



ToolTalk ユーザーズガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 816-3975-11
2002 年 12 月

Copyright 2002 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

本製品およびそれに関連する文書は著作権法により保護されており、その使用、複製、頒布および逆コンパイルを制限するライセンスのもとにおいて頒布されます。サン・マイクロシステムズ株式会社の書面による事前の許可なく、本製品および関連する文書のいかなる部分も、いかなる方法によっても複製することが禁じられます。

本製品の一部は、カリフォルニア大学からライセンスされている Berkeley BSD システムに基づいていることがあります。UNIX は、X/Open Company, Ltd. が独占的にライセンスしている米国ならびに他の国における登録商標です。フォント技術を含む第三者のソフトウェアは、著作権により保護されており、提供者からライセンスを受けているものです。

Federal Acquisitions: Commercial Software-Government Users Subject to Standard License Terms and Conditions.

本製品に含まれる HG 明朝 L、HG-MincyoL-Sun、HG ゴシック B、および HG-GothicB-Sun は、株式会社リコーがリコービマジクス株式会社からライセンス供与されたタイプフェイスマスタをもとに作成されたものです。HG 平成明朝体 W3@X12 は、株式会社リコーが財団法人日本規格協会からライセンス供与されたタイプフェイスマスタをもとに作成されたものです。フォントとして無断複製することは禁止されています。

Sun、Sun Microsystems、docs.sun.com、AnswerBook、AnswerBook2、ToolTalk は、米国およびその他の国における米国 Sun Microsystems, Inc. (以下、米国 Sun Microsystems 社とします) の商標もしくは登録商標です。

サンロゴマークおよび Solaris は、米国 Sun Microsystems 社の登録商標です。

すべての SPARC 商標は、米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標が付いた製品は、米国 Sun Microsystems 社が開発したアーキテクチャに基づくものです。

OPENLOOK、OpenBoot、JLE は、サン・マイクロシステムズ株式会社の登録商標です。

Wnn は、京都大学、株式会社アステック、オムロン株式会社で共同開発されたソフトウェアです。

Wnn6 は、オムロン株式会社、オムロンソフトウェア株式会社で共同開発されたソフトウェアです。© Copyright OMRON Co., Ltd. 1995-2000. All Rights Reserved. © Copyright OMRON SOFTWARE Co., Ltd. 1995-2002 All Rights Reserved.

「ATOK」は、株式会社ジャストシステムの登録商標です。

「ATOK Server/ATOK12」は、株式会社ジャストシステムの著作物であり、「ATOK Server/ATOK12」にかかる著作権その他の権利は、株式会社ジャストシステムおよび各権利者に帰属します。

本製品に含まれる郵便番号辞書 (7 桁/5 桁) は郵政事業庁が公開したデータを元に制作された物です (一部データの加工を行なっています)。

本製品に含まれるフェイスマーク辞書は、株式会社ビレッジセンターの許諾のもと、同社が発行する『インターネット・パソコン通信フェイスマークガイド '98』に添付のものを使用しています。© 1997 ビレッジセンター

Unicode は、Unicode, Inc. の商標です。

本書で参照されている製品やサービスに関しては、該当する会社または組織に直接お問い合わせください。

OPEN LOOK および Sun Graphical User Interface は、米国 Sun Microsystems 社が自社のユーザおよびライセンス実施権者向けに開発しました。米国 Sun Microsystems 社は、コンピュータ産業用のビジュアルまたはグラフィカル・ユーザインタフェースの概念の研究開発における米国 Xerox 社の先駆者としての成果を認めるものです。米国 Sun Microsystems 社は米国 Xerox 社から Xerox Graphical User Interface の非独占的ライセンスを取得しており、このライセンスは米国 Sun Microsystems 社のライセンス実施権者にも適用されます。

DtComboBox ウィジェットと DtSpinBox ウィジェットのプログラムおよびドキュメントは、Interleaf, Inc. から提供されたものです。(© 1993 Interleaf, Inc.)

本書は、「現状のまま」をベースとして提供され、商品性、特定目的への適合性または第三者の権利の非侵害の黙示の保証を含みそれに限定されない、明示的であるか黙示的であるかを問わない、なんらの保証も行われぬものとします。

本製品が、外国為替および外国貿易管理法 (外為法) に定められる戦略物資等 (貨物または役務) に該当する場合、本製品を輸出または日本国外へ持ち出す際には、サン・マイクロシステムズ株式会社の事前の書面による承諾を得ることのほか、外為法および関連法規に基づく輸出手続き、また場合によっては、米国商務省または米国所轄官庁の許可を得ることが必要です。

原典: *ToolTalk User's Guide*

Part No: 816-1324-10

Revision A



020927@4660



目次

はじめに	13
1 ToolTalk サービスの紹介	19
概要	19
ToolTalk のシナリオ	20
ToolTalk デスクトップサービスメッセージセットの使用法	20
ToolTalk 文書メディア交換メッセージセットの使用法	22
CASE 連携メッセージセットの使用	23
ToolTalk ファイル名マッピング機能の使い方	25
マルチスレッド環境での ToolTalk の使い方	25
アプリケーションの ToolTalk メッセージの使用法	25
ToolTalk メッセージの送信	25
メッセージパターン	26
ToolTalk メッセージの受信	26
ToolTalk メッセージの配布	27
プロセス指向メッセージ	27
オブジェクト指向メッセージ	27
メッセージ配信の判別	27
ToolTalk サービスを使用するためのアプリケーションの変更	29
2 ToolTalk サービスの概要	31
ToolTalk アーキテクチャ	31
ToolTalk セッションの開始	32
バックグラウンドおよびバッチセッション	34
X Window System	35

	ttsession の検出	35
	ToolTalk ファイルとデータベースの管理	35
	デモンストレーションプログラム	35
3	メッセージパターン	37
	メッセージパターンの属性	37
	配信範囲属性	40
	セッションだけの配信範囲指定	40
	ファイルだけの配信範囲指定	41
	セッション内のファイルへの配信範囲指定	41
	ファイル、セッション、または両方への配信範囲指定	43
	配信範囲指定したパターンへのファイルの追加	43
	コンテキスト属性	44
	パターン引数の属性	44
	処置属性	45
4	ToolTalk プロセスの設定と管理	47
	ToolTalk サービスファイルの場所	47
	バージョン	49
	必要条件	49
	環境変数	49
	ToolTalk 環境変数	49
	その他の環境変数	51
	リモートホスト上のプログラムの起動に必要な環境変数	51
	コンテキストスロットの使用による環境変数の作成	53
	ToolTalk データベースサーバーのインストール	53
	rpc.ttdserverd がシステムにインストールされていることの確認	54
	rpc.ttdbserverd がシステム上で動作していることを確認	54
	Solaris CD-ROM	55
	新しい ToolTalk データベースサーバーの実行	55
	ToolTalk データベースサーバーのリダイレクト	55
	ホストマシンのリダイレクト	56
	ファイルシステムのパーティションのリダイレクト	56
5	アプリケーション情報の管理	59
	アプリケーション型のインストール	59
	ToolTalk 型情報の検査	61

ToolTalk 型情報の削除	61
ToolTalk サービスの更新	62
▼ ttsession プロセスに SIGUSR2 を送信するには	62
プロセス型エラー	62
ttsnop の使用によるメッセージおよびパターンのデバッグ	63
メッセージの構成と送信	64
パターンの構成と登録	66
メッセージ構成要素の表示	68
作成済みメッセージの送信	69
メッセージの受信	69
メッセージ受信の停止	70
6 ToolTalk メッセージで参照するファイルとオブジェクトの保守	71
ToolTalk の拡張シェルコマンド	71
ToolTalk データベースの保守と更新	72
データベースの表示、検査、および修復	73
7 ToolTalk セッションの結合	75
ToolTalk API ヘッダーファイルの組み込み	75
ToolTalk サービスへの登録	76
初期セッションへの登録	76
指定セッションへの登録	77
複数のセッションへの登録	78
メッセージ受信の設定	78
同一プロセス内でのメッセージの送受信	79
ネットワーク環境でのメッセージの送受信	80
ToolTalk サービスからの登録解除	80
マルチスレッド環境における ToolTalk の使用	81
初期化	81
ToolTalk procid とセッション	81
ToolTalk 記憶領域	82
共通の問題	82
8 メッセージの送信	83
ToolTalk サービスによるメッセージのルーティング方法	83
通知の送信	83
要求の送信	84

オフアの送信	85
送信したメッセージの状態の変化	85
メッセージの属性	85
アドレス属性	86
配信範囲属性	86
構造化データのシリアル化	89
ToolTalk のメッセージ配信アルゴリズム	90
プロセス指向メッセージの配信	90
オブジェクト指向メッセージの配信	92
otype のアドレス指定	94
ToolTalk メッセージを送信するためのアプリケーションの変更	95
メッセージの作成	95
メッセージコールバックの追加	101
メッセージの送信	104
例	104

9 動的メッセージパターン	107
動的メッセージの定義	107
メッセージパターンの作成	109
メッセージパターンコールバックの追加	109
メッセージパターンの登録	110
メッセージパターンの削除と登録解除	110
現在のセッションまたはファイルによるメッセージパターンの更新	111
デフォルトセッションの結合	111
複数セッションの結合	112
処理の対象とするファイルの結合	113
10 静的メッセージパターン	115
静的メッセージの定義	115
プロセス型の定義	115
シグニチャ	116
ptype ファイルの作成	117
ツールの自動起動	120
オブジェクト型の定義	120
シグニチャ	121
otype ファイルの作成	122
型情報のインストール	124

	既存のプロセス型の確認	125
	プロセス型の宣言	125
	プロセス型の宣言解除	127
11	メッセージの受信	129
	メッセージの検索	129
	メッセージの容易な識別方法と処理方法	131
	応答の容易な認識方法と処理方法	131
	メッセージ状態の検査	131
	メッセージの検査	132
	コールバックルーチン	134
	ハンドラにアドレス指定されたメッセージのコールバック	135
	静的パターンへのコールバックの付加	135
	要求の処理	136
	要求への応答	136
	要求の拒否または異常終了	137
	オフアの監視	138
	メッセージの削除	139
12	オブジェクト	141
	オブジェクト指向メッセージ方式	141
	オブジェクトデータ	141
	オブジェクト仕様の作成	142
	otype の割り当て	143
	オブジェクト仕様プロパティの決定	143
	仕様プロパティの格納	144
	プロパティへの値の追加	144
	オブジェクト仕様の書き込み	144
	オブジェクト仕様の更新	144
	オブジェクト仕様の管理	145
	仕様情報の検査	145
	オブジェクト仕様の比較	146
	ファイル内の特定の仕様の照会	146
	オブジェクト仕様の移動	148
	オブジェクト仕様の削除	149
	オブジェクトおよびファイル情報の管理	149
	オブジェクトデータが入ったファイルの管理	149

	ToolTalk 情報が入ったファイルの管理	150
	オブジェクト指向メッセージの例	150
13	情報記憶領域の管理	153
	ToolTalk サービスに提供される情報	153
	ToolTalk サービスが提供する情報	153
	情報の記憶領域管理のための呼び出し	154
	情報のマーク付けと解放	154
	記憶領域の割り当てと解放	155
	特殊な実装方法: コールバックルーチンとフィルタルーチン	156
	コールバックルーチン	156
	フィルタルーチン	156
14	エラー処理	159
	ToolTalk エラー状態の検索	159
	ToolTalk エラー状態の確認	160
	戻り値の状態	160
	通常の戻り値を伴う関数	160
	通常 of 戻り値を伴わない関数	160
	返されたポインタの状態	161
	返された整数の状態	162
	接続の異常終了	163
	エラーの伝達	163
A	分類機構データベースから ToolTalk 型データベースへの移行	165
	ttce2xdr スクリプト	165
	ユーザーデータベースの変換	165
	システムデータベースの変換	166
	ネットワークデータベースの変換	167
B	ToolTalk サービスのデモンストレーション	169
	簡易化されたアプリケーション間通信	169
	連携機能の追加	170
	Xedit アプリケーションの変更	170
	Xfontsel アプリケーションの変更	171
	ツールコミュニケーション	172

アプリケーションへの ToolTalk コードの追加	173
Xedit ファイルへの ToolTalk コードの追加	173
Xfontsel ファイルへの ToolTalk コードの追加	180
C ToolTalk の標準メッセージセット	187
ToolTalk デスクトップサービスメッセージセット	187
開発目的	188
特長	188
ToolTalk 文書メディア交換メッセージセット	188
開発目的	189
特長	189
ToolTalk メッセージの一般的な定義と表記法	190
エラー	192
ToolTalk 開発の一般的な指針と表記法	193
要求相手を特定しない	193
ツールは必要なときだけ起動する	193
操作が完了したら応答する	194
できるだけ内部状態を持たないようにする	194
役割ごとにプロセス型を 1 つ宣言する	194
ToolTalk アプリケーションの開発	195
メッセージ方式に関する問い合わせ先	196
D Q & A	197
Q & A	199
ToolTalk サービスとは何ですか	199
ToolTalk サービスは、Common Object Request Broker Architecture (CORBA) の Sun 版ですか	199
ToolTalk サービスに入っているファイルの種類を教えてください	200
X ベースの ttsession が最初に起動されるのはいつですか	201
rpc.ttdbserverd はどこで起動されますか	201
ToolTalk 型データベースはどこにありますか	201
ToolTalk サービスを使用するには、X Window System が必要ですか	202
MIT X で ToolTalk サービスを使用できますか	202
X セッションのセッション ID はどこにありますか	203
tt_open はどのように ttsession に接続しますか	203
tt_open を呼び出した後、セッションが実際に始まるのはいつですか	204
別のセッションが接続されると最初のセッションは終了しますか	204

動作するマシンが異なるプロセス同士は、ToolTalk サービスをどのように使用すれば通信できますか 204

tt_default_session_set の目的は何ですか 206

1つのプロセスで2つ以上のセッションに接続するには、どうすればいいですか 206

あらかじめ決めたセッション ID で tsession を起動できますか 207

セッション ID には、どのような情報が入っていますか 207

プログラムが新しくセッションに参加したことを通知する標準的な方法はありませんか 207

メッセージの行き先を教えてください 207

メッセージの基本的なフローを教えてください 208

アプリケーションにメッセージが到着するとどうなりますか 209

メッセージを区別する方法を教えてください 210

プロセスは自分自身に要求を送信できますか 211

tt_message_callback_add で登録した関数に、自分自身のデータを渡せますか 212

任意のデータをメッセージで送信する方法を教えてください 212

ToolTalk サービスでファイルを転送できますか 212

ToolTalk サービスは、メモリー (バイト) の順序の問題をどのように処理しますか 213

メッセージは再使用できますか 213

メッセージを破棄するとどうなりますか 213

1つのメッセージを2つ以上のハンドラで処理できますか 213

1つの ptype のハンドラを2つ以上実行できますか 214

メッセージの処置の値とは何ですか 215

メッセージ状態要素とは何ですか 215

tt_free はいつ使用しますか 215

ptype とは何ですか 216

新しく作成した型が認識されない理由を教えてください 216

ptype のプロセスが既に存在する場合、ptype 情報は使用されますか 216

(インスタンスが既に動作中かどうかにかかわらず) インスタンスを常に起動するよう ptype の定義を変更できますか 217

tt_ptype_declare は何を行いますか 217

TT_TOKEN とは何ですか 217

パターンはいつ有効になりますか 217

応答を入手するにはパターンの登録が必要ですか 218

要求を監視するにはどうすればいいですか 218

静的パターンの属性値と照合させる方法を教えてください 218

TT_HANDLER に対してワイルドカードでパターンを指定できない理由を教えてください	218
ファイルを配信範囲とする任意のメッセージを監視するようなパターンを設定できますか	218
静的パターンのファイル配信範囲は file_in_session と同じですか	219
arg_add、barg_add、iarg_add の違いを教えてください	219
メッセージ引数の type や vtype とは何ですか	219
コンテキストの使用方法を教えてください	220
ttsession はどのように照合をチェックしますか	220
ToolTalk サービスには、配信範囲が何種類ありますか	221
TT_DB ディレクトリとは何ですか、また、型データベースと TT_DB ディレクトリは何が違いますか	222
tt_db データベースには何を入れればいいですか	222
rpc.ttdbserverd は何を実行しますか	222
ttsession と rpc.ttdbserverd は通信しますか	223
どのような帯域幅のメッセージをサポートできますか	223
メッセージサイズや引数の個数に制限はありますか	223
メッセージを送信する際、最も時間効率のよい方法は何ですか	223
ネットワークのオーバーヘッドの種類を教えてください	224
ToolTalk サービスは、要求を処理するのに負荷分散を使用しますか	224
ToolTalk アプリケーションに必要なリソースには何がありますか	224
ttsession が異常終了するとどうなりますか	224
rpc.ttdbserverd が異常終了するとどうなりますか	225
ホストやリンクが停止するとどうなりますか	225
tt_close は何をしますか	226
メッセージの配信は、ネットワーク上でも保証されますか	226
メッセージの配信には、時間的な順序がありますか	226
unix、xauth、des とは何ですか	226
アプリケーションが互いにメッセージを隠すことはできますか	227
横取りや模造に対して保護機構はありますか	227
待ち行列に入ったメッセージはどこに格納されていますか、また、その記憶領域はどのくらい安全ですか	227
ToolTalk サービスは C2 保証されていますか	227
メッセージの行方を追跡するには、どうすればいいですか	228
ToolTalk サービスを使用して、他のすべてのツールから自分のデバッグツールを隔離するには、どうすればいいですか	228
C++ で ToolTalk サービスを使用できますか	229
ファイル名を修飾する必要はありますか	229
ToolTalk オブジェクトとは何ですか	229

ToolTalk ニュースグループはありますか 230

用語集 231

索引 235

はじめに

このマニュアルは、ToolTalk™ サービスについての説明と、ToolTalk メッセージを送受信できるようにアプリケーションを変更する方法について記述しています。このマニュアルでは、次の内容について説明しています。

- ToolTalk サービスの概念
- ToolTalk サービスとは何か、どのように機能するのか
- ToolTalk サービスを設定および管理するための必要事項
- ToolTalk サービスにアプリケーションを統合するための必要事項
- プロセスまたは ToolTalk オブジェクトへメッセージを送信するようにアプリケーションを変更する方法
- アプリケーションが受け取りたいメッセージパターンに関する情報を登録する方法
- ToolTalk サービスからアプリケーションに配信されたメッセージを受信して処理する方法
- アプリケーションデータ内で ToolTalk オブジェクトを作成し管理する方法
- ToolTalk オブジェクトをシステム管理者レベルで管理する方法。ユーザーが、オブジェクトとオブジェクトが格納されているファイルを管理するための必要事項
- ToolTalk サービスを使用したアプリケーション間通信の実現方法

対象読者

このマニュアルは、ToolTalk サービスを使用して他のアプリケーションと連携するアプリケーションを作成し管理する開発者を対象としています。また、ワークステーションを設定するシステム管理者にも役立ちます。このマニュアルでは、読者が Solaris 操作環境のコマンド、システム管理者のコマンド、およびシステム用語についての知識があると想定しています。

内容の紹介

このマニュアルは、次のように構成されています。

第1章では、ToolTalk サービスの働き、アプリケーションが提供した情報を使って ToolTalk サービスがメッセージを配信する方法、およびアプリケーションの ToolTalk サービスの使用法について説明します。

第2章では、このリリースにおける新しい機能、変更された機能、およびアプリケーションと ToolTalk の構成要素について説明します。

第3章では、メッセージパターンの属性について説明します。

第4章では、ToolTalk ファイルの配置、ハードウェアとソフトウェアの必要事項、ToolTalk バージョン情報の検索方法、および ToolTalk データベースのインストール上の注意事項について説明します。

第5章では、アプリケーション情報の管理方法について説明します。

第6章では、ToolTalk メッセージにおけるファイル参照の保守方法、システム管理者とユーザーが ToolTalk オブジェクトを保守する方法、および ToolTalk データベースの保守方法について説明します。

第7章では、ToolTalk API ヘッダーファイルの配置、アプリケーションを初期化して ToolTalk サービスとのセッションを開始する方法、ファイルおよびセッションについての情報を ToolTalk サービスに対して指定する方法、記憶領域の管理方法とエラーの対処方法、およびプロセス終了時にメッセージパターンを登録解除して ToolTalk サービスとの通信を終了する方法について説明します。

第8章では、メッセージの配信、ToolTalkメッセージの属性とアルゴリズムについて説明します。また、メッセージの作成、メッセージ内容の記入、要求に対するコールバックの添付、メッセージの送信などそれぞれの方法についても説明します。

第9章では、動的メッセージパターンを作成する方法、ToolTalk サービスに動的メッセージパターンを登録する方法、および動的メッセージパターンにコールバックを追加する方法について説明します。

第10章では、インストール時にプロセスおよびオブジェクトタイプについての情報を指定する方法、ToolTalk サービスで静的メッセージパターンを使用できるようにする方法、ptype の宣言方法、Sun Vendor Type Registration プログラムで ptype を登録する方法について説明します。

第11章では、アプリケーションに配信されたメッセージを検索する方法、検査の済んだメッセージを処理する方法、応答を送信する方法、およびどのような場合にメッセージを破棄すべきかについて説明します。

第12章では、ユーザーのプロセスが作成および管理するオブジェクトに対する ToolTalk 仕様のオブジェクトを作成する方法について説明します。

第 13 章 では、オブジェクトを管理および削除する方法について説明します。

第 14 章 では、エラー状態を処理する方法について説明します。

付録 A では、分類機構データベースから ToolTalk 型データベースへの移行方法について説明します。

付録 B では、ToolTalk サービスがどのような方法でアプリケーション間の通信を実現しているかを示します。

付録 C では、連携させたいアプリケーションと同じプロトコルのアプリケーションを開発するための ToolTalk メッセージセットについて説明します。

付録 D では、ToolTalk サービスについてよく質問を受ける項目とその解答をまとめています。

関連マニュアル

次に、ToolTalk の関連マニュアルおよび関連文書を示します。

- 『ToolTalk リファレンスマニュアル』
- 『ToolTalk Message Sets』
- 『CASE Inter-Operability Message Sets』
- 『ToolTalk and Open Protocols』、ISBN 013-031055-7、SunSoft Press/Prentice Hall 発行

Sun のオンラインマニュアル

<http://docs.sun.com> では、Sun が提供しているオンラインマニュアルを参照することができます。マニュアルのタイトルや特定の主題などをキーワードとして、検索をおこなうこともできます。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P-1 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% su password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm <i>filename</i> と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第5章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% grep `^#define \ XV_VERSION_STRING`

コード例は次のように表示されます。

■ C シェルプロンプト

```
system% command y|n [filename]
```

■ Bourne シェルおよび Korn シェルのプロンプト

```
system$ command y|n [filename]
```

■ スーパーユーザーのプロンプト

```
system# command y|n [filename]
```

[] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

一般規則

- このマニュアルでは、英語環境での画面イメージを使っています。このため、実際に日本語環境で表示される画面イメージとこのマニュアルで使っている画面イメージが異なる場合があります。本文中で画面イメージを説明する場合には、日本語のメニュー、ボタン名などの項目名と英語の項目名が、適宜併記されています。

第 1 章

ToolTalk サービスの紹介

この章では、ToolTalk サービスの基本概念について説明します。

概要

ToolTalk サービスを使用すると、独立したアプリケーションが、互いに直接認識していても通信できます。アプリケーションは、ToolTalk メッセージを作成し、送信することで相互に通信します。ToolTalk サービスは、これらのメッセージを受信し、受信側を判別してから、そのメッセージを適切なアプリケーションに配信します。この通信の様子を図 1-1 に示します。

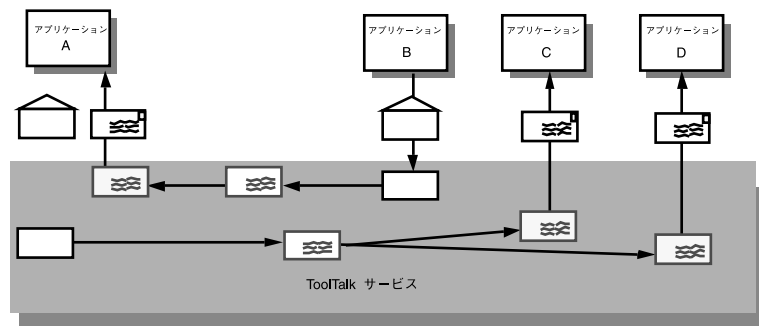


図 1-1 ToolTalk サービスを使ったアプリケーション

ToolTalk のシナリオ

次に示すシナリオは、ToolTalk サービスを利用することによって、業務上の問題をどのように解決できるかを示す例です。これらのシナリオで使用するプロトコルは、架空のものであります。

ToolTalk デスクトップサービスメッセージセット の使用方法

ToolTalk デスクトップサービスメッセージセットを使用することにより、アプリケーションは、ユーザーの介入なしで他のアプリケーションを統合および制御できます。この節では、デスクトップサービスメッセージセットの実行方法を示す2つのシナリオを説明します。

スマートデスクトップ

フロントエンドのグラフィカルユーザーインターフェース (GUI) が、データファイルがアプリケーションに気づく (または「知っている」) ように要求するには、アプリケーションレベルのプログラムが、ユーザーの要求を翻訳する必要があります。このアプリケーションレベルのプログラム (「スマートデスクトップ」と言います) には、Solaris のファイルマネージャ、Apple 社の Macintosh ファインダ、Microsoft 社の Windows ファイルマネージャなどがあります。スマートデスクトップの主な共通要件は、次のとおりです。

1. ファイルを取得する
2. アプリケーションを決定する
3. アプリケーションを起動する

ToolTalk サービスは、ツールのクラスが特定のデータ型を編集できるようにすることによって柔軟性を増します。次のシナリオでは、デスクトップサービスメッセージセットを、エンドユーザーに対して透過的なスマートデスクトップとして実行する方法を説明します。

1. 「ファイルマネージャ」アイコンをダブルクリックします。
ファイルマネージャが開き、カレントディレクトリ内のファイルが表示されます。
2. データファイルのアイコンをダブルクリックします。
 - a. ファイルマネージャは、アイコンで表現されるファイルの表示を要求します。また、「表示」メッセージ内のファイルタイプを符号化します。
 - b. ToolTalk セッションマネージャは、登録されたアプリケーション (この場合はアイコンエディタ) に「表示」メッセージ内のパターンを照合して、デスクトップ上で実行中の、アプリケーションのインスタンスを見つけます。

