SPP.NET または SPP へのアクセス権限がありません。

Hitachi.OpenTP1.Client.ErrServiceTerminatedException

サービス要求した SPP.NET または SPP が，処理を完了する前に異常終了しました。

この例外は rap サービス定義の rpc_extend_function オペランドに 00000001 を指定した場合だけ返されます。

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，ErrTimedOutException または ErrServiceNotUpException が返されます。

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。rap サーバが問い合わせ間隔の時間監視でタイムアウトし，コネクションを切断したことも考えられます。

Hitachi.OpenTP1.Client.ErrTestmodeException

テストモードの SPP に対してサービス要求を行いました。

Hitachi.OpenTP1.Client.ErrConnfreeException

rap サーバとの常設コネクションが切断されました。

Hitachi.OpenTP1.Client.ErrHostUndefException

通信先となる TP1/Server のホスト名が Client .NET 構成定義の<TP1Server>要素に指定されていないか，または指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrInvalidPortException

次の要因が考えられます。

- リモート API 機能を使用した RPC を行う場合，Client .NET 構成定義で<rapService>要素の port 属性が指定されていない。
- スケジューラダイレクト機能を使用した RPC を行う場合，Client .NET 構成定義で<scheduleService>要素の port 属性が指定されていない。

注意事項

入力パラメタ（in_data パラメタの指定値）と，サービスメソッドまたはサービス関数の応答（out_data パラメタの指定値）に同じ領域を指定しないでください。

flags パラメタに DCRPC_NOREPLY を指定した場合，次の例外は返されません。

- 発生しない例外

  ErrReplyTooBigException

  ErrInvalidReplyException
- 発生しても検出できない例外

  ErrNoSuchServiceException

  ErrServiceClosedException

  ErrServiceTerminatingException

  ErrSyserrAtServerException

  ErrNoBufsAtServerException

  ErrNotUpException

ErrTimedOutException が返される場合，次に示す要因が考えられます。

- Client .NET 構成定義または TP1/Server の定義で指定した最大応答待ち時間が短い。
- サービス要求先の SPP.NET から発行したサービスメソッド，または SPP から発行したサービス関数が異常終了した。
- サービス要求先の SPP.NET または SPP が存在するノードに障害が発生した。
- サービス要求先の SPP.NET または SPP の処理完了前に異常終了した。
- ネットワーク障害が発生した。

上記の場合，サービス要求先の SPP.NET または SPP から開始したトランザクションの処理はコミットされて，データが更新されていることがあります。データが更新されているかどうか確認してください。

このメソッドの呼び出し時，何らかの障害が発生した場合には例外が返されますが，その障害によってrap リスナーおよび rap サーバとの常設コネクションが切断されたかどうかは，その時点では判断できません。この場合，再びこのメソッドを呼び出すと，rap リスナーおよび rap サーバとの常設コネクションが切断されていれば，ErrConnfreeException が返されます。

このメソッドの呼び出し時，何らかの障害で rap リスナーおよび rap サーバとの常設コネクションが切断された場合は，再び OpenConnection メソッドを呼び出して rap リスナーおよび rap サーバとの常設コネクションを確立する必要があります。

オートコネクトモードを使用した常設コネクションの場合，サービス要求送信時に rap サーバとのネットワーク障害が発生すると，一度だけリトライ処理を行います。リトライしてもコネクションが確立されない場合は，ErrNetDownException が返されます。

スケジューラダイレクト機能を使用した RPC，およびネームサービスを使用した RPC では，flags パラメタに DCRPC_CHAINED を指定できません。

●CallTo

説明

Call メソッドと同様に，SPP.NET または SPP のサービスを要求します。

CallTo メソッドでは，サービスグループ名，サービス名，およびホスト名を検索のキーにして，該当するサービスメソッドまたはサービス関数をサービスの要求先に限定します。

このメソッドを呼び出す前に CreateScdDirectObject メソッドを呼び出し，DCRpcBindTbl インスタンスを作成しておく必要があります。

それ以外のインタフェースは，Call メソッドと同じです。

宣言

【C#の場合】

```
public void CallTo(
  Hitachi.OpenTP1.Client.DCRpcBindTbl direction,
  string group,
  string service,
  byte[] in_data,
  int in_length,
  byte[] out_data,
  ref int out_length,
  int flags
);
```

【Visual Basic の場合】

```
Public Sub CallTo( _
  ByVal direction As Hitachi.OpenTP1.Client.DCRpcBindTbl, _
  ByVal group As String, _
  ByVal service As String, _
  ByVal in_data() As Byte, _
  ByVal in_length As Integer, _
  ByVal out_data() As Byte, _
  ByRef out_length As Integer, _
```

```
  ByVal flags As Integer _
)
```

パラメタ

direction

DCRpcBindTbl オブジェクトを指定します。

このメソッドを呼び出す前に CreateScdDirectObject メソッドを呼び出し，通信先情報を設定しておきます。

group

サービスグループ名を 1 から 31 文字の識別子で指定します。

service

サービス名を 1 から 31 文字の識別子で指定します。

in_data

サービスの入力パラメタを指定します。

in_length

サービスの入力パラメタの応答領域長を，1 から DCRPC_MAX_MESSAGE_SIZE の指定値までの範囲で指定します。ただし，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に 2 以上を指定した場合，指定できる最大値は DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に指定した値になります。

out_data

サービスメソッドまたはサービス関数で指定した応答が返される領域を指定します。

非応答型 RPC の場合は null を指定します。非応答型 RPC の場合，null 以外の値を指定しても何も格納されません。

out_length

サービスの応答の長さを，1 から DCRPC_MAX_MESSAGE_SIZE の指定値までの範囲で指定します。ただし，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に 2 以上を指定した場合，指定できる最大値は DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に指定した値になります。

flags

RPC の形態を指定します。

- DCNOFLAGS：同期応答型 RPC
- DCRPC_NOREPLY：非応答型 RPC

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。詳細メッセージに誤ったパラメタ名が設定されます。

Hitachi.OpenTP1.Client.ErrProtoException

メソッドの発行順序に誤りがあります。OpenConnection メソッドが呼び出されていません。

Hitachi.OpenTP1.Client.ErrNoBufsException

メモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。または，通信先の TP1/Server が稼働していません。

Hitachi.OpenTP1.Client.ErrTimedOutException

このメソッドの処理でタイムアウトが発生しました。または，サービス要求先の SPP.NET もしくは SPP が処理を完了する前に異常終了しました。

Hitachi.OpenTP1.Client.ErrMessageTooBigException

in_length パラメタに指定した入力パラメタ長が最大値を超えています。

Hitachi.OpenTP1.Client.ErrReplyTooBigException

サーバから返された応答の長さが，CUP.NET で用意した領域（out_data パラメタの指定値）の長さを超えています。

Hitachi.OpenTP1.Client.ErrNoSuchServiceGroupException

group パラメタに指定したサービスグループ名は定義されていません。

Hitachi.OpenTP1.Client.ErrNoSuchServiceException

service パラメタに指定したサービス名は定義されていません。

Hitachi.OpenTP1.Client.ErrServiceClosedException

service パラメタに指定したサービス名が存在するサービスグループは閉塞されています。

Hitachi.OpenTP1.Client.ErrServiceTerminatingException

service パラメタに指定したサービスは終了処理中です。

Hitachi.OpenTP1.Client.ErrServiceNotUpException

サービス要求した SPP.NET もしくは SPP は稼働していません。または，サービス要求した SPP.NET もしくは SPP が処理を完了する前に異常終了しました。

この例外は，Client .NET 構成定義の<rpc>要素の WatchTime 属性に 0 を指定した（応答を無限に待つ）場合に返されます。

Hitachi.OpenTP1.Client.ErrNotUpException

指定したサービスが存在するノードの TP1/Server が稼働していません。

この場合，異常終了，停止中，終了処理中，またはネットワーク障害の発生が考えられます。

Hitachi.OpenTP1.Client.ErrSyserrAtServerException

指定したサービスでシステムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrNoBufsAtServerException

指定したサービスでメモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrInvalidReplyException

サービスメソッドまたはサービス関数が返した応答の長さが，1 から DCRPC_MAX_MESSAGE_SIZE で指定した値までの範囲にありません。

Hitachi.OpenTP1.Client.ErrInitializingException

サービス要求したノードにある TP1/Server は開始処理中です。

Hitachi.OpenTP1.Client.ErrTrnchkException

ノード間負荷バランス機能を使用している環境で，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 TP1/Server のバージョンが，Client .NET 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を実行できません。

この例外は，ノード間負荷バランス機能を使用している SPP.NET または SPP にサービス要求した場合だけ返されます。

Hitachi.OpenTP1.Client.ErrNotTrnExtendException

トランザクション処理の連鎖 RPC を使用したあとで，flags パラメタに DCRPC_TPNOTRAN を指定したサービスを要求しています。

Hitachi.OpenTP1.Client.ErrTrnchkExtendException

次のどれかの要因が考えられます。

- 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ない。
- 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ない。
- 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 DCRPC_TPNOTRAN を設定していない。

Hitachi.OpenTP1.Client.ErrServerBusyException

サービス要求先のソケット受信型サーバが，サービス要求を受信できません。

Hitachi.OpenTP1.Client.ErrSecchkException

サービス要求先の SPP.NET または SPP は，セキュリティ機能で保護されています。Call メソッドを呼び出した CUP.NET には，SPP.NET または SPP へのアクセス権限がありません。

Hitachi.OpenTP1.Client.ErrServiceTerminatedException

サービス要求した SPP.NET または SPP が，処理を完了する前に異常終了しました。

この例外は，rap サービス定義の rpc_extend_function オペランドに 00000001 を指定した場合だけ返されます。

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，ErrTimedOutException または ErrServiceNotUpException が返されます。

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。rap サーバが問い合わせ間隔の時間監視でタイムアウトし，コネクションを切断したことも考えられます。

Hitachi.OpenTP1.Client.ErrTestmodeException

テストモードの SPP に対してサービス要求を行いました。

Hitachi.OpenTP1.Client.ErrConnfreeException

rap サーバとの常設コネクションが切断されました。

Hitachi.OpenTP1.Client.ErrHostUndefException

通信先となる TP1/Server のホスト名が Client .NET 構成定義の<TP1Server>要素に指定されていないか，または指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrInvalidPortException

次の要因が考えられます。

- リモート API 機能を使用した RPC を行う場合
  Client .NET 構成定義の<rapService>要素の port 属性が指定されていない。
- スケジューラダイレクト機能を使用した RPC を行う場合
  Client .NET 構成定義の<scheduleService>要素の port 属性が指定されていない。

注意事項

ソケット受信型のユーザサーバにサービス要求を行った場合，このメソッドは，ErrServiceNotUpException を返します。

サービス要求先の TP1/Server のバージョンは，03-03 以降でなければなりません。これ以前のバージョンの TP1/Server をサービス要求先に指定した場合は，動作の保証はできません。

このメソッドは，リモート API 機能を使用した RPC では使用できません。リモート API 機能を使用しているときにこのメソッドを発行した場合，このメソッドは，ErrProtoException を返します。

このメソッドは，連鎖 RPC では使用できません。flags パラメタに DCRPC_CHAINED を指定した場合，このメソッドは ErrInvalidArgsException を返します。

CallTo メソッドの呼び出し時は，Client .NET 構成定義の次のオペランドは参照されません。

- <TP1Server>要素
- <nameService>要素の port 属性
- <scheduleService>要素の port 属性，および usePriority 属性

また，SetTP1Server メソッドで指定されたホスト名およびポート番号も参照されません。

これらの指定値は，次回 Call メソッドの呼び出し時に有効になります。

サービス要求先のホスト名に誤りがあった場合，このメソッドは ErrHostUndefException を返します。

サービス要求先のポート番号に誤りがあった場合，このメソッドは ErrInvalidPortException を返します。

●CancelNotification

説明

サーバからの一方通知受信待ち状態（AcceptNotification メソッド，または AcceptNotificationChained メソッドの発行）を解除します。解除するときに，inf 引数に指定したメッセージを，一方通知受信待ち状態の CUP.NET に通知できます。

宣言

【C#の場合】

```
public void CancelNotification(
  byte[] inf,
  int inf_len,
  string hostname,
  int port
);
```

【Visual Basic の場合】

```
Public Sub CancelNotification( _
  ByVal inf() As Byte, _
  ByVal inf_len As Integer, _
  ByVal hostname As String, _
  ByVal port As Integer _
)
```

パラメタ

inf

CUP.NET に通知するメッセージを指定します。

inf_len

CUP.NET に通知するメッセージ長（inf 引数の長さ）を指定します。0 から DCRPC_MAX_MESSAGE_SIZE の範囲で指定します。0 を指定した場合は CUP.NET にメッセージを通知しません。

ただし，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に 2 以上を指定した場合，指定できる最大値は DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，Client .NET 構成定義の<rpc>要素の maxMessageSize 属性に指定した値になります。

hostname

一方通知受信待ち状態の CUP.NET が存在するホスト名を指定します。ホスト名として，10 進ドット記法（例：10.209.15.124）の IP アドレスを指定することもできます。

port

一方通知受信待ち状態の CUP.NET のポート番号を，5001 から 65535 の範囲で指定します。

戻り値

なし

例外

ErrInvalidArgsException

メソッドの引数の指定に誤りがあります。

ErrProtoException

OpenRpc メソッドが実行されていません。

ErrIOErrException

何らかの入出力例外が発生しました。

ErrInvalidPortException

port 引数で指定したポート番号が不正です。

ErrClientTimedOutException

Client .NET でタイムアウトが発生しました。

ErrNetDownAtClientException

CUP.NET 間でネットワーク障害が発生しました。

ErrHostUndefException

hostname 引数で指定したホスト名が不正です。

ErrSyserrException

システムエラーが発生しました。

●CloseConnection

説明

CUP.NET と rap リスナーおよび rap サーバとの間で確立されている常設コネクションを切断します。

宣言

【C#の場合】

```
public void CloseConnection(
);
```

【Visual Basic の場合】

```
Public Sub CloseConnection( _
)
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrProtoException

メソッドの発行順序に誤りがあります。

すでにコネクションが切断されている状態で，再度 CloseConnection メソッドが呼び出されました。

Hitachi.OpenTP1.Client.ErrTimedOutException

rap リスナー，rap サーバとのコネクション切断中に，タイムアウトが発生しました。

Hitachi.OpenTP1.Client.ErrNetDownException

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。

注意事項

このメソッドが例外を返した場合でも，常設コネクションは切断されています。

●CloseNotification

説明

一方通知連続受信機能を使用するための環境を削除します。このメソッドは，OpenNotification メソッドと対で発行します。OpenNotification メソッドが正常終了した場合は，必ずこのメソッドを発行してください。

宣言

【C#の場合】

```
public void CloseNotification(
);
```

【Visual Basic の場合】

```
Public Sub CloseNotification( _
)
```

パラメタ

なし

戻り値

なし

例外

ErrNetDownAtClientException

TP1/Server と CUP.NET の間でネットワーク障害が発生しました。

ErrSyserrException

システムエラーが発生しました。

●CloseRpc

説明

CUP.NET の各機能を使用する環境を解放します。

CUP.NET の各機能を使用する場合は，OpenRpc メソッドを呼び出します。

CUP.NET の実行の最後にこのメソッドを呼び出します。

宣言

【C#の場合】

```
public void CloseRpc(
);
```

【Visual Basic の場合】

```
Public Sub CloseRpc( _
)
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

●Commit

説明

トランザクションの同期点を取得します。

Commit メソッドが正常に終了すると，グローバルトランザクションは終了します。

グローバルトランザクションの範囲外からは，SPP.NET または SPP をトランザクションとして実行できません。

宣言

【C#の場合】

```
public void Commit(
);
```

【Visual Basic の場合】

```
Public Sub Commit( _
)
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrProtoException

Commit メソッドを不正なコンテキストから呼び出しています。

Hitachi.OpenTP1.Client.ErrRollbackException

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。

Hitachi.OpenTP1.Client.ErrHeuristicException

Commit メソッドを実行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。

この例外は，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い場合に返されます。

この例外が返される原因になった 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，TP1/Server のメッセージログファイルを参照してください。

この例外が返されたあと，このプロセスはトランザクション下にありません。

プロセスはグローバルトランザクションの範囲外です。

Hitachi.OpenTP1.Client.ErrHazardException

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。

この例外が返される原因になった 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，TP1/Server のメッセージログファイルを参照してください。

この例外が返されたあと，このプロセスはトランザクション下にありません。プロセスはグローバルトランザクションの範囲外です。

Hitachi.OpenTP1.Client.ErrNoBufsException

メモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrNotUpException

TP1/Server が稼働していません。

Hitachi.OpenTP1.Client.ErrTimedOutException

タイムアウトが発生しました。

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrConnfreeException

CUP.NET 実行プロセス側から常設コネクションが解放されました。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

●CommitChained

説明

トランザクションの同期点を取得します。

CommitChained メソッドが正常終了すると新しいグローバルトランザクションが発生し，以降実行するメソッドは新しいグローバルトランザクションの範囲になります。

宣言

【C#の場合】

```
public void CommitChained(
);
```

【Visual Basic の場合】

```
Public Sub CommitChained( _
)
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrProtoException

CommitChained メソッドを不正なコンテキストから呼び出しています。

Hitachi.OpenTP1.Client.ErrRollbackException

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。

この例外が返されたあとも，このプロセスはトランザクション下にあり，グローバルトランザクションの範囲内です。

Hitachi.OpenTP1.Client.ErrHeuristicException

CommitChained メソッドを実行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。

この例外は，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い場合に返されます。

この例外が返される原因になった 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，TP1/Server のメッセージログファイルを参照してください。

この例外が返されたあとも，このプロセスはトランザクション下にあり，グローバルトランザクションの範囲内です。

Hitachi.OpenTP1.Client.ErrHazardException

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。

この例外が返される原因になった 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，TP1/Server のメッセージログファイルを参照してください。

この例外が返されたあとも，このプロセスはトランザクション下にあり，グローバルトランザクションの範囲内です。

Hitachi.OpenTP1.Client.ErrNoBeginException

コミットまたは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。

この例外が返されたあと，このプロセスはトランザクション下にありません。

Hitachi.OpenTP1.Client.ErrRollbackNoBeginException

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。

新しいトランザクションは開始できませんでした。

この例外が返されたあと，このプロセスはトランザクション下にありません。

Hitachi.OpenTP1.Client.ErrHeuristicNoBeginException

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。

この例外は，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い場合に返されます。

この例外が返される原因になった 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，TP1/Server のメッセージログファイル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。

この例外が返されたあと，このプロセスはトランザクション下にありません。

Hitachi.OpenTP1.Client.ErrHazardNoBeginException

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。

この例外が返される原因になった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，TP1/Serverの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。

この例外が返されたあと，このプロセスはトランザクション下にありません。

Hitachi.OpenTP1.Client.ErrNoBufsException

メモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrNotUpException

TP1/Serverが稼働していません。

Hitachi.OpenTP1.Client.ErrTimedOutException

タイムアウトが発生しました。

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrConnfreeException

CUP.NET実行プロセス側から常設コネクションが解放されました。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

●CreateScdDirectObject

説明

通信先スケジューラのホスト名およびポート番号を設定したDCRpcBindTblを作成します。

宣言

【C#の場合】

```
public Hitachi.OpenTP1.Client.DCRpcBindTbl
 CreateScdDirectObject(
  string host,
  int scdport,
  int flags
);
```

【Visual Basicの場合】

```
Public Function CreateScdDirectObject( _
  ByVal host As String, _
  ByVal scdport As Integer, _
```

```
  ByVal flags As Integer _
) As Hitachi.OpenTP1.Client.DCRpcBindTbl
```

パラメタ

host

通信先スケジュールサーバのホスト名を指定します。

不正なホスト名，および null が指定された場合，このメソッドのあとに呼び出す CallTo メソッドは，ErrHostUndefException を返します。

scdport

通信先スケジュールサーバのポート番号を指定します。

ポート番号は，5001 から 65535 までの範囲で指定します。

不正なポート番号が指定された場合，このメソッドのあとに呼び出す CallTo メソッドは，ErrInvalidPortException を返します。

flags

DCNOFLAGS を指定します。

戻り値

DCRpcBindTbl オブジェクト。

例外

なし

●GetTransactionID

説明

現在のトランザクショングローバル識別子およびトランザクションブランチ識別子を取得します。

このメソッドは内部の変数を参照するだけで，rap サーバとは通信しません。

現在のトランザクショングローバル識別子およびトランザクションブランチ識別子は，次に示すメソッドを呼び出してトランザクションが起動したときに，TP1/Server が割り当てたものです。

- Begin メソッド
- CommitChained メソッド
- RollbackChained メソッド

宣言

【C#の場合】

```
public void GetTransactionID(
  byte[] trngid,
  byte[] trnbid
);
```

【Visual Basic の場合】

```
Public Sub GetTransactionID( _
  ByVal trngid() As Byte, _
  ByVal trnbid() As Byte _
)
```

パラメタ

trngid

トランザクショングローバル識別子を格納する，16 バイト以上の byte 型配列を指定します。

trnbid

トランザクションブランチ識別子を格納する，16 バイト以上の byte 型配列を指定します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrProtoException

GetTransactionID メソッドを不正なコンテキストから呼び出しています。

●GetTransactionInfo

説明

GetTransactionInfo メソッドを呼び出した TP1Client オブジェクトが，現在トランザクションとして稼働しているかどうかを報告します。

このメソッドは内部の変数を参照するだけで，rap サーバとは通信しません。

宣言

【C#の場合】

```
public System.Boolean GetTransactionInfo(
);
```

【Visual Basic の場合】

```
Public Function GetTransactionInfo( _
) As System.Boolean
```

パラメタ

なし

戻り値

true

GetTransactionInfo メソッドを呼び出した TP1Client オブジェクトは，トランザクションの範囲内にあります。

false

GetTransactionInfo メソッドを呼び出した TP1Client オブジェクトは，トランザクションの範囲外にあります。

例外

なし

●OpenConnection

説明

Client .NET 構成定義の<TP1Server>要素および<rapService>要素の port 属性で指定された rap サーバとの間に常設コネクションを確立します。なお，一つの CUP.NET から，同時に複数の常設コネクションを確立することはできません。

CUP.NET と接続要求先 rap リスナーとの間にファイアウォールがある場合は，接続要求先 rap リスナーにはファイアウォールのホスト名とポート番号を指定してください。

宣言

【C#の場合】

```
public void OpenConnection(
);
```

【Visual Basic の場合】

```
Public Sub OpenConnection( _
)
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrHostUndefException

rap リスナーのホスト名が，Client .NET 構成定義の<TP1Server>要素に指定されていません。

Hitachi.OpenTP1.Client.ErrTimedOutException

rap リスナーとのコネクション確立中にタイムアウトが発生しました。

Hitachi.OpenTP1.Client.ErrNetDownException

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。

または，通信先の TP1/Server が稼働していません。

Hitachi.OpenTP1.Client.ErrNoBufsException

rap リスナー，rap サーバでメモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrNotUpException

rap リスナー，rap サーバが稼働していません。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrProtoException

メソッドの発行順序に誤りがあります。

すでにコネクションを確立している状態で，再度 OpenConnection メソッドが呼び出されました。

Hitachi.OpenTP1.Client.ErrInvalidPortException

rap リスナーのポートが定義されていません。

注意事項

接続先の rap リスナーが起動していない場合，ErrIOErrException または ErrNetDownException が返されます。

このメソッドが例外を返した場合，常設コネクションは確立されていません。

●OpenConnection

説明

リモート API 機能を使用した RPC を行うために，CUP.NET と rap リスナーおよび rap サーバとの間に常設コネクションを確立します。

常設コネクションの確立先はパラメタで指定された値を使用します。

宣言

【C#の場合】

```
public void OpenConnection(
  string host,
  int port
);
```

【Visual Basic の場合】

```
Public Sub OpenConnection( _
  ByVal host As String, _
  ByVal port As Integer _
)
```

パラメタ

host

rap リスナーまたはファイアウォールのホスト名を指定します。

port

rap リスナーまたはファイアウォールのポート番号を，5001 から 65535 までの範囲で指定します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrHostUndefException

host パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrTimedOutException

rap リスナーとのコネクション確立中にタイムアウトが発生しました。

Hitachi.OpenTP1.Client.ErrNoBufsException

rap リスナー，rap サーバでメモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrNotUpException

指定されたサービスがあるノードの TP1/Server が稼働していません。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrInvalidPortException

port パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrProtoException

メソッドの発行順序に誤りがあります。

すでにコネクションを確立している状態で，再度 OpenConnection メソッドが呼び出されました。

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrNetDownException

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。

または，通信先の TP1/Server が稼働していません。

注意事項

接続先の rap リスナーが起動していない場合，ErrIOErrException または ErrNetDownException が返されます。

このメソッドが例外を返した場合，常設コネクションは確立されていません。

●OpenNotification

説明

一方通知連続受信機能を使用するための環境を作成します。このメソッドは，CloseNotification メソッドと対で発行します。このメソッドが正常終了した場合は，必ず CloseNotification メソッドを発行してください。

宣言

【C#の場合】

```
public void OpenNotification(
int port
);
```

【Visual Basic の場合】

```
Public Sub OpenNotification( _
ByVal port As Integer _
)
```

パラメタ

port

サーバからの通知メッセージを受信するポート番号を，5001 から 65535 の範囲で指定します。

同一マシン内で，複数のプロセス，または複数のスレッドを同時に実行する場合，port 引数にはそれぞれ異なるポート番号を指定してください。また，port 引数に指定できるポート番号でも，OS，またはほかのプログラムが使用するポート番号は指定しないでください。指定した場合，応答データを正しく受信できないことがあります。なお，OS が使用するポート番号は，OS ごとに異なります。OS が使用するポート番号については，ご利用の OS の関連ドキュメントを参照してください。

戻り値

なし

例外

ErrInvalidArgsException

メソッドの引数の指定に誤りがあります。

ErrProtoException

OpenRpc メソッドが実行されていない，あるいは OpenNotification メソッドがすでに実行されています。

ErrInvalidPortException

port 引数で指定したポート番号は，すでに使用されています。

ErrNetDownAtClientException

TP1/Server と CUP.NET の間でネットワーク障害が発生しました。

ErrSyserrException

システムエラーが発生しました。

●OpenRpc

説明

CUP.NET の各機能を使用する環境を初期化します。

CUP.NET を実行するときは，最初にこのメソッドを呼び出します。

宣言

【C#の場合】

```
public void OpenRpc(
);
```

【Visual Basic の場合】

```
Public Sub OpenRpc( _
)
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrProtoException

メソッドの発行順序に誤りがあります。

CloseRpc メソッドが呼び出されないで，再度 OpenRpc メソッドが呼び出されました。

Hitachi.OpenTP1.Client.ErrFatalException

Client .NET 構成定義の指定に誤りがあります。

または，TP1/Server との通信環境の初期化に失敗しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

●OpenRpc

説明

TP1/Server の SPP.NET または SPP を呼び出すための環境を初期化します。

CUP.NET を実行するときは，最初にこのメソッドを呼び出します。

宣言

【C#の場合】

```
public void OpenRpc(
  string profileId
);
```

【Visual Basic の場合】

```
Public Sub OpenRpc( _
  ByVal profileId As String _
)
```

パラメタ

profileId

プロファイル ID を指定します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrProtoException

メソッドの発行順序に誤りがあります。

CloseRpc メソッドを呼び出さないで，再度 OpenRpc メソッドが呼び出されました。

Hitachi.OpenTP1.Client.ErrFatalException

Client .NET 構成定義に誤りがあります。

または，TP1/Server との通信環境の初期化に失敗しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

●Receive

説明

MHP が送信したメッセージを受信します。

Receive メソッドを実行する場合，Client .NET 構成定義の<tcpip>要素の type 属性に recv または sendrecv を指定して，OpenRpc メソッドをあらかじめ実行しておく必要があります。

宣言

【C#の場合】

```
public void Receive(
  byte[] buff,
  ref int recvleng,
  int timeout,
  int flags
);
```

【Visual Basic の場合】

```
Public Sub Receive( _
  ByVal buff() As Byte, _
  ByRef recvleng As Integer, _
  ByVal timeout As Integer, _
  ByVal flags As Integer _
)
```