注 – ToolTalk セッションマネージャが、アプリケーションで実行中のインスタンスを見つけられない場合は、静的に定義した ptype を検査し、メッセージ内のパターンに最も一致するアプリケーションを起動します。一致する ptype がないと、ファイルマネージャに失敗を報告します。

- c. アイコンエディタは「表示」メッセージを受け取り、自分のアイコン化を解除し、自分を一番上に表示します。
3. ファイルを手作業で編集します。

統合ツールセット

デスクトップサービスメッセージセットを実行できるもう1つの重要なアプリケーションは、「統合ツールセット」です。これらの環境は、垂直のアプリケーション (CASE ソフトウェア開発者用ツールセットなど) または水平の環境 (複合文書など) に適用できます。その両方のアプリケーションの共通点は、解決法全体が、1つの特定のタスクをうまく実行するように設計されている専門のアプリケーション以外で構築されたという前提があることです。統合ツールセットアプリケーションには、テキストエディタ、描画パッケージ、ビデオディスプレイツール、オーディオディスプレイツール、コンパイラのフロントエンド、デバッガなどがあります。統合ツールセット環境には、相互に呼び出して対話し、ユーザーからの要求を処理するアプリケーションが必要です。たとえば、ビデオを表示するには、エディタがビデオディスプレイプログラムを呼び出します。また、完了コードのブロックを確認するには、エディタがコンパイラを呼び出します。次のシナリオでは、デスクトップサービスメッセージセットを統合ツールセットとして実行する方法を説明します。

1. エディタを使用して複合文書を扱う作業をします。
ソースコードテキストの一部を変更します。
2. 「ソースコードテキスト」をダブルクリックします。
 - a. 文書エディタは、まずソースコードを表すテキストを判定し、その後そのソースコードがどのファイルに入っているかを判定します。
 - b. 文書エディタは、ファイル名をメッセージのパラメタとして使用し、「編集」メッセージ要求を送信します。
 - c. ToolTalk セッションマネージャは、登録されたアプリケーション (この場合はソースコードエディタ) に「編集」メッセージ内のパターンを照合して、デスクトップ上で実行中のアプリケーションのインスタンスを見つけます。

注 – ToolTalk セッションマネージャが、アプリケーションの実行中のインスタンスを見つけられない場合は、静的に定義した ptype を検査し、メッセージ内のパターンに最も一致するアプリケーションを起動します。一致する ptype がないと、文書エディタアプリケーションに失敗を報告します。

- d. ソースコードエディタが「編集」メッセージ要求を受け取ります。
 - e. ソースコードエディタは、ソースコードファイルが構成制御を受けていることを判定し、ファイルを検査するためのメッセージを送信します。
 - f. そのメッセージをソースコード制御アプリケーションが受け取り、要求されたファイルの読み取りまたは書き込み用コピーを作成します。その後、ファイル名をソースコードエディタに戻します。
 - g. ソースコードエディタは、ソースファイルが入っているウィンドウを開きます。
3. ソースコードテキストを編集します。

ToolTalk 文書メディア交換メッセージセットの使用法

ToolTalk 文書メディア交換メッセージセットは、非常に柔軟性があり、強力です。この節では、次のような ToolTalk 文書メディア交換メッセージセットの3つのアプリケーションについて説明します。

- マルチメディアとオーサリングアプリケーションの統合
- 既存のアプリケーションへのマルチメディア拡張機能の追加
- メディア変換機能による「X」の「カット&ペースト」機能の拡張

マルチメディア機能の統合

マルチメディア機能をアプリケーションに統合することによって、アプリケーションのユーザーは、さまざまなメディアの型をそれらの文書に埋め込むことができます。

通常、メディアオブジェクトを表すアイコンは、文書に埋め込まれます。埋め込まれたオブジェクトを選択すると、ToolTalk サービスは自動的に適切な外部メディアアプリケーションを起動し、オブジェクトは次のように処理されます。

1. マルチメディアオブジェクトが入っている文書を開きます。
2. ウィンドウが、さまざまなメディアの種類 (音声、画像、グラフィックスなど) を表す複数のアイコンで文書を表示します。
3. 音声アイコンをダブルクリックします。
音声アプリケーション (「プレイヤ」と呼びます) が起動され、録音済みの音声再生されます。
4. 録音状態を編集するために、アイコンを1回クリックして選択し、3番目のマウスボタンを使用して「編集」メニューを表示します。
編集アプリケーションが起動され、メディアオブジェクトを編集します。

既存のアプリケーションへのマルチメディア拡張機能の追加

ToolTalk 文書メディア交換メッセージセットによって、アプリケーションは他のマルチメディアアプリケーションを使用して、その機能または性能を拡張することもできます。たとえば、次に示すように、カレンダーマネージャを拡張して、オーディオツールを使って音声ファイルをアポイントメントの覚え書きとして再生することもできます。

1. 自分のカレンダーマネージャを開いて、アポイントメントを設定します。
2. 音声応答ボタンをクリックすると、サウンドツールが表示されます。
3. たとえば、「レポートを持ち返る」というメッセージを記録します。

アポイントの覚え書きを実行すると、カレンダーマネージャはオーディオツールを起動し、録音した覚え書きを再生します。

X のカット & ペースト機能の拡張

ToolTalk 文書メディア変換メッセージセットは、拡張可能な無制限の変換機能をサポートできます。次に、拡張可能なマルチメディアの「カット & ペースト」機能の動作を示します。

1. メディア型の異なる 2 つの文書を開きます。
2. 「文書 A」の一部分を選択し、標準の「X」Window System の「カット」機能を使用してその部分を切り取ります。
3. カットした部分を「文書 B」に貼り付けます。
 - a. 「文書 B」は、カットしたデータの転送について「文書 A」と折衝します。
 - b. 文書 A が提供するデータのどの型についても「文書 B」が認識しない場合、「文書 B」は「タグ付きメディア型」を要求します。「文書 B」は、タグ付きメディア型を使用して、そのメディア型を認識可能なメディア型へ変換するように要求する ToolTalk メッセージを送ります。
 - c. 登録されている変換ユーティリティはその要求を受け、変換された後のバージョンのメディア型を「文書 B」へ返します。
 - d. 変換されたデータの「文書 B」へのペーストが実行されます。

CASE 連携メッセージセットの使用

CASE 連携メッセージセットによって、アプリケーションは、ユーザーの介入なしで他のアプリケーションを統合または制御できます。この節では、CASE 連携メッセージセットの使用方法を示すいくつかのシナリオを示します。

ユーザーシナリオ「バグの修正」

このシナリオでは、リリースされたアプリケーションに対するバグの修正方法の完全なサイクルを通して進みます。まず、バグレポートを受け取り、バグ修正に必要な処理を記述することから始まります。

1. 自分のアプリケーションに問題があるというバグレポートを受け取ります。
2. 自分の CASE 環境を起動します。
CASE ユーザーインターフェースが表示されます。実行したい機能は、この CASE ユーザーインターフェースで利用可能です。
3. バグレポートに記述されている問題点の写しとして、テストケースを書きます。
4. デバッグ機能を選択して、テストケースに対してアプリケーションを実行します。
 - a. デバッグ要求が送信されます。
 - b. メッセージ環境は、デバッグアプリケーションを選択します。CASE 環境で実行中のアプリケーションのインスタンスが見つけれられない場合は、自動的にデバッグを起動します。
 - c. デバッグアプリケーションは、要求を受信し、バイナリをロードします。
5. コードをテストし、デバッグウィンドウでデバッグ状態を再検査します。
関数呼び出しが誤った引数で渡されているところを見つけます。
6. 編集機能を選択して、このコードを編集します。
 - a. デバッグツールは、編集要求を送信します。
 - b. ソースエディタは、指定されたソースファイルを編集するようにというメッセージを受信します。
7. ソースコードを修正したいので、チェックアウト機能を選択します。
 - a. ソースコードエディタは、チェックアウト要求を送信します。
 - b. ソースコードエディタは、チェックアウト通知を受信し、バッファ状態を修正可能にします。
8. ソースコードを編集してバグを修正し、構築機能を選択して、アプリケーションを構築します。
 - a. 構築要求が送信されます。
 - b. 構築アプリケーションは、構築要求を受信し、構築を実行します。
 - c. 構築を完了すると、構築アプリケーションは「構築完了」通知を送信します。
 - d. デバッグは、「構築完了」通知を受信し、新たに構築されたアプリケーションバイナリを再ロードします。
9. アプリケーションを再テストして、バグ修正がうまくいったかどうかを確認します。
10. CASE 環境を終了します。
終了要求は、ソースコードエディタ、デバッグ、バージョンマネージャ、および構築アプリケーションへ送信されます。

ToolTalk ファイル名マッピング機能の使い方

ファイル名マッピング機能は、ToolTalk APL の一部を構成するものです。ToolTalk メッセージパッシングとは、独立してこれらを使用することによって、ネットワークから見えるファイル名の標準形式のエンコードまたはデコードを行います。共通ファイルにアクセスする必要があるアプリケーション内に、これらの標準形式を渡すことができます。ファイルの参照を行うホストによって、このファイルのパス名は異なる可能性があります。たとえば、ホスト A のファイルが `/home/fred/test.c` でも、他のホストでは `/net/A/export/home/fred/test.c` の場合もあります。ファイル名マッピング API は、NFS から見ることができるファイル名を標準的な名前に変換し (`tt_host_file_netfile()` で)、それが各ホストのアプリケーションに渡されます。別のホストは、その標準名を API (`tt_host_netfile_file()`) に渡して起動プログラムで使える UNIX パス名に戻します。

マルチスレッド環境での ToolTalk の使い方

Solaris 2.6 リリースおよびその互換バージョンでは、ToolTalk ライブラリは、マルチスレッド対策が施されています。マルチスレッド環境で ToolTalk を実行したいユーザーは、そのままシングルスレッドで使用することも、必要に応じてスレッドごとのデータを提供する (たとえば、`procid` やセッション ID など) 複数の新規 API 呼び出しを使用することもできます。

アプリケーションの ToolTalk メッセージの使用法

アプリケーションは、ToolTalk メッセージを作成、送信、または受信することによって、他のアプリケーションと通信します。送信側は、メッセージを作成、書き込み、または送信します。ToolTalk サービスは受信側を判別し、そのメッセージを受信側に配信します。受信側はメッセージを検出し、メッセージ内の情報を検査してから、メッセージを破棄するか、操作を実行してその結果を応答します。

ToolTalk メッセージの送信

ToolTalk メッセージの構造は簡単で、アドレス、サブジェクト、および配信情報を含んでいます。ToolTalk メッセージを送信するために、アプリケーションは空のメッセージを取得し、メッセージ属性を書き込んだ後、メッセージを送信します。送信を行うアプリケーションは、次の情報を提供する必要があります。

- メッセージが通知か、要求か、提供物なのか (すなわち受信側がメッセージに応答するかどうか)

- 受信側と送信側は、どのような処理対象を共有しているか (たとえば、受信側は、特定のユーザーセッションで実行されているか、または特定のファイルを処理の対象としているか)

送信側アプリケーションは、メッセージ配信の範囲を限定するために、メッセージ内にさらに情報を指定できます。

メッセージパターン

ToolTalk の重要な特徴は、送信側が受信側について何も知らなくてかまわないということです。これは、メッセージを受信したいアプリケーションの方が、受け取りたいメッセージがどのようなものかを明示的に示すからです。メッセージの型に関する情報は、「メッセージパターン」という形で ToolTalk サービスに登録されます。

アプリケーションはインストール時または実行時に、メッセージパターンを ToolTalk サービスに対して指定します。メッセージパターンは、メッセージと同様に作成します。つまり、どちらの場合も同種の情報を使用します。アプリケーションは受信したいそれぞれの型のメッセージについて、空のメッセージパターンを取得し、属性を書き込み、そのパターンを ToolTalk サービスに登録します。これらのメッセージパターンは通常、アプリケーションが相互に合意しているメッセージプロトコルと一致します。アプリケーションは、個々の使用に応じてさらにパターンを追加できます。

ToolTalk サービスは、送信側アプリケーションからメッセージを受信すると、メッセージ内の情報と登録されているパターンとを比較します。一致するものが見つかり、ToolTalk サービスは、パターンがメッセージと一致するすべての受信側アプリケーションにメッセージのコピーを配信します。

アプリケーションは、受信したいメッセージを記述したパターンごとに、メッセージを「処理」または「監視」できるかを宣言しています。多数のアプリケーションがメッセージを監視できますが、メッセージを処理できるアプリケーションは1つだけです。これは、要求された操作が確実に一度だけ行われるようにするためです。ToolTalk サービスが、メッセージを処理するハンドラを見つけ出せなかった場合は、そのメッセージを送信側アプリケーションに返し、配信が失敗したことを示します。

ToolTalk メッセージの受信

ToolTalk サービスは、メッセージを特定のプロセスに配信する必要があると判断すると、メッセージのコピーを作成し、受信待ちメッセージがあることをそのプロセスに通知します。受信側アプリケーションが実行中でない場合は、ToolTalk サービスは、アプリケーション起動方法に関する指示 (インストール時にアプリケーションが指定したもの) を検索します。

プロセスは、メッセージを検索し、その内容を検査します。

- 操作が完了したという情報がメッセージに含まれている場合は、プロセスは、その情報を読み取り、メッセージを破棄します。

- 操作の実行要求が含まれている場合は、プロセスはその操作を実行し、元のメッセージへの応答という形で操作の結果を返します。応答が返されると、プロセスは元のメッセージを破棄します。

ToolTalk メッセージの配布

ToolTalk サービスは、2つのメッセージのアドレス方式を提供します。「プロセス指向メッセージ」方式と「オブジェクト指向メッセージ」方式です。

プロセス指向メッセージ

「プロセス指向」メッセージとは、プロセスにアドレス指定されたメッセージのことです。プロセス指向メッセージを作成するアプリケーションは、そのメッセージを指定されたプロセスまたは特定の型のプロセスのどちらかにアドレス指定します。プロセス指向メッセージ方式は、既存のアプリケーションが他のアプリケーションと通信するのに便利な方法です。プロセス指向メッセージ方式をサポートするための修正は簡単で、通常は実行するのに時間はかかりません。

オブジェクト指向メッセージ

「オブジェクト指向」メッセージは、アプリケーションが管理するオブジェクトにアドレス指定されます。オブジェクト指向メッセージを作成するアプリケーションは、そのメッセージを指定されたオブジェクトまたは特定の型のオブジェクトのどちらかにアドレス指定します。オブジェクト指向メッセージ方式は、特に現在オブジェクトを使用しているアプリケーションまたはオブジェクトを対象として設計されたアプリケーションに便利です。既存のアプリケーションがオブジェクト指向でない場合は、ToolTalk サービスを使えば、アプリケーションがアプリケーションのデータの一部をオブジェクトとして識別することにより、これらのオブジェクトに関する通信ができるようになります。

注 - ToolTalk オブジェクト指向メッセージインタフェース用にコーディングされたプログラムは、ソースコードを変更しなければ CORBA 準拠のシステムに移植できません。

メッセージ配信の判別

メッセージを受信するグループを判別するために、メッセージの配信範囲を「指定」できます。配信範囲指定を行うことにより、メッセージの配信を特定のセッションまたはファイルに限定します。

セッション

「セッション」は、同じ ToolTalk メッセージサーバーのインスタンスを持つプロセスのグループのことです。プロセスが ToolTalk サービスと通信を開始する場合、デフォルトのセッションが配置され (またはセッションが存在していない場合は作成される)、プロセスには「プロセス識別子」(procid) が割り当てられます。デフォルトセッションは、環境変数または X ディスプレイによって配置されます。前者はプロセスツリーセッション、後者は X セッションと呼ばれます。

セッションの概念は、メッセージの配信において重要です。送信側は、あるセッションをメッセージの配信範囲とすることができます。ToolTalk サービスは、現在のセッションを参照するメッセージパターンを持つすべてのプロセスにメッセージを配信します。アプリケーションは、現在の「セッション識別子」(sessid) でメッセージパターンを更新するときに、そのセッションを結合します。

ファイル

このマニュアルでは、アプリケーションの処理対象であるデータを入れる入れ物のことを「ファイル」と呼びます。

ファイルの概念は、メッセージの配信において重要です。送信側は、あるファイルをメッセージの配信範囲とすることができます。また、ToolTalk サービスは、プロセスのデフォルトセッションに関わらず、そのファイルを参照するメッセージパターンを持つすべてのプロセスにメッセージを配信します。アプリケーションは、現在のファイルのパス名でメッセージパターンを更新するときに、そのファイルを結合します。

また、1つのセッション内で、あるファイルをメッセージの配信範囲とすることもできます。ToolTalk サービスは、そのファイルとセッションの両方を参照するメッセージパターンを持つすべてのプロセスにメッセージを配信します。

注 - ファイルの配信範囲指定機能を使用できるのは、NFS™ ファイルシステムと UFS ファイルシステムだけです。たとえば、tmpfs ファイルシステムでは、この機能は使用できません。

ToolTalk サービスを使用するためのアプリケーションの変更

ToolTalk サービスを使用できるようにアプリケーションを変更する前に、ToolTalk 「メッセージプロトコル」を定義(または配置)する必要があります。メッセージプロトコルとは、アプリケーションが実行を認めた操作を記述した ToolTalk メッセージの集合です。メッセージプロトコル仕様には、メッセージの設定とアプリケーションがメッセージを受信したときの動作が含まれます。

ToolTalk サービスを使用するために、アプリケーションは、ToolTalk アプリケーションプログラミングインタフェース (API) の ToolTalk 関数を呼び出します。ToolTalk API には、ToolTalk サービスに登録する機能、メッセージパターンを作成する機能、メッセージを送信する機能、メッセージを受信する機能、メッセージ情報を検査する機能などがあります。ToolTalk サービスを使用できるようにアプリケーションを変更するには、プログラムに ToolTalk API のヘッダーファイルを組み込む必要があります。また、アプリケーションを変更して、次のことを実現する必要があります。

- ToolTalk サービスを初期化し、セッションに参加する
- メッセージパターンを ToolTalk サービスに登録する
- メッセージを送信する
- メッセージパターンの登録を解除し ToolTalk セッションを終了する

第 2 章

ToolTalk サービスの概要

コンピュータのユーザーは、独立して開発した個々のアプリケーションの共同作業に対する要求をますます高めています。このため、ソフトウェア開発者にとっては「連携」が重要なテーマになっています。連携アプリケーションは、相互の機能を協調して使用することによって、1つのアプリケーションでは提供しきれないユーザー機能を提供します。ToolTalk サービスは、個人やグループに提供する連携アプリケーションの開発を容易にするように設計されています。

ToolTalk アーキテクチャ

次の ToolTalk サービスの構成要素が一緒に動作して、アプリケーション間の通信とオブジェクト情報の管理を提供します。

- `ttsession` は、ToolTalk 通信プロセスです。
このプロセスは、同一の X サーバーを使用しているか同一のファイルを処理対象とする送信側と受信側を結合します。メッセージを他のセッションのアプリケーションに配信する必要がある場合、1つの `ttsession` は他の `ttsession` と通信します。
- `rpc.ttdbserverd` は、ToolTalk データベースのサーバープロセスです。
`rpc.ttdbserverd` は、ToolTalk クライアントの対象となるファイルまたは ToolTalk オブジェクトを含むファイルを格納したディスクパーティションを持つ各マシンに、1つずつインストールされます。
ファイル情報と ToolTalk オブジェクト情報は、`rpc.ttdbserverd` が管理するレコードデータベースに格納されています。
- `libtt` は、ToolTalk アプリケーションプログラミングインタフェース (API) のライブラリです。
アプリケーションは、プログラム内に API ライブラリを持っており、そのライブラリの中から ToolTalk 関数を呼び出します。

ToolTalk サービスは、SunSoft™ ONC™ 遠隔手続き呼び出し (RPC) を使用して、これらの ToolTalk 構成要素間で通信します。

アプリケーションは、ToolTalk サービスにプロセス型情報とオブジェクト型情報を提供します。この情報は、XDR 形式のファイルに格納されます。このマニュアルでは、このファイルのことを ToolTalk 型データベースと呼びます。

ToolTalk セッションの開始

ToolTalk サーバーとの通信を開始する際に、ToolTalk のメッセージサーバー `ttsession` は自動的に起動します。メッセージの送受信の前に、このバックグラウンドプロセスが実行中でなければなりません。各メッセージサーバーは、セッションを 1 つずつ定義します。

注 - 1 つのセッションは、複数のセッション識別子を持つことができます。

セッションを手入力で開始するには、次のようにコマンドを入力します。

```
ttsession [-a level][-d display][-spStvh][{-E|X}][-c command]
```

`ttsession` コマンド行のオプションについては、表 2-1 を参照してください。

表 2-1 `ttsession` コマンド行のオプション

引数	説明
-a <i>level</i>	サーバー認証レベルを設定する。 レベルは、 <i>unix</i> 、 <i>gss</i> 、または <i>des</i> でなければならない
-d <i>display</i>	指定されたディスプレイの X セッションの開始を <code>ttsession</code> に指示する。 通常、 <code>ttsession</code> は <code>\$DISPLAY</code> 環境変数を使用する
-h	<code>ttsession</code> の呼び出し方法と終了方法についてヘルプを出力する
-p	新しい <code>ttsession</code> を開始して、そのセッション ID を出力する

表 2-1 ttsession コマンド行のオプション (続き)

引数	説明
-S	ttsession に、そのセッションを管理するために、バックグラウンドインスタンスをフォークしないように指示する
-o allow_unaut_types_load=<yes no>	デフォルトでは、ToolTalk API の <code>tt_session_types_load(3)</code> への呼び出しは、 <code>TT_ERR_ACCESS</code> では失敗する。システム全体のデフォルト値は、 <code>ttsession_file(4)</code> で変更できる。このオプションを使用した特定の <code>ttsession</code> を変更できるのは、 <code>ttsession_file(4)</code> がセッションごとの変更をこのオプションに「固定」されていない場合のみ
-s	サイレント操作を可能にする。警告メッセージは出力されない
-N	開いているファイル記述の制限値を引き上げることによって、このセッションに接続可能なクライアント数を最大限にする。使用可能なファイル記述子の正確な数は、システムによって異なる。Solaris 2.6 オペレーティング環境およびその互換バージョンでは、 <code>ttsession</code> は常にクライアント数を最大にするので、このオプションを指定する必要はない
-t	<p>トレースモードを設定する。<code>ttsession</code> 実行中にトレースモードが設定されると、コンソール上にメッセージが表示される。</p> <p>トレースモードは、メッセージが添付または送信される様子を確認したい場合に設定する。<code>ttsession</code> コマンドによって最初に確認された時のメッセージの状態を表示する。また、与えられたプロセスへのメッセージの送信の試行も、成功した場合には表示する。</p> <p>トレースモードの設定と解除を切り替えるには、<code>USR1</code> シグナルを使用する</p>
-v	バージョン番号を出力して終了する
-E	分類機構データベースからタイプを読み取る。このオプションは使用不可 (現在の ToolTalk サービスは XDR 形式のデータベースを使用している)
-X	<code>\$TTHOME/.tt/types.xdr</code> と <code>/etc/tt/types.xdr</code> 内の XDR 形式のデータベースからタイプを読み取る。このオプションはデフォルト設定

表 2-1 `ttsession` コマンド行のオプション (続き)

引数	説明
<code>-c command</code>	<p>プロセスのツリーセッションを開始して与えられたコマンドを実行する。</p> <p>特別な環境変数 <code>TT_SESSION</code> が本セッション名に設定される。この環境変数で開始されたプロセスは、デフォルト状態で本セッションに含まれる。</p> <p>このオプションは、コマンド行の最後で指定しなければならない。コマンド行の <code>-c</code> オプションの後に置かれた文字は、すべて実行するコマンドと解釈される。</p> <p><code>command</code> を省略すると <code>\$SHELL</code> の値が代わりに使用される</p>

注 `-c`、`-d`、または `-p` オプションのどれも指定しない場合、`ttsession` は、`$DISPLAY` 環境変数で指定したディスプレイの X セッションを開始します。

`ttsession` は、次の 2 つのシグナルに応答します。

- SIGUSR1 シグナルを受信する場合は、トレースモードをオンまたはオフに切り替えます。
- SIGUSR2 シグナルを受信する場合は、型ファイルをもう一度読み取ります。

バックグラウンドおよびバッチセッション

バッチセッション内または文字端末に確立したセッション内で、アプリケーションをバックグラウンドジョブとして実行する場合は、このアプリケーションを自身のセッションとして実行します。アプリケーションを自身のセッションで実行するには、次のように `ttsession` コマンドで `-c` パラメタを使用します。

```
ttsession -c [command-to-non-in-batch]
```

このコマンドは、ユーザーのアプリケーションを実行できるシェルをフォークします。

注 `-c` パラメタは、コマンド行上の最後のオプションでなければなりません。コマンド行上の `-c` パラメタの後に置かれる文字は、実行コマンドと見なされません。

X Window System

X Window System でセッションを確立するには、`ttsession` を引数なし (`$DISPLAY` 環境変数からディスプレイ設定をとる) で実行するか、次のようにディスプレイを `-d` パラメタ付きで指定します。

```
ttsession -d :0
```

`ttsession` が呼び出されると、すぐにフォークして親コピーが終了します。セッションを管理するプロセスは、バックグラウンドで実行されます。セッションはプロパティとして登録され、画面 0 のルートウィンドウ上で `TT_SESSION` によって命名されます。ホスト番号とポート番号が指定され、セッションを管理するプロセスとの通信に使用されます。

ttsession の検出

X ディスプレイのセッションの `sessid` を表示するには、次のように入力します。

```
xprop -root | grep TT_SESSION
```

ToolTalk ファイルとデータベースの管理

ToolTalk パッケージには、ToolTalk ファイル (つまり、メッセージで述べられているファイルと ToolTalk オブジェクトが入っているファイル) のコピー、移動、および削除に使用できる一連の特定のシェルコマンドが入っています。標準シェルコマンド (`cp`、`mv`、`rm`) の実行後、ToolTalk サービスは、ファイルの場所が変更されたという通知を受けます。

また、ToolTalk パッケージには、ToolTalk データベース用の、データベースの検査および修復のユーティリティである `ttdbck` も入っています。これを使用すれば、ToolTalk データベースの検査と修復ができます。

デモンストレーションプログラム

ToolTalk サービスソースファイルには、2 つの Motif 型のデモンストレーションプログラムが含まれています。

- `ttsample1` — ToolTalk サービスの送受信メッセージの簡単なデモンストレーションです。

- `edit_demo` — ToolTalk のオブジェクト指向メッセージのデモンストレーションを行う、`cnt1` と `edit` の 2 つのプログラムです。