パラメタ

buff

受信したメッセージを格納する領域を指定します。

recvleng で指定する長さ以上の領域を指定してください。

メソッド実行後は受信したメッセージが返されます。

recvleng

受信するメッセージの長さを指定します。

メソッド実行後は受信したメッセージの長さが返されます。

timeout

メッセージ受信時の最大待ち時間を，-1 から 65535 までの整数（単位：秒）で指定します。

-1 を指定した場合は，メッセージを受信するまで無制限に待ちます。

0 を指定した場合は，メッセージの受信を待ちません。受信するメッセージがなかった場合は，ErrTimedOutException が返されます。

1 から 65535 の値を指定した場合は，指定した秒数だけメッセージの受信を待ちます。指定した時間を過ぎてもメッセージを受信できない場合は，ErrTimedOutException が返されます。

flags

メッセージ受信後に，コネクションを解放するかどうかを指定します。

- DCNOFLAGS：メッセージ受信後，コネクションを解放しません。

  DCNOFLAGS を指定した場合は，障害時を除き，CloseRpc メソッドを実行するまでコネクションを解放しません。
- DCCLT_RCV_CLOSE：メッセージ受信後，コネクションを解放します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrProtoException

OpenRpc メソッドが実行されていません。または，OpenRpc メソッドは実行されていますが，Client .NET 構成定義の<tcpip>要素の type 属性に recv または sendrecv を指定していません。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

Hitachi.OpenTP1.Client.ErrTimedOutException

メッセージ受信時にタイムアウトしました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrInvalidPortException

受信用 port の指定に誤りがあります。

この例外は，Client .NET 構成定義の<tcpip>要素の openPortAtRecv 属性に true を指定し，recvPort 属性に指定したポートがすでに使用中の場合に発生します。

openPortAtRecv 属性に false を指定した場合は，OpenRpc メソッドが ErrFatalException を返します。

Hitachi.OpenTP1.Client.ErrConnfreeException

MHP からコネクションが解放されました。

注意事項

Receive メソッドは，次に示す場合に CUP.NET に制御を戻します。

- MHP から recvleng パラメタで指定した長さ分のメッセージを受信した場合
- MHP からのメッセージ受信時に，タイムアウトが発生した場合
- MHP からコネクションが解放された場合
- ネットワーク障害が発生した場合

Receive メソッドの発行時に，MHP からコネクションが解放された場合，ErrConnfreeException でエラーリターンします。

●ReceiveAssembledMessage

説明

受信メッセージの組み立て機能を使用してメッセージを受信します。ReceiveAssembledMessage メソッドを実行する場合，Client .NET 構成定義の<tcpip>要素の type 属性に recv または sendrecv を指定して，OpenRpc メソッドをあらかじめ実行しておく必要があります。

受信メッセージの組み立て機能を使用した場合，メッセージの先頭 4 バイトは，buff 引数に指定されたバッファに格納されません。このメソッドが正常終了した場合，recvleng の長さのユーザデータを含むメッセージを受信し，ユーザデータが buff[0]～buff[recvleng－1]に格納されます。

宣言

【C#の場合】

```
public void ReceiveAssembledMessage(
  byte[] buff,
  ref int recvleng,
  int timeout,
```

```
  int flags
);
```

【Visual Basic の場合】

```
Public Sub ReceiveAssembledMessage( _
  ByVal buff() As Byte, _
  ByRef recvleng As Integer, _
  ByVal timeout As Integer, _
  ByVal flags As Integer _
)
```

パラメタ

buff

受信したメッセージを格納する領域を指定します。

受信するメッセージの長さ以上の領域を指定してください。

recvleng

メソッド実行後に，受信したメッセージの長さが recvleng に返されます。

timeout

メッセージ受信時の最大待ち時間を，-1 から 65535 までの整数（単位：秒）で指定します。

-1 を指定した場合は，メッセージを受信するまで無制限に待ちます。

0 を指定した場合は，メッセージの受信を待ちません。受信するメッセージがなかった場合は，ErrTimedOutException が返されます。

1 から 65535 を指定した場合は，指定した秒数だけメッセージの受信を待ちます。指定した秒数を過ぎてもメッセージを受信できない場合は，ErrTimedOutException が返されます。

flags

メッセージ受信後に，コネクションを解放するかどうかを指定します。

- DCNOFLAGS：メッセージ受信後，コネクションを解放しません。

  DCNOFLAGS を指定した場合は，障害時を除き，CloseRpc メソッドを実行するまでコネクションを解放しません。
- DCCLT_RCV_CLOSE：メッセージ受信後，コネクションを解放します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrProtoException

OpenRpc メソッドが実行されていません。または，OpenRpc メソッドは実行されていますが，Client .NET 構成定義の<tcpip>要素の type 属性に recv または sendrecv を指定していません。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

Hitachi.OpenTP1.Client.ErrTimedOutException

メッセージ受信時にタイムアウトしました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrInvalidPortException

受信用 port の指定に誤りがあります。

この例外は，Client .NET 構成定義の<tcpip>要素の openPortAtRecv 属性に true を指定し，recvPort 属性に指定したポートがすでに使用中の場合に発生します。

openPortAtRecv 属性に false を指定した場合は，OpenRpc メソッドが ErrFatalException を返します。

Hitachi.OpenTP1.Client.ErrConnfreeException

MHP からコネクションが解放されました。

注意事項

ReceiveAssembledMessage メソッドは，次に示す場合に CUP.NET に制御を戻します。

- メッセージ受信が完了した場合
- 不正なメッセージ長のメッセージを受信した場合
- 不正なセグメント情報のメッセージを受信した場合
- buff パラメタで指定した長さを超えるメッセージを受信した場合
- メッセージ受信時に，タイムアウトが発生した場合
- 相手システムからコネクションが解放された場合
- ネットワーク障害が発生した場合

ReceiveAssembledMessage メソッドの発行時に，MHP からコネクションが解放された場合，ErrConnfreeException でエラーリターンします。

●Rollback

説明

トランザクションをロールバックします。

Rollback メソッドを呼び出すと，グローバルトランザクションは終了します。

グローバルトランザクションの範囲外からは，SPP.NET または SPP をトランザクションとして実行できません。

宣言

【C#の場合】

```
public void Rollback(
);
```

【Visual Basic の場合】

```
Public Sub Rollback( _
)
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrProtoException

Rollback メソッドを不正なコンテキストから呼び出しています。

Hitachi.OpenTP1.Client.ErrHeuristicException

Rollback メソッドを実行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。

この例外は，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い場合に返されます。

この例外が返される原因になった 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，TP1/Server のメッセージログファイルを参照してください。

この例外が返されたあと，このプロセスはトランザクション下にありません。プロセスはグローバルトランザクションの範囲外です。

Hitachi.OpenTP1.Client.ErrHazardException

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。

この例外が返される原因になった 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，TP1/Server のメッセージログファイルを参照してください。

この例外が返されたあと，このプロセスはトランザクション下にありません。プロセスはグローバルトランザクションの範囲外です。

Hitachi.OpenTP1.Client.ErrNoBufsException

メモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrNotUpException

TP1/Server が稼働していません。

Hitachi.OpenTP1.Client.ErrTimedOutException

タイムアウトが発生しました。

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrConnfreeException

CUP.NET 実行プロセス側から常設コネクションが解放されました。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

●RollbackChained

説明

トランザクションをロールバックします。

RollbackChained メソッドが正常終了すると，新しいグローバルトランザクションが発生し，以降呼び出すメソッドは新しいグローバルトランザクションの範囲になります。

宣言

【C#の場合】

```
public void RollbackChained(
);
```

【Visual Basic の場合】

```
Public Sub RollbackChained( _
)
```

パラメタ

なし

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrProtoException

RollbackChained メソッドを不正なコンテキストから呼び出しています。

Hitachi.OpenTP1.Client.ErrHeuristicException

RollbackChained メソッドを実行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。

この例外は，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い場合に返されます。

この例外が返される原因になった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，TP1/Serverのメッセージログファイルを参照してください。

この例外が返されたあとも，このプロセスはトランザクション下にあり，グローバルトランザクションの範囲内です。

Hitachi.OpenTP1.Client.ErrHazardException

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。

この例外が返される原因になった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，TP1/Serverのメッセージログファイルを参照してください。

この例外が返されたあとも，このプロセスはトランザクション下にあり，グローバルトランザクションの範囲内です。

Hitachi.OpenTP1.Client.ErrNoBeginException

コミットまたは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。

この例外が返されたあと，このプロセスはトランザクション下にありません。

Hitachi.OpenTP1.Client.ErrHeuristicNoBeginException

RollbackChainedメソッドを実行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。

この例外は，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い場合に返されます。

この例外が返される原因になった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，TP1/Serverの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。

この例外が返されたあと，このプロセスはトランザクション下にありません。

Hitachi.OpenTP1.Client.ErrHazardNoBeginException

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。

この例外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，TP1/Serverの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。

この例外が返されたあと，このプロセスはトランザクション下にありません。

Hitachi.OpenTP1.Client.ErrNoBufsException

メモリ不足が発生しました。

Hitachi.OpenTP1.Client.ErrNotUpException

TP1/Server が稼働していません。

Hitachi.OpenTP1.Client.ErrIOErrException

何らかの入出力例外が発生しました。

Hitachi.OpenTP1.Client.ErrTimedOutException

タイムアウトが発生しました。

Hitachi.OpenTP1.Client.ErrConnfreeException

CUP.NET 実行プロセス側から常設コネクションが解放されました。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

●Send

説明

MHP にメッセージを送信します。

Send メソッドを実行する場合は，Client .NET 構成定義の<tcpip>要素の type 属性に send または sendrecv を指定して，OpenRpc メソッドをあらかじめ実行しておく必要があります。

宣言

【C#の場合】

```
public void Send(
  byte[] buff,
  int sendleng,
  string hostname,
  int portnum,
  int flags
);
```

【Visual Basic の場合】

```
Public Sub Send( _
  ByVal buff() As Byte, _
  ByVal sendleng As Integer, _
  ByVal hostname As String, _
  ByVal portnum As Integer, _
  ByVal flags As Integer _
)
```

パラメタ

buff

送信するメッセージが格納されている領域を指定します。

sendleng で指定する長さ以上の領域を指定してください。

sendleng

送信するメッセージの長さを指定します。

hostname

コネクションが確立されていない場合，接続する MHP が存在するノードのホスト名を指定します。

null を指定した場合は，OpenRpc メソッドを実行したときに取得した Client .NET 構成定義の<tcpip>要素の sendHost 属性の内容を参照します。

ホスト名として，10 進ドット記法の IP アドレスを指定することもできます。

コネクションが確立されている場合，このパラメタは無視されます。

portnum

コネクションが確立されていない場合，接続する MHP のポート番号を，0 から 65535 までの整数で指定します。

0 を指定すると，OpenRpc メソッドを実行したときに取得した Client .NET 構成定義の<tcpip>要素の sendPort 属性の内容を参照します。

コネクションが確立されている場合，このパラメタは無視されます。

flags

メッセージ送信後にコネクションを解放するかどうかを指定します。

- DCNOFLAGS：メッセージ送信後，コネクションを解放しません。

  DCNOFLAGS を指定した場合は，障害時を除き，CloseRpc メソッドを実行するまでコネクションを解放しません。
- DCCLT_SND_CLOSE：メッセージ送信後，コネクションを解放します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrProtoException

OpenRpc メソッドが実行されていません。または，OpenRpc メソッドは実行されていますが，Client .NET 構成定義の<tcpip>要素の type 属性に send または sendrecv を指定していません。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrHostUndefException

hostname パラメタの指定に誤りがあります。または，hostname パラメタと Client .NET 構成定義の<tcpip>要素の sendHost 属性の両方に，ホスト名が指定されていません。

Hitachi.OpenTP1.Client.ErrInvalidPortException

portnum パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrConnRefusedException

MHP に対するコネクションの確立要求が拒否されました。監視していないポートに対してコネクションを確立しようとしました。

注意事項

Send メソッドを実行してメッセージを送信しているときに，MHP からコネクションが解放されると，次に実行する Send メソッドは正常終了または異常終了します。

正常終了した場合は，その次に実行する Send メソッドで初めて異常終了します。そのため，CUP.NET を作成するときは注意してください。

●SendAssembledMessage

説明

受信メッセージの組み立て機能を使用してメッセージを送信します。SendAssembledMessage メソッドを実行する場合，Client .NET 構成定義の<tcpip>要素の type 属性に send または sendrecv を指定して，OpenRpc メソッドをあらかじめ実行しておく必要があります。

受信メッセージの組み立て機能を使用した場合，メッセージの先頭４バイトにメッセージ長（4 ＋ sendleng 引数の指定値）を付けて，buff[0]～buff[sendleng－1]の長さのメッセージを送信します。相手システムとのコネクションが確立されていない場合，hostname 引数および portnum 引数の指定に従ってコネクションを確立し，メッセージを送信します。

宣言

【C#の場合】

```
public void SendAssembledMessage(
  byte[] buff,
  int sendleng,
  string hostname,
  int portnum,
  int timeout,
  int flags
);
```

【Visual Basic の場合】

```
Public Sub SendAssembledMessage( _
  ByVal buff() As Byte, _
  ByVal sendleng As Integer, _
  ByVal hostname As String, _
  ByVal portnum As Integer, _
  ByVal timeout As Integer, _
  ByVal flags As Integer _
)
```

パラメタ

buff

送信するメッセージが格納されている領域を指定します。

sendleng パラメタで指定する長さ以上の領域を指定してください。

sendleng

送信するメッセージの長さを指定します。

hostname

コネクションが確立されていない場合，接続する相手システムのホスト名を指定します。

null を指定した場合は，OpenRpc メソッドを実行したときに取得した Client .NET 構成定義の<tcpip>要素の sendHost 属性の内容を参照します。

ホスト名として，10 進ドット記法の IP アドレスを指定することもできます。

コネクションが確立されている場合，このパラメタは無視されます。

portnum

コネクションが確立されていない場合，接続する相手システムのポート番号を，0 から 65535 までの整数で指定します。0 を指定すると，OpenRpc メソッドを実行したときに取得した Client .NET 構成定義の<tcpip>要素の sendPort 属性の内容を参照します。

コネクションが確立されている場合，このパラメタは無視されます。

timeout

予約引数です。指定した値は無視されます。

flags

メッセージ送信後にコネクションを解放するかどうかを指定します。

- DCNOFLAGS：メッセージ送信後，コネクションを解放しません。

  DCNOFLAGS を指定した場合は，障害時を除き，CloseRpc メソッドを実行するまでコネクションを解放しません。

- DCCLT_SND_CLOSE：メッセージ送信後，コネクションを解放します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrProtoException

OpenRpc メソッドが実行されていません。または，OpenRpc メソッドは実行されていますが，Client .NET 構成定義の<tcpip>要素の type 属性に send または sendrecv を指定していません。

Hitachi.OpenTP1.Client.ErrNetDownException

ネットワーク障害が発生しました。

Hitachi.OpenTP1.Client.ErrSyserrException

システムエラーが発生しました。

Hitachi.OpenTP1.Client.ErrHostUndefException

hostname パラメタの指定に誤りがあります。または，hostname パラメタと Client .NET 構成定義の<tcpip>要素の sendHost 属性の両方に，ホスト名が指定されていません。

Hitachi.OpenTP1.Client.ErrInvalidPortException

portnum パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrConnRefusedException

相手システムに対するコネクションの確立要求が拒否されました。監視していないポートに対してコネクションを確立しようとしました。

Hitachi.OpenTP1.Client.ErrTimedOutException

応答メッセージの受信時にタイムアウトになりました。

コネクションは解放されます。

Hitachi.OpenTP1.Client.ErrConnfreeException

相手システムからコネクションが解放されました。

Hitachi.OpenTP1.Client.ErrInvalidMessageException

不正なメッセージを受信しました。

Hitachi.OpenTP1.Client.ErrCollisionMessageException

送受信メッセージの衝突が発生しました。

注意事項

SendAssembledMessage メソッドを実行してメッセージを送信しているときに相手システムからコネクションが解放されると，次に実行する SendAssembledMessage メソッドは正常終了または異常終了します。

正常終了した場合は，その次に実行する SendAssembledMessage メソッドで初めて異常終了します。そのため，CUP.NET を作成するときは注意してください。

●SetConnectInformation

説明

端末識別情報を設定します。

このメソッドに指定した端末識別情報は，Client .NET 構成定義の<tp1Server>要素の host 属性に DCCM3 論理端末のホスト名を，<tp1Server>要素の port 属性または<rapService>要素の port 属性に DCCM3 論理端末のポート番号を指定し，次のどちらか方法で DCCM3 論理端末との常設コネクションを確立した場合に有効となります。

- OpenConnection メソッドを呼び出します。引数有りの OpenConnection メソッドの場合，引数 host に DCCM3 のホスト名，引数 port に DCCM3 のポート番号を指定します。

- Client .NET 構成定義の<rapService>要素の autoConnect 属性に true を指定し，Call メソッドを呼び出します。

このメソッドを呼び出した場合，Client .NET 構成定義の<rapService>要素の connectInformation 属性に指定した端末識別情報は，再び OpenRpc メソッドを呼び出すまで無視されます。

なお，このメソッドに指定した端末識別情報は，DCCM3 論理端末との常設コネクション確立時に認識されます。このメソッドを複数回呼び出した場合は，DCCM3 論理端末との常設コネクションを確立する直前に指定した端末識別情報が有効となります。

また，ネームサービスを使用した RPC，またはスケジューラダイレクト機能を使用した RPC の場合は，このメソッドで指定した端末識別情報は無視されます。

宣言

【C#の場合】

```
public void SetConnectInformation(
  byte[] inf,
  short inf_len
);
```

【Visual Basic の場合】

```
Public Sub SetConnectInformation( _
  ByVal inf() As Byte, _
  ByVal inf_len As Short _
)
```

パラメタ

inf

端末識別情報として，DCCM3 論理端末の論理端末名称を 64 バイト以内の EBCDIK コードで指定します。

ただし，DCCM3 側では先頭 8 バイト目までに指定した値だけが有効になり，9 バイト目以降に指定された値は無視されます。

inf_len

端末識別情報長を指定します。

1 から 64 までの範囲のバイト長を指定します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

●SetDataTraceMode

説明

データトレースを取得するかどうかを指定します。

宣言

【C#の場合】

```
public void SetDataTraceMode(
  string TrcPath,
  int size,
  int DataSize,
  System.Boolean flag
);
```

【Visual Basic の場合】

```
Public Sub SetDataTraceMode( _
  ByVal TrcPath As String, _
  ByVal size As Integer, _
  ByVal DataSize As Integer, _
  ByVal flag As System.Boolean _
)
```

パラメタ

TrcPath

データトレースを出力するディレクトリを指定します。

flag パラメタに false を指定した場合は無視されます。

size

出力するデータトレースファイルのサイズを 4096 から 1048576（単位：バイト）までの範囲で指定します。

flag パラメタに false を指定した場合は無視されます。

DataSize

出力する一つのデータトレースの最大データ長を 16 から 1048576 までの範囲で指定します。

flag パラメタに false を指定した場合は無視されます。

flag

データトレースを取得するかどうかを指定します。

- true：データトレースを取得します。
- false：データトレースを取得しません。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

●SetErrorTraceMode

説明

エラートレースを取得するかどうかを指定します。

宣言

【C#の場合】

```
public void SetErrorTraceMode(
  string TrcPath,
  int size,
  System.Boolean flag
);
```

【Visual Basic の場合】

```
Public Sub SetErrorTraceMode( _
  ByVal TrcPath As String, _
  ByVal size As Integer, _
  ByVal flag As System.Boolean _
)
```

パラメタ

TrcPath

エラートレースを出力するディレクトリを指定します。

flag パラメタに false を指定した場合は無視されます。

size

出力するエラートレースファイルのサイズを 4096 から 1048576（単位：バイト）までの範囲で指定します。

flag パラメタに false を指定した場合は無視されます。

flag

エラートレースを取得するかどうかを指定します。

- true：エラートレースを取得します。
- false：エラートレースを取得しません。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

●SetExtendLevel

説明

Client .NET の機能の拡張レベルを指定します。

機能の拡張レベルを複数指定する場合，それぞれの指定値の論理和を指定します。

宣言

【C#の場合】

```
public void SetExtendLevel(
  int flags
);
```

【Visual Basic の場合】

```
Public Sub SetExtendLevel( _
  ByVal flags As Integer _
)
```

パラメタ

flags

Client .NET の機能の拡張レベルを指定します。

- 0x00000000

  Client .NET の機能を拡張しません。

- 0x00000001

  Client .NET 機能を拡張します。Call メソッド呼び出し時，自 CUP.NET の IP アドレスをサービスに連絡します。

  呼び出したサービスで dc_rpc_get_callers_address メソッドを実行し，CUP.NET のアドレスを求める必要がある場合に指定してください。

戻り値

なし

例外

なし

●SetMethodTraceMode

説明

メソッドトレースを取得するかどうかを指定します。

宣言

【C#の場合】

```
public void SetMethodTraceMode(
  string TrcPath,
  int size,
```

```
  System.Boolean flag
);
```

【Visual Basic の場合】

```
Public Sub SetMethodTraceMode( _
  ByVal TrcPath As String, _
  ByVal size As Integer, _
  ByVal flag As System.Boolean _
)
```

パラメタ

TrcPath

メソッドトレースを出力するディレクトリを指定します。

flag パラメタに false を指定した場合は無視されます。

size

出力するメソッドトレースファイルのサイズを 4096 から 1048576（単位：バイト）までの範囲で指定します。

flag パラメタに false を指定した場合は無視されます。

flag

メソッドトレースを取得するかどうかを指定します。

- true：メソッドトレースを取得します。
- false：メソッドトレースを取得しません。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

●SetRapDelay

説明

rap サーバと CUP.NET 間の通信遅延時間を設定します。

このメソッドを設定すると，rap サーバ側の最大応答待ち時間の時間監視を指定した値の分だけ早く終了させ，CUP.NET 側の最大応答待ち時間の監視時間タイムアウトによるメッセージのすれ違いを防止できます。

SetRpcWatchTime メソッドで指定した最大応答待ち時間が 0 の場合，このメソッドで指定した通信遅延時間は無視されます。

また，最大応答待ち時間から rap サーバ通信時間を引いた値が 0 または負の値になるときは，rap サーバ側での最大応答待ち時間を 1 と仮定して rap サーバが動作します。

なお，このメソッドの指定を有効にするには，Client .NET 構成定義で<rapService>要素の watchTimeInheritance 属性に true を指定するか，または SetRpcExtend メソッドの extendoption パラメタに DCRPC_WATCHTIMINHERIT を指定してください。

宣言

【C#の場合】

```
public void SetRapDelay(
  int sec
);
```

【Visual Basic の場合】

```
Public Sub SetRapDelay( _
  ByVal sec As Integer _
)
```

パラメタ

sec

rap サーバと CUP.NET 間の通信遅延時間を 0 から 65535（単位：秒）までの範囲で指定します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

sec パラメタに指定された値が，0 から 65535 までの範囲にありません。

●SetRapInquireTime

説明

CUP.NET がサーバに対して問い合わせを行ってから，次の問い合わせをするまでの間隔の最大時間を設定します。この値は rap サーバで監視するタイマです。

指定時間を超えても問い合わせがない場合，rap サーバは強制的に常設コネクションを切断します。

このメソッドは TP1Client クラスのインスタンスが存在している間，呼び出せます。

宣言

【C#の場合】

```
public void SetRapInquireTime(
  int sec
);
```

【Visual Basic の場合】

```
Public Sub SetRapInquireTime( _
  ByVal sec As Integer _
)
```

パラメタ

sec

rap サーバが監視する問い合わせ間隔最大時間を 0 から 1048575（単位：秒）までの範囲で指定します。

0 を指定した場合は，rap サーバは CUP.NET からの問い合わせを無限に待ちます。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

sec パラメタに指定された値が，0 から 1048575 までの範囲にありません。

注意事項

常設コネクションの確立前に「問い合わせ間隔最大時間」を変更できます。

常設コネクションの確立後に「問い合わせ間隔最大時間」を変更しても，rap サーバ側の監視タイマには反映されません。

この場合，確立中の常設コネクションを切断し，次回常設コネクションを確立するタイミングで rap サーバ側の監視タイマに反映されます。

●SetRpcExtend

説明

Client .NET から発行する RPC の機能拡張オプションを指定します。

RPC 機能の拡張レベルを複数指定する場合，それぞれの指定値の論理和を指定します。

宣言

【C#の場合】

```
public void SetRpcExtend(
  int extendoption
);
```

【Visual Basic の場合】

```
Public Sub SetRpcExtend( _
  ByVal extendoption As Integer _
)
```

パラメタ

extendoption

RPC 機能の拡張レベルを指定します。

- DCRPC_SCD_LOAD_PRIORITY

  サービス要求を受け付けた窓口となる TP1/Server を優先して負荷分散するかどうかを指定します。

オプションの真偽	動作
真の場合	Client .NET 構成定義で，<scheduleService>要素の usePriority 属性に true を指定したときと同じ動作をします。
偽の場合	Client .NET 構成定義で，<scheduleService>要素の usePriority 属性に false を指定したときと同じ動作をします。

- DCRPC_WATCHTIMINHERIT

  リモート API 機能を使用した RPC を行う場合に，CUP.NET の最大応答待ち時間を rap サーバ側に引き継ぐかどうかを指定します。

オプションの真偽	動作
真の場合	Client .NET 構成定義で，<rapService>要素の watchTimeInheritance 属性に true を指定したときと同じ動作をします。
偽の場合	Client .NET 構成定義で，<rapService>要素の watchTimeInheritance 属性に false を指定したときと同じ動作をします。

- DCRPC_RAP_AUTOCONNECT

  リモート API 機能を使用した RPC を行う場合に，Client .NET が自動的にコネクションを確立するかどうかを指定します。

  すでに非オートコネクトモードを使用し，OpenConnection メソッドを呼び出してコネクションを確立している場合は，このオプションは無視されます。

オプションの真偽	動作
真の場合	Client .NET 構成定義で，<rapService>要素の autoConnect 属性に true を指定したときと同じ動作をします。
偽の場合	Client .NET 構成定義で，<rapService>要素の autoConnect 属性に false を指定したときと同じ動作をします。

- DCRPC_WATCHTIMRPCINHERIT

  CUP.NET の最大応答待ち時間を，サーバ側に引き継ぐかどうかを指定します。

オプションの真偽	動作
真の場合	Client .NET 構成定義で，<rpc>要素の watchTimeNotification 属性に true を指定したときと同じ動作をします。
偽の場合	Client .NET 構成定義で，<rpc>要素の watchTimeNotification 属性に false を指定したときと同じ動作をします。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

extendoption パラメタの指定に誤りがあります。

●SetRpcWatchTime

説明

同期応答型 RPC の場合に，CUP.NET から SPP.NET または SPP へサービス要求を送ってからサービスの応答が返るまでの最大応答待ち時間を設定します。

このメソッドで設定する値は，Client .NET の内部通信での最大応答待ち時間としても使用されます。

宣言

【C#の場合】

```
public void SetRpcWatchTime(
  int sec
);
```

【Visual Basic の場合】

```
Public Sub SetRpcWatchTime( _
  ByVal sec As Integer _
)
```

パラメタ

sec

最大応答待ち時間を 0 から 65535（単位：秒）までの範囲で指定します。

0 を指定した場合は，応答を受信するまで無限に待ちます。使用する環境によっては，サーバがダウンしても TCP/IP コネクションの切断を検出できない場合があります。この場合，待ち時間を無限にすると存在しないサーバからの応答を永久に待ち続けることになるため，適当な待ち時間を設定してください。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

sec パラメタに指定された値が，0 から 65535 までの範囲にありません。

●SetTP1Server

説明

窓口となる TP1/Server のホスト名とポート番号を設定します。

宣言

【C#の場合】

```
public void SetTP1Server(
  string host,
  int port
);
```

【Visual Basic の場合】