第 3 章

メッセージパターン

この章では、メッセージパターン情報を ToolTalk サービスに提供する方法について説明します。ToolTalk サービスは、メッセージパターンを使用してメッセージの受信側を判定します。メッセージを受信した後で、ToolTalk サービスは、メッセージを現在のメッセージパターンのすべてと比較して一致するパターンを見つけます。パターンが一致すると、メッセージは、メッセージパターンを登録したアプリケーションに配信されます。

動的、静的、またはその両方を使用して、メッセージパターン情報を ToolTalk サービスに提供できます。選択できる方法は、受信したいメッセージの型によって異なります。

- 受信したいメッセージの型がアプリケーションの実行中に変わると思われる場合は、「動的な方法」によって、アプリケーションの起動後もメッセージパターン情報を追加、変更、または削除できます。
- アプリケーションが実行中でないときに、メッセージにアプリケーションを起動させたい場合またはメッセージを待ち行列に入れたい場合は、「静的な方法」がこれらの指示を簡単に指定できる方法を提供します。また静的な方法は、定義した一連のメッセージを受信したい場合に、メッセージパターン情報を簡単に指定できる方法も提供します。詳細は、第 10 章を参照してください。

ToolTalk サービスにメッセージパターンを提供する選択方法に関係なく、現在の各セッションおよびファイル情報によってこれらのパターンを更新すると、処理の対象とするセッションまたはファイルを参照する、すべてのメッセージを受信します。

メッセージパターンの属性

メッセージパターンの属性は、ユーザーが受信したいメッセージの型を指定します。1 つの値しか設定されない属性があっても、1 つのパターンに追加する属性のほとんどに、複数の値を設定できます。

メッセージパターンに設定できる属性の完全なリストを表 3-1 に示します。

表 3-1 ToolTalk メッセージパターンの属性

パターン属性	値	説明
カテゴリ	TT_OBSERVE, TT_HANDLE, TT_HANDLE_PUSH, TT_HANDLE_ROTATE	メッセージに表示されている操作を実行するか、 メッセージの監視だけを行うかを宣言する
配信範囲	TT_SESSION, TT_FILE, TT_FILE_IN_SESSION, TT_BOTH	セッションおよびファイル、またはそのどちらかに 関するメッセージを対象としていることを宣言する。 sessid およびファイル名を更新するためにメッセージパ ターンを登録した後、セッションまたはファイルを結 合する
引数	引数または結果	処理の対象とする操作の定位置引数を宣言する
コンテキスト	<名前, 値>	処理の対象とする操作のキーワードまたは非定位置引 数を宣言する
クラス	TT_NOTICE, TT_REQUEST, TT_OFFER	通知、要求、提供、またはそれらすべてを受信する かどうかを宣言する
ファイル	<i>char *pathname</i>	処理の対象とするファイルを宣言する。配信範囲が ファイルに関係しない場合、このパターン属性は無効 である
オブジェクト	<i>char *objid</i>	処理の対象とするオブジェクトを宣言する
操作	<i>char *opname</i>	処理の対象とする操作を宣言する
otype	<i>char *otype</i>	処理の対象とするオブジェクトの型を宣言する
アドレス	TT_PROCEDURE, TT_OBJECT, TT_HANDLER, TT_OTYPE	処理の対象とするアドレスの型を宣言する
処置	TT_DISCARD, TT_QUEUE, TT_START, TT_START+TT_QUEUE	インスタンスを実行中でなければ、アプリケーション へのメッセージの処理方法を ToolTalk サービスに指示 する
送信側	<i>char *procid</i>	処理の対象とする送信側を宣言する
sender_ptype	<i>char *ptype</i>	処理の対象とする送信側プロセスの型を宣言する
セッション	<i>char *sessid</i>	処理の対象とするセッションを宣言する
状態	TT_CREATED, TT_SENT, TT_HANDLED, TT_FAILED, TT_QUEUED, TT_STARTED, TT_REJECTED, TT_RETURNED, TT_ACCEPTED, TT_ABSTAINED	処理の対象とするメッセージの状態を宣言する

少なくとも指定しなければならないすべてのメッセージパターンは次のとおりです。

- カテゴリ — アプリケーションが、メッセージに表示されている操作を実行するか、またはメッセージの表示だけを行うか
 - メッセージの監視だけを行う場合は、TT_OBSERVE を使用します。
 - メッセージによって要求される操作を実行する場合は、TT_HANDLE を使用します。
 - 別の HANDLE カテゴリの任意のパターンを使用する前に TT_HANDLE_PUSH のカテゴリがあり、その最後に登録したパターンを使用したい場合は、TT_HANDLE_PUSH を使用します。
 - TT_HANDLE を使用したいときは、TT_HANDLE_ROTATE を使用します。適切な TT_HANDLE_PUSH パターンが見つからないときは、他の TT_HANDLE パターンを使う前に、一番古く使用された TT_HANDLE_ROTATE パターンを使ってメッセージを配信します。
- 配信範囲 — アプリケーションが、特定のセッションまたはファイルに関するメッセージを処理の対象としているか
 - 自分のセッション内で他のプロセスからメッセージを受信するには、TT_SESSION を使用します。
 - 結合したファイルに関するメッセージを受信するには、TT_FILE を使用します。
 - このセッションにいる間に結合したファイルへのメッセージを受信するには、TT_FILE_IN_SESSION を使用します。
 - ファイル、セッション、または結合したファイルとセッションの両方のメッセージを受信するには、TT_BOTH を使用します。

ToolTalk サービスは、次のようにメッセージ属性をパターン属性と比較します。

- 次の場合は、ToolTalk サービスはメッセージ属性が一致したと見なします。
 - パターン属性が指定されていない
 - パターンが、コンテキストスロットを指定しない
 - パターンが、空のコンテキストスロットを持っている

注 — 指定するパターン属性が少ないほど、より多くのメッセージを受信する資格を持つようになります。

- パターン属性用に指定された値が複数ある場合は、値の1つがメッセージ属性値に一致しなければなりません。値が一致しない場合、ToolTalk サービスは、アプリケーションを受信側とは見なしません。
- メッセージにコンテキストスロットが含まれる場合、ToolTalk サービスは、アプリケーションを受信側とは見なしません。ただし、次のような場合は除きます。
 - パターンのコンテキストスロットに指定した値が、メッセージのコンテキストスロットに指定された値と一致する

- 複数のコンテキストスロットがメッセージに指定された場合、メッセージの各コンテキストスロットの値は、パターン内の対応するコンテキストスロットの値に一致する

配信範囲属性

メッセージパターンに次のような配信範囲の型を指定できます。

1. セッションだけの配信範囲
2. ファイルだけの配信範囲
3. 特定のセッションのファイルだけの配信範囲
4. ファイルおよびセッションのどちらか、または両方への配信範囲

注 – ファイルの配信範囲指定機能を使用できるのは、NFS ファイルシステムと UFS ファイルシステムだけです。たとえば、`tmpfs` ファイルシステムなどの他の型のファイルシステムでは、ファイルへの配信範囲は指定できません。

セッションだけの配信範囲指定

`TT_SESSION` 型は、セッションだけを配信範囲とします。静的セッション配信範囲指定のパターンは、明示的な `tt_session_join` 呼び出しに配信範囲の値を設定するように要求します。動的セッション配信範囲指定のパターンは、`tt_session_join` 呼び出しまたは `tt_pattern_session_add` 呼び出しのどちらかによって設定できます。

注 – これらの呼び出しが指定するセッションは、デフォルトセッションでなければなりません。

例 3-1 は、静的セッション配信範囲指定のパターンを示します。例 3-2 は、動的セッション配信範囲指定のパターンを示します。

例 3-1 静的セッション配信範囲指定のパターン

Obtain procid	<code>tt_open();</code>
Ptype is scoped to session	<code>tt_ptype_declare(ptype);</code>

例 3-1 静的セッション配信範囲指定のパターン (続き)

Join session	<code>tt_session_join(tt_default_session());</code>
--------------	-----------------------------------------------------

例 3-2 ファイル属性による動的セッション配信範囲指定のパターン

Obtain procid	<code>tt_open();</code>
Create pattern	<code>Tt_pattern pat = tt_create_pattern();</code>
Add scope to pattern	<code>tt_pattern_scope_add(pat, TT_SESSION);</code>
Add session to pattern	<code>tt_pattern_session_add(tt_default_session());</code>
Register pattern	<code>tt_pattern_register(pat);</code>

ファイルだけの配信範囲指定

TT_FILE 型は、ファイルだけを配信範囲とします。例 3-3 は、静的ファイル配信範囲指定のパターンを示します。例 3-4 は、動的ファイル配信範囲指定のパターンを示します。

例 3-3 静的ファイル配信範囲指定のパターン

Obtain procid	<code>tt_open();</code>
Ptype is scoped to file	<code>tt_ptype_declare(ptype);</code>
Join file	<code>tt_file_join(file);</code>

例 3-4 動的ファイル配信範囲指定のパターン

Obtain procid	<code>tt_open();</code>
Create pattern	<code>Tt_pattern pat = tt_create_pattern();</code>
Add scope to pattern	<code>tt_pattern_scope_add(pat, TT_FILE);</code>
Add file to pattern	<code>tt_pattern_file_add(pat, file);</code>
Register pattern	<code>tt_pattern_register(pat);</code>

セッション内のファイルへの配信範囲指定

TT_FILE_IN_SESSION 型は、指定セッション内の指定ファイルだけを配信範囲とします。この配信範囲設定によってパターンは、ファイルとセッションの両方を配信範囲とするメッセージだけに一致します。例 3-5 では、セッションを追加してからパターンを登録します。

例 3-5 TT_FILE_IN_SESSION 配信範囲指定のパターンへのセッション追加

Obtain procid	tt_open();
Create pattern	Tt_pattern pat = tt_create_pattern();
Add scope to pattern	tt_pattern_scope_add(pat,TT_FILE_IN_SESSION);
Add file to pattern	tt_pattern_file_add(pat, file);
Add session to pattern	tt_pattern_session_add(pat, tt_default_session());
Register pattern	tt_pattern_register(pat);

例 3-6 は、パターンを登録してからセッションを結合します。

例 3-6 TT_FILE_IN_SESSION 配信範囲指定のパターンのセッションを設定するためのセッション結合

Obtain procid	tt_open();
Create pattern	Tt_pattern pat = tt_create_pattern();
Add scope to pattern	tt_pattern_scope_add(pat, TT_FILE_IN_SESSION);
Add file to pattern	tt_pattern_file_add(pat, file);
Register pattern	tt_pattern_register(pat);
Join session	tt_session_join(tt_default_session());

例 3-7 は、静的パターンの配信範囲の値を設定します。

例 3-7 TT_FILE_IN_SESSION 静的パターンの配信範囲値の設定

Obtain procid	tt_open();
Declare Ptype	Tt_ptype_declare(ptype);
Join File	tt_file_join(file);
Join session	tt_session_join(tt_default_session());

ファイル、セッション、または両方への配信範囲指定

TT_BOTH 配信範囲指定のパターンは、ファイル、セッション、またはその両方へ配信範囲指定したメッセージと一致します。しかし、この配信範囲指定を使用する場合は、`tt_file_join` 呼び出しを明示的に行う必要があります。呼び出しを行わないと、ToolTalk サービスは登録されたパターンのファイルとセッションの両方に配信範囲指定したメッセージだけに一致します。例 3-8 と例 3-9 は、この配信範囲の使用例を示します。

例 3-8 TT_BOTH 配信範囲を使用する動的パターン

Obtain procid	<code>tt_open();</code>
Create pattern	<code>Tt_pattern pat = tt_create_pattern();</code>
Add scope to pattern	<code>tt_pattern_scope_add(pat, TT_BOTH);</code>
Add session to pattern	<code>tt_pattern_session_add(pat, tt_default_session());</code>
Add file to pattern	<code>tt_pattern_file_add (pat, file);</code>
Register pattern	<code>tt_pattern_register(pat);</code>

例 3-9 TT_BOTH 配信範囲を使用する静的パターン

Obtain procid	<code>tt_open();</code>
Declare Ptype	<code>Tt_ptype_declare(ptype);</code>
Join file	<code>tt_file_join(file)</code>
Join session	<code>tt_session_join(tt_default_session());</code>

配信範囲指定したパターンへのファイルの追加

同じファイル属性を持つ TT_SESSION 配信範囲指定のメッセージと TT_SESSION 配信範囲指定のパターンを一致させるために、`tt_pattern_file_add` 呼び出しによって、ファイル属性を TT_SESSION 配信範囲指定のパターンに追加できます (詳細は、例 3-10 を参照してください)。

注 - ファイル属性の値は、パターンの配信範囲に影響しません。

例 3-10 2つのファイル属性のセッション配信範囲指定したパターンへの追加

Obtain procid	<code>tt_open();</code>
Create pattern	<code>Tt_pattern pat = tt_create_pattern();</code>
Add scope to pattern	<code>tt_pattern_scope_add(pat, TT_SESSION);</code>
Add session to pattern	<code>tt_pattern_session_add(tt_default_session());</code>
Add first file attribute to pattern	<code>tt_pattern_file_add(pat, file1);</code>
Add second file attribute to pattern	<code>tt_pattern_file_add(pat, file2);</code>
Register pattern	<code>tt_pattern_register(pat);</code>

コンテキスト属性

ToolTalk の「コンテキスト」は、メッセージとパターンの両方に明示的に含まれる一連の <名前, 値> のペアです。ToolTalk コンテキストによって、詳細な照合処理を実行できます。

コンテキストを使用して、任意のペアを ToolTalk メッセージとパターンに関連付けることができます。また、一連のメッセージの受信側を制限できます。ToolTalk コンテキスト属性が提供する、制限付きパターン照合の一般的な使用法は、サブセッションを作成することです。たとえば、2つの異なるプログラムは、各プログラムのために動作するブラウザ、エディタ、デバッガ、構成マネージャなどのツールによって同時にデバッグできます。各ツールセットのメッセージとパターンのコンテキストスロットは、異なる値を持っています。これらの値の正常な ToolTalk パターンの照合は、2つのサブセッションを別々に保持します。

ToolTalk コンテキスト属性が提供する、制限付きパターン照合の別の使用法は、環境変数とコマンド行引数の情報を、ToolTalk サービスが起動するツールに提供することです。

パターン引数の属性

ToolTalk のパターン引数には、文字列、バイナリデータ、および整数値があり、受信メッセージと照合をとるのに ToolTalk サービスが使用します。

引数が配置されたパラメタであるのに対して、コンテキストは名前付きのパラメタですので、引数はコンテキストによって異なります。メッセージに設定されているコンテキストの順番で、受信および送信メッセージ内の順番が決まります。つまり、パ

ターンで設定されている引数の順番とタイプに一致します。引数は位置が重要な意味を持つため、受信メッセージの最初の引数ではない引数のマッチングをとりたい場合は、パターンの中間引数としての「ワイルドカード」引数を追加する必要があります。ワイルドカード引数は、「ALL」の `vtype` で NULL 値を持ちます。

パターン引数を使用して、自分の引数 (整数、バイナリデータ、または ASCII 文字列) のタイプに一致する API 呼び出しを追加します。特に、`tt_pattern_iarg_add()` の NULL のワイルドカード値を使ってパターン引数を加えることはできないので注意してください。NULL つまり 0 は、有効な整数の引数であるためです。ワイルドカードの引数を加えるには、`tt_pattern_arg_add()` を使用してください。

処置属性

ユーザーアプリケーションのインスタンスが動作していない場合、処置属性で、そのアプリケーションに送られたメッセージの処理方法を ToolTalk サービスに指示します。

パターンの静的型定義で指定した処置の値がデフォルトになりますが、メッセージの処置でハンドラの `ptype` を指定すると、このデフォルトは上書きされます。たとえば、「Display」というメッセージングニチャの入った「UWriteIt」という `ptype` に対して、メッセージの処置で静的型定義を指定するとします。このメッセージングニチャは、そのパターンのどの静的シグニチャとも一致しないため、ToolTalk サービスはそのメッセージ中に設定された処置に従います。たとえば、メッセージの処置が `TT_START` で、「UWriteIt」という `ptype` で開始文字列を指定すると、ToolTalk サービスはアプリケーションのインスタンスが 1 つも動作していない場合、アプリケーションのインスタンスを起動します。

第 4 章

ToolTalk プロセスの設定と管理

注 - ToolTalk データベースのサーバープログラムは、ToolTalk オブジェクトまたは ToolTalk メッセージに関連するファイルが入ったマシンすべてにインストールされなければなりません。

ToolTalk サービスファイルの場所

ToolTalk バイナリとライブラリは、`/usr/dt` のシンボリックリンクとともに `/usr/openwin` にあります。したがって、Solaris 共通デスクトップ環境 (Common Desktop Environment, CDE) または OpenWindows (OW) のユーザーは、同じバージョンの ToolTalk を利用できます。オンラインのマニュアルページと ToolTalk デモプログラムのソースは、`/usr/openwin` にあります。

表 4-1 に ToolTalk サービスファイルを示します。

表 4-1 ToolTalk サービスファイル

ファイルまたは場所	説明
<code>ttsession</code>	メッセージの配信のため、ネットワーク上で他の <code>ttsession</code> と通信
<code>rpc.ttdbserverd</code>	ToolTalk オブジェクト仕様と情報を ToolTalk メッセージで参照されるファイルに保存して管理

表 4-1 ToolTalk サービスファイル (続き)

ファイルまたは場所	説明
ttcp, ttmv, ttrm, ttrmdir, tttar	これらのコマンドは標準の OS シェルコマンドで、ToolTalk オブジェクトを含むファイルまたは ToolTalk メッセージの主体であるファイルがコピー、移動、または削除されたとき、その旨を ToolTalk に伝える
ttdbck	データベースチェックおよび ToolTalk データベースのためのリカバリツール
tt_type_comp	ptype および otype 用コンパイラ。ptype と otype をコンパイルし、ToolTalk 型データベースに自動的にインストールする
ttce2xdr	ToolTalk 型データを分類機構データベースフォーマットから XDR データベースフォーマットに変換する
ttsnoop	ToolTalk パターンを登録したり、ToolTalk メッセージを送信したりする Motif アプリケーション。ToolTalk メッセージトラフィックの一般的な監視も行う。これは、既存のアプリケーションのデバッグを行う際や、チュータとしてパターンの各部分がどのように受信メッセージをフィルタ処理するかを理解するのに便利
tttrace	tttrace は、truss(1) に似ている。このアプリケーションは、その使用方法に 2 つの方法がある。これを使用して、特定の ttssession で発生しているメッセージパッシングとパターンマッチングをトレースすることも、ToolTalk API へのすべてのコールのトレースをプログラムごとに行うこともできる
libtt.so.2	アプリケーションプログラミングインタフェース (API) ライブラリ
tttk.h と tt_c.h (/usr/dt/include/Tt にある)	メッセージの送受信用のアプリケーションで使用する ToolTalk 関数を含むヘッダーファイル
/usr/openwin/man/man1	ttssession, ttdbck, tt_type_comp などのユーザーコマンド用 ToolTalk のマニュアルページ
/usr/openwin/man/man1m	rpc.ttdbserverd, ttdbck などの ToolTalk 管理コマンド用 ToolTalk のマニュアルページ
/usr/openwin/man/man3	ToolTalk API コール用 ToolTalk のマニュアルページ
/usr/openwin/man/man4	ToolTalk メッセージセットおよび ttssession(1) と rpc.ttdbserverd(1M) で使用される構成ファイル用の ToolTalk のマニュアルページ

表 4-1 ToolTalk サービスファイル (続き)

ファイルまたは場所	説明
/usr/openwin/man/man5	インクルードファイル用 ToolTalk のマニュアルページ
ttsample, edit_demo, Makefile (/usr/openwin/share/src/tooltalk/demo にある)	ToolTalk 簡易デモプログラム用ソースコード

バージョン

すべての ToolTalk コマンドは、バージョンを出力する `-v` オプションをサポートしています。

必要条件

ToolTalk サービスに必要なソフトウェアには、ONC RPC があります。

環境変数

ここでは、ToolTalk と関連環境変数について説明します。

ToolTalk 環境変数

いくつかの ToolTalk 環境変数を設定できます。これらの変数について、表 4-2 に説明します。

表 4-2 環境変数

変数	説明
TTSESSION_CMD	ツールが <code>ttsession</code> を自動的に開始するときに、指定された標準オプションを無効にする。この変数を設定すると、すべての ToolTalk クライアントはこのコマンドを使用して、X セッションを自動的に開始させる
TT_ARG_TRACE_WIDTH	トレースモードを使用する際に、表示する引数およびコンテキストの値の文字数を定義する。デフォルトでは最初の 40 字を表示する
TT_FILE	定義済みのファイルに範囲指定したメッセージがツールを呼び出す際に、 <code>ttsession</code> がこの変数にパス名を設定する
TT_HOSTNAME_MAP	マップファイルを指す。定義したマップファイルは、ホストマシンのリダイレクトのために ToolTalk クライアントが読み取る
TT_PARTITION_MAP	マップファイルを指す。定義したマップファイルは、ファイルパーティションのリダイレクトのために ToolTalk クライアントが読み取る
TT_SESSION	<code>ttsession</code> は、それが起動するツールにセッション識別子を通知する。この変数を設定すると、ToolTalk クライアントライブラリは、その値をデフォルトのセッション識別子として使用する。この変数に記憶された文字列は、 <code>tt_default_session_set</code> へ渡すことができる
TT_TOKEN	ToolTalk クライアントライブラリに、 <code>ttsession</code> によって起動したことを通知する。これによってクライアントは、起動が正常であったことを <code>ttsession</code> に確認できる
TT_TRACE_SCRIPT	<code>tttrace(1)</code> についてトレーススクリプトで指定したように、 <code>libtt</code> がクライアントサイドのトレースを行うようにする
TTPATH	<code>tt_type_comp(1)</code> と <code>rpc.ttdbserverd(1M)</code> で使用する ToolTalk 型データベースがどこに常駐するかを ToolTalk に通知する
CEPATH	ToolTalk 型データベースの常駐場所を分類機構に通知する
DISPLAY	<p>TT_SESSION 変数を設定しない場合、<code>ttsession</code> は起動するツールにセッション識別子を通知する。</p> <p>DISPLAY 変数を設定すると、ToolTalk クライアントライブラリは、その値をデフォルトのセッション識別子として使用する。この変数は、<code>ttsession</code> が OpenWindows 上で動作しているときに自動起動すると通常設定される。</p> <p>注 - Solaris 2.6 オペレーティング環境およびその互換バージョンのオペレーティングシステムでは、アカウントによってはこの変数が渡されない場合がある。つまり、コンソールにユーザー A またはユーザー B のスイッチユーザーとしてログオンしている場合は、通常 <code>ttsession</code> を自動起動する ToolTalk プログラムを実行しようとしても <code>ttsession</code> が自動起動しないことがある。この問題を回避するためには、この変数を手入力で設定するか、<code>.login</code> ファイルに含める必要がある。</p>

表 4-2 環境変数 (続き)

変数	説明
DTMOUNTPOINT	この変数を設定すると、構築されているパス名の /net の代わりに、この環境変数の値を使用して、rpc.ttdbserverd(1M) による tt_host_netfile_file(3) 照会への応答を行う

ToolTalk サービスが自動的に起動すると環境が変わります。この環境の中には、\$TT_SESSION と \$TT_TOKEN という環境変数や、キーワードがドル記号 (\$) で始まる開始メッセージのコンテキストがすべて入っています。また、ファイルを配信範囲とするメッセージの場合には、\$TT_FILE という環境変数もこの環境に入ることができます。

注 - 子プロセスが tt_open を起動する場合、その親プロセスはこの環境の変化を子プロセスに伝播しなければなりません。

その他の環境変数

ToolTalk の開発環境を操作するのに TMPDIR という環境変数も使用できます。たとえば、次のように入力すると、ディレクトリ /var/tmp にファイルがリダイレクトされます。

```
TMPDIR=/var/tmp
```

リモートホスト上のプログラムの起動に必要な環境変数

開始文字列によってプロセスが起動されるのは ttsession が動作しているホスト上だけですが、起動されたプロセスは他のホスト上の他のプロセスを起動できます。

そのためには、まず開始文字列を次のようにします。

```
# rsh farhost myprog
```

次に、myprog を確実に正しいセッションに入れて初期メッセージを受信させるために、ToolTalk の重要な環境変数をいくつか伝播する必要があります。これには、例 4-1 に示すように、シェルスクリプトの ttrsh を使用します。

例 4-1 ToolTalk 環境変数の伝播

```
#!/bin/sh
# Runs a command remotely in background, by pointing stdout and stderr
# at /dev/null. By running this through the Bourne shell at the other end,
# we get rid of the rsh and rshd.
```

例 4-1 ToolTalk 環境変数の伝播 (続き)

```
#set -x
user=
debug=
HOST=${HOST-`hostname`}
if [ "$1" = "-debug" ]; then
    debug=1
    shift
fi
if [ $# -lt 2 -o "$1" = "-h" -o "$1" = "-help" ];
then
    echo "Usage: ttrsh [-debug] remotehost [-l username] \  
remotecommand"
    echo "Usage: ttrsh [-h | -help]"
    exit 1
else
    host=$1
    shift
    if test "$1" = "-l" ; then
        shift
        user=$1
        shift
    fi
fi
xhostname=`expr "$DISPLAY" : "\([^:]*\)".*"`
xscreen=`expr "$DISPLAY" : "[:]*\(.*)"`
if test x$xscreen = x; then
    xscreen=":0.0"
fi
if test x$xhostname = x -o x$xhostname = x"unix";
then
    DISPLAY=$HOST$xscreen
fi
if [ "$user" = "" ]; then
    userOption=""
else
    userOption="-l $user"
fi
if [ $debug ]; then
    outputRedirect=
else
    outputRedirect='> /dev/null 2>&1 &'
fi
(
    echo "OPENWINHOME=$OPENWINHOME;export OPENWINHOME;\
    TT_SESSION=$TT_SESSION;export TT_SESSION;\
    TT_TOKEN=$TT_TOKEN;export TT_TOKEN;TT_FILE=$TT_FILE;\
    export TT_FILE;DISPLAY=$DISPLAY;export DISPLAY;($*)" \  
$outputRedirect | rsh $host $userOption /bin/sh &
) &
```