```
Public Sub SetTP1Server( _
  ByVal host As String, _
  ByVal port As Integer _
)
```

パラメタ

host

窓口となる TP1/Server のホスト名を指定します。

リモート API 機能を使用した RPC を行う場合は，通信先の rap リスナーのホスト名を指定します。

CUP.NET と TP1/Server との間にファイアウォールがある場合は，ファイアウォールのホスト名を指定します。

port

窓口となる TP1/Server 上で動作しているスケジュールサーバのポート番号，またはネームサーバのポート番号を指定します。リモート API 機能を使用した RPC を行う場合は，通信先の rap リスナーのポート番号を指定します。

ポート番号は，5001 から 65535 までの範囲で指定します。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrHostUndefException

host パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrInvalidPortException

port パラメタの指定に誤りがあります。

Hitachi.OpenTP1.Client.ErrProtoException

メソッドの発行順序に誤りがあります。リモート API 機能を使用した RPC を行う場合で，すでに rap サーバとの間に常設コネクションが確立されている状態でこのメソッドが呼び出されました。

●SetTraceArray

説明

パラメタに指定された配列にエラートレースを取得するかどうかを指定します。

宣言

【C#の場合】

```
public void SetTraceArray(
  string[] array
);
```

【Visual Basic の場合】

```
Public Sub SetTraceArray( _
  ByVal array() As String _
)
```

パラメタ

array

エラートレースを格納する String 配列を指定します。

null を指定した場合，エラートレースを取得しません。

戻り値

なし

例外

なし

●SetUapTraceMode

説明

UAP トレースを取得するかどうかを指定します。

宣言

【C#の場合】

```
public void SetUapTraceMode(
  string TrcPath,
  int size,
  System.Boolean flag
);
```

【Visual Basic の場合】

```
Public Sub SetUapTraceMode( _
  ByVal TrcPath As String, _
  ByVal size As Integer, _
  ByVal flag As System.Boolean _
)
```

パラメタ

TrcPath

UAP トレースを出力するディレクトリを指定します。

flag パラメタに false を指定した場合は無視されます。

size

出力する UAP トレースファイルのサイズを 4096 から 1048576（単位：バイト）までの範囲で指定します。

flag パラメタに false を指定した場合は無視されます。

flag

UAP トレースを取得するかどうかを指定します。

- true：UAP トレースを取得します。
- false：UAP トレースを取得しません。

戻り値

なし

例外

Hitachi.OpenTP1.Client.ErrInvalidArgsException

パラメタの指定に誤りがあります。

TP1ClientError

TP1ClientError の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Client.TP1ClientError
```

説明

Client .NET で使用するエラーコードを定義するクラスです。

フィールドの一覧

名称	説明
DCCLNER_BUFFER_OVERFLOW	受信バッファのオーバフローが発生しました。
DCCLNER_COLLISION_MSG	送受信メッセージの衝突が発生しました。
DCCLNER_CONNFREE	コネクションが切断されました。
DCCLNER_CONNREFUSED	接続がリモートで拒否されました。
DCCLNER_EOF	入力の途中で，予期しないで終了しました。
DCCLNER_FATAL	Client .NET 構成定義の指定に誤りがあります。
DCCLNER_INVALID_ARGS	メソッドのパラメタの指定に誤りがあります。
DCCLNER_INVALID_HOST	ホスト名が指定されていないか，または指定に誤りがあります。
DCCLNER_INVALID_MSG	不正なメッセージを受信しました。
DCCLNER_INVALID_PORT	ポート番号が指定されていないか，または指定に誤りがあります。
DCCLNER_IO	何らかの入出力例外が発生しました。
DCCLNER_MARSHAL	カスタムレコードのマーシャリング/アンマーシャリング処理に失敗しました。
DCCLNER_MESSAGE_TOO_BIG	Call メソッドに指定した入力パラメタ長が最大値を超えています。
DCCLNER_NETDOWN	ネットワーク障害が発生しました。または，通信先の TP1/Server が稼働していません。
DCCLNER_NETDOWN_C	Client .NET 側でタイムアウトが発生しました。
DCCLNER_NETDOWN_S	TP1/Server と，SPP.NET または SPP との間でネットワーク障害が発生しました。
DCCLNER_NOBUFS	メモリ不足が発生しました。

名称	説明
DCCLNER_NOT_UP	指定したサービスが存在するノードの TP1/Server が稼働していません。
DCCLNER_PARAM	パラメタエラーが発生しました。
DCCLNER_PROTO	メソッドの発行順序に誤りがあります。
DCCLNER_REPLY_TOO_BIG	サーバから返された応答の長さが，CUP.NET で用意した領域（out_data パラメタの指定値）の長さを超えています。
DCCLNER_SYS	システムエラーが発生しました。
DCCLNER_TIMEOUT	タイムアウトが発生しました。
DCCLNER_TIMEOUT_C	Client .NET 側でタイムアウトが発生しました。
DCCLNER_TIMEOUT_S	TP1/Server 側でサービスの実行中にタイムアウトが発生しました。 または，サービス要求先の SPP.NET もしくは SPP が処理を終了する前に異常終了しました。
DCCLNER_UNEXPECT	予期しないエラーです。
DCCLNTRNER_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。
DCCLNTRNER_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。 新しいトランザクションは開始できませんでした。
DCCLNTRNER_HEURISTIC	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ません。
DCCLNTRNER_HEURISTIC_NO_BEGIN	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ません。 新しいトランザクションは開始できませんでした。
DCCLNTRNER_NO_BEGIN	コミットまたは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。
DCCLNTRNER_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。
DCCLNTRNER_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。
DCCLNTRNER_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。 新しいトランザクションは開始できませんでした。
DCCLNTRNER_TM	トランザクションサービスでエラーが発生したため，トランザクションを開始できませんでした。

フィールドの詳細

●DCCLNER_BUFFER_OVERFLOW

説明

受信バッファのオーバフローが発生しました。

宣言

【C#の場合】

```
public const int DCCLNER_BUFFER_OVERFLOW
```

【Visual Basic の場合】

```
Public Const DCCLNER_BUFFER_OVERFLOW As Integer
```

●DCCLNER_COLLISION_MSG

説明

送受信メッセージの衝突が発生しました。

宣言

【C#の場合】

```
public const int DCCLNER_COLLISION_MSG
```

【Visual Basic の場合】

```
Public Const DCCLNER_COLLISION_MSG As Integer
```

●DCCLNER_CONNFREE

説明

コネクションが切断されました。

宣言

【C#の場合】

```
public const int DCCLNER_CONNFREE
```

【Visual Basic の場合】

```
Public Const DCCLNER_CONNFREE As Integer
```

●DCCLNER_CONNREFUSED

説明

接続がリモートで拒否されました。

宣言

【C#の場合】

```
public const int DCCLNER_CONNREFUSED
```

【Visual Basic の場合】

```
Public Const DCCLNER_CONNREFUSED As Integer
```

●DCCLNER_EOF

説明

入力の途中で，予期しないで終了しました。

宣言

【C#の場合】

```
public const int DCCLNER_EOF
```

【Visual Basic の場合】

```
Public Const DCCLNER_EOF As Integer
```

●DCCLNER_FATAL

説明

Client .NET 構成定義の指定に誤りがあります。

宣言

【C#の場合】

```
public const int DCCLNER_FATAL
```

【Visual Basic の場合】

```
Public Const DCCLNER_FATAL As Integer
```

●DCCLNER_INVALID_ARGS

説明

メソッドのパラメタの指定に誤りがあります。

宣言

【C#の場合】

```
public const int DCCLNER_INVALID_ARGS
```

【Visual Basic の場合】

```
Public Const DCCLNER_INVALID_ARGS As Integer
```

●DCCLNER_INVALID_HOST

説明

ホスト名が指定されていないか，または指定に誤りがあります。

宣言

【C#の場合】

```
public const int DCCLNER_INVALID_HOST
```

【Visual Basic の場合】

```
Public Const DCCLNER_INVALID_HOST As Integer
```

●DCCLNER_INVALID_MSG

説明

不正なメッセージを受信しました。

宣言

【C#の場合】

```
public const int DCCLNER_INVALID_MSG
```

【Visual Basic の場合】

```
Public Const DCCLNER_INVALID_MSG As Integer
```

●DCCLNER_INVALID_PORT

説明

ポート番号が指定されていないか，または指定に誤りがあります。

宣言

【C#の場合】

```
public const int DCCLNER_INVALID_PORT
```

【Visual Basic の場合】

```
Public Const DCCLNER_INVALID_PORT As Integer
```

●DCCLNER_IO

説明

何らかの入出力例外が発生しました。

宣言

【C#の場合】

```
public const int DCCLNER_IO
```

【Visual Basic の場合】

```
Public Const DCCLNER_IO As Integer
```

●DCCLNER_MARSHAL

説明

カスタムレコードのマーシャリング/アンマーシャリング処理に失敗しました。

宣言

【C#の場合】

```
public const int DCCLNER_MARSHAL
```

【Visual Basic の場合】

```
Public Const DCCLNER_MARSHAL As Integer
```

●DCCLNER_MESSAGE_TOO_BIG

説明

Call メソッドに指定した入力パラメタ長が最大値を超えています。

宣言

【C#の場合】

```
public const int DCCLNER_MESSAGE_TOO_BIG
```

【Visual Basic の場合】

```
Public Const DCCLNER_MESSAGE_TOO_BIG As Integer
```

●DCCLNER_NETDOWN

説明

ネットワーク障害が発生しました。または，通信先の TP1/Server が稼働していません。

宣言

【C#の場合】

```
public const int DCCLNER_NETDOWN
```

【Visual Basic の場合】

```
Public Const DCCLNER_NETDOWN As Integer
```

●DCCLNER_NETDOWN_C

説明

Client .NET 側でタイムアウトが発生しました。

宣言

【C#の場合】

```
public const int DCCLNER_NETDOWN_C
```

【Visual Basic の場合】

```
Public Const DCCLNER_NETDOWN_C As Integer
```

●DCCLNER_NETDOWN_S

説明

TP1/Server と，SPP.NET または SPP との間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 DCCLNER_NETDOWN_S
```

【Visual Basic の場合】

```
Public Const DCCLNER_NETDOWN_S As Integer
```

●DCCLNER_NOBUFS

説明

メモリ不足が発生しました。

宣言

【C#の場合】

```
public const int DCCLNER_NOBUFS
```

【Visual Basic の場合】

```
Public Const DCCLNER_NOBUFS As Integer
```

●DCCLNER_NOT_UP

説明

指定したサービスが存在するノードの TP1/Server が稼働していません。

宣言

【C#の場合】

```
public const int DCCLNER_NOT_UP
```

【Visual Basic の場合】

```
Public Const DCCLNER_NOT_UP As Integer
```

●DCCLNER_PARAM

説明

パラメタエラーが発生しました。

宣言

【C#の場合】

```
public const int DCCLNER_PARAM
```

【Visual Basic の場合】

```
Public Const DCCLNER_PARAM As Integer
```

●DCCLNER_PROTO

説明

メソッドの発行順序に誤りがあります。

宣言

【C#の場合】

```
public const int DCCLNER_PROTO
```

【Visual Basic の場合】

```
Public Const DCCLNER_PROTO As Integer
```

●DCCLNER_REPLY_TOO_BIG

説明

サーバから返された応答の長さが，CUP.NET で用意した領域（out_data パラメタの指定値）の長さを超えています。

宣言

【C#の場合】

```
public const int DCCLNER_REPLY_TOO_BIG
```

【Visual Basic の場合】

```
Public Const DCCLNER_REPLY_TOO_BIG As Integer
```

●DCCLNER_SYS

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 DCCLNER_SYS
```

【Visual Basic の場合】

```
Public Const DCCLNER_SYS As Integer
```

●DCCLNER_TIMEOUT

説明

タイムアウトが発生しました。

宣言

【C#の場合】

```
public const int DCCLNER_TIMEOUT
```

【Visual Basic の場合】

```
Public Const DCCLNER_TIMEOUT As Integer
```

●DCCLNER_TIMEOUT_C

説明

Client .NET 側でタイムアウトが発生しました。

宣言

【C#の場合】

```
public const int DCCLNER_TIMEOUT_C
```

【Visual Basic の場合】

```
Public Const DCCLNER_TIMEOUT_C As Integer
```

●DCCLNER_TIMEOUT_S

説明

TP1/Server 側でサービスの実行中にタイムアウトが発生しました。

または，サービス要求先の SPP.NET もしくは SPP が処理を終了する前に異常終了しました。

宣言

【C#の場合】

```
public const int DCCLNER_TIMEOUT_S
```

【Visual Basic の場合】

```
Public Const DCCLNER_TIMEOUT_S As Integer
```

●DCCLNER_UNEXPECT

説明

予期しないエラーです。

宣言

【C#の場合】

```
public const int DCCLNER_UNEXPECT
```

【Visual Basic の場合】

```
Public Const DCCLNER_UNEXPECT As Integer
```

●DCCLNTRNER_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。

宣言

【C#の場合】

```
public const int DCCLNTRNER_HAZARD
```

【Visual Basic の場合】

```
Public Const DCCLNTRNER_HAZARD As Integer
```

●DCCLNTRNER_HAZARD_NO_BEGIN

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。
新しい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 DCCLNTRNER_HAZARD_NO_BEGIN
```

【Visual Basic の場合】

```
Public Const DCCLNTRNER_HAZARD_NO_BEGIN As Integer
```

●DCCLNTRNER_HEURISTIC

説明

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ません。

宣言

【C#の場合】

```
public const int DCCLNTRNER_HEURISTIC
```

【Visual Basic の場合】

```
Public Const DCCLNTRNER_HEURISTIC As Integer
```

●DCCLNTRNER_HEURISTIC_NO_BEGIN

説明

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ません。
新しい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 DCCLNTRNER_HEURISTIC_NO_BEGIN
```

【Visual Basic の場合】

```
Public Const DCCLNTRNER_HEURISTIC_NO_BEGIN As Integer
```

●DCCLNTRNER_NO_BEGIN

説明

コミットまたは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。

宣言

【C#の場合】

```
public const int DCCLNTRNER_NO_BEGIN
```

【Visual Basic の場合】

```
Public Const DCCLNTRNER_NO_BEGIN As Integer
```

●DCCLNTRNER_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 DCCLNTRNER_RM
```

【Visual Basic の場合】

```
Public Const DCCLNTRNER_RM As Integer
```

●DCCLNTRNER_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 DCCLNTRNER_ROLLBACK
```