コンテキストスロットの使用による環境変数の作成

メッセージコンテキストは、ToolTalk サービスがアプリケーションを起動する際に特別な意味を持ちます。コンテキストスロット名がドル記号 (\$) で始まる場合、ToolTalk サービスはこの値を環境変数として解釈します。たとえば、次のようにコンテキストスロット \$CON1 の値が使用された場合です。

```
start "my_application $CON1"
```

ToolTalk データベースサーバーのインストール

ToolTalk データベースサーバーは、次の 3 種類の情報を格納するのに使用します。

1. ToolTalk オブジェクト仕様
2. `tt_file_join` 呼び出しでファイルを結合したクライアントを持つセッションの ToolTalk セッション ID
3. メッセージの処置が `TT_QUEUED` であり、処理できるハンドラがまだ起動されていないために待ち行列に入れられている、ファイルを配信範囲とするメッセージ

さらに、ToolTalk データベースサーバーは、ToolTalk ファイル名マッピング API コール (`tt_host_file_netfile()` と `tt_host_netfile_file()`) の照会に回答します。

注 - ToolTalk データベースサーバーは、セッション中のファイルを配信範囲とするメッセージは格納しません。

ToolTalk サービスは、ToolTalk オブジェクトが入ったファイルまたは ToolTalk メッセージの対象であるファイルを格納している各マシンで、データベースサーバーを実行することが必要です。アプリケーションがデータベースサーバーを含まないマシンのファイルを参照しようとした場合は、次のメッセージと同様のエラーが表示されます。

```
% Error: Tool Talk database server on integral is not running: tcp
```

`integral` はホスト名で、`tcp` はアプリケーションプロトコルです。このエラーメッセージは、接続が異常終了したことを示します。接続の異常終了は、ネットワーク上の問題で生じることもあります。

rpc.ttdserverd がシステムにインストールされていることの確認

SUNWtltk と SUNWdtcor パッケージに ToolTalk メッセージに出てくるファイルが含まれている場合は、全マシンにそれらがインストールされている必要があります。rpc.ttdbserverd がシステムにインストールされているかどうかは、次の手順で確認します。

1. システムにログインします。
2. pkginfo(1) を使用して、**SUNWtltk** と **SUNWdtcor** パッケージがインストールされていることを確認します。



注意 – 下記の /etc/inetd.conf は、SUNWdtcor Solaris パッケージによってインストールされます。システムに SUNWtltk がインストールされていないときは、SUNWtltk のインストール前に SUNWdtcor があることを確認してください。Solaris 1.0 (SunOS 4.0/4.1 またはその互換) オペレーティングシステムを実行しているマシンに、Solaris 7 (SunOS 5.7 またはその互換) サーバーをコピーしないでください。

3. /etc/inetd.conf ファイルに次の行があることを確認してください。

```
100083/1 tli rpc/tcp wait root /usr/openwin/bin/rpc.ttdbserverd
rpc.ttdbserverd がないことがわかって、pkgadd(1M) を使って SUNWtltk と
SUNWdtcor パッケージを追加すればインストールできます。これらのパッケージを追加すれば、inetd で下記の構成ファイルを再読み取りできます。

# ps -ef | grep inetd
# kill -HUP inetd-pid
```

注 – inetd-pid は ps リストからのものです。

rpc.ttdbserverd がシステム上で動作していることを確認

特定のシステムで ToolTalk サーバーが実際に動作しているかどうかを知るには、次のように rpcinfo(1M) コマンドで確認できます。

```
% rpcinfo -T tcp -t <hostname> 100083
program 100083 version 1 ready and waiting
%
```

注 - *hostname* は `hostname(1)` からのものです。

Solaris CD-ROM

Solaris CD-ROM から ToolTalk ソフトウェアパッケージをインストールするには、`pkgadd` コマンドを使用します。ToolTalk ソフトウェアは `SUNWtltk`、開発者用パッケージ名は `SUNWtltkd`、マニュアルページのパッケージ名は `SUNWtltkm` です。

新しい ToolTalk データベースサーバーの実行

新しいバージョンの ToolTalk データベースサーバーが一度でもマシン上で実行されると、ToolTalk データベースサーバーの以前のバージョンには戻すことはできません。ToolTalk データベースサーバーの以前のバージョンを実行しようとする、次のエラーメッセージが表示されます。

```
rpc.ttdbserverd[pid #]: rpc.ttdbserverd version (1.0.x)
does not match the version (1.1) of the database tables.
Please install an rpc.ttdbserverd version 1.1 (or greater).
```

ToolTalk データベースサーバーのリダイレクト

データベースのホストマシンとファイルシステムのパーティションは、両方ともリダイレクトできます。

- データベースのホストマシンのリダイレクトによって、ToolTalk クライアントは、ToolTalk データベースサーバーを実行していないマシンから ToolTalk データにアクセスすることが物理的に可能になります。
- ファイルシステムのパーティションをリダイレクトすると、ToolTalk データベースは、異なるファイルシステムのパーティションに物理的にアクセスすることによって、ToolTalk データを読み取り専用ファイルシステムのパーティション (たとえば CD-ROM など) に対して論理的に読み取ったり書き込んだりできます。

ファイルシステムのパーティションのリダイレクトは、システム管理者が、デフォルトのローカルパーティション1つごとに1つの ToolTalk データベースを常駐させる代わりに、1つのパーティションにすべての ToolTalk データベースを常駐させたいときにも実行できます。

ホストマシンのリダイレクト

ToolTalk クライアントは、データベースのホストマシンをリダイレクトする際に、ToolTalk データベースサーバーを実行していないマシンから ToolTalk データにアクセスすることが物理的に可能になります。ホストマシンをリダイレクトするには、ToolTalk クライアントがアクセスするマシンのホスト名をマップする必要があります。データベース照会を行う ToolTalk クライアントを実行しているマシン上で、次の手順を実行します。

1. `hostname_map` ファイルを作成します。

たとえば、次のように作成します。

```
# Map first host machine
oldhostname1 newhostname1
```

```
# Map second host machine
oldhostname2 newhostname2
```

`oldhostname` は ToolTalk クライアントがアクセスしなければならないマシン名、`newhostname` は ToolTalk データベースサーバーを実行しているマシン名です。

2. **ToolTalk** 型データベースが格納されているのと同じ場所にファイルを格納します。マップファイルは、ToolTalk 型データベースと同じ優先順位を持ちます (詳細は、`tt_type_comp`(1) のマニュアルページを参照してください。)

注 - `TT_HOSTNAME_MAP` 環境変数で定義したファイルは、ユーザーデータベース内のマップより優先順位が高くなります。

マップファイルは、クライアントが `tt_open` 呼び出しを行う際に、ToolTalk クライアントが読み取ります。ホストリダイレクションの詳細は、`hostname_map`(4) のマニュアルページを参照してください。

ファイルシステムのパーティションのリダイレクト

ファイルシステムのパーティションをリダイレクトする際に、ToolTalk データベースは、異なるファイルシステムのパーティションに物理的にアクセスすることによって、ToolTalk データを読み取り専用ファイルシステムのパーティションに対して

論理的に読み取ったり書き込んだりできます。ファイルパーティションをリダイレクトするには、ToolTalk データベースが書き込みを行うパーティションをマップする必要があります。ToolTalk データベースサーバーを実行しているマシン上で、次の手順を実行します。

1. `partition_map` ファイルを作成します。
たとえば、次のように作成します。

```
# Map first partition
/cdrom /usr

# Map second partition
/sr0/export/home /export/home
```

これによって、読み取り専用パーティション `/cdrom` を読み取り書き込みパーティション `/usr` にマッピングし、読み取り専用パーティション `/sr0/export/home` を読み取り書き込みパーティション `/export/home` にマッピングします。

2. **ToolTalk** 型データベースが格納されているのと同じ場所に、マッピングファイルを格納します。

注 - `TT_PARTITION_MAP` 環境変数で定義したファイルパーティションは、このマッピングファイルで定義したファイルパーティションより優先順位が高くなります。

マッピングファイルは、ToolTalk データベースサーバーの起動時、または ToolTalk データベースサーバーが `USR2` シグナルを受信するときに読み取られます。パーティションのリダイレクションに関する詳細は、`partition_map(4)` のマニュアルページを参照してください。

第 5 章

アプリケーション情報の管理

ToolTalk メッセージを受信したいアプリケーションは、ToolTalk サービスに対して、どの種類のメッセージを受信したいかという情報を提供します。この情報(メッセージパターンと言います)は、実行時にアプリケーションによって動的に提供されます。プロセス型 (ptype) とオブジェクト型 (otype) のファイルから提供される場合があります。

アプリケーション型のインストール

アプリケーション型のインストールは、頻繁に行う必要はありません。型情報をインストールする必要があるのは、新しい型が作成されたときか、アプリケーションにエラーが発生したときだけです。ptype ファイルと otype ファイルは、インストール時に ToolTalk 型コンパイラを通して実行されます。tt_type_comp は、この情報を ToolTalk 型データベースに併合します。その後アプリケーションは、ToolTalk サービスにデータベース内の型情報を読み取るように指示します。

アプリケーションの ptype ファイルと otype ファイルをインストールするには、次の手順に従います。

1. 型ファイル上で tt_type_comp を実行します。

```
% tt_type_comp your-file
```

tt_type_comp は、cpp を介して your-file を実行し、型定義をコンパイルして、ToolTalk 型テーブルにその情報を併合します。表 5-1 は XDR 形式のテーブルの場所、表 5-2 は分類機構形式のテーブルの場所を示します。

注 - 分類機構インタフェースは、互換性のためだけのものです。デフォルトは XDR です。

表 5-1 XDR 形式の ToolTalk 型テーブル

データベース	XDR テーブルの使用
ユーザー (user)	~/.tt/types.xdr
システム (system)	/etc/tt/types.xdr
デスクトップ (desktop)	/usr/dt/appconfig/tttypes/types.xdr
ネットワーク (network)	\$OPENWINHOME/etc/tt/types.xdr

表 5-2 分類機構形式の ToolTalk 型テーブル

データベース	分類機構テーブルの使用
ユーザー (user)	~/.cetables/cetables
システム (system)	/etc/cetables/cetables
ネットワーク (network)	\$OPENWINHOME/lib/cetables/cetables

XDR データベースには、4 種類あります。\$TTPATH 環境変数によって、どの 3 つを使用するかを決めます。\$TTPATH エントリのフォーマットと優先順位に関する詳細は、tt_type_comp (1) のマニュアルページを参照してください。

デフォルトでは、tt_type_comp はユーザーデータベースを使用します。他のデータベースを指定する場合は、-d オプションを使用します。たとえば、次のように入力します。

```
% tt_type_comp -d user|system|[network|desktop] your_file
```

注 -ptype ファイルまたは otype ファイル上で tt_type_comp を実行する場合、tt_type_comp は、データを ToolTalk 型データベースの形式にする前に、まずファイル上の cpp を実行し、次に構文を検査します。構文エラーが検出された場合、cpp ファイルの行番号を示すメッセージが表示されます。行検索を行うには、**cpp -P source-file temp-file** と入力します。これによって temp-file が表示され、tt_type_comp が報告する行のエラーを検出できます。

2. ttsession は、ToolTalk 型データベースを自動的に再読み取りします。

ttsession に ToolTalk 型データベースを再読み取りさせる方法については、62 ページの「ToolTalk サービスの更新」を参照してください。

ToolTalk 型情報の検査

指定された ToolTalk 型データベースのすべての型情報を検査することも、ptype 情報だけまたは otype 情報だけを検査することもできます。検査したいデータベースを指定するには、`-d` オプションを使用し、`user`、`system`、`network` のいずれかを入力して、検査したいデータベースを示します。`-d` オプションを使用しない場合、`tt_type_comp` はデフォルトのユーザーデータベースを使用します。

- ToolTalk 型データベース内のすべての ToolTalk 型情報を検査するには、次のように入力します。

```
% tt_type_comp -d user|system|network -p
```

型情報は、ソース形式で出力されます。

- ToolTalk 型データベース内のすべての ptype を表示するには、次のように入力します。

```
% tt_type_comp -d user|system|network -P
```

ptype の名前は、ソース形式で出力されます。

- ToolTalk 型データベース内のすべての otype を表示するには、次のように入力します。

```
% tt_type_comp -d user|system|network -O
```

otype の名前は、ソース形式で出力されます。

ToolTalk 型情報の削除

ToolTalk 型データベースから、ptype 情報と otype 情報を削除できます。

- `tt_type_comp` を使用して、型情報を削除します。次のように入力します。

```
% tt_type_comp -d user|system|network -r type
```

たとえば、サンプルアプリケーションの ToolTalk 型ネットワークデータベースから `EditDemo` という ptype を削除するには、次のように入力します。

```
% tt_type_comp -d network -r EditDemo
```

型情報の削除後は、実行中の `ttsession` に ToolTalk 型データベースを再読み取りさせて、ToolTalk サービスを最新の状態にします。詳細は、62 ページの「ToolTalk サービスの更新」を参照してください。

ToolTalk サービスの更新

tt_type_comp(1) で ToolTalk タイプのデータベースに変更を加える場合、ToolTalk サービスは、タイプファイルを再読み取りするように自動的に指示されます。すでに実行している ToolTalk セッションに対して、明示的にデータベースを再読み取りさせたいときは、ttsession プロセスに SIGUSR2 を送信します。

▼ ttsession プロセスに SIGUSR2 を送信するには

1. ps コマンドを入力して、ttsession プロセスのプロセス識別子 (pid) を検索します。

```
% ps -ef | grep ttsession
```

2. kill コマンドを入力して、ttsession に SIGUSR2 シグナルを送信します。

```
% kill -USR2 ttsession_pid
```

プロセス型エラー

アプリケーションがこのエラーを報告する場合は、次の状態のどちらか、または両方が存在すると考えられます。

```
Application is not an installed ptype.
```

1. ToolTalk サービスは、ToolTalk 型データベース内で最近更新された型情報を再読み取りする指示を、アプリケーションから受けていません。ToolTalk サービスに ToolTalk 型データベースの型情報を再読み取りさせる手順については、62 ページの「ToolTalk サービスの更新」を参照してください。
2. アプリケーションの ptype と otype が、ToolTalk 型データベースにコンパイルおよび併合されていません。型情報のコンパイルと併合の手順については、59 ページの「アプリケーション型のインストール」を参照してください。

ttsnoop の使用によるメッセージおよびパターンのデバッグ

ttsnoop は、カスタム構築の ToolTalk メッセージを作成して送信するためのツールです。ttsnoop を、ToolTalk メッセージの 1 つまたはすべてを選択的に監視するツールとして使用することもできます。ttsnoop プログラムは、ディレクトリ /usr/dt/bin/ttsnoop にあります。プログラムを起動するには、コマンド行に次のコマンドを入力します。

```
% /usr/dt/bin/ttsnoop [ -t ]
```

-t オプションは、特定のパターンまたはメッセージの構築に使用する ToolTalk API 呼び出しを表示します。図 5-1 は、ttsnoop を起動する際に表示されるウィンドウを示します。

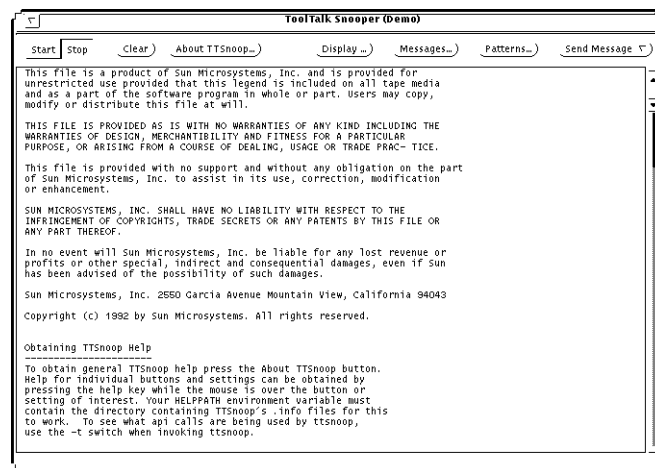


図 5-1 ttsnoop 表示ウィンドウ

- **Start** ボタン
メッセージの受信を起動する場合に押します。ttsnoop は、登録するパターンに一致する着信メッセージを表示します。
- **Stop** ボタン
メッセージの受信を停止する場合に押します。
- **Clear** ボタン
ウィンドウをクリアする場合に押します。
- **About TTSnoop** ボタン
ttsnoop についてのヘルプを見る場合に押します。

注 – 各ボタンおよび設定についてのヘルプを見るには、そのボタンまたは設定の上にマウスを置いて、キーボードの F1 キーまたはヘルプキーを押します。
TTSnoop.info というファイルの入ったディレクトリを、環境変数 HELPPATH に入れておく必要があります。

- **Display** ボタン
チェックボックスのパネルを表示して、ttsnoop ディスプレイのサブウィンドウ上で特定の ToolTalk メッセージ構成要素を強調表示する場合に押します。
- **Messages** ボタン
ToolTalk メッセージの作成、格納、および送信を可能にするパネルを表示する場合に押します。
- **Patterns** ボタン
ToolTalk パターンの構成と登録を可能にするパネルを表示する場合に押します。
- **Send Messages** ボタン
Messages ウィンドウを使用して格納したメッセージを送信する場合に押します。

メッセージの構成と送信

初期表示ウィンドウで Messages ボタンを押すと、図 5-2 に示すポップアップパネルが表示されます。

図 5-2 ポップアップメッセージパネル

- **Add Message** ボタン

現在のメッセージ設定を格納する場合に押します。メッセージが格納されてしまうと、初期表示ウィンドウ上の Send Message ボタンを使用して、これらのメッセージの再呼び出しと送信ができます。

- **Edit Contexts** ボタン

送信するメッセージコンテキストを追加、変更、または削除する場合に押します。図 5-3 で示すポップアップウィンドウによって、メッセージとともに送信するコンテキストを編集できます。

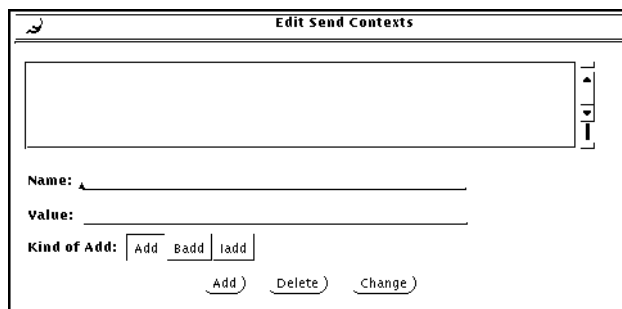


図 5-3 「Edit Send Contexts」ウィンドウ

- **Send Message** ボタン

新規に作成したメッセージを送信する場合に押します。

注 - この Send Message ボタンは、メインメニュー上の Send Message ボタンと同じ機能を実行します。

パターンの構成と登録

初期表示ウィンドウの Patterns ボタンを押すと、図 5-4 に示すポップアップパネルが表示されます。

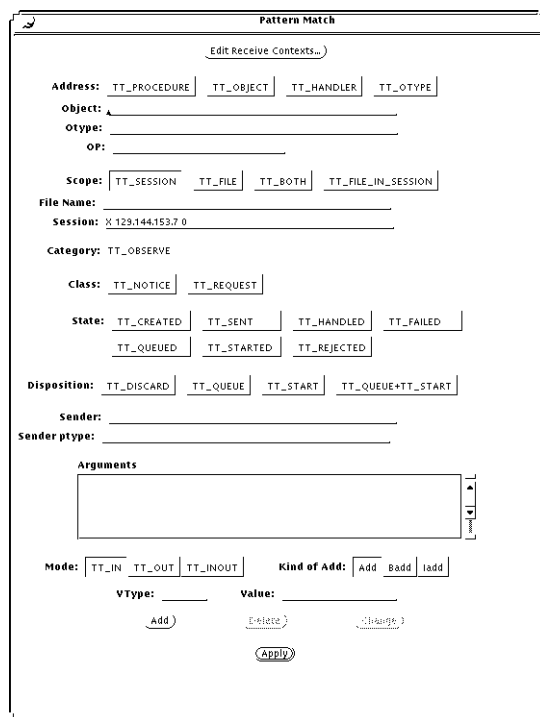


図 5-4 ポップアップパターンパネル

パターンを登録するには、Apply ボタンを押します。パターンが登録されてしまうと、デバッグツールとして ttsnoop を使用して、他のアプリケーションがどのようなメッセージを送信しているかを監視できます。

- Edit Receive Contexts ボタン

パターン内の受信メッセージコンテキストを追加、変更、または削除する場合に押します。図 5-5 に示すポップアップウィンドウによって、パターンとともに登録するコンテキストを編集できます。

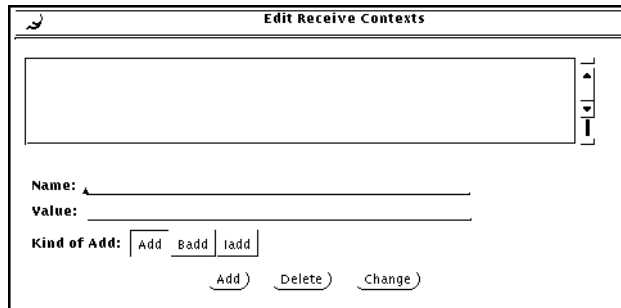


図 5-5 「Edit Receive Contexts」ウィンドウ

メッセージ構成要素の表示

初期表示ウィンドウ上の Display ボタンを押すと、図 5-6 に示すチェックボックスパネルが表示されます。

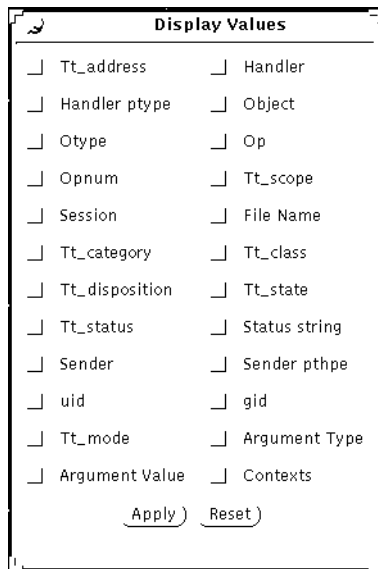


図 5-6 メッセージ構成要素の値を表示するチェックボックス

チェックボックスを選択すると、指定された ToolTalk メッセージの構成要素は、表示したメッセージ上で、表示されたメッセージ構成要素の左に矢印(--->)を付けて示されます。図 5-7 は、表示されたメッセージ構成要素を示します。

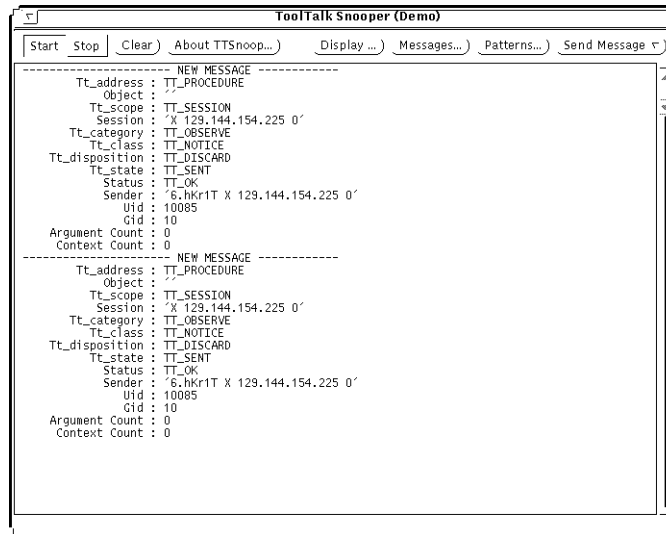


図 5-7 メッセージ構成要素の表示

作成済みメッセージの送信

初期表示ウィンドウ上の Send Message ボタンを押すと、Messages ポップアップを使用して作成または格納したメッセージを送信できます。

メッセージの受信

初期表示ウィンドウ上の Start ボタンを押すと、ttsnoop は登録したパターンに一致する着信メッセージを表示します。図 5-8 は、表示された着信メッセージの例です。

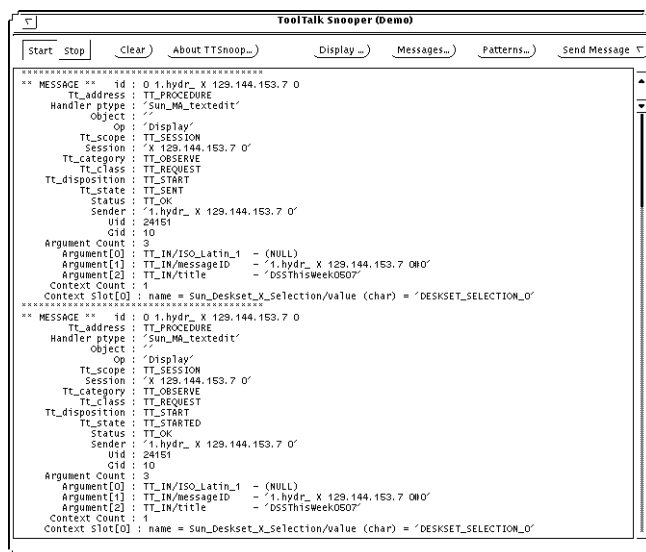


図 5-8 着信メッセージの表示

メッセージ受信の停止

初期表示ウィンドウ上の Stop ボタンを押すと、ttsnoop はメッセージの受信を停止します。

第 6 章

ToolTalk メッセージで参照するファイルとオブジェクトの保守

ToolTalk メッセージは、対象とするファイルや ToolTalk オブジェクトを参照できます。ToolTalk サービスは、ファイルとオブジェクトに関する情報を保持しており、ファイルまたはオブジェクトに変更があった場合は、通知を受ける必要があります。

ToolTalk サービスは、ファイルの移動、コピー、および削除用の、ラップされたシェルコマンドを提供します。これらのコマンドは、ToolTalk サービスに変更を通知します。

ToolTalk の拡張シェルコマンド

表 6-1 に示す ToolTalk の拡張シェルコマンドは、それらが関連付けられている標準シェルコマンドを最初に呼び出し (たとえば、`ttmv` は `mv` を呼び出します)、その後ファイルの変更に伴い ToolTalk サービスを更新します。ToolTalk の拡張シェルコマンドは、ToolTalk オブジェクトを含むファイルを操作する際に必要です。

表 6-1 ToolTalk の拡張シェルコマンド

コマンド	定義	構文
<code>ttcp</code>	オブジェクトを含むファイルをコピーする	<code>ttcp source-file destination-file</code>
<code>ttmv</code>	オブジェクトを含むファイル名を変更する	<code>ttmv old new</code>
<code>ttrm</code>	オブジェクトを含むファイルを削除する	<code>ttrm file</code>

表 6-1 ToolTalk の拡張シェルコマンド (続き)

コマンド	定義	構文
ttrmdir	<p>ToolTalk オブジェクトと関連付けられた空ディレクトリを削除する。</p> <p>このコマンドは、たとえば、ファイルに配信範囲指定したメッセージの中にディレクトリが記述されている場合に、ディレクトリのオブジェクト仕様を作成するためにも使用する。オブジェクト仕様を作成する際には、ファイルまたはディレクトリのパス名を指定する</p>	ttrmdir <i>directory</i>
tttar	ToolTalk オブジェクトを含むファイルのアーカイブとアーカイブの解除を行う	tttar c t x <i>pathname1 pathname2</i>

ToolTalk の拡張シェルコマンドは、標準シェルコマンドを呼び出すことによって実行できます。そのためには、シェルの起動ファイルで ToolTalk の拡張シェルコマンドに別名を付け、拡張コマンドが標準シェルコマンドとして表示されるようにします。

```
# ToolTalk-aware shell commands in .cshrc
alias mv      ttmv
alias cp      ttcp
alias rm      ttrm
alias rmdir   ttrmdir
alias tar     tttar
```

ToolTalk データベースの保守と更新

ToolTalk の拡張シェルコマンドを使ってコピー、移動、または削除していない場合は、ToolTalk データベース内のファイルとオブジェクトに関する情報は古くなっています。たとえば、ToolTalk オブジェクトを含むファイル「old_file」は、標準の rm コマンドを使ってファイルシステムから削除できます。しかし、標準シェルコマンドは「old_file」の削除を ToolTalk サービスに通知しないので、ファイルと個々のオブジェクトに関する情報は、ToolTalk データベースに残ります。

ToolTalk データベースからファイルとオブジェクトの情報を削除するには、次のコマンドを使用します。

```
ttrm -I old_file
```

データベースの表示、検査、および修復

ToolTalk データベースを表示、検査、または修復するには、ToolTalk データベースユーティリティの `ttdbck` を使用します。`ttdbck` ユーティリティは、次の操作にも使用します。

- 特定の `otype`、たとえば取り除いた `otype` のすべての ToolTalk オブジェクトの削除
- あるファイルから他のファイルへの特定の ToolTalk オブジェクトの移動
- 実在しないファイルを参照するすべての ToolTalk オブジェクトの検索

注 - 通常、ToolTalk データベースはスーパーユーザーとしてだけアクセス可能です。したがって、`ttdbck` ユーティリティは、通常はスーパーユーザーとして実行します。

第 7 章

ToolTalk セッションの結合

この章では、ToolTalk セッションを結合する方法を説明します。同様に、ToolTalk サービスから渡される値の記憶を管理する方法、および ToolTalk サービスが返すエラーを処理する方法を説明します。

アプリケーションが ToolTalk サービスを使用するためには、ToolTalk API ライブラリの ToolTalk 関数を呼び出します。ToolTalk サービスを使用するためにアプリケーションを変更するには、最初に ToolTalk API ヘッダーファイルをプログラムに組み込まなければなりません。ToolTalk サービスを初期化し、セッションを結合した後、ファイルを結合し、追加のユーザーセッションを結合できます。処理を終了する準備ができたなら、メッセージパターンの登録を解除し、ToolTalk セッションを終了します。

ToolTalk API ヘッダーファイルの組み込み

ToolTalk サービスを使用するためにアプリケーションを変更するには、最初に ToolTalk API ヘッダーファイルの `tt_c.h` をプログラムに組み込まなければなりません。このファイルは、`/usr/dt/include/Tt` ディレクトリにあります。

次のコード例は、このファイルをプログラムに組み込む方法を示しています。

```
#include <stdio.h>
#include <sys/param.h>
#include <sys/types.h>
#include <Tt/tt_c.h>
```

ToolTalk サービスへの登録

ToolTalk セッションを結合するためには、ToolTalk サービスにユーザープロセスを登録しなければなりません。アプリケーションを起動した ToolTalk セッション(「初期セッション」)に登録するか、他のセッションを検索して登録できます。

ToolTalk サービスに登録しなければならない ToolTalk 関数を表 7-1 に示します。

表 7-1 ToolTalk サービスへの登録

返される型	ToolTalk 関数	機能
char *	tt_open(void)	プロセス識別子
int	tt_fd(void)	ファイル記述子
char *	tt_X_session(const char *xdisplay)	指定した X ディスプレイサーバーのセッション識別子を返す
Tt_status	tt_default_session_set(const char *sessid)	tt_open の接続先セッションを設定する

初期セッションへの登録

アプリケーションは、プロセスを初期化して初期 ToolTalk セッションに登録するために、プロセス識別子 (procid) を取得する必要があります。その後、新規に初期化した ToolTalk プロセスに対応するファイル記述子 (fd) を取得できます。

次のコード例では、まずサンプルプログラムを初期化し、ToolTalk サービスに登録します。その後、対応するファイル記述子を取得します。

```
int ttfd;
char *my_procid;

/*
 * Initialize ToolTalk, using the initial default session
 */

my_procid = tt_open();

/*
 * obtain the file descriptor that will become active whenever
 * ToolTalk has a message for this process.
 */
```

```
ttfd = tt_fd();
```

tt_open はプロセスの procid を返し、それをデフォルトの procid として設定します。tt_fd は、現在の procid のファイル記述子を返します。このファイル記述子は、アプリケーションにメッセージが到着すると使用可能になります。



注意 – アプリケーションは他の tt_呼び出しをする前に、tt_open を呼び出さなければなりません。そうしないと、エラーが発生する場合があります。ただし、いくつかの例外があります。tt_default_session_set と tt_X_session は、tt_open の前に呼び出して、接続するセッションを制御できます。ToolTalk をマルチスレッド環境で使用する場合は、tt_feature_required と tt_feature_enabled を呼び出すことができます。ToolTalk ファイル名マッピング API コールである tt_file_netfile、tt_netfile_file、tt_host_file_netfile、および tt_host_netfile_file を、tt_open を呼び出すことなく呼び出すことができます。

ToolTalk サービスに対する最初の呼び出しが tt_open である場合、デフォルトのセッションとして初期セッションが設定されます。デフォルトのセッション識別子 (sessid) は、ToolTalk メッセージの配信にとって重要です。アプリケーションがセッションのメッセージ属性を明示的に設定しない場合、ToolTalk サービスは、デフォルトの sessid を自動的に設定します。メッセージを TT_SESSION に配信範囲と指定すると、メッセージは、この型のメッセージを処理対象として登録したデフォルトのセッション内のすべてのアプリケーションに配信されます。

指定セッションへの登録

プログラムは、初期セッション以外のセッションに登録する場合には、そのセッション名を検索し、新しいセッションをデフォルトとして設定し、ToolTalk サービスに登録しなければなりません。

次のコード例は、somehost:0 と命名された X セッションを結合する方法を示しています。このセッションは、初期セッションではありません。

```
char *my_session;
char *my_procid;

my_session = tt_X_session("somehost:0");
tt_default_session_set(my_session);
my_procid = tt_open();
ttfd = tt_fd();
```

注 – 必要な呼び出しは、指定されている順番で行わなければなりません。

1. tt_X_session();

この呼び出しは、Xディスプレイサーバーに関連付けられているセッション名を検索します。tt_X_session() は、引数 char *xdisplay_name をとります。

xdisplay_name は、Xディスプレイサーバー名です (この例では、somehost:0 です)。

2. tt_default_session_set();
この呼び出しは、新しいセッションをデフォルトのセッションとして設定します。
3. tt_open();
この呼び出しは、プロセスの procid を返し、それをデフォルトの procid として設定します。
4. tt_fd();
この呼び出しは、現在の procid のファイル記述子を返します。

複数のセッションへの登録

メッセージを複数のセッション間で送受信したい場合もあります。複数のセッションに登録する場合、プログラムは接続したいセッションの識別子を検索し、新しいセッションを設定し、ToolTalk サービスに登録しなければなりません。

次のコード例は、procid を sessid1 に、procid2 を sessid2 に接続する方法を示しています。

```
tt_default_session_set(sessid1);  
my_procid1 = tt_open();  
tt_default_session_set(sessid2);  
my_procid2 = tt_open();  
tt_fd2 = tt_fd();
```

こうしておくで、tt_default_procid_set() を使用して、セッション間を切り替えることができます。

メッセージ受信の設定

アプリケーションが他のアプリケーションからのメッセージを受信するためには、着信メッセージの有無を監視するプロセスを設定しなければなりません。メッセージがアプリケーションに到着すると、ファイル記述子が使用可能になります。ファイル記述子が使用可能になったことをアプリケーションに通知するコードは、アプリケーションの構成によって異なります。

たとえば、xv_main_loop または notify_start 呼び出しによって XView 通知子を使用するプログラムは、ファイル記述子が使用可能になったときにコールバック関数を呼び出すことができます。次のコード例は、メッセージオブジェクトのハンドルをパラメタとして notify_set_input_func を呼び出しています。

```

/*
 * Arrange for XView to call receive_tt_message when the
 * ToolTalk file descriptor becomes active.
 */
notify_set_input_func(base_frame,
                      (Notify_func)receive_tt_message,
                      ttfd);

```

表 7-2 は、各種のウィンドウツールキットと着信メッセージの有無を監視する呼び出しを示したものです。

表 7-2 着信メッセージの有無を監視するモード

ウィンドウツールキット	使用されるコード
XView	notify_set_input_func()
X Window System Xt (イントリンシクス)	XtAddInput() または XtAddAppInput()
他のツールキット (select(2) または poll(2) のシステムコールに基づいて構成された Xlib など)	tt_fd() が返すファイル記述子 注 - ファイル記述子が使用可能になってから select 呼び出しを終了した場合は、tt_message_receive() を使用して、着信メッセージのハンドルを取得します。

同一プロセス内でのメッセージの送受信

受信側は、要求された操作を完了すると、通常そのメッセージを削除します。しかし、ToolTalk サービスでは受信側にも要求側にも同じメッセージ ID を使用するため、同じプロセス内で送受信を行うと、要求側にあるメッセージも削除されてしまいます。

この対処方法の 1 つは、メッセージに参照カウントを付加することです。そのためには、tt_message_user[_set]() 関数を使用します。

もう 1 つの対処方法は、送信側が現在の procid でない場合だけ、受信側のメッセージを破棄することです。次に例を示します。

```

Tt_callback_action
my_pattern_callback(Tt_message m, Tt_pattern p)
{
    /* normal message processing goes here */

    if (0 != strcmp(tt_message_sender(m), tt_default_procid())) {
        tt_message_destroy(m);
    }
    return TT_CALLBACK_PROCESSED;
}

```

ネットワーク環境でのメッセージの送受信

ToolTalk サービスをネットワーク環境で使用できます。たとえば、別のマシン上のツールを実行したり、別のマシン上で動作するセッションを結合したりできます。そのためには `ttsession` を起動する際に、`-c` または `-p` オプションを指定してください。

- `-c` オプションを指定すると指定したプログラムが起動され、そのプログラムの `TT_SESSION` 環境変数に正しいセッション ID が設定されます。たとえば、次のコマンドを入力します。

```
ttsession -c dtterm
```

`cmdtool` の `TT_SESSION` 環境変数が定義されるため、その値を `$TT_SESSION` 環境変数に設定して実行する ToolTalk クライアントはすべて、この `ttsession` が所有するセッションに結合されます。

- `-p` オプションを指定すると、セッション ID が標準出力に出力されます。その後 `ttsession` は、バックグラウンドにセッションを生成し、そのセッションを起動します。

セッションを結合する際、アプリケーションは `tt_open` 関数を呼び出す前に、`tt_default_session_set` でセッション ID を設定するか、`TT_SESSION` 環境変数にセッション ID を設定しなければなりません。`tt_open` は `TT_SESSION` 環境変数を確認し、(値が設定されていれば) そのセッションを結合します。

ToolTalk サービスからの登録解除

ToolTalk サービスおよび他の ToolTalk セッション参加者との対話を停止したい場合は、アプリケーションを終了する前に、プロセスを「登録解除」しなければなりません。

```
/*
 * Before leaving, allow ToolTalk to clean up.
 */
tt_close();

exit(0);
}
```

`tt_close` は `Tt_status` を返し、現在のデフォルトの `procid` を閉じます。

マルチスレッド環境における ToolTalk の使用

ここでは、マルチスレッド環境における ToolTalk の使用方法について説明します。

初期化

マルチスレッドクライアントで ToolTalk ライブラリを使用するには、次のような初期化コールを必要とします。

```
tt_feature_required(TT_FEATURE_MULTITHREADED);
```

このコールは、他のどの ToolTalk コールよりも前に呼び出しておく必要があります。他の ToolTalk API コール後に `tt_feature_required(TT_FEATURE_MULTITHREADED)` を呼び出そうとすると、`TT_ERR_TOOLATE` エラーとなります。

ToolTalk を使用するライブラリやその他の再利用可能なモジュールは、ToolTalk ライブラリに照会して、呼び出しアプリケーションが ToolTalk のマルチスレッド機能を有効にしているかどうかを確認できます。`tt_feature_enabled()` API コールが、この目的のために追加されました。アプリケーションが初期化されているか知ることができるので、トップレベルのアプリケーションは `tt_feature_enabled()` をほとんど使う必要はありません。

ToolTalk procid とセッション

ToolTalk クライアントが `tt_open()` または `tt_session_join()` を呼び出すと、新規 `procid` またはセッションがそのスレッドのデフォルトとなります。マルチスレッド以外の ToolTalk クライアントのときは、全プロセスが対象でしたが、ここではそうではありません。`tt_open()` または `tt_session_join()` への呼び出しが行われる前のスレッドの `procid` およびセッションのデフォルト値は、最初はスレッドの作成者のものと同様です。`tt_open()` または `tt_session_join()` によるデフォルト値の変更の他に、`tt_thread_procid_set()` と `tt_thread_session_set()` を使うと、以前に作成した他のデフォルト値を変更できます。`procid` とセッションのデフォルト値は、`tt_thread_procid()` と `tt_thread_session()` を使って検索できます。スレッドに特有の `procid` とセッション値は、スレッドに特有の記憶領域を使って管理します。スレッドに値が作成されていない場合は、ToolTalk プロセス全体におけるデフォルト値は代替値です。

注 - `tt_default_procid()` と `tt_thread_procid()` (同様に、`tt_default_session()` と `tt_thread_session()`) によって返される値が、特定の時間においてスレッドによって違う場合があります。これは、`tt_default_procid_set()` と `tt_default_session_set()` が、スレッドのデフォルト値に影響しないためです。影響を受けるのは、ToolTalk 全体のプロセスのデフォルト値だけです。

ToolTalk が有効なアプリケーションでスレッドを使用することは、`procid` とセッションを切り換えるプログラムの自然な実装技術の 1 つです。これらのプログラムを使用して、ある時点でどの ToolTalk `procid` が ToolTalk セッションと関連を持っているかを簡単に判断できます。これは、ToolTalk の以前のバージョンで実行するのは困難でした。これを行うために、`tt_procid_session()` が提供されています。`tt_procid_session()` は、スレッドに依存しているわけではありませんが、ToolTalk でスレッドを使用するアプリケーションには有効です。

ToolTalk 記憶領域

`tt_mark()` と `tt_release()` は、スレッド 1 つごとに割り当てている記憶領域に影響を与えますが、プロセス単位で割り当てている領域には影響ありません。したがって、`tt_mark()` を使って別のスレッドでマークを付けられた記憶領域を 1 つのスレッドが `tt_release()` を使って解放することはできません。

共通の問題

1 つのスレッドでメッセージを送信し、もう 1 つのスレッドでメッセージを処理する方法は、よく使われる技術です。ただし、別のスレッドがメッセージの内容を検査または処理している最中に `tt_message_destroy()` を使ってメッセージを破棄する場合には注意が必要です。受信スレッドが、別のスレッドが解放した記憶領域にアクセスしようとする、プログラムクラッシュを引き起こします。これは、マルチスレッドにおけるアプリケーションの ToolTalk 以外の記憶領域の管理に類似していますが、ToolTalk API を使うと、より簡単になります。

第 8 章

メッセージの送信

この章では、メッセージの経路を指定する方法と ToolTalk メッセージの属性とアルゴリズムについて説明します。メッセージの作成、メッセージ内容の書き込み、要求へのコールバックの付加、およびメッセージの送信方法についても説明します。

ToolTalk サービスによるメッセージのルーティング方法

アプリケーションは、「通知」と「要求」という 2 つのクラスの ToolTalk メッセージを送信できます。通知は情報メッセージであり、アプリケーションがイベントについて連絡する手段です。通知を受信したアプリケーションは、結果を送信側へ返さずにメッセージを吸収します。要求とは、処理を求めるメッセージです。処理の結果がメッセージに記録され、それが応答として送信側に返されます。

通知の送信

通知メッセージは、一方方向で送信されます。この様子を図 8-1 に示します。



図 8-1 通知のルーティング

送信側プロセスは、メッセージを作成し、属性値を書き込んでから送信します。ToolTalk サービスは、メッセージとパターン属性値を照合して、1つのハンドラと、一致するすべてのオブザーバにメッセージを送信します。ファイルに配信範囲指定したメッセージは、セッションの境界を越えて、ファイルを処理対象として宣言しているプロセスに自動転送されます。

要求の送信

要求メッセージを送信する場合、要求は送信側とハンドラの間を往復します。要求メッセージのコピーは、処理の対象としているオブザーバへ一方方向で送信されます。図 8-2 は、要求のルーティングの手順を示しています。

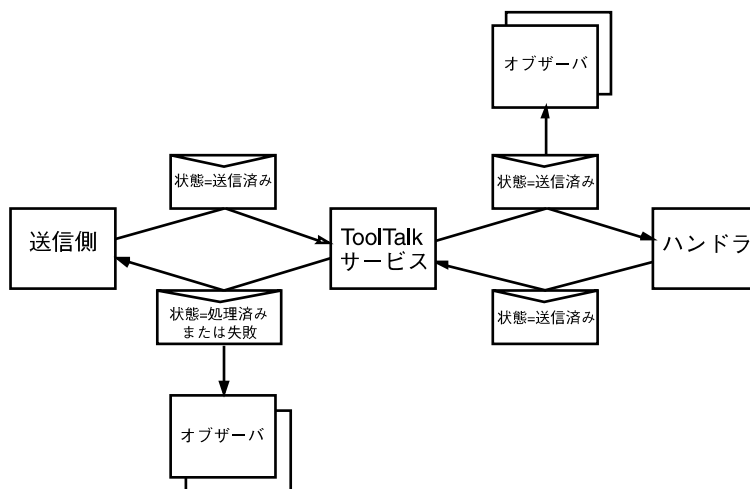


図 8-2 要求のルーティング

ToolTalk サービスは、1つのハンドラだけに要求を送信します。ハンドラは、メッセージに結果を追加して返信します。その他のプロセスは、要求が処理される前後、またはその両方の時点で、要求を監視できます。オブザーバは、要求を送り返さずに要求を吸収します。

オフアの送信

オフアは要求に似たメッセージですが、データを送信しても応答は期待できません。また、メッセージの送信時に、その受信者数をあらかじめ予測できます。また、これらの受信者がそのメッセージを受信、拒絶、または忌避したかを知ることができます。したがって、オフアは通知や要求に比べて、一般性が低いと言えます。

送信したメッセージの状態の変化

送信した要求を追跡できるよう、要求の状態が変化するたびに送信側にメッセージが送られます。状態の変化を知らせるこのメッセージは、パターンを登録していなくても、メッセージのコールバックを指定していなくても送られます。

メッセージの属性

ToolTalk メッセージには、メッセージ情報を格納して配信情報を ToolTalk サービスに提供する属性が含まれています。配信情報は、メッセージを適切な受信側ヘルディングするために使用されます。

ToolTalk メッセージは、アドレス、サブジェクト(「操作」や「引数」など)、および配信情報(「クラス」や「配信範囲」など)の属性を含む単純な構造体です。メッセージの属性を表 8-1 に示します。

表 8-1 ToolTalk メッセージの属性

メッセージの属性	値	機能	使用できる人
引数	引数または結果	操作に使用する引数を指定する。 メッセージが応答である場合、これらの引数は操作の結果を含む	送信側と受信側
クラス	TT_NOTICE, TT_REQUEST, TT_OFFER	受信側が操作を実行する必要があるかどうかを指定する	送信側
ファイル	char *pathname	操作に関連するファイルを指定する。 メッセージの配信範囲がファイルに関係しない場合、このメッセージ属性は無効である	送信側と ToolTalk
オブジェクト	char *objid	操作に関係するオブジェクトを指定する	送信側と ToolTalk
操作	char *opname	実行する操作名を指定する	送信側

表 8-1 ToolTalk メッセージの属性 (続き)

メッセージの属性	値	機能	使用できる人
otype	char *otype	操作に関するオブジェクトの型を指定する	送信側と ToolTalk
アドレス	TT_PROCEDURE, TT_OBJECT, TT_HANDLER, TT_OTYPE	メッセージを送信する場所を指定する	送信側
ハンドラ	char *procid	受信側プロセスを指定する	送信側と ToolTalk
handler_ptype	char *ptype	受信側プロセスの型を指定する	送信側と ToolTalk
処置	TT_DISCARD, TT_QUEUE, TT_START TT_START+TT_QUEUE	実行中のプロセスがメッセージを受信できない場合に取りべき手段を指定する	送信側と ToolTalk
配信範囲	TT_SESSION, TT_FILE, TT_BOTH, TT_FILE_IN_SESSION	セッションまたはファイルに対して登録された内容に基づいて、受信する可能性のあるアプリケーションを指定する	送信側と ToolTalk
sender_ptype	char *ptype	送信側プロセスの型を指定する	送信側と ToolTalk
セッション	char *sessid	送信側プロセスのセッションを指定する	送信側と ToolTalk
状態	int status, char *status_str	メッセージの状態に関する追加情報を指定する	受信側と ToolTalk

アドレス属性

他のアプリケーションにアドレス指定されたメッセージは、特定のプロセス、またはメッセージに一致するパターンを登録してあるプロセスにアドレス指定できます。メッセージをプロセスにアドレス指定するときは、他のアプリケーションのプロセス ID (procid) を知っている必要があります。ただしプロセスは、通常は互いの procid を知りません。ほとんどの場合、送信側はどのプロセスが操作 (要求メッセージ) を実行するのか、またはどのプロセスがイベント (通知メッセージ) を認識するのかを知らなくてもかまいません。

配信範囲属性

通常、ToolTalk サービスを使用して通信するアプリケーションには共通点があります。アプリケーションが同じセッションで実行されていたり、同じファイルまたはデータを対象としていたりします。アプリケーションは、処理の対象を登録するために、ToolTalk サービスにセッションまたはファイル (あるいはその両方) を結合します。ファイル情報とセッション情報は、メッセージパターンが指定された ToolTalk サービスが、メッセージを受信するアプリケーションを判定するために使用します。

注 – 配信範囲属性機能を使用できるのは、NFS ファイルシステムと UFS ファイルシステムだけです。tmpfs ファイルシステムでは、ファイルへの配信範囲は指定できません。

ファイル配信範囲

メッセージをファイルに配信範囲指定した場合は、そのファイルを結合している (かつ残りの属性に一致している) アプリケーションだけがメッセージを受信します。1つのファイルを処理の対象として共有するアプリケーションは、同じセッションで実行している必要はありません。

パターンへの設定

メッセージがファイル配信範囲であることをパターンに指定する際、使用できる型を表 8-2 に示します。

表 8-2 メッセージのファイル配信範囲をパターンに指定する際の型

配信範囲の型	説明
TT_FILE	指定したファイルだけを配信範囲とする。同時にセッション属性も設定すればセッション中のファイルと同様な配信範囲を指定できるが、配信範囲が TT_FILE になっていると、tt_session_join を呼び出してもそのパターンのセッション属性は更新されない
TT_BOTH	処理の対象となるファイルとセッションの「論理和」を配信範囲とする。ファイル、セッション、またはその両方を配信範囲とするメッセージに一致するのは、この配信範囲を指定したパターンだけである
TT_FILE_IN_SESSION	処理の対象となるファイルとセッションの「論理積」を配信範囲とする。ファイルとセッションを同時に配信範囲とするメッセージにだけ一致するのは、この配信範囲を指定したパターンだけである

TT_FILE_IN_SESSION と TT_SESSION の論理和を配信範囲に指定するには、例 8-1 のように両方の配信範囲を同じパターンに追加します。

例 8-1 TT_FILE_IN_SESSION と TT_SESSION の論理和を配信範囲に指定する場合

```
tt_open();

Tt_pattern pat = tt_create_pattern();
tt_pattern_scope_add(pat, TT_FILE_IN_SESSION);
tt_pattern_scope_add(pat, TT_SESSION);
tt_pattern_file_add(pat, file);
tt_pattern_session_add(pat, tt_default_session());
tt_pattern_register(pat);
```

メッセージへの設定

メッセージのファイル配信範囲の型も、パターンと同じです。この型を表 8-3 に示します。

表 8-3 メッセージ用の配信範囲

配信範囲の型	説明
TT_FILE	処理対象のファイルを登録した全クライアントをメッセージの配信範囲とする
TT_BOTH	処理対象のセッション、ファイル、またはセッションとファイルの両方を登録した全クライアントをメッセージの配信範囲とする
TT_FILE_IN_SESSION	処理対象のファイルとセッションを同時に登録した全クライアントをメッセージの配信範囲とする
TT_SESSION + tt_message_file_set()	処理対象のセッションを登録した全クライアントをメッセージの配信範囲とする。パターンが一致するクライアントは、メッセージを受信すると tt_message_file を呼び出して、そのファイル名を入手できる

メッセージの配信範囲が TT_FILE または TT_BOTH の場合は、ToolTalk クライアントライブラリが、そのファイルを処理対象とするクライアントのセッションをデータベースサーバーからすべて探し出し、それらの ToolTalk セッションにメッセージを送信します。ToolTalk セッションは、その後でメッセージとクライアントを照合します。メッセージの送信側は、tt_file_join を明示的に呼び出す必要はありません。

配信範囲が TT_FILE_IN_SESSION または TT_SESSION のメッセージにファイルを指定しても、ToolTalk データベースサーバーには連絡されず、メッセージは指定したセッション配信範囲のクライアントにしか送信されません。