【Visual Basic の場合】

```
Public Const DCCLNTRNER_ROLLBACK As Integer
```

●DCCLNTRNER_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。

新しい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 DCCLNTRNER_ROLLBACK_NO_BEGIN
```

【Visual Basic の場合】

```
Public Const DCCLNTRNER_ROLLBACK_NO_BEGIN As Integer
```

●DCCLNTRNER_TM

説明

トランザクションサービスでエラーが発生したため，トランザクションを開始できませんでした。

宣言

【C#の場合】

```
public const int DCCLNTRNER_TM
```

【Visual Basic の場合】

```
Public Const DCCLNTRNER_TM As Integer
```

TP1ClientException

TP1ClientException の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- System.Exception
    +- Hitachi.OpenTP1.TP1Exception
      +- Hitachi.OpenTP1.Client.TP1ClientException
```

実装インタフェース

System.Runtime.Serialization.ISerializable

説明

TP1ClientException クラスは，Client .NET が返すすべての例外のスーパークラスです。

TP1ClientFlags

TP1ClientFlags の概要

名前空間

Hitachi.OpenTP1.Client

継承関係

```
System.Object
  +- Hitachi.OpenTP1.Client.TP1ClientFlags
```

説明

Client .NET で使用するメソッドのパラメタで用いるフラグを定義するクラスです。

フィールドの一覧

名称	説明
DCCLT_RCV_CLOSE	メッセージ受信後，コネクションを解放します。
DCCLT_SND_CLOSE	メッセージ送信後，コネクションを解放します。
DCNOFLAGS	RPC の形態に同期応答型 RPC を指定します。
DCRPC_CHAINED	RPC の形態に連鎖 RPC を指定します。
DCRPC_NOREPLY	RPC の形態に非応答型 RPC を指定します。
DCRPC_RAP_AUTOCONNECT	リモート API 機能を利用した RPC を行う場合に，Client .NET が自動的にコネクションを確立するかどうかを指定します。
DCRPC_SCD_LOAD_PRIORITY	サービス要求を受け付けた窓口となる TP1/Server を優先して負荷分散するかどうかを指定します。
DCRPC_TPNOTRAN	RPC の形態にトランザクションを引き継がない RPC を指定します。
DCRPC_WATCHTIMINHERIT	リモート API 機能を使用した RPC を行う場合に，CUP.NET の最大応答待ち時間を rap サーバ側に引き継ぐかどうかを指定します。
DCRPC_WATCHTIMRPCINHERIT	CUP.NET の最大応答待ち時間を，サーバ側に引き継ぐかどうかを指定します。

フィールドの詳細

●DCCLT_RCV_CLOSE

説明

メッセージ受信後，コネクションを解放します。

宣言

【C#の場合】

```
public const int DCCLT_RCV_CLOSE
```

【Visual Basic の場合】

```
Public Const DCCLT_RCV_CLOSE As Integer
```

●DCCLT_SND_CLOSE

説明

メッセージ送信後，コネクションを解放します。

宣言

【C#の場合】

```
public const int DCCLT_SND_CLOSE
```

【Visual Basic の場合】

```
Public Const DCCLT_SND_CLOSE As Integer
```

●DCNOFLAGS

説明

RPC の形態に同期応答型 RPC を指定します。

宣言

【C#の場合】

```
public const int DCNOFLAGS
```

【Visual Basic の場合】

```
Public Const DCNOFLAGS As Integer
```

●DCRPC_CHAINED

説明

RPC の形態に連鎖 RPC を指定します。

宣言

【C#の場合】

```
public const int DCRPC_CHAINED
```

【Visual Basic の場合】

```
Public Const DCRPC_CHAINED As Integer
```

●DCRPC_NOREPLY

説明

RPC の形態に非応答型 RPC を指定します。

宣言

【C#の場合】

```
public const int DCRPC_NOREPLY
```

【Visual Basic の場合】

```
Public Const DCRPC_NOREPLY As Integer
```

●DCRPC_RAP_AUTOCONNECT

説明

リモート API 機能を利用した RPC を行う場合に，Client .NET が自動的にコネクションを確立するかどうかを指定します。

宣言

【C#の場合】

```
public const int DCRPC_RAP_AUTOCONNECT
```

【Visual Basic の場合】

```
Public Const DCRPC_RAP_AUTOCONNECT As Integer
```

●DCRPC_SCD_LOAD_PRIORITY

説明

サービス要求を受け付けた窓口となる TP1/Server を優先して負荷分散するかどうかを指定します。

宣言

【C#の場合】

```
public const int DCRPC_SCD_LOAD_PRIORITY
```

【Visual Basic の場合】

```
Public Const DCRPC_SCD_LOAD_PRIORITY As Integer
```

●DCRPC_TPNOTRAN

説明

RPC の形態にトランザクションを引き継がない RPC を指定します。

宣言

【C#の場合】

```
public const int DCRPC_TPNOTRAN
```

【Visual Basic の場合】

```
Public Const DCRPC_TPNOTRAN As Integer
```

●DCRPC_WATCHTIMINHERIT

説明

リモート API 機能を使用した RPC を行う場合に，CUP.NET の最大応答待ち時間を rap サーバ側に引き継ぐかどうかを指定します。

宣言

【C#の場合】

```
public const int DCRPC_WATCHTIMINHERIT
```

【Visual Basic の場合】

```
Public Const DCRPC_WATCHTIMINHERIT As Integer
```

●DCRPC_WATCHTIMRPCINHERIT

説明

CUP.NET の最大応答待ち時間を，サーバ側に引き継ぐかどうかを指定します。

宣言

【C#の場合】

```
public const int DCRPC_WATCHTIMRPCINHERIT
```

【Visual Basic の場合】

```
Public Const DCRPC_WATCHTIMRPCINHERIT As Integer
```

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	パラメタに設定した値に誤りがあります。
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。
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	パラメタに設定した値に誤りがあります。
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。
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。
DCPRFER_PARAM	パラメタ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付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。
DCRAPER_PANIC_SV	rap リスナーでシステム障害が発生しました。
DCRAPER_PARAM	パラメタに誤りがあります。
DCRAPER_PROTO	プロトコル不正です。

名称	説明
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	パラメタに設定した値に誤りがあります。
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は定義の範囲外です。
DCRPCER_MESSAGE_TOO_BIG	設定したメッセージ長が，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_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，ドメイン名に誤りがあります。
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。
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。

名称	説明
DCRPCER_SECCHK	サービス要求先の 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。サービス要求した CUP.NET には，SPP.NET または SPP へのアクセス権限がありません。
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.NET もしくは SPP に，またはテストモードでない UAP からテストモードの SPP に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。
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。
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。

名称	説明
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，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	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。
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を，コミットできず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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●DCADMER_PARAM

説明

パラメタに設定した値に誤りがあります。

宣言

【C#の場合】

```
public const int DCADMER_PARAM
```

【Visual Basic の場合】

```
Public Const DCADMER_PARAM As Integer
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●DCLOGER_PARAM_ARGS

説明

パラメタに設定した値に誤りがあります。

宣言

【C#の場合】

```
public const int DCLOGER_PARAM_ARGS
```

【Visual Basic の場合】

```
Public Const DCLOGER_PARAM_ARGS As Integer
```

●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
```

●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
```

●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
```

●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
```

●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
```

●DCPRFER_PARAM

説明

パラメタ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
```

●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
```

●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
```

●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付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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●DCRAPER_PARAM

説明

パラメタに誤りがあります。

宣言

【C#の場合】

```
public const int DCRAPER_PARAM
```

【Visual Basic の場合】

```
Public Const DCRAPER_PARAM As Integer
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●DCRPCER_INVALID_ARGS

説明

パラメタに設定した値に誤りがあります。

宣言

【C#の場合】

```
public const int DCRPCER_INVALID_ARGS
```

【Visual Basic の場合】

```
Public Const DCRPCER_INVALID_ARGS As Integer
```

●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
```

●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は定義の範囲外です。

宣言

【C#の場合】

```
public const int DCRPCER_INVALID_REPLY
```

【Visual Basic の場合】

```
Public Const DCRPCER_INVALID_REPLY As Integer
```

●DCRPCER_MESSAGE_TOO_BIG

説明

設定したメッセージ長が，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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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，ドメイン名に誤りがあります。

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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●DCRPCER_SECCHK

説明

サービス要求先の 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。サービス要求した CUP.NET には，SPP.NET または SPP へのアクセス権限がありません。

宣言

【C#の場合】

```
public const int DCRPCER_SECCHK
```

【Visual Basic の場合】

```
Public Const DCRPCER_SECCHK As Integer
```

●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
```

●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
```

●DCRPCER_SERVICE_CLOSED

説明

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。

宣言

【C#の場合】

```
public const int DCRPCER_SERVICE_CLOSED
```

【Visual Basic の場合】

```
Public Const DCRPCER_SERVICE_CLOSED As Integer
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●DCRPCER_TESTMODE

説明

テストモードの UAP からテストモードでない SPP.NET もしくは SPP に，またはテストモードでない UAP からテストモードの SPP に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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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
```

●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，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_HEURISTIC_NO_BEGIN
```

【Visual Basic の場合】

```
Public Const DCTRNER_HEURISTIC_NO_BEGIN As Integer
```

●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
```

●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
```

●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
```

●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
```

●DCTRNER_ROLLBACK_NO_BEGIN

説明

コミットしようとしたトランザクションを，コミットできず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
```

●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
```

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
```

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
```

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が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
```

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
```

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
```

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パラメタです。

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( _
)
```

パラメタ

なし

例外

なし

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 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 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 _
)
```

パラメタ

errCode

エラーコードです。

argInfo

エラー情報です。

例外

なし

●TP1UserException

説明

エラーコード，エラー情報，および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 _
)
```

パラメタ

errCode

エラーコードです。

argInfo

エラー情報です。

message

例外の原因を説明するエラーメッセージです。

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
```

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
```

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

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()
```

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
```

例外

なし

7 障害運用

この章では，Client .NET で障害が発生した場合の対処方法を説明します。

7.1　障害の種類と対処方法

Client .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 7-1　障害の種類とユーザの取る処置

障害の種類	障害の内容	Client .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トレースファイルに出力します。 2. 例外を CUP.NET 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，再度クライアント処理を実行してください。 必要に応じて構成定義の内容，または TP1/Server 側の定義の内容を見直してください。 また，ネームサービスを使用した RPC では，CUP と通信先 TP1/Server 間で，NAT（Network Address Translator：アドレス変換）をしている場合に通信障害が発生することがあります。これを回避するためには，Client .NET 側で，<nameService>要素の multiHomedHost 属性に true を指定してください。
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）	TP1UserException を CUP.NET 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）	TP1RemoteException を CUP.NET 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	クライアントスタブでのデータ変換エラー	TP1MarshalException を CUP.NET 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。
クライアント環境障害	・ファイル障害 ・メモリ不足 など	1. 障害発生時の情報をトレースファイルに出力します。※2 2. 例外を CUP.NET に通知します。	障害の要因を取り除いてください。
メッセージ受信障害	不正データ（4 バイト以下のメッセージ）の受信	1. 障害発生時の情報をトレースファイルに出力します。 2. コネクションを解放します。 3. ErrInvalidMessageException を CUP.NET に通知します。	相手システムを見直してください。
	受信メッセージ格納領域の超過	1. 障害発生時の情報をトレースファイルに出力します。 2. コネクションを解放します。 3. ErrBufferOverflowException を CUP.NET に通知します。	ReceiveAssembledMessage メソッドに指定した受信メッセージ格納領域のサイズ，または相手システムを見直してください。

障害の種類	障害の内容	Client .NET の処理	ユーザが取る処置
メッセージ受信障害	受信メッセージ格納領域の超過	なお，受信メッセージ格納領域にメッセージは格納されません。	ReceiveAssembledMessage メソッドに指定した受信メッセージ格納領域のサイズ，または相手システムを見直してください。
	最大待ち時間の超過	1. コネクションを解放します。 2. ErrTimedOutException を CUP.NET に通知します。 なお，受信メッセージ格納領域には，最大待ち時間を超過するまでに受信したメッセージが格納されます。	ReceiveAssembledMessage メソッドに指定した最大待ち時間，または相手システムを見直してください。

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

トレースファイルへの出力で障害が発生した場合は，トレース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，トレースは出力されません。

7.2　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。

7.2.1　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。トレースもあわせて取得してください。

7.2.2　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。

7.2.3　トレースを取得している場合

取得しているすべてのトレースファイルを保存してください。デバッグトレースが出力されている場合は保存してください。

付録

付録 A　DCCM3 との接続

Client .NET では，RPC や TCP/IP 通信機能を使用して，VOS3 や VOS1 上の DCCM3 と接続できます。

付録 A.1　DCCM3 との RPC

Client .NET では，OpenTP1 のサーバだけでなく，DCCM3 のサーバとも RPC を使用した通信ができます。相手サーバが DCCM3 の場合の特徴は次のとおりです。

- 使用できる RPC の形態は，同期応答型 RPC と非応答型 RPC です。連鎖 RPC は使用できません。
- トランザクション制御機能は使用できません。
- 常設コネクションを使用して DCCM3 論理端末へ RPC を行う場合，負荷分散機能を使用できます。詳細については，「付録 A.1(4)　DCCM3 論理端末に対して RPC を行う場合の負荷分散」を参照してください。
- DCCM3 に対して RPC を行う場合，サービス名がトランザクション名称と評価されます。
- DCCM3 のサーバ側に，OpenTP1 の RPC 要求を解釈する機能が組み込まれていることが前提になります。この機能は，DCCM3/SERVER/TP1，および DCCM3/Internet が提供します。

(1)　相手サーバの指定方法

DCCM3 のサーバと RPC を使用した通信をする場合，相手サーバはサービスグループ名とサービス名で指定します。この指定方法は OpenTP1 のサーバに対する RPC と同じです。

- サービスグループ名

  サービスグループ名として不正でないダミーの文字列を指定します。

  1～31 文字の識別子を指定してください。
- サービス名

  DCCM3 側のトランザクション名称を指定します。

  使用できる文字は，アルファベット（A～Z，a～z）と数字（0～9）です。文字数の合計が 1～8 文字になるように指定してください。

(2)　相手サーバのアドレス定義

DCCM3 のサーバと RPC を使用した通信をする場合，OpenTP1 のネームサービスの管理外にあるサーバを呼び出すため，Client .NET 側でサービス名ごとに定義を分けて，サーバのアドレスを定義する必要があります。