セッション配信範囲

メッセージをセッションに配信範囲指定した場合は、そのセッションに接続しているアプリケーションだけが受信可能側であると見なされます。

例 8-2 セッションの設定

```
Tt_message m= tt_message_create();
tt_message_scope_set(m, TT_SESSION);
tt_message_file_set(m, file);
```

最初の行でメッセージを作成しています。2行目ではメッセージの配信範囲を追加し、3行目ではメッセージの範囲に影響を与えないファイル属性を追加しています。

セッション中のファイル配信範囲

アプリケーションは、メッセージ配信範囲として `TT_FILE_IN_SESSION` を指定することによって、メッセージ配布の範囲をかなり限定できます。指定したファイルとセッションの両方を結合しているアプリケーションだけが受信可能側であると見なされます。

また、アプリケーションは、メッセージのセッションを処理対象として登録しているすべてのクライアントを、`TT_SESSION` と `tt_message_file_set` を指定することによって、メッセージの配信範囲とすることもできます。パターンが一致するクライアントがメッセージを受信した場合、その受信側クライアントは `tt_message_file` を呼び出すことによってファイル名を取得できます。

例 8-3 ファイルの設定

```
Tt_message m= tt_message_create();
tt_message_scope_set(m, TT_FILE_IN_SESSION);
tt_message_file_set(m, file);
```

最初の行でメッセージを作成しています。2行目ではメッセージの配信範囲を追加し、3行目ではメッセージ配信範囲にファイルを追加しています。

構造化データのシリアル化

ToolTalk サービスではメッセージ引数のデータ型として、整数、NULL 終了文字列、バイト文字列の3種類をサポートしています。

ToolTalk メッセージで他のデータ型を送信する際、クライアントはデータをシリアル化して文字列またはバイト文字列に変換し、受信時はそのシリアル化を解除しなければなりません。今回の ToolTalk サービスとともに新しく提供する XDR 引数 API 呼び出しでは、これらの処理を関数として提供します。クライアントが用意するのは、XDR ルーチンとデータへのポインタだけです。ToolTalk サービスは、データをシリアル化して内部バッファに入れた後、そのデータをバイトストリームと同じように処理します。

ToolTalk のメッセージ配信アルゴリズム

この節では、ToolTalk サービスがメッセージの受信側を判定する方法について理解できるように、プロセス指向メッセージとオブジェクト指向メッセージの両方について、作成と配信方法を説明します。

プロセス指向メッセージの配信

プロセス指向メッセージに関して、メッセージを処理すべきプロセスの `ptype` または `procid` が送信側アプリケーションにわかっている場合があります。これに該当しないメッセージに関して、ToolTalk サービスはメッセージの操作と引数からハンドラを判定します。

1. 初期化

送信側はメッセージハンドルを取得し、アドレス属性、配信範囲属性、およびクラス属性を書き込みます。

送信側は、操作属性と引数属性を書き込みます。

送信側が `ptype` を 1 つだけ宣言した場合、ToolTalk サービスはデフォルトとして `sender_ptype` を書き込みます。それ以外の場合は、送信側が `sender_ptype` を書き込まなければなりません。

配信範囲が `TT_FILE` の場合は、ファイル名を書き込む必要があります。書き込まない場合は、デフォルトのファイル名が使用されます。配信範囲が `TT_SESSION` の場合は、セッション名を書き込む必要があります。書き込まない場合は、デフォルトのファイル名が使用されます。配信範囲が `TT_BOTH` または `TT_FILE_IN_SESSION` の場合は、ファイル名とセッション名の両方を書き込む必要があります。書き込まない場合は、デフォルトのファイル名とセッション名が使用されます。

注 - 配信に関して検査される一連のパターンは、メッセージの配信範囲によって異なります。配信範囲が `TT_SESSION` の場合は、同じセッション内のプロセスのパターンだけが検査されます。配信範囲が `TT_FILE` の場合は、ファイルを監視するすべてのプロセスのパターンが検査されます。配信範囲が `TT_FILE_IN_SESSION` または `TT_BOTH` の場合は、その両者に該当するプロセスが検査されます。

送信側は、`handler_ptype` がわかっている場合は書き込んでもかまいません。ただしその場合は、ある `ptype` のプロセスを別の `ptype` に置き換えられなくなるので、柔軟性がかなり低くなります。またこの場合、送信側が処置属性を指定しなければなりません。

2. ハンドラへのディスパッチ

ToolTalk サービスは、アドレス、配信範囲、メッセージクラス、操作、および引数のモードと型を、各 `ptype` の `Handle` セクションのすべてのシグニチャと比較します。

操作と引数に一致し、ハンドルを指定するメッセージパターンを含んでいる ptype は、通常 1 つだけです。ToolTalk サービスは、ハンドラの ptype を検索すると、ptype のメッセージパターンから *opnum*、*handler_ptype*、および「処置」を書き込みます。

アドレスが TT_HANDLER の場合、ToolTalk サービスは指定された *procid* を探索し、ハンドラのメッセージ待ち行列にメッセージを追加します。パターン照合は行わないので、TT_HANDLER メッセージを監視できません。

3. オブザーバへのディスパッチ

ToolTalk サービスは、配信範囲、クラス、操作、および引数の型を、各 ptype の Observe セクションのすべてのメッセージパターンと比較します。

メッセージに一致して、TT_QUEUE または TT_START を指定するすべての監視シグニチャに関して、ToolTalk サービスは、ptype と待ち行列を指定するメッセージまたは開始オプションを指定するメッセージに、レコード（「約束監視」と呼ばれます）を追加します。その後 ToolTalk サービスは、内部の ObserverPtypeList に ptype を追加します。

4. ハンドラへの配信

実行中のプロセスがメッセージに一致する登録済みハンドラメッセージパターンを持っている場合、ToolTalk サービスはメッセージをそのプロセスに配信します。それ以外の場合、ToolTalk サービスは、処置（開始または待ち行列）オプションに従います。

ハンドラ情報に一致する動的パターンを複数のプロセスが登録している場合は、より限定的なパターン（ワイルドカード以外に一致しているものの数を数えることによって判定する）が優先されます。2 つのパターンが同等に限定的である場合、ハンドラが任意に選択します。

5. オブザーバへの配信

ToolTalk サービスは実行中のプロセスのうち、メッセージに一致するオブザーバパターンを登録したすべてのプロセスにメッセージを渡します。ToolTalk サービスは配信のたびに、オブザーバの ptype について約束監視の完了状態を検査します。このプロセスが終わった時点で未完了の約束監視が残っている場合、ToolTalk サービスは、その約束に指定されている開始オプションと待ち行列オプションに従います。

例

この例では、デバッガはエディタを使用して、ToolTalk メッセージを介してブレークポイント周辺のソースを表示します。

エディタは、ptype の中に次のハンドルパターンを持っています。

```
(HandlerPtype: TextEditor;  
  Op: ShowLine;  
  Scope: TT_SESSION;  
  Session: my_session_id;  
  File: /home/butterfly/astrid/src/ebe.c)
```

1. デバッガはブレークポイントに到達すると、操作属性 (ShowLine)、引数属性 (行番号)、ファイル属性 (ファイル名)、セッション属性 (現在のセッション ID)、および配信範囲属性 (TT_SESSION) を含むメッセージを送信します。
2. ToolTalk サービスは、登録されているすべてのパターンとメッセージを照合して、エディタが登録したパターンを検索します。
3. ToolTalk サービスは、エディタにメッセージを配信します。
4. その後エディタは、引数で示した行へスクロールします。

オブジェクト指向メッセージの配信

ToolTalk サービスが処理するメッセージの多くはオブジェクト宛てですが、実際にはオブジェクトを管理するプロセスに配信されます。メッセージは、特定のメッセージを処理できるプロセスの `ptype` を含んだ `otype` にシグニチャを書き込み、ToolTalk サービスがオブジェクト指向メッセージを配信する必要があるプロセスを判定するのを助けます。

1. 初期化

送信側はクラス属性、操作属性、引数属性、および対象とする `objid` 属性を書き込みます。

送信側属性は、ToolTalk サービスによって自動的に書き込まれます。送信側は、`sender_ptype` 属性とセッション属性を書き込むことも、ToolTalk サービスにデフォルト値を書き込ませることもできます。

配信範囲が `TT_FILE` の場合は、ファイル名を書き込む必要があります。書き込まない場合は、デフォルトのファイル名が使用されます。配信範囲が `TT_SESSION` の場合は、セッション名を書き込む必要があります。書き込まない場合は、デフォルトのセッション名が使用されます。配信範囲が `TT_BOTH` または `TT_FILE_IN_SESSION` の場合は、ファイル名とセッション名の両方を書き込む必要があります。書き込まない場合は、デフォルトのファイル名とセッション名が使用されます。

注 - 配信に関して検査される一連のパターンは、メッセージの配信範囲によって異なります。配信範囲が `TT_SESSION` の場合は、同じセッション内のプロセスのパターンだけが検査されます。配信範囲が `TT_FILE` の場合は、ファイルを監視するすべてのプロセスのパターンが検査されます。配信範囲が `TT_FILE_IN_SESSION` または `TT_BOTH` の場合は、その両者に該当するプロセスが検査されます。

2. 解釈処理

ToolTalk サービスは、ToolTalk データベースで `objid` を検索し、`otype` 属性とファイル属性を書き込みます。

3. ハンドラへのディスパッチ

ToolTalk は、`otype` 定義内でメッセージの操作属性と引数属性に一致するハンドラのメッセージパターンを検索します。一致するものが見つかり、ToolTalk サービスは `otype` のメッセージパターンから、配信範囲属性、`opnum` 属性、`handler_ptype` 属性、および処置属性を書き込みます。

4. オブジェクト指向オブザーバへのディスパッチ

ToolTalk サービスは、メッセージのクラス属性、操作属性、および引数属性を、`otype` のすべての `Observe` メッセージのパターンと比較します。一致するものが見つかり、メッセージパターンが `TT_QUEUE` または `TT_START` を指定している場合には、ToolTalk サービスは `ptype` と待ち行列を指定するメッセージ、または開始オプションを指定するメッセージに、レコード(「約束監視」と呼ばれる)を付加します。

5. 手順オブザーバへのディスパッチ

ToolTalk サービスは、メッセージのクラス属性、操作属性、および引数属性を、すべての `ptypes` のすべての `Observe` メッセージのパターンと比較します。一致するものが見つかり、シグニチャが `TT_QUEUE` または `TT_START` を指定している場合には、ToolTalk サービスは `ptype` と待ち行列を指定するメッセージ、または開始オプションを指定するメッセージに、約束監視のレコードを付加します。

6. ハンドラへの配信

実行中のプロセスがメッセージに一致した登録済みのハンドラパターンを持っている場合、ToolTalk サービスはメッセージをそのプロセスに配信します。それ以外の場合、ToolTalk サービスは、処置(待ち行列または開始)オプションに従います。

ハンドラ情報に一致する動的パターンを複数のプロセスが登録している場合は、より限定的なパターン(ワイルドカード以外に一致するものの数を数えることによって判定する)が優先されます。2つのパターンが同時に限定的である場合は、ハンドラが任意に選択します。

7. オブザーバへの配信

ToolTalk サービスは、実行中のプロセスのうち、メッセージに一致するオブザーバパターンを登録したすべてのプロセスにメッセージを渡します。ToolTalk サービスは、配信のたびにオブザーバの `ptype` について約束監視の完了状態を検査します。このプロセスが終わった時点で未完了の約束監視がある場合、ToolTalk サービスは、その約束に指定されている処置(待ち行列または開始)オプションに従います。

例

この例では、ToolTalk サービスに `FinnogaCalc` という架空のスプレッドシートアプリケーションを統合します。

1. `FinnogaCalc` は ToolTalk サービスを起動し、`FinnogaCalc` という `ptype` を宣言してデフォルトのセッションに参加することによって ToolTalk へ登録します。

2. FinnogaCalc は、hatsize.wks ワークシートをロードし、ワークシートファイルに結合することによって、ワークシートを監視することを ToolTalk サービスに通知します。
3. 第 2 のインスタンスの FinnogaCalc (FinnogaCalc2 と呼びます) が起動し、ワークシート wardrobe.wks をロードし、同じ方法で ToolTalk に登録します。
4. hatsize.wks のセル B2 の値を、wardrobe.wks のセル C14 に代入します。
5. FinnogaCalc2 は、FinnogaCalc が FinnogaCalc2 に値を送信できるように ToolTalk 関数を呼び出して、セル C14 のオブジェクト仕様を作成します。このオブジェクトは、objid によって識別されます。
6. その後 FinnogaCalc2 は、FinnogaCalc にその objid を (たとえばクリップボードを介して) 指示します。
7. FinnogaCalc は、この objid によって識別されるオブジェクトにセル B2 を表示し、値を持つメッセージを送信することを記憶します。
8. ToolTalk はメッセージの経路を指定します。ToolTalk サービスは、メッセージを配信するために次のことを行います。
 - a. objid に関連付けられている仕様を検査して、objid の型が FinnogaCalc_cell であることと、対応するオブジェクトが wardrobe.wks ファイルの中にあることを確認します。
 - b. FinnogaCalc_cell について otype の定義を参照します。ToolTalk サービスは otype から、ptype である FinnogaCalc のプロセスがこのメッセージを監視することと、メッセージの配信範囲が TT_FILE であることを判定します。
 - c. メッセージを登録済みのパターンと照合して、適切なファイルを監視しているプロセスのうち、この ptype を持つすべてのプロセスを検索します。FinnogaCalc2 は一致しますが、FinnogaCalc は一致しません。
 - d. FinnogaCalc2 にメッセージを配信します。
9. FinnogaCalc2 は、セル C14 に対応するオブジェクトがメッセージに入っていることを認識します。FinnogaCalc2 は、wardrobe.wks の値を更新し、新しい値を表示します。

otype のアドレス指定

objid が不明なままオブジェクト指向メッセージを送信しなければならない場合があります。このような場合に備えて、ToolTalk サービスでは otype のアドレス指定ができます。このアドレス指定モードでは、送信側が操作、引数、配信範囲、および otype を指定する必要があります。ToolTalk サービスは、メッセージの操作と引数に一致するメッセージパターンについて指定した otype の定義を調べ、処理プロセスと監視プロセスを検索します。その後、特定のオブジェクト宛てのメッセージの場合と同様に、ディスパッチと配信を続けます。

ToolTalk メッセージを送信するためのアプリケーションの変更

アプリケーションは、ToolTalk メッセージを送信するために、いくつかの操作を実行しなければなりません。つまり、ToolTalk メッセージの作成と完了、メッセージコールバックルーチンの追加、および完了したメッセージの送信などの操作が必要です。

メッセージの作成

ToolTalk サービスは、メッセージを作成し完了するために次の3通りの方法を用意しています。

1. 汎用関数
 - `tt_message_create()`
2. プロセス指向の通知関数と要求関数
 - `tt_pnotice_create()`
 - `tt_prequest_create()`
3. オブジェクト指向の通知関数と要求関数
 - `tt_onotice_create()`
 - `tt_orequest_create()`

プロセス指向およびオブジェクト指向の通知関数と要求関数を使用すると、一般的なメッセージが簡単に作成できます。これらの関数は、機能的には他の `tt_message_create()` および `tt_message_attribute_set()` 呼び出しの文字列と同じですが、より簡単に書き込みと読み取りができます。表 8-4 と表 8-5 は、メッセージの作成および完成用に使用する ToolTalk 関数を示しています。

表 8-4 メッセージ作成用の関数

ToolTalk 関数	機能
<code>tt_onotice_create(const char *objid, const char *op)</code>	オブジェクト指向の通知を作成する
<code>tt_orequest_create(const char *objid, const char *op)</code>	オブジェクト指向の要求を作成する
<code>tt_pnotice_create(Tt_scope scope, const char *op)</code>	プロセス指向の通知を作成する

表 8-4 メッセージ作成用の関数 (続き)

ToolTalk 関数	機能
<code>tt_prequest_create(Tt_scope scope, const char *op)</code>	プロセス指向の要求を作成する
<code>tt_message_create(void)</code>	メッセージを作成する。この関数は、メッセージ作成用の ToolTalk 汎用関数である

注 - すべての作成用関数が返す型は `Tt_message` です。

表 8-5 メッセージ完成用の関数

ToolTalk 関数	機能
<code>tt_message_address_set(Tt_message m, Tt_address p)</code>	アドレス指定モードを設定する (たとえば、ポイントツーポイント)
<code>tt_message_arg_add(Tt_message m, Tt_mode n, const char *vtype, const char *value)</code>	NULL 終了文字列の引数を追加する
<code>tt_message_arg_bval_set(Tt_message m, int n, const unsigned char *value, int len)</code>	引数の値を、指定されたバイト配列に設定する
<code>tt_message_arg_ival_set(Tt_message m, int n, int value)</code>	引数の値を、指定された整数に設定する
<code>tt_message_arg_val_set(Tt_message m, int n, const char *value)</code>	引数の値を、指定された NULL 終了文字列に設定する
<code>tt_message_barg_add(Tt_message m, Tt_mode n, const char *vtype, const unsigned char *value, int len)</code>	バイト配列の引数を追加する
<code>tt_message_iarg_add(Tt_message m, Tt_mode n, const char *vtype, int value)</code>	整数の引数を追加する
<code>tt_message_context_bval(Tt_message m, const char *slotname, unsigned char **value, int *len);</code>	コンテキストの値を、指定されたバイト配列に取得する
<code>tt_message_context_ival(Tt_message m, const char *slotname, int *value);</code>	コンテキストの値を、指定された整数に取得する
<code>tt_message_context_val(Tt_message m, const char *slotname);</code>	コンテキストの値を、指定された文字列に取得する
<code>tt_message_icontext_set(Tt_message m, const char *slotname, int value);</code>	コンテキストを、指定された整数に設定する
<code>tt_message_bcontext_set(Tt_message m, const char *slotname, unsigned char *value, int length);</code>	コンテキストを、指定されたバイト配列に設定する。

表 8-5 メッセージ完成用の関数 (続き)

ToolTalk 関数	機能
<code>tt_message_context_set(Tt_message m, const char *slotname, const char *value);</code>	コンテキストを、指定された NULL 終了文字列に設定する
<code>tt_message_class_set(Tt_message m, Tt_class c)</code>	メッセージの型 (通知または要求) を設定する
<code>tt_message_file_set(Tt_message m, const char *file)</code>	メッセージの配信範囲として指定するファイルを設定する
<code>tt_message_handler_ptype_set(Tt_message m, const char *ptid)</code>	メッセージを受信する ptype を設定する
<code>tt_message_handler_set(Tt_message m, const char *procid)</code>	メッセージを受信する procid を設定する
<code>tt_message_object_set(Tt_message m, const char *objid)</code>	メッセージを受信するオブジェクトを設定する
<code>tt_message_op_set(Tt_message m, const char *opname)</code>	メッセージを受信する操作を設定する
<code>tt_message_otype_set(Tt_message m, const char *otype)</code>	メッセージを受信するオブジェクト型を設定する
<code>tt_message_scope_set(Tt_message m, Tt_scope s)</code>	メッセージを受信する受信側 (ファイルまたはセッション、あるいはその両方) を設定する
<code>tt_message_sender_ptype_set(Tt_message m, const char *ptid)</code>	メッセージの送信元であるアプリケーションの ptype を設定する
<code>tt_message_session_set(Tt_message m, const char *sessid)</code>	メッセージの配信範囲として指定するセッションを設定する
<code>tt_message_status_set(Tt_message m, int status)</code>	メッセージの状態を設定する。この状態は、受信側アプリケーションが参照する
<code>tt_message_status_string_set(Tt_message m, const char *status_str)</code>	メッセージの状態を記述するテキストを設定する。このテキストは、受信側アプリケーションが参照する
<code>tt_message_user_set(Tt_message m, int key, void *v)</code>	送信側アプリケーションの内部メッセージを設定する。この内部メッセージは、受信側アプリケーションが参照できない隠されたデータである
<code>tt_message_abstainer(Tt_message m, int n)</code>	指定したメッセージの n 番目の忌避者の procid を返す
<code>tt_message_abstainers_count(Tt_message m)</code>	オフアを忌避した、TT_OFFER m に記録されている procid の数を返す
<code>tt_message_accepter(Tt_message m, int n)</code>	特定のメッセージを受信した n 番目の procid を返す

表 8-5 メッセージ完成用の関数 (続き)

ToolTalk 関数	機能
<code>tt_message_accepters_count (Tt_message m)</code>	オファを受信した、 <code>TT_OFFER m</code> に記録されている <code>procid</code> の数を返す
<code>tt_message_rejecter (Tt_message m, int n)</code>	指定したメッセージの <code>n</code> 番目の拒絶者の <code>procid</code> を返す
<code>tt_message_rejecters_count (Tt_message m)</code>	オファを拒絶した、 <code>TT_OFFER m</code> に記録されている <code>procid</code> の数を返す

注 - メッセージの完成用に使用するすべての関数が返す型は `Tt_status` です。

汎用関数を使った ToolTalk メッセージの作成

汎用関数 `tt_message_create ()` を使用して、ToolTalk メッセージを作成し完成できます。`tt_message_create ()` を使用してプロセス指向またはオブジェクト指向のメッセージを作成する場合は、`tt_message_attribute_set ()` 呼び出しを使用して属性を設定します。

クラス

- 値または状態を返すメッセージ用には、`TT_REQUEST` を使用します。メッセージが処理された場合、メッセージが待ち行列に入れられた場合、または要求を処理するプロセスを起動した場合には通知されます。
- イベントの他のプロセスに通知するだけのメッセージ用には、`TT_NOTICE` を使用します。
- メッセージの受信者としていくつかのタイプを予想して、それらの中の受信者、拒絶者、忌避者の割合を知りたい場合は、`TT_OFFER` を使用します。

アドレス

- `TT_PROCEDURE` を使用して、これらの引数を使用してこの操作を実行できるすべてのプロセスにメッセージを送信します。このメッセージの操作属性と引数属性を書き込みます。
- `TT_OTYPE` を使用して、これらの引数を使用してこの操作を実行できるオブジェクト型にメッセージを送信します。このメッセージの `otype` 属性、操作属性、および引数属性を書き込みます。
- `TT_HANDLER` を使用して、特定のプロセスにメッセージを送信します。ハンドラ属性の値を指定します。

通常、1つのプロセスは一般的な要求を作成し、応答からハンドラ属性を選択し、そのハンドラに追加のメッセージを送信します。ハンドラの正確な `procid` を指定した場合、ToolTalk サービスはメッセージを直接配信します。この場合、パターン照合は行われず、他のアプリケーションがメッセージを監視できません。このポイントツーポイント (PTP) のメッセージ渡し機能によって、2つのプロセスがブロードキャストメッセージを渡すことによって認識し合い、明示的な通信を実行できます。

注 – オファは、`TT_PROCEDURE` アドレスでのみ送信可能です。それ以外のアドレスで送信しようとする、`TT_ERR_ADDRESS` エラーとなります。

- `TT_OBJECT` を使用して、これらの引数を使用して操作を実行する特定のオブジェクトにメッセージを送信します。このメッセージのオブジェクト属性、操作属性、および引数属性を書き込みます。

配信範囲

メッセージ配信の配信範囲を書き込みます。受信可能側は、次の値に結合できます。

- `TT_SESSION`
- `TT_FILE`
- `TT_BOTH`
- `TT_FILE_IN_SESSION`

ToolTalk サービスは、配信範囲に応じてデフォルトのセッションまたはファイル、あるいはその両方をメッセージに追加します。

オファは、`TT_SESSION` の配信範囲でのみ送信可能です。

操作

作成中の通知または要求について記述した操作を書き込みます。操作名を判定するには、ターゲットとなる受信側の `ptype` 定義またはメッセージプロトコル定義を参照します。

引数

操作に固有の引数を書き込みます。引数のデータ型に最も適した関数を使用します。

- `tt_message_arg_add()`
値がゼロ終了文字列の引数を追加します。
- `tt_message_barg_add()`
値がバイト文字列の引数を追加します。

- `tt_message_iarg_add()`
値が整数の引数を追加します。

追加する引数ごとに、次の値を指定します (値の型には関係ありません)。

- `Tt_mode`
`TT_IN` または `TT_INOUT` を指定します。`TT_IN` は、送信側が引数を書き込み、ハンドラおよびオブザーバがこの引数を読み取れることを示します。`TT_INOUT` は、送信側とハンドラが引数を書き込み、誰でも引数を読み取れることを示します。応答としてハンドラからの引数を必要とする要求を送信する場合は、`TT_INOUT` を使用します。
- 値の型

値の型 (`vtype`) は、追加する引数のデータ型について説明します。ToolTalk サービスは、登録済みのパターンとメッセージを比較してメッセージの受信側を判定する際に `vtype` 名を使用します。しかし、メッセージまたはパターンの引数の値を処理する際には `vtype` を使用しません。

`vtype` 名は、メッセージの受信側がデータを解釈するのに役立ちます。たとえば、ワードプロセッサがメモリ内のパラグラフを `PostScript` 表現に変換した場合は、`tt_message_arg_add` に次の引数を付けて呼び出すことができます。

```
tt_message_arg_add (m, "PostScript", buf);
```

この場合 ToolTalk サービスは、ゼロ終了文字列を指す `buf` を想定し送信します。

同様に、アプリケーションは、ToolTalk メッセージで `enum` 値 (たとえば、次のような `Tt_status` の要素) を送信します。

```
tt_message_iarg_add(m, "Tt_status", (int) TT_OK);
```

ToolTalk サービスは、値を整数として送信します。ただし、`Tt_status` という `vtype` は、値が何を意味するかを受信側に示します。

注 - 特に送信側と受信側で特定の `vtype` 名を定義し、受信側が他の形式で格納された値を検索できないようにしなければなりません。たとえば、整数として格納した値が、文字列として検索されないようにしなければなりません。

プロセス指向メッセージの作成

プロセス指向の通知と要求は、簡単に作成できます。手続き型の通知または要求については、新しいメッセージオブジェクトのハンドルまたは隠されたポインタを取得するには、`tt_pnotice_create` 関数または `tt_prerequest_create` 関数を使用します。このハンドルは、以降の呼び出しでメッセージを参照するために使用できます。

`tt_pnotice_create` または `tt_prerequest_create` を使用してメッセージを作成する場合は、引数として次の 2 つの属性を指定しなければなりません。

- 配信範囲

メッセージ配信の配信範囲を書き込みます。受信可能側は、次の値に結合できません。

- TT_SESSION
- TT_FILE
- TT_BOTH
- TT_FILE_IN_SESSION

ToolTalk サービスは配信範囲に応じて、デフォルトのセッションまたはファイル(あるいはその両方)を書き込みます。

■ 操作

作成中の通知または要求について記述した操作を書き込みます。操作名を判定するには、ターゲットプロセスの `ptype` 定義またはその他のプロトコル定義を参照します。