サーバのアドレスの定義方法を次に示します。

1. Client .NET 構成定義の<tp1Server>要素の host 属性および port 属性に，それぞれ RPC 受け付け窓口のホスト名およびポート番号を指定します。

(3) RPC の実行手順

相手サーバのアドレスを定義したあと RPC を実行します。Client .NET 構成定義の指定内容によって RPC の実行手順は異なります。

(a) <rapService>要素の autoConnect 属性に true を指定した場合

1. OpenRpc メソッドを実行して，定義を読み込みます。
2. Call メソッドを実行します。
   Client .NET 構成定義の<tp1Server>要素で指定した RPC 受け付け窓口に，自動でコネクションを確立します。コネクションの確立後，RPC を行います。

(b) <rapService>要素の autoConnect 属性に false を指定，または指定を省略した場合

1. OpenRpc メソッドを実行して，定義を読み込みます。
2. 引数なしの OpenConnection メソッドを実行します。
   Client .NET 構成定義の<tp1Server>要素で指定した RPC 受け付け窓口に，コネクションを確立します。
3. コネクションの確立後，Call メソッドを実行して RPC を行います。

(c) <tp1Server>要素で指定したサーバ以外と通信する場合

1. OpenRpc メソッドを実行して，定義を読み込みます。
   この手順は省略できます。省略した場合でも，2.および 3.は実行できます。
2. OpenConnection メソッドの引数に RPC 受け付け窓口（ホスト名，ポート番号）を指定して，OpenConnection メソッド実行します。
3. コネクションの確立後，Call メソッドを実行して RPC を行います。

(4) DCCM3 論理端末に対して RPC を行う場合の負荷分散

Client .NET と DCCM3 論理端末が常設コネクションを使用して RPC を行う場合，コネクション確立時に接続先を複数の DCCM3 に振り分けて負荷を分散できます。Client .NET は，Client .NET 構成定義の<tp1Server>要素に指定された複数の DCCM3 論理端末のホスト名およびポート番号の中から，Client .NET 構成定義の<rapService>要素の randomSelect 属性に Y を指定することで，接続先をランダムに選択し，接続を試みます。ある DCCM3 論理端末との接続に失敗すると，それ以外の DCCM3 論理端末を<tp1Server>要素に指定された順で選択し，接続を試みます。これを繰り返し，Client .NET 構成定義の<tp1Server>要素に指定されたすべての DCCM3 論理端末との接続にすべて失敗したときに，初めてエラーを検知します。

Client .NET と DCCM3 論理端末との通信方法を次に示します。

1. Client .NET 構成定義の<tp1Server>要素に DCCM3 論理端末のホスト名およびポート番号を指定し，引数なしの OpenConnection メソッドを実行します。

   この場合，常設コネクションを使用します。

2. Client .NET 構成定義の<tp1Server>要素に DCCM3 論理端末のホスト名およびポート番号，<rapService>要素の randomSelect 属性に true を指定します。

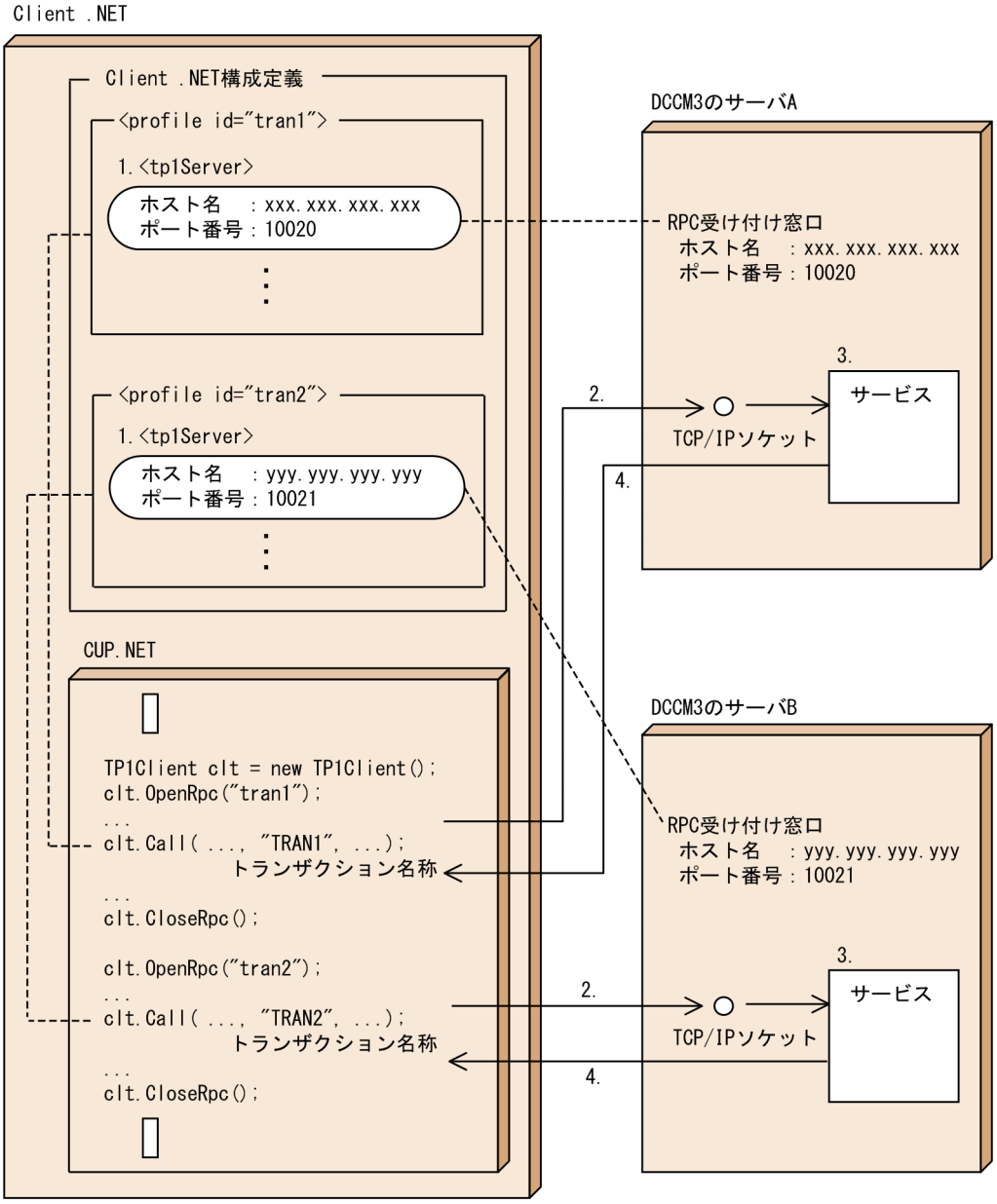
3. Client .NET 構成定義の<rapService>要素の autoConnect 属性に true を指定した場合，Call メソッドを実行します。Call メソッドを実行すると，コネクション未確立時には，自動でコネクションを確立します。

   Client .NET 構成定義の<rapService>要素の autoConnect 属性に false を指定，または指定を省略した場合，引数なしの OpenConnection メソッドを実行します。引数なしの OpenConnection メソッドを実行すると，コネクションを確立します。

(5) Client .NET 構成定義の定義例

次の図で動くような，DCCM3 接続時の Client .NET 構成定義の定義例を示します。

図 A-1　DCCM3 のサーバとの RPC



1. Client .NET 構成定義で，各トランザクションを処理するサーバのアドレス（RPC 受け付け窓口）を，プロファイル ID を分けて<tpiServer>要素に定義します。
2. RPC を行うとき，プロファイル ID から RPC 受け付け窓口のアドレスを求め，RPC のメッセージを送信します。
3. RPC のメッセージを解釈し，要求されたサービスを実行します。
4. 同期応答型 RPC の場合，サーバからの応答メッセージを受信します。

定義例

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section
```

```
      name="hitachi.opentp1.client"
      type="Hitachi.OpenTP1.Common.Util.ProfileSectionHandler,
            Hitachi.OpenTP1.Client,Version=7.0.0.0,
            Culture=neutral,PublicKeyToken=2440cf5f0d80c91c,Custom=null" />
  </configSections>

  <hitachi.opentp1.client>
    <common>
      <rpc use="rap" watchTime="180" />
      <rapService autoConnect="true" randomSelect="true" />
    </common>
    <profile id="tran1">
      <!--"TRAN1"のトランザクションを処理できるサーバのアドレスを定義-->
      <tp1Server host="xxx.xxx.xxx.xxx" port="10020" />
      <tp1Server host="zzz.zzz.zzz.zzz" port="10022" />
    </profile>
    <profile id="tran2">
      <!--"TRAN2"のトランザクションを処理できるサーバのアドレスを定義-->
      <tp1Server host="yyy.yyy.yyy.yyy" port="10021" />
      <tp1Server host="zzz.zzz.zzz.zzz" port="10022" />
    </profile>
  </hitachi.opentp1.client>
</configuration>
```

上記の Client .NET 構成定義を使用したプログラム例を次に示します。

プログラム例

```
using Hitachi.OpenTP1;
using Hitachi.OpenTP1.Client;

public class DCCM3Caller
{
  ...
  public void Function1(){
    TP1Client clt = new TP1Client();

    // TRAN1のトランザクションを呼び出すRPC
    clt.OpenRpc("tran1");
    ...
    // 同期応答型RPC
    clt.Call("dummysvg", "TRAN1", ..., TP1ClientFlags.DCNOFLAGS);
    ...
    clt.CloseRpc();