操作引数などの他のメッセージ属性を完成するには、`tt_message_attribute_set()` 呼び出しを使用します。

オブジェクト指向メッセージの作成および完成

オブジェクト指向の通知と要求は、簡単に作成できます。オブジェクト指向の通知または要求について、新しいメッセージオブジェクトのハンドルまたは隠されたポインタを取得するには、`tt_onotice_create` 関数または `tt_orequest_create` 関数を使用します。このハンドルは、以降の呼び出しでメッセージを参照するために使用できます。

`tt_onotice_create` または `tt_orequest_create` を使用してメッセージを作成する場合は、引数として次の2つの属性を指定しなければなりません。

■ Objid

一意のオブジェクト識別子を書き込みます。

■ 操作

作成中の通知または要求について記述した操作を書き込みます。操作名を判定するには、ターゲットプロセスの `ptype` 定義またはその他のプロトコル定義を参照します。

操作引数などの他のメッセージ属性を完成するには、`tt_message_attribute_set()` 呼び出しを使用します。

メッセージコールバックの追加

ある要求がメッセージコールバックルーチンを含んでいる場合、コールバックルーチンは、応答が受信されると自動的に呼び出され、応答結果を調査して適切な処置を行います。

注 - コールバックは、登録した順序と逆の順序で呼び出されます (つまり、最後に追加したコールバックが最初に呼び出されます)。

コールバックルーチンを要求に追加するには、`tt_message_callback_add` を使用します。応答が返され、コールバックルーチンが応答メッセージを処理し終わった場合、コールバックルーチンが `TT_CALLBACK_PROCESSED` を返す前に、その応答メッセージを破棄しなければなりません。応答メッセージを破棄するには、例 8-4 で示すように `tt_message_destroy` を使用します。

例 8-4 メッセージの破棄

```
Tt_callback_action
sample_msg_callback(Tt_message m, Tt_pattern p)
{
    ... process the reply msg ...

    tt_message_destroy(m);
    return TT_CALLBACK_PROCESSED;
}
```

次のコード例は、`cntl_msg_callback` というコールバックルーチンです。このルーチンは、応答の状態フィールドを調査し、状態が起動済み、処理済み、または失敗である場合に処理を実行します。

```

/*
 * Default callback for all the ToolTalk messages we send.
 */

Tt_callback_action
cntl_msg_callback(m, p)
    Tt_message m;
    Tt_pattern p;
{
    int      mark;
    char     msg[255];
    char     *errstr;

    mark = tt_mark();
    switch (tt_message_state(m)) {
        case TT_STARTED:
            xv_set(cntl_ui_base_window, FRAME_LEFT_FOOTER,
                "Starting editor...", NULL);
            break;
        case TT_HANDLED:
            xv_set(cntl_ui_base_window, FRAME_LEFT_FOOTER, "", NULL);
            break;
        case TT_FAILED:
            errstr = tt_message_status_string(m);
            if (tt_pointer_error(errstr) == TT_OK && errstr) {
                sprintf(msg, "%s failed: %s", tt_message_op(m), errstr);
            } else if (tt_message_status(m) == TT_ERR_NO_MATCH) {
                sprintf(msg, "%s failed: Couldn't contact editor",
                    tt_message_op(m),
                    tt_status_message(tt_message_status(m)));
            } else {
                sprintf(msg, "%s failed: %s",
                    tt_message_op(m),
                    tt_status_message(tt_message_status(m)));
            }
            xv_set(cntl_ui_base_window, FRAME_LEFT_FOOTER, msg, NULL);
            break;
        default:
            break;
    }
}
/*
 * no further action required for this message. Destroy it
 * and return TT_CALLBACK_PROCESSED so no other callbacks will
 * be run for the message.
 */
tt_message_destroy(m);
tt_release(mark);
return TT_CALLBACK_PROCESSED;
}

```

ptype のシグニチャの opnum に接続すると、静的パターンにコールバックを追加することもできます。opnum を持つ静的パターンと一致したためにメッセージが配信されると、ToolTalk サービスはその opnum に付加されているコールバックをすべて調べて起動します。

- osignature の opnum にコールバックルーチンを付加するには、tt_otype_opnum_callback_add を使用してください。
- psignature の opnum にコールバックルーチンを付加するには、tt_ptype_opnum_callback_add を使用してください。

メッセージの送信

メッセージが完成したら、tt_message_send を使用して送信します。

ToolTalk サービスが TT_WRN_STALE_OBJID を返した場合は、ToolTalk データベースで転送ポインタを検索したことを意味します。そのポインタは、メッセージに指定されたオブジェクトが移動したことを示します。ただし、ToolTalk サービスは、新しい objid を使用してメッセージを送信します。その後、tt_message_object を使用してメッセージから新しい objid を検索し、内部のデータ構造体に入れることができます。

将来必要としないメッセージ (たとえば通知など) がある場合は、tt_message_destroy を使用して、メッセージを削除して記憶領域を解放できます。

注 - メッセージに対する応答がありそうな場合は、応答を処理する前にメッセージを破棄しないでください。

例

例 8-5 は、pnotice の作成方法と送信方法を示しています。

例 8-5 pnotice の作成と送信

```
/*
 * Create and send a ToolTalk notice message
 * ttsample1_value(in int <new value)
 */

msg_out = tt_pnotice_create(TT_SESSION, "ttsample1_value");
tt_message_arg_add(msg_out, TT_IN, "integer", NULL);
tt_message_arg_ival_set(msg_out, 0, (int)xv_get(slider,
        PANEL_VALUE));
tt_message_send(msg_out);

/*
 * Since this message is a notice, we don't expect a reply, so
 * there's no reason to keep a handle for the message.
 */

tt_message_destroy(msg_out);
```

例 8-6 は、`cntl_ui_hilite_button` のコールバックルーチン呼び出す場合の `orequest` の作成と送信方法を示しています。

例 8-6 orequest の作成と送信

```
/*
 * Notify callback function for 'cntl_ui_hilite_button'.
 */
void
cntl_ui_hilite_button_handler(item, event)
    Panel_item    item;
    Event         *event;
{
    Tt_message    msg;

    if (cntl_objid == (char *)0) {
        xv_set(cntl_ui_base_window, FRAME_LEFT_FOOTER,
            "No object id selected", NULL);
        return;
    }
    msg = tt_orequest_create(cntl_objid, "hilite_obj");
    tt_message_arg_add(msg, TT_IN, "string", cntl_objid);
    tt_message_callback_add(msg, cntl_msg_callback);
    tt_message_send(msg);
}
```


第 9 章

動的メッセージパターン

動的メッセージ方式は、アプリケーションプログラムの実行中にメッセージパターン情報を提供するという方法です。まず、メッセージパターンを作成して、ToolTalk サービスに登録します。ToolTalk サービスがメッセージとパターンを照合するときに呼び出すコールバックルーチンを、動的メッセージパターンに追加できます。

動的メッセージの定義

動的メッセージパターンを作成し登録するためには、新しいパターンオブジェクトを割り当て、適切な情報を書き込んでから登録します。パターンがなくなった(つまり、そのパターンに一致するメッセージを対象としなくなった)場合は、パターンを登録解除するか削除します。必要に応じて、動的メッセージパターンを登録および登録解除できます。

動的メッセージパターンの作成、登録、および登録解除に使用される ToolTalk 関数を表 9-1 に示します。

表 9-1 メッセージパターンの作成、更新、および削除用の関数

ToolTalk 関数	機能
<code>tt_pattern_create(void)</code>	パターンの作成
<code>tt_pattern_arg_add(Tt_pattern p, Tt_mode n, const char *vtype, const char *value)</code>	文字列引数の追加
<code>tt_pattern_barg_add(Tt_pattern m, Tt_mode n, const char *vtype, const unsigned char *value, int len)</code>	バイト配列引数の追加
<code>tt_pattern_iarg_add(Tt_pattern m, Tt_mode n, const char *vtype, int value)</code>	整数引数の追加

表 9-1 メッセージパターンの作成、更新、および削除用の関数 (続き)

ToolTalk 関数	機能
<code>tt_pattern_xarg_add(Tt_pattern m, Tt_mode n, const char *vtype, xdrproc_t xdr_proc, void *value)</code>	バイト配列に xdr 引数を追加
<code>tt_pattern_bcontext_add(Tt_pattern p, const char *slotname, const unsigned char *value, int length);</code>	バイト配列コンテキストの追加
<code>tt_pattern_context_add(Tt_pattern p, const char *slotname, const char *value);</code>	文字列コンテキストの追加
<code>tt_pattern_icontext_add(Tt_pattern p, const char *slotname, int value);</code>	整数コンテキストの追加
<code>tt_pattern_address_add(Tt_pattern p, Tt_address d)</code>	アドレスの追加
<code>tt_pattern_callback_add(Tt_pattern p, Tt_message_callback_action f)</code>	メッセージコールバックの追加
<code>tt_pattern_category_set(Tt_pattern p, Tt_category c)</code>	カテゴリの設定
<code>tt_pattern_class_add(Tt_pattern p, Tt_class c)</code>	クラスの追加
<code>tt_pattern_disposition_add(Tt_pattern p, Tt_disposition r)</code>	処置の追加
<code>tt_pattern_file_add(Tt_pattern p, const char *file)</code>	ファイルの追加
<code>tt_pattern_object_add(Tt_pattern p, const char *objid)</code>	オブジェクトの追加
<code>tt_pattern_op_add(Tt_pattern p, const char *opname)</code>	操作の追加
<code>tt_pattern_opnum_add(Tt_pattern p, int opnum)</code>	操作番号の追加
<code>tt_pattern_otype_add(Tt_pattern p, const char *otype)</code>	オブジェクト型の追加
<code>tt_pattern_scope_add(Tt_pattern p, Tt_scope s)</code>	配信範囲の追加
<code>tt_pattern_sender_add(Tt_pattern p, const char *procid)</code>	送信側プロセス識別子の追加
<code>tt_pattern_sender_ptype_add(Tt_pattern p, const char *ptid)</code>	送信側プロセス型の追加
<code>tt_pattern_session_add(Tt_pattern p, const char *sessid)</code>	セッション識別子の追加
<code>tt_pattern_state_add(Tt_pattern p, Tt_state s)</code>	状態の追加
<code>tt_pattern_user_set(Tt_pattern p, int key, void *v)</code>	ユーザーの設定
<code>tt_pattern_register(Tt_pattern p)</code>	パターンの登録
<code>tt_pattern_unregister(Tt_pattern p)</code>	パターンの登録解除
<code>tt_pattern_destroy(Tt_pattern p)</code>	メッセージパターンの削除

注 - `tt_pattern_create` を除くすべての関数が返す型は、`Tt_status` です。
`tt_pattern_create` は、`Tt_pattern` を返します。

メッセージパターンの作成

メッセージパターンを作成するには、`tt_pattern_create` 関数を使用します。この関数を使用して、新しいパターンオブジェクトのハンドルまたは隠されたポインタを入手できます。後続の呼び出しでは、このハンドルを使用してパターンを参照できます。

パターン情報を書き込むには、`tt_pattern_attribute_add` および `tt_pattern_attribute_set` 呼び出しを使用します。パターンに追加する各属性に複数の値を指定できます。パターンの値のどれか 1 つがメッセージの値に一致すれば、パターン属性はメッセージ属性に一致します。属性の値が指定されていない場合、ToolTalk サービスは任意の値を一致させるものと仮定します。すでに設定されているために値を 1 つだけ持つ属性もあります。

メッセージパターンコールバックの追加

コールバックルーチンをユーザーのパターンに追加するには、`tt_pattern_callback_add` 関数を使用します。

注 - コールバックは、登録したのと逆の順序で呼び出されます (つまり、最後に追加したコールバックが最初に呼び出されます)。

ToolTalk サービスがメッセージを照合する場合、コールバックルーチンを自動的に呼び出してメッセージを検査し、適切な処置を行います。パターンに一致するメッセージは、コールバックとともに配信されると、コールバックルーチンを介して処理されます。ルーチンが終了すると、`TT_CALLBACK_PROCESSED` を返し、操作に使用された API オブジェクトは解放されます。その後、`tt_message_destroy` を使用してメッセージを削除します。それによって、メッセージが使用していた記憶領域が解放されます。次にコード例を示します。

```

Tt_callback_action
sample_msg_callback(Tt_message m, Tt_pattern p)
{
    ... process the reply msg ...

    tt_message_destroy(m);
    return TT_CALLBACK_PROCESSED;
}

```

メッセージパターンの登録

完成したパターンを登録するには、`tt_pattern_register()` 関数を使用します。パターンを登録してから、処理の対象とするセッションまたはファイルを結合します。

次のコード例では、パターンを作成して登録しています。

```

/*
 * Create and register a pattern so ToolTalk
 * knows we are interested
 * in "ttsample1_value" messages within
 * the session we join.
 */

pat = tt_pattern_create();
tt_pattern_category_set(pat, TT_OBSERVE);
tt_pattern_scope_add(pat, TT_SESSION);
tt_pattern_op_add(pat, "ttsample1_value");
tt_pattern_register(pat);

```

メッセージパターンの削除と登録解除

注 - アプリケーションプログラムが、削除したパターンに一致する配信済みメッセージを検索していない場合 (たとえば、メッセージが待ち行列に入れられた場合)、ToolTalk サービスは、それらのメッセージを削除しません。

メッセージパターンを削除するには、`tt_pattern_destroy()` 関数を使用します。この関数は最初にパターンを登録解除し、その後パターンオブジェクトを削除します。

パターンオブジェクトを削除せずに、メッセージパターンに一致するメッセージの受信を停止するには、`tt_pattern_unregister()` を使用してパターンを登録解除します。

tt_close を呼び出すと、ToolTalk サービスは、すべてのメッセージパターンオブジェクトを自動的に登録解除して削除します。

現在のセッションまたはファイルによるメッセージパターンの更新

現在、処理の対象とするセッションまたはファイルによってメッセージパターンを更新するには、セッションまたはファイルを結合します。

デフォルトセッションの結合

セッションを結合するとき、ToolTalk サービスは、sessid によってメッセージパターンを更新します。たとえば、ptype を宣言する場合、あるいは TT_SESSION または TT_FILE_IN_SESSION を指定したメッセージパターンを登録する場合は、tt_session_join を使用してデフォルトセッションを結合します。次のコード例は、デフォルトセッションを結合する方法を示しています。

```
/*
 * Join the default session
 */

tt_session_join(tt_default_session());
```

表 9-2 は、処理の対象とするセッションを結合する場合に使用する ToolTalk 関数を示しています。

表 9-2 デフォルトセッションを結合するための ToolTalk 関数

返される型	ToolTalk 関数	機能
char *	tt_default_session(void)	デフォルトセッションの ID を返す
Tt_status	tt_default_session_set(const char *sessid)	デフォルトセッションを設定する
char *	tt_initial_session(void)	初期セッションの ID を返す
Tt_status	tt_session_join(const char *sessid)	このセッションを結合する

表 9-2 デフォルトセッションを結合するための ToolTalk 関数 (続き)

返される型	ToolTalk 関数	機能
Tt_status	tt_session_quit(const char *sessid)	セッションを終了する

パターンが更新されてしまうと、結合したセッションを配信範囲としたメッセージを受信します。

注 - あらかじめセッションを結合しており、ptype または新しいメッセージパターンを登録した場合、新しいパターンに一致するメッセージを受信するためには、再び同じセッションまたは新しいセッションを結合してパターンを更新しなければなりません。

デフォルトセッションを参照するメッセージを受信する必要がなくなった場合は、tt_session_quit() 関数を使用します。この関数は、セッションに配信範囲指定されたメッセージパターンから sessid を削除します。

複数セッションの結合

複数のセッションを結合する場合、ユーザーは要求とポイントツーポイントメッセージに対する応答を自動的に受け取りますが、新しいセッションを明示的に結合しないかぎり通知を受け取ることはありません。次のコード例は、複数のセッションを結合する方法を示します。

```
tt_default_session_set(new_session_identifier);
tt_open();
tt_session_join(new_session);
```

複数のセッションを効果的に使用するには、ユーザーが、対象とするセッションのセッション ID を格納しなければなりません。このセッションは、tt_open を使用して新しいセッションを開始する前に、tt_default_session_set にこれらの識別子を渡すためのものです。つまり、ユーザーは、システム上のファイルに値 (ttsession が環境変数 _SUN_TT_SESSION に格納する値) を配置する必要があり、これによって他の ToolTalk クライアントがそのファイルに含まれるセッション ID の値にアクセスでき、その値を使用して非デフォルトセッションを開始できます。たとえばユーザーは、セッション ID を「明らかに認識されている」ファイルに格納し、その後ファイル範囲指定のメッセージ (このファイルを示す) を、適切なパターンを登録したすべてのクライアントへ送信できます。その場合クライアントは、配信先ファイルを開始する情報を得て、そこから 1 つ以上のセッション ID を読み取りま

す。そしてこれらのセッション ID (`tt_open` を伴う) を使用して、非デフォルトセッションを開始します。これに代わる方法としては、たとえば、ネームサービスまたは第三者データベースを使用してセッション ID を通知するなどの手段があります。

注 - `ttsession` セッション ID が、どのように格納されて対象とするクライアントに渡されるかは、ToolTalk プロトコルの範囲を越えており、システムのアーキテクチャに基づいて決定されなければなりません。

処理の対象とするファイルの結合

ファイルを結合する場合、ToolTalk サービスは、ファイルに配信範囲指定されたメッセージパターンにファイル名を自動的に追加します。たとえば、プロセス型を宣言する、あるいは `TT_FILE` または `TT_FILE_IN_SESSION` を指定したメッセージパターンを登録する場合、`tt_file_join()` 関数を使用して処理の対象とするファイルを結合します。表 9-3 は、特定の処理対象のファイルを示すために使用する ToolTalk 関数を示しています。

表 9-3 処理対象とするファイルを結合するための ToolTalk 関数

返される型	ToolTalk 関数	機能
<code>char *</code>	<code>tt_default_file(void)</code>	デフォルトファイルを結合する
<code>Tt_status</code>	<code>tt_default_file_set(const char *docid)</code>	デフォルトファイルを設定する
<code>Tt_status</code>	<code>tt_file_join(const char *filepath)</code>	このファイルを結合する
<code>Tt_status</code>	<code>tt_file_quit(const char *filepath)</code>	ファイルを終了する

ファイルを参照するメッセージを受信する必要がなくなった場合、`tt_file_quit()` 関数を使用して、ファイルに配信範囲指定したメッセージパターンからファイル名を削除します。

第 10 章

静的メッセージパターン

定義済みのメッセージの集合を受信する場合、静的メッセージ方式は、メッセージパターン情報を指定する簡単な方法です。

静的メッセージの定義

静的メッセージ方式を使用するには、ユーザーのプロセス型とオブジェクト型を定義し、ToolTalk 型コンパイラの `tt_type_comp` によってコンパイルします。プロセス型を宣言すると、ToolTalk サービスはその型に基づいたメッセージパターンを作成します。静的メッセージパターンは、ToolTalk サービスとの通信を閉じるまで有効です。

プロセス型の定義

アプリケーションは、アプリケーションを実行しているプロセスがない場合でも、潜在的なメッセージの受信者であると考えられます。そのために、プロセス型 (`p_type`) ファイルにアプリケーションの起動方法に関するメッセージパターンと指示を提供します。これらの指示は、メッセージがアプリケーションを利用可能だがアプリケーションが起動していない場合、次の処置の 1 つを行うよう ToolTalk サービスに指示します。

- アプリケーションを起動し、メッセージを配信する。
- アプリケーションが起動するまで、メッセージを待ち行列に入れる。
- メッセージを破棄する。

ToolTalk サービスが情報を利用できるようにするため、アプリケーションのインストール時に、ToolTalk 型コンパイラの `tt_type_comp` によって `ptype` ファイルをコンパイルします。

アプリケーションが ToolTalk サービスに `ptype` を登録すると、`ptype` に表示されたメッセージパターンも自動的に登録されます。

`ptype` は、アプリケーションが起動していないときに ToolTalk サービスが使用する、アプリケーション情報を提供します。この情報は、必要に応じてプロセスを起動する場合のほか、メッセージを受信する場合またはプロセスが開始するまでメッセージを待ち行列に入れる場合に使用されます。

`ptype` は、プロセス型識別子 (`ptid`) で始まります。`ptid` の後には、次の項目が続きます。

1. オプションの開始文字 — ToolTalk サービスは、必要に応じてこのコマンドを実行し、プログラムを実行するプロセスを起動します。
2. シグニチャー プログラムが受信対象とする、`TT_PROCEDURE` にアドレス指定したメッセージを記述します。監視対象のメッセージは、処理対象のメッセージとは別に記述します。

シグニチャ

「シグニチャ」は、プログラムが受信対象とするメッセージについて記述します。シグニチャは、矢印 (`=>`) によって 2 つの部分に分けられます。シグニチャの前半部分は、照合する属性値を指定します。シグニチャに指定する属性値の数が増えるほど、シグニチャが一致するメッセージの数は少なくなります。シグニチャの後半部分は、シグニチャの前半部分に一致したメッセージに ToolTalk サービスがコピーする受信側の値を指定します。

`ptype` のシグニチャは、処置および操作番号 (`opnum`) の値を持つことができます。ToolTalk サービスは、処置の値 (開始、待ち行列、またはデフォルトの破棄) を使用し、プログラムを実行しているプロセスがない場合に、シグニチャに一致したメッセージに対する処理を決定します。`opnum` 値は、メッセージの受信側にとって便利です。2 つのシグニチャの操作名が同じで引数が異なる場合は、異なる `opnum` によって着信メッセージを簡単に識別できます。

ptype ファイルの作成

次のコード例は、ptype ファイルを示しています。

```
#include "Sun_EditDemo_opnums.h"

ptype Sun_EditDemo {
    /* setenv Sun_EditDemo_HOME to install dir for the demo */
    start    "${Sun_EditDemo_HOME}/edit";
    handle:
    /* edit file named in message, start editor if necessary */
    session Sun_EditDemo_edit(void)
        => start opnum=Sun_EditDemo_EDIT;

    /* tell editor viewing file in message to save file */
    session Sun_EditDemo_save(void)
        => opnum=Sun_EditDemo_SAVE;

    /* save file named in message to new filename */
    session Sun_EditDemo_save_as(in string new_filename)
        => opnum=Sun_EditDemo_SAVE_AS;

    /* bring down editor viewing file in message */
    session Sun_EditDemo_close(void)
        => opnum=Sun_EditDemo_CLOSE;
};
```

次に ptype ファイルの構文を示します。

```

ptype ::= 'ptype' ptid '{'
        property* ['observe:' psignature*]
        ['handle:' psignature* ]
        ['handle_push:' psignature*]
        ['handle_rotate:' psignature*]
        '}' [';']
property ::= property_id value ';'
property_id ::= 'start'
value ::= string
ptid ::= identifier
psignature ::= [scope] op args [contextdcl]
            ['=>'
             ['start']['queue']
             ['opnum='number']]
            ';'
scope ::= 'file'
        | 'session'
        | 'file_in_session'
args ::= '(' argspec {, argspec}* ')'
        | '(void)'
        | '()'
contextdcl ::= 'context' '(' identifier {, identifier}* ')' ';'
argspec ::= mode type name
mode ::= 'in' | 'out' | 'inout'
type ::= identifier
name ::= identifier

```

property_id 情報

ptid — プロセス型識別子 (ptid) は、プロセスの型を識別します。ptid は、導入システムごとに一意でなければなりません。この識別子はインストール後は変更できません。したがって、選択する個々の名前は一意でなければなりません。たとえば、Sun_EditDemo のように、製品または会社の商標名を使用できます。ptid には、32 文字以内という制限があり、予約済み識別子 (ptype、otype、start、opnum、queue、file、session、observe、または handle) は使用できません。

start — プロセスの起動文字列。ToolTalk サービスが、プロセスを起動する必要がある場合は、シェルとして使われるコマンド /bin/sh を実行します。

コマンドを実行する前に、ToolTalk サービスは、アプリケーションを起動したメッセージのファイル属性値を使って、TT_FILE を環境変数として定義します。このコマンドは、アプリケーションを起動したメッセージ送信側の環境ではなく、ttsession の環境内で実行します。したがって、コンテキスト情報がある場合、メッセージ引数またはコンテキストによって渡される必要があります。

psignature 照合情報

scope — このパターン属性は、メッセージ内の配信範囲属性と照合されます。

op — 操作名。この名前は、メッセージ内の操作属性と照合されます。

注 - ptype および otype の両方にメッセージシグニチャを指定する場合は、それぞれに固有の操作名を使用してください。たとえば、ptype と otype の両方に表示操作を指定することはできません。

args — 操作のための引数。args リストが void の場合、シグニチャは引数のないメッセージだけに一致します。args リストが空 (つまり ()) の場合は、シグニチャは引数とは無関係に一致します。

contextdcl — コンテキスト名。この名前の付いたコンテキストを持つパターンがシグニチャから生成される場合、パターンは空の値リストを持ちます。

psignature 処置情報

start — psignature がメッセージに一致しても、この ptype を持つ実行中のプロセスの中に、メッセージに一致したパターンを持つものがない場合は、この ptype のプロセスを起動します。

queue — psignature がメッセージに一致しても、この ptype を持つ実行中のプロセスの中にメッセージに一致したパターンを持つものがない場合は、この ptype のプロセスがメッセージに一致したパターンを登録するまでメッセージを待ち行列に入れます。

opnum — 指定された数をメッセージの opnum 属性に書き込み、メッセージと一致したシグニチャを識別できるようにします。

メッセージがシグニチャと一致するとき、シグニチャの opnum がメッセージに組み込まれます。これでアプリケーションは、tt_message_opnum 呼び出しによって opnum を検索できます。シグニチャごとに固有の opnum を指定すると、どのシグニチャがメッセージと一致したかすぐに判定できます。

tt_ptype_opnum_callback_add 呼び出しを使用すると、opnum にコールバックルーチンを接続できます。メッセージが一致すると、ToolTalk サービスはその opnum に接続されたコールバックをすべて調べ、それらを起動します。

Sun_EditDemo_opnums.h ファイルは、edit.c が使用するすべての opnum のシンボリック定義を定義します。これにより、edit.types ファイルと edit.c ファイルは、同じ定義を共有できます。