    // TRAN2のトランザクションを呼び出すRPC
    // 要求先のサーバアドレス取得のために定義を読み直す
    clt.OpenRpc("tran2");
    ...
    // 非応答型RPC
    clt.Call("dummysvg", "TRAN2", ..., TP1ClientFlags.DCRPC_NOREPLY);
    ...
    clt.CloseRpc();
  }
}
```

(6) DCCM3 論理端末に対して RPC を行う場合の注意事項

- リモート API 機能を使用した RPC だけ使用できます。
- 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。
- データ圧縮機能を使用した 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。
- Client .NET 構成定義で，<rapService>要素の inquireTime 属性を指定しても無効になります。CUP.NET からサーバに対する問い合わせの間隔最大時間は，DCCM3 の「端末放置監視時間」で指定してください。
- 常設コネクションかつオートコネクトモードを使用したシステムで，次のタイミングが重なった場合，RPC の実行に失敗して例外を返すことがあります。
  - rap サーバが常設コネクションを解放するタイミング
  - CUP.NET 実行プロセスが RPC を実行するタイミング

  この現象を回避するには，次の値を指定してください。
  - <rapService>要素の inquireTimeCheck 属性※：true
  - <rapService>要素の inquireTime 属性：DCCM3 の「端末放置監視時間」と同じ値

  inquireTimeCheck 属性，および inquireTime 属性の説明については「3. 構成定義」の<rapService>要素を参照してください。

  注※

  TP1/Client for .NET Framework のバージョン 07-03-07 以降で指定できます。
- DCCM3 側の注意事項については，マニュアル「VOS3 データマネジメントシステム XDM E2 系 解説」および「VOS1 データコミュニケーションマネジメントシステム DCCM3 解説」を参照してください。

付録 A.2　DCCM3 との TCP/IP 通信

DCCM3 論理端末と TCP/IP 通信を行う場合の注意事項は次のとおりです。

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。

付録 B　Client .NET で利用できるクラスのフィールド

Client .NET で利用できる各クラスのフィールドを次の表に示します。

表 B-1　Client .NET で利用できるクラスのフィールド

クラス名	フィールド名	値
Hitachi.OpenTP1.TP1Error	DCADMER_COMM	-1851
	DCADMER_PARAM	-1852
	DCADMER_STS_IO	-1853
	DCADMER_PROTO	-1854
	DCADMER_STATNOTZERO	-1855
	DCADMER_MEMORY_OUT	-1856
	DCADMER_MEMORY_ERR	-1857
	DCADMER_MEMORY_OUTERR	-1858
	DCADMER_SYSTEMCALL	-1859
	DCADMER_SUBAREA_NOT_EXIST	-1860
	DCADMER_MEMORY	-1861
	DCADMER_DEF	-1862
	DCADMER_MULTI_DEF	-1864
	DCADMER_NO_MORE_ENTRY	-1865
	DCADMER_REMOTE	-1866
	DCADMER_NODE_NOT_EXIST	-1867
	DCADMER_SWAP	-1868
	DCJNLER_PARAM	-1101
	DCJNLER_SHORT	-1102
	DCJNLER_LONG	-1103
	DCJNLER_MODE	-1104
	DCJNLER_PROTO	-1105
	DCLCKER_PARAM	-401
	DCLCKER_WAIT	-450
	DCLCKER_DLOCK	-452
	DCLCKER_TIMOUT	-453
	DCLCKER_MEMORY	-454

クラス名	フィールド名	値
Hitachi.OpenTP1.TP1Error	DCLCKER_OUTOFTRN	-455
	DCLCKER_NOTHING	-456
	DCLCKER_VERSION	-457
	DCLOGER_PARAM_ARGS	-1900
	DCLOGER_COMM	-1901
	DCLOGER_MEMORY	-1902
	DCLOGER_DEFFILE	-1904
	DCLOGER_NOT_UP	-1905
	DCLOGER_HEADER	-1906
	DCLOGER_TIMEOUT	-1907
	DCLOGER_PROTO	-1999
	DCMCFER_INVALID_ARGS	-11900
	DCMCFER_PROTO	-11901
	DCMCFRTN_71002	-12002
	DCMCFRTN_71003	-12003
	DCMCFRTN_71004	-12004
	DCMCFRTN_71108	-12108
	DCMCFRTN_72000	-13000
	DCMCFRTN_72001	-13001
	DCMCFRTN_72012	-13012
	DCMCFRTN_72013	-13013
	DCMCFRTN_72016	-13016
	DCMCFRTN_72017	-13017
	DCMCFRTN_72026	-13026
	DCMCFRTN_72036	-13036
	DCMCFRTN_72041	-13041
	DCMCFRTN_72073	-13073
	DCMCFRTN_73001	-14001
	DCMCFRTN_73002	-14002
	DCMCFRTN_73003	-14003
	DCMCFRTN_73005	-14005

クラス名	フィールド名	値
Hitachi.OpenTP1.TP1Error	DCMCFRTN_73010	-14010
	DCMCFRTN_73015	-14015
	DCMCFRTN_73018	-14018
	DCMCFRTN_73019	-14019
	DCMCFRTN_73020	-14020
	DCNJSER_XALIB_LOAD	-7519
	DCNJSER_XALIB_INVALID	-7520
	DCNJSER_SYSTEM	-7549
	DCNJS2ER_INVALID_DATA	-7551
	DCNJS2ER_INTERNAL	-7552
	DCNJS2ER_INVALID_ARGS	-7553
	DCNJS2ER_XML_ANALYSIS_ERR	-7554
	DCPRFER_PARAM	-4601
	DCRAPER_PARAM	-5501
	DCRAPER_PROTO	-5502
	DCRAPER_NOMEMORY	-5503
	DCRAPER_NETDOWN	-5505
	DCRAPER_TIMEDOUT	-5506
	DCRAPER_NOSOCKET	-5507
	DCRAPER_NOHOSTNAME	-5508
	DCRAPER_MAX_CONNECTION	-5517
	DCRAPER_NOMEMORY_SV	-5520
	DCRAPER_SHUTDOWN	-5521
	DCRAPER_NOCONTINUE	-5522
	DCRAPER_SYSCALL	-5523
	DCRAPER_NOSERVICE	-5528
	DCRAPER_ALREADY_CONNECT	-5529
	DCRAPER_UNKNOWN_NODE	-5531
	DCRAPER_TIMEOUT_SV	-5532
	DCRAPER_PANIC_SV	-5533
	DCRAPER_MAX_CONNECTION_SV	-5534

クラス名	フィールド名	値
Hitachi.OpenTP1.TP1Error	DCRPCER_INVALID_ARGS	-301
	DCRPCER_PROTO	-302
	DCRPCER_FATAL	-303
	DCRPCER_NO_BUFS	-304
	DCRPCER_NET_DOWN	-306
	DCRPCER_TIMED_OUT	-307
	DCRPCER_MESSAGE_TOO_BIG	-308
	DCRPCER_REPLY_TOO_BIG	-309
	DCRPCER_NO_SUCH_SERVICE_GROUP	-310
	DCRPCER_NO_SUCH_SERVICE	-311
	DCRPCER_SERVICE_CLOSED	-312
	DCRPCER_SERVICE_TERMINATING	-313
	DCRPCER_SERVICE_NOT_UP	-314
	DCRPCER_OLTF_NOT_UP	-315
	DCRPCER_SYSERR_AT_SERVER	-316
	DCRPCER_NO_BUFS_AT_SERVER	-317
	DCRPCER_SYSERR	-318
	DCRPCER_INVALID_REPLY	-319
	DCRPCER_OLTF_INITIALIZING	-320
	DCRPCER_ALL_RECEIVED	-321
	DCRPCER_INVALID_DES	-322
	DCRPCER_NO_BUFS_RB	-323
	DCRPCER_SYSERR_RB	-324
	DCRPCER_SYSERR_AT_SERVER_RB	-325
	DCRPCER_REPLY_TOO_BIG_RB	-326
	DCRPCER_TRNCHK	-327
	DCRPCER_NO_SUCH_DOMAIN	-328
	DCRPCER_NO_PORT	-329
	DCRPCER_SERVER_BUSY	-356
	DCRPCER_TESTMODE	-366
	DCRPCER_NOT_TRN_EXTEND	-367

クラス名	フィールド名	値
Hitachi.OpenTP1.TP1Error	DCRPCER_STANDBY_END	-369
	DCRPCER_SECCHK	-370
	DCRPCER_SEC_INIT	-371
	DCRPCER_TRNCHK_EXTEND	-372
	DCRPCER_RETRY_COUNT_OVER	-377
	DCRPCER_SERVICE_TERMINATED	-378
	DCTAMER_PARAM_TID	-1700
	DCTAMER_PARAM_TBL	-1701
	DCTAMER_PARAM_KEY	-1702
	DCTAMER_PARAM_KNO	-1703
	DCTAMER_PARAM_BFA	-1704
	DCTAMER_PARAM_BFS	-1705
	DCTAMER_PARAM_DTA	-1706
	DCTAMER_PARAM_DTS	-1707
	DCTAMER_PARAM_FLG	-1708
	DCTAMER_NOTTAM	-1709
	DCTAMER_UNDEF	-1710
	DCTAMER_TAMEND	-1720
	DCTAMER_PROTO	-1721
	DCTAMER_TRNOPN	-1722
	DCTAMER_RMTBL	-1723
	DCTAMER_NOLOAD	-1724
	DCTAMER_OPENED	-1725
	DCTAMER_NOOPEN	-1726
	DCTAMER_LOGHLD	-1727
	DCTAMER_OBSHLD	-1728
	DCTAMER_IDXTYP	-1729
	DCTAMER_ACSATL	-1730
	DCTAMER_NOREC	-1731
	DCTAMER_SEQENCE	-1732
	DCTAMER_EXWRITE	-1733

クラス名	フィールド名	値
Hitachi.OpenTP1.TP1Error	DCTAMER_EXREWRT	-1734
	DCTAMER_EXKEY	-1735
	DCTAMER_LOCK	-1736
	DCTAMER_DLOCK	-1737
	DCTAMER_TBLVR	-1760
	DCTAMER_FLSVR	-1761
	DCTAMER_TAMVR	-1762
	DCTAMER_NOAREA	-1763
	DCTAMER_RECOBS	-1764
	DCTAMER_TRNNUM	-1765
	DCTAMER_OPENNUM	-1766
	DCTAMER_ACCESSS	-1767
	DCTAMER_ACCESSF	-1768
	DCTAMER_MEMORY	-1769
	DCTAMER_IO	-1770
	DCTAMER_TMERR	-1771
	DCTAMER_NO_ACL	-1772
	DCTAMER_ACCESS	-1773
	DCTRNER_ROLLBACK	-902
	DCTRNER_HEURISTIC	-903
	DCTRNER_HAZARD	-904
	DCTRNER_PROTO	-905
	DCTRNER_RM	-906
	DCTRNER_TM	-907
	DCTRNER_NO_BEGIN	-924
	DCTRNER_ROLLBACK_NO_BEGIN	-925
	DCTRNER_HEURISTIC_NO_BEGIN	-926
	DCTRNER_HAZARD_NO_BEGIN	-927
Hitachi.OpenTP1.Client.TP1Client	DCRPC_MAX_MESSAGE_SIZE	1048576
Hitachi.OpenTP1.Client.TP1ClientFlags	DCNOFLAGS	0
	DCRPC_NOREPLY	1

クラス名	フィールド名	値
Hitachi.OpenTP1.Client.TP1ClientFlags	DCRPC_CHAINED	4
	DCRPC_TPNOTRAN	32
	DCRPC_SCD_LOAD_PRIORITY	8
	DCRPC_WATCHTIMINHERIT	16
	DCRPC_RAP_AUTOCONNECT	32
	DCRPC_WATCHTIMRPCINHERIT	64
	DCCLT_SND_CLOSE	1
	DCCLT_RCV_CLOSE	2
Hitachi.OpenTP1.Client.TP1ClientError	DCCLNER_INVALID_ARGS	-7601
	DCCLNER_INVALID_HOST	-7602
	DCCLNER_INVALID_PORT	-7603
	DCCLNER_IO	-7604
	DCCLNER_TIMEOUT	-7605
	DCCLNER_REPLY_TOO_BIG	-7606
	DCCLNER_CONNFREE	-7608
	DCCLNER_PROTO	-7609
	DCCLNER_EOF	-7612
	DCCLNER_FATAL	-7613
	DCCLNER_NOBUFS	-7614
	DCCLNER_NOT_UP	-7616
	DCCLNER_PARAM	-7617
	DCCLNER_SYS	-7618
	DCCLNER_NETDOWN	-7620
	DCCLNTRNER_RM	-7622
	DCCLNTRNER_TM	-7623
	DCCLNTRNER_ROLLBACK	-7624
	DCCLNTRNER_HEURISTIC	-7625
	DCCLNTRNER_HAZARD	-7626
	DCCLNTRNER_NO_BEGIN	-7627
	DCCLNTRNER_ROLLBACK_NO_BEGIN	-7628
	DCCLNTRNER_HEURISTIC_NO_BEGIN	-7629

クラス名	フィールド名	値
Hitachi.OpenTP1.Client.TP1ClientError	DCCLNTRNER_HAZARD_NO_BEGIN	-7630
	DCCLNER_TIMEOUT_S	-7631
	DCCLNER_TIMEOUT_C	-7632
	DCCLNER_NETDOWN_S	-7633
	DCCLNER_NETDOWN_C	-7634
	DCCLNER_CONNREFUSED	-7635
	DCCLNER_MESSAGE_TOO_BIG	-7637
	DCCLNER_INVALID_MSG	-7638
	DCCLNER_BUFFER_OVERFLOW	-7639
	DCCLNER_COLLISION_MSG	-7640
	DCCLNER_ACCEPT_CANCELED	-7642
	DCCLNER_VERSION	-7643
	DCCLNER_MARSHAL	-7650
	DCCLNER_UNEXPECT	-7699

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　07-51 での変更点

Client .NET 07-51 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　Client .NET 07-51 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	Client .NET 構成定義 • <socket>要素の sendBufferSize 属性 • <socket>要素の receiveBufferSize 属性
	コマンド	なし
削除	なし	

Client .NET 07-51 での動作の変更はありません。

付録 C.2　07-50 での変更点

Client .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–2　Client .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	Client .NET 構成定義 • <nameService>要素の usePriority 属性 • <socket>要素の backlogCount 属性 • <rpc>要素の extendOpt 属性 • <nameService>要素の hostChangeMode 属性 • <nameService>要素の sysWatchTime 属性 • <rapService>要素の inquireTimeCheck 属性

種別	分類	内容
追加	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

Client .NET 07-50 での動作の変更点を次の表に示します。

表 C–3　Client .NET 07-50 での動作の変更点

分類	内容
API	なし
定義	Client .NET 構成定義 <rapService>要素の inquireTime 属性に，0 以外を指定し，オートコネクトモードで Call メソッドを使用して RPC を実行した場合，CUP 実行プロセスでも問い合わせ間隔最大時間を監視するように変更
コマンド	if2cstub コマンド，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	UAP トレースに TP1/Client.NET のインスタンスごとに一意である識別情報を出力するように変更
	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
	適用 OS を Windows 7，および Windows Server 2008 R2 以降に変更
	Client .NET を使用する場合の前提ソフトウェアを.NET Framework 3.5 SP1 に変更

Client .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 C.3　07-03 での変更点

Client .NET 07-03 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–4　Client .NET 07-03 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	TP1Client クラス • AcceptNotification メソッド

種別	分類	内容
追加	API	• AcceptNotificationChained メソッド • CancelNotification メソッド • CloseNotification メソッド • OpenNotification メソッド
		ErrAcceptCanceledException クラス
		ErrVersionException クラス
	定義	Client .NET 構成定義 • <rpc>要素の cupRecvPort 属性 • <rpc>要素の useMultiScheduler 属性 • <scheduleService>要素の multiSchedulerCount 属性 • <socket>要素の cupSendHost 属性 • <transaction>要素の completionLimitTime 属性
	コマンド	なし
削除	なし	

Client .NET 07-03 での動作の変更点を次の表に示します。

表 C-5　Client .NET 07-03 での動作の変更点

分類	内容
API	SetUapTraceMode メソッド，SetErrorTraceMode メソッド，SetMethodTraceMode メソッド，SetDataTraceMode メソッドで，UAP トレースを取得するように変更
定義	なし
コマンド	なし
メッセージ	なし
その他	リモート API 機能を使用した RPC で，TP1/Server の性能検証用トレースに Client .NET のインスタンスごとに一意である識別情報を出力するように変更
	開発言語として COBOL 言語に対応
	適用 OS に Windows Server 2008 を追加

Client .NET 07-03 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 C.4　07-02 での変更点

Client .NET 07-02 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–6　Client .NET 07-02 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	Client .NET 構成定義 • <xarTransaction>要素
	コマンド	なし
削除	なし	

Client .NET 07-02 での API，定義およびコマンドの動作，ならびにデフォルト値の変更はありません。

付録 C.5　07-01 での変更点

Client .NET 07-01 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–7　Client .NET 07-01 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	TP1Client クラス • SetConnectInformation メソッド
	定義	Client .NET 構成定義 • <rapService>要素の connectInformation 属性 • <socket>要素
	コマンド	なし
削除	なし	

Client .NET 07-01 での API，定義およびコマンドの動作の変更点を次の表に示します。

表 C–8　Client .NET 07-01 での API，定義およびコマンドの動作の変更点

分類	内容
API	なし
定義	なし
コマンド	なし
メッセージ	なし
その他	Client .NET から TP1/Server への接続時，自システムの自動割当てポートの不足によるエラー（System.Net.Sockets.SocketException 例外）が発生した場合にリトライ処理を行うように変更

分類	内容
その他	SPP から応答メッセージ受信処理中に一時クローズ要求メッセージを受信した場合，その一時クローズ要求メッセージを破棄して本来の応答メッセージを受信し，かつ受信待ちを不当に行い，ErrClientTimedOutException 例外が発生する問題を修正
	適用 OS に Windows Vista を追加

Client .NET 07-01 での API，定義およびコマンドの動作，ならびにデフォルト値の変更はありません。

付録 C.6　07-00 での変更点

Client .NET 07-00 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–9　Client .NET 07-00 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	Client .NET 構成定義 • <nameService>要素の multiHomedHost 属性
	コマンド	なし
削除	なし	

Client .NET 07-00 での API，定義およびコマンドの動作の変更点を次の表に示します。

表 C–10　Client .NET 07-00 での API，定義およびコマンドの動作の変更点

分類	内容
API	なし
定義	なし
コマンド	なし
メッセージ	なし
その他	.NET Framework 2.0 をサポート
	RPC 送受信メッセージの最大長拡張機能を追加

Client .NET 07-00 での API，定義およびコマンドの動作，ならびにデフォルト値の変更はありません。

付録 D　用語解説

Client .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）

Client .NET 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。

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。

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。

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。

（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

こ

さ

し

す

ろ