ツールの自動起動

ToolTalk サービスにツールを自動的に起動させる ptype 宣言の簡単な例を次に示します。これは、表示、編集、および構成対象のメッセージを受信し、そのメッセージを処理できるツールのインスタンスが1つも動作していない場合は、`/home/toone/tools/mytest` を起動し、そのメッセージを配信するというコード例です。



注意 – この例を実行すると、ToolTalk サービスはハンドラを無限に検索し続けます。

```
ptype My_Test {
    start "/home/toone/tools/mytest";
    handle:

    session Display (in Ascii text) => start;
    session Edit (inout Ascii text) => start;
    session Compose (out Ascii text) => start;

    file Display (in Ascii file_name) => start;
    file Edit (inout Ascii file_name) => start;
    file Compose (out Ascii file_name) => start;
};
```

オブジェクト型の定義

メッセージが、特定のオブジェクトまたはオブジェクト型にアドレス指定されている場合、ToolTalk サービスは、メッセージを配信するアプリケーションを判定できなければなりません。アプリケーションは、「オブジェクト型」(otype) にこの情報を提供します。otype はアプリケーションの ptype を指定します。ptype はオブジェクトを管理し、オブジェクトと関連するメッセージパターンを記述しています。

メッセージパターンには、メッセージは利用可能だがアプリケーションが起動していない場合の処理を ToolTalk サービスに示す指示が入っています。この場合、ToolTalk は次の指示のいずれか1つを実行します。

- アプリケーションを起動し、メッセージを配信する。
- アプリケーションが起動するまで、メッセージを待ち行列に入れる。
- メッセージを破棄する。

ToolTalk サービスがこのような情報を利用できるようにするために、アプリケーションのインストール時に ToolTalk 型コンパイラの `tt_type_comp` で `otype` ファイルをコンパイルします。オブジェクトを管理するアプリケーションは、ToolTalk サービスに登録する際に `potype` を宣言します。`potype` を登録するとき、ToolTalk サービスは `potype` を示す `otype` の有無を調べ、`otype` 内で検索したパターンに登録します。

アプリケーションの `otype` は、ToolTalk サービスが、オブジェクト指向のメッセージを配信するときに使用するアドレス指定情報を提供します。ユーザーが持つ `otype` の数とそれらの `otype` が表すものは、ユーザーのアプリケーションの性質によって異なります。たとえば、ワープロ用のアプリケーションの `otype` は文字、単語、パラグラフ、文書などになり、図編集用のアプリケーションの `otype` はノード、円弧、注釈ボックス、ダイアグラムなどになります。

`otype` は、オブジェクト型識別子 (`otid`) で始まります。`otid` の後には、次の項目が続きます。

1. オプションの開始文字列 — ToolTalk は、必要に応じてこのコマンドを実行し、プログラムを実行するプロセスを起動します。
2. シグニチャ — その型のオブジェクトにアドレス指定できるメッセージを定義するコード (つまり、その型のオブジェクトに対して呼び出せる動作)

シグニチャ

「シグニチャ」は、その型のオブジェクトにアドレス指定できるメッセージを定義します。シグニチャは、矢印 (`=>`) によって 2 つの部分に分けられます。シグニチャの前半部分は、着信メッセージの照合基準を定義します。シグニチャの後半部分は、シグニチャの前半部分に一致した各メッセージに ToolTalk サービスが追加する受信側の値を定義します。これらの値は、操作およびメッセージの配信範囲と処置を実行するプログラムの `ptid` を指定します。

otype ファイルの作成

次のコード例は、otype ファイルの構文を示しています。

```
otype ::= obj_header '{' objbody* '}' [';']
obj_header ::= 'otype' otid [':' otid+]
objbody ::= 'observe:' osignature*
          | 'handle:' osignature*
          | 'handle_push:' osignature*
          | 'handle_rotate:' osignature*
osignature ::= op args [contextdcl] [rhs][inherit] ';'
rhs ::= [=>' ptid [scope]]
        ['start']['queue']
        ['opnum='number]
inherit ::= 'from' otid
args ::= '(' argspec {, argspec}* ')'
        | '(void)'
        | '()'
contextdcl ::= 'context' '(' identifier {, identifier}* ')' ';'
argspec ::= mode type name
mode ::= 'in' | 'out' | 'inout'
type ::= identifier
name ::= identifier
otid ::= identifier
ptid ::= identifier
```

obj_Header 情報

otid — 「オブジェクト型識別子」。オブジェクト型を識別します。otid は、導入システムごとに一意でなければなりません。この識別子は、インストール後は変更できません。したがって、選択する個々の名前は一意でなければなりません。たとえば、otype を実行するツールの ptid で始めることができます。otid には、64 文字以内という制限があり、予約済み識別子 (ptype、otype、start、opnum、queue、file、session、observe、または handle) は使用できません。

osignature 情報

otype の定義のオブジェクト本体の部分は、アプリケーションが監視および処理の対象とするオブジェクトに関する、メッセージの osignature を記述したものです。

op — 操作名。この名前は、メッセージの操作属性と照合されます。

注 - `ptype` と `otype` の両方にメッセージシグニチャを指定する場合は、それぞれに一意の操作名を使用します。たとえば、`ptype` と `otype` の両方に表示操作は指定できません。

`args` — 操作のための引数。`args` リストが `void` の場合、シグニチャは引数のないメッセージだけに一致します。`args` リストが空で (つまり `()`) である場合は、シグニチャは引数とは無関係に一致します。

`contextdcl` — コンテキスト名。この名前の付いたコンテキストを持つパターンがシグニチャから生成される場合、パターンは空の値リストを持ちます。

`ptid` — この型のオブジェクトを管理するアプリケーションのプロセス型識別子

`opnum` — 指定された数をメッセージの `opnum` 属性に書き込み、メッセージと一致したシグニチャを識別できるようにします。

メッセージがシグニチャと一致するとき、シグニチャの `opnum` がメッセージに組み込まれます。これでアプリケーションは、`tt_message_opnum` 呼び出しによって `opnum` を検索できます。シグニチャごとに固有の `opnum` を指定すると、どのシグニチャがメッセージと一致したかを判定できます。

`tt_otype_opnum_callback_add` 呼び出しを使用すれば、`opnum` にコールバックルーチンを接続できます。メッセージが一致すると、ToolTalk サービスは、その `opnum` に接続されたコールバックをすべて調べ、それらを起動します。

`inherit` — `otype` は、操作を基本の型から引き継げる引き継ぎ階層を形成します。ToolTalk サービスでは `otype` を定義する場合、引き継いだすべての操作と引き継ぎ元の `otype` を明示的に指定しなければなりません。この明示的な指定によって、後からの変更 (階層に新しいレベルを追加することや、基本の型に新しい操作を追加することなど) が、`otype` の動作に予想外の影響を与えるのを防止できます。

`scope` — このパターン属性は、メッセージの配信範囲属性と照合されます。配信範囲は矢印の右端にあり、メッセージのディスパッチのときに ToolTalk サービスが書き込みます。これは `otype` を定義するときに、属性を指定できることを意味します。つまり、メッセージの送信側は、メッセージの配信方法を理解していなくてもかまいません。

osignature 処理情報

`start` — `osignature` がメッセージに一致しても、この `otype` を持つ実行中プロセスの中にメッセージに一致したパターンを持つものがない場合は、この `otype` のプロセスを起動します。

queue — osignature がメッセージに一致しても、この otype を持つ実行中プロセスの中に、メッセージに一致したパターンを持つものがない場合は、この otype のプロセスがメッセージに一致したパターンを持つものを登録するまで、メッセージを待ち行列に入れます。

次に otype ファイルの例を示します。

```
#include "Sun_EditDemo_opnums.h"

otype Sun_EditDemo_object {
  handle:
  /* hilite object given by objid, starts an editor if necessary */
  hilite_obj (in string objid)
    => Sun_EditDemo session start opnum=Sun_EditDemo_HILITE_OBJ;
};
```

Sun_EditDemo_opnums.h ファイルは、edit.c が使用するすべての opnum のシンボリック定義を定義します。これによって、edit.types ファイルと edit.c ファイルは同じ定義を共有できます。

型情報のインストール

ToolTalk 型データベースを使用すると、送信処理を実行するホスト、受信処理を実行するホスト、およびプロセスが結合しているセッションを実行しているホストが、ptype と otype の情報を利用できます。

- アプリケーションを起動してメッセージを待ち行列に入れるには、ptype の定義を ToolTalk 型データベースに格納しなければなりません。
- アプリケーションが作成し管理するオブジェクトにアドレス指定されたメッセージを受信するには、otype の定義も ToolTalk 型データベースにインストールしなければなりません。

ToolTalk 型データベースに型情報を格納し、ToolTalk サービスが利用できるようにするには、ToolTalk 型コンパイラの tt_type_comp で型ファイルをコンパイルします。このコンパイラは、型情報の ToolTalk 型定義を作成し、ToolTalk 型データベースに格納します。詳細は、第 5 章を参照してください。

今回のバージョンの ToolTalk サービスには、現在動作中の ttsession にコンパイル済み ToolTalk 型ファイルを併合する次のような関数があります。

```
tt_session_types_load(current_session, compiled_types_file)
```

current_session は現在のデフォルト ToolTalk セッション、*compiled_types_file* はコンパイル済み ToolTalk 型ファイルの名称です。この関数は新しい型は追加し、同じ名前を持つ型が既にあれば置き換えます。他の既存の型は元のままです。



注意 - `tt_session_types_load()` の動作は、`ttsession(1)` と `ttsession_file(4)` の両方の引数によって制御されます。
`tt_session_types_load()` を使用する前に、これらのマニュアルページを参照してください。

既存のプロセス型の確認

ToolTalk サービスには、指定された `ptype` が、現在のセッションにすでに登録されているかどうかを確認する簡単な関数があります。

```
tt_ptype_exists(const char *ptid)
```

`ptid` は、登録されているかどうかを確認するセッションの識別子です。

プロセス型の宣言

型情報は、(アプリケーションのインストール時に) 一度しか指定しません。したがってアプリケーションは、起動するたびに `ptype` の宣言だけを行う必要があります。

`ptype` を宣言するには、アプリケーションの ToolTalk 初期化ルーチンで `tt_ptype_declare` を使用します。ToolTalk サービスは、`ptype` と指定した `ptype` を参照する `otype` に記述されたメッセージパターンを作成します。

`ptype` を宣言すると作成されるメッセージパターンは、アプリケーションが ToolTalk セッションを終了するまでメモリーに存在します。

注 - ptype 情報を宣言するときに作成されるメッセージパターンは、`tt_pattern_unregister` で登録を解除できません。ただし、`tt_ptype_undeclare` を使用すると、これらのパターンの登録を解除できます。

次にプログラム初期設定時に ptype を登録する方法を示します。

```
/*
 * Initialize our ToolTalk environment.
 */
int
edit_init_tt()
{
    int    mark;
    char   *procid = tt_open();
    int    ttfd;
    void    edit_receive_tt_message();

    mark = tt_mark();

    if (tt_pointer_error(procid) != TT_OK) {
        return 0;
    }
    if (tt_ptype_declare("Sun_EditDemo") != TT_OK) {
        fprintf
(stderr, "Sun_EditDemo is not an installed ptype.\n");
        return 0;
    }
    ttfd = tt_fd();
    tt_session_join(tt_default_session());
    notify_set_input_func(edit_ui_base_window,
                          (Notify_func)edit_receive_tt_message,
                          ttfd);

    /*
     * Note that without tt_mark() and tt_release(), the above
     * combination would leak storage -- tt_default_session() returns
     * a copy owned by the application, but since we don't assign the
     * pointer to a variable we could not free it explicitly.
     */

    tt_release(mark);
    return 1;
}
```

プロセス型の宣言解除

宣言した ptype を宣言解除する必要が生じる場合があります。たとえば、CASE 環境における次のような場合です。

- 導入システムでは、コンパイルサーバーを設定しています。このサーバーは、コンパイルが要求されたとき、そのコンパイル要求を優先的に受け付けることを宣言しています。サーバーは、要求を受け付けると状態を変更し、それ以降は新しいコンパイル要求を受け付けません。
- 通常カプセル化されたプロセスは、それ自身を複数の ptype として宣言し、下位層のツールへ要求を送ります。下位層のツールが終了した場合は、総称ラッパーは、そのツールに関連付けられている ptype として宣言する必要はありません。

ptype を登録解除するには、tt_ptype_undeclare を使用します。この呼び出しは、tt_ptype_declare と逆の動作をします。つまり、ptype から生成されたすべてのパターンを登録解除します。さらにセッションが保持する、この ptype を持つ有効プロセスのリストからプロセスを削除します。指定した ptype が呼び出し側プロセスによって宣言されていない場合、この呼び出しは TT_ERR_PTYPE の状態を返します。



注意 - tt_ptype_undeclare を 1 回呼び出すと、プロセスが ptype を宣言した回数に関係なく、ptype は完全に登録解除されます。つまり、ptype の複数の宣言は、1 回の宣言と同じです。

例 10-1 は、宣言した ptype を宣言解除する方法の例です。

例 10-1 ptype の宣言解除

```
/*
 * Obtain procid
 */
tt_open();

/*
 * Undeclared Ptype
 */
tt_ptype_undeclare(ptype);
```


第 11 章

メッセージの受信

この章では、ユーザーのアプリケーションに配信されたメッセージの検索方法と検査済みメッセージの処理方法について説明します。さらに、受信した要求に対して応答を送信する方法についても説明します。

ToolTalk メッセージの検索と処理を行うには、ユーザーのアプリケーションはさまざまな操作を実行しなければなりません。ToolTalk メッセージの検索、メッセージの検査、コールバックルーチンの提供、要求への応答、および不要になったメッセージの削除を実行できる必要があります。

メッセージの検索

メッセージがユーザーのプロセスに届くと、ToolTalk が提供したファイル記述子が使用可能になります。このファイル記述子が使用可能だと通知されると、プロセスは `tt_message_receive` を呼び出し、着信メッセージのハンドルを取得しなければなりません。

メッセージの受信方法を例 11-1 に示します。

例 11-1 メッセージの受信

```
/*
 * When a ToolTalk message is available, receive it; if it's a
 * ttsample1_value message, update the gauge with the new value.
 */
void
receive_tt_message()
{
    Tt_message msg_in;
    int mark;
    int val_in;

    msg_in = tt_message_receive();

    /*
     * It's possible that the file descriptor would become active
     * even though ToolTalk doesn't really have a message for us.
     * The returned message handle is NULL in this case.
     */

    if (msg_in == NULL) return;
```

メッセージのハンドルは不変です。たとえば、プロセスがメッセージを送信するとき、メッセージとそのメッセージに対する応答は、送信されたメッセージと同じハンドルを持ちます。例 11-2 に、TT_HANDLED のメッセージ状態を検査する方法の例を示します。

例 11-2 メッセージ状態のコード検査

```
Tt_message m, n;
m = tt_message_create();
...
tt_message_send(m);

... wait around for tt_fd to become active

n = tt_message_receive();
if (m == n) {
    /* This is the reply to the message we sent */
    if (TT_HANDLED == tt_message_state(m) ) {
        /* The receiver has handled the message so we can go
on */
        ...
    }
} else {
    /* This is a new message coming in */
}
```

メッセージの容易な識別方法と処理方法

ユーザーが受信したメッセージを容易に識別および処理するには、次のようにします。

- `tt_pattern_callback_add` を使って、コールバックを動的パターンに追加します。メッセージを検索する場合、ToolTalk サービスは任意のメッセージコールバックまたはパターンコールバックを呼び出します。コールバックをパターンに置く方法については、第9章を参照してください。
- `ptype` のメッセージパターンに一致するメッセージを受信している場合、メッセージの `opnum` を検索します。

応答の容易な認識方法と処理方法

ユーザーが送信したメッセージに対する応答を容易に認識および処理するには、次のようにします。

- `tt_message_callback_add` を使って、送信する前に特定のコールバックを要求します。コールバックをメッセージに置く方法については、第8章を参照してください。
- 受信したばかりのメッセージと送信したメッセージのハンドルを比較します。そのメッセージが応答の場合は、ハンドルは同じになります。
- `tt_message_user_set` を使って、ユーザーのアプリケーションにとって意義のある情報を要求に追加します。

メッセージ状態の検査

メッセージを受信したら、その状態を検査しなければなりません。メッセージ状態が `TT_WRN_START_MESSAGE` の場合は、応答、拒否、異常終了、`tt_message_accept` 呼び出しのいずれかの処理を実行しなければなりません。`tt_message_accept` 以外は、メッセージが通知であっても行います。

ToolTalk サービスを使って起動したプログラムで `TT_WRN_START_MESSAGE` の状態を受信したプログラムは、`tt_message_uid` と `tt_message_gid` を確認してください。UNIX UID と GID のいずれか一方、あるいはその両方が要求と合わない場合は、`TT_DESKTOP_EACCES` で要求が失敗することがあります。同様に、UID や GID に不一致がある場合は、すでに実行済みのアプリケーションでは、`TT_DESKTOP_EACCES` で要求を拒絶することがあります。このため、マッチング ID ハンドラが見つかるか、自動起動ハンドラが要求を拒絶するまで、メッセージは連続して拒絶されます。

メッセージの検査

ユーザープロセスがメッセージを受信する場合、ユーザーはメッセージを検査して適切な処置を行います。

値の検索を開始する前に ToolTalk API スタック上のマークを取得し、ToolTalk サービスがユーザーに同時に返す情報を解放できるようにします。例 11-3 では、記憶領域を割り当て、メッセージの内容を検査し、記憶領域を解放します。

例 11-3 記憶領域の割り当て、検査、および解放

```
/*
 * Get a storage mark so we can easily free all the data
 * ToolTalk returns to us.
 */

mark = tt_mark();

if (0==strcmp("ttsample1_value", tt_message_op(msg_in))) {
    tt_message_arg_ival(msg_in, 0, &val_in);
    xv_set(gauge, PANEL_VALUE, val_in, NULL);
}

tt_message_destroy(msg_in);
tt_release(mark);
return;
```

表 11-1 に、受信したメッセージの属性を検査するためにユーザーが使用する ToolTalk 関数を示します。

表 11-1 メッセージ属性を検査する関数

返される型	ToolTalk 関数	説明
Tt_address	tt_message_address(Tt_message m)	メッセージのアドレス
Tt_status	tt_message_arg_bval(Tt_message m, int n, unsigned char **value, int *len)	バイト配列としての引数値
Tt_status	tt_message_arg_ival(Tt_message m, int n, int *value)	整数としての引数値
Tt_status	tt_message_arg_xval(Tt_message m, int n, xdrproc_t xdr_proc, void *value)	xdr としての引数値
Tt_mode	tt_message_arg_mode(Tt_message m, int n)	引数のモード (入力、出力、入出力)

表 11-1 メッセージ属性を検査する関数 (続き)

返される型	ToolTalk 関数	説明
char *	tt_message_arg_type(Tt_message m, int n)	引数の型
char *	tt_message_arg_val(Tt_message m, int n)	文字列としての引数値
int	tt_message_args_count(Tt_message m)	引数の数
Tt_class	tt_message_class(Tt_message m)	メッセージの型 (通知または要求)
int	tt_message_contexts_count(Tt_message m);	コンテキストの数
char *	tt_message_context_slotname(Tt_message m, int n);	メッセージの「n 番目」のコンテキスト名
Tt_disposition	tt_message_disposition(Tt_message m)	メッセージを受信するアプリケーションが動作していない場合のメッセージの処理方法
char *	tt_message_file(Tt_message m)	メッセージの配信範囲指定となるファイル名
gid_t	tt_message_gid(Tt_message m)	送信側アプリケーションのグループ化した識別子
char *	tt_message_handler(Tt_message m)	ハンドラの procid
char *	tt_message_handler_ptype(Tt_message m)	ハンドラの ptype
char *	tt_message_object(Tt_message m)	メッセージが送られたオブジェクト
char *	tt_message_op(Tt_message m)	操作名
int	tt_message_opnum(Tt_message m)	操作番号
char *	tt_message_otype(Tt_message m)	メッセージが送られたオブジェクトタイプ
Tt_pattern	tt_message_pattern(Tt_message m)	メッセージが照合されるパターン
Tt_scope	tt_message_scope(Tt_message m)	メッセージの受信側 (FILE、SESSION、BOTH)
char *	tt_message_sender(Tt_message m)	送信側アプリケーションの procid
char *	tt_message_sender_ptype(Tt_message m)	送信側アプリケーションの ptype
char *	tt_message_session(Tt_message m)	メッセージを送信したセッション

表 11-1 メッセージ属性を検査する関数 (続き)

返される型	ToolTalk 関数	説明
Tt_state	tt_message_state(Tt_message m)	メッセージの現在の状態
int	tt_message_status(Tt_message m)	メッセージの現在の状態
char *	tt_message_status_string(Tt_message m)	メッセージの現在の状態を記述するテキスト
uid_t	tt_message_uid(Tt_message m)	送信側アプリケーションのユーザー識別子
void *	tt_message_user(Tt_message m, int key)	アプリケーション内部の隠されたデータ

コールバックルーチン

パターンが一致してメッセージが到着した際、ToolTalk サービスにコールバックを起動するよう指示できます。

```
p = tt_pattern_create();
tt_pattern_op_add(p, "EDIT");
... other pattern attributes
tt_pattern_callback_add(p, do_edit_message);
tt_pattern_register(p);
```

注 - コールバックは、登録したときと逆の順序で呼び出されます (つまり、最後に追加したコールバックが最初に呼び出されます)。

新しいメッセージを検索するために tt_message_receive を呼び出した場合、ToolTalk サービスがメッセージとパターンコールバックをどのように起動するかを図 11-1 に示します。

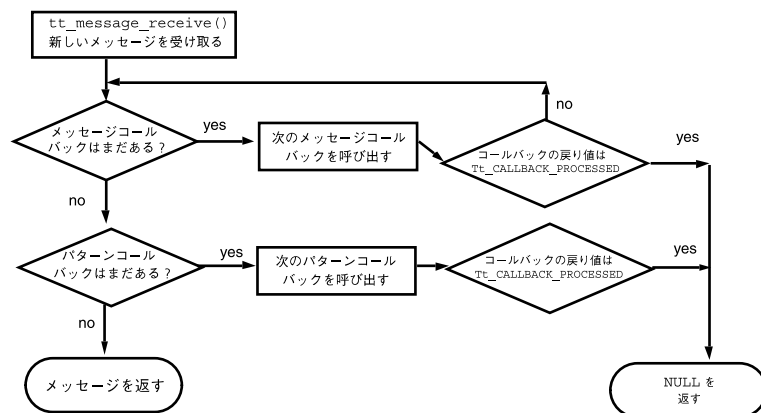


図 11-1 コールバックの起動方法

ハンドラにアドレス指定されたメッセージのコールバック

ToolTalk サービスは、ハンドラにアドレス指定されたメッセージの受信側を判別後、その受信側が登録したすべてのパターンに対してそのメッセージを照合します。(明示的にハンドラにアドレス指定されたメッセージは「ポイントツーポイント」のメッセージであるため、パターンの照合は行いません。)

- メッセージがパターンに一致しなかった場合、そのメッセージは通常の方法で配信されます。
- メッセージがパターンに一致した場合、そのパターンに付加されているコールバックがすべて起動されます。

静的パターンへのコールバックの付加

静的パターンを作成する際、数値タグ (opnum) を ptype の各シングニチャに付加できます。今回のバージョンでは、opnum にコールバックを付加できます。opnum を持つ静的パターンに一致したためにメッセージが配信されると、ToolTalk サービスはその opnum に付加されたコールバックをすべて調べ、ある場合は起動します。

要求の処理

ユーザーのプロセスが要求 (class=TT_REQUEST) を受信した場合、ユーザーは要求に
応答するか、要求を拒否するか、または要求を異常終了するかのどれかを選択しなけ
ればなりません。

要求への応答

要求に応答する場合は次の操作が必要です。

1. 要求された操作を実行します。
2. 任意の引数に TT_OUT または TT_INOUT のモードを埋め込みます。
3. メッセージに対して応答を送信します。

表 11-2 に、要求への応答時に使用する ToolTalk 関数を示します。

表 11-2 要求に応答するための関数

ToolTalk 関数	説明
<code>tt_message_arg_mode(Tt_message m, int n)</code>	引数のモード (入力、出力、入出力)。返される型は <code>Tt_mode</code>
<code>tt_message_arg_bval_set(Tt_message m, int n, const unsigned char *value, int len)</code>	指定されたバイト配列に引数の値を設定する。返される型は <code>Tt_status</code>
<code>tt_message_arg_ival_set(Tt_message m, int n, int value)</code>	指定された整数に引数の値を設定する。返される型は <code>Tt_status</code>
<code>tt_message_arg_val_set(Tt_message m, int n, const char *value)</code>	指定された文字列に引数の値を設定する。返される型は <code>Tt_status</code>
<code>tt_message_arg_xval_set(Tt_message m, int n, xdrproc_t xdr_proc, void *value)</code>	返される型は <code>Tt_status</code>
<code>tt_message_context_set(Tt_message m, const char *slotname, const char *value);</code>	指定された文字列にコンテキストを設定する。返される型は <code>Tt_status</code>
<code>tt_message_bcontext_set(Tt_message m, const char *slotname, unsigned char *value, int length);</code>	指定されたバイト配列にコンテキストを設定する。返される型は <code>Tt_status</code>

表 11-2 要求に応答するための関数 (続き)

ToolTalk 関数	説明
<code>tt_message_icontext_set(Tt_message m, const char *slotname, int value);</code>	指定された整数にコンテキストを設定する。返される型は <code>Tt_status</code>
<code>tt_message_xcontext_set(Tt_message m, const char *slotname, xdrproc_t xdr_proc, void *value)</code>	返される型は <code>Tt_status</code>
<code>tt_message_reply(Tt_message m)</code>	メッセージに応答する。返される型は <code>Tt_status</code>

要求の拒否または異常終了

要求の検査後、ユーザーのアプリケーションが現在その要求を処理できない場合は、表 11-3 に一覧表示された ToolTalk 関数を使って、要求を拒否または異常終了できます。

表 11-3 要求の拒否または異常終了

ToolTalk 関数	説明
<code>tt_message_reject(Tt_message m)</code>	メッセージを拒否する
<code>tt_message_fail(Tt_message m)</code>	メッセージを異常終了する
<code>tt_message_status_set(Tt_message m, int status)</code>	メッセージの状態を設定する。受信側アプリケーションは、この状態を見ることができる
<code>tt_message_status_string_set(Tt_message m, const char *status_str)</code>	メッセージの状態を記述するテキストを設定する。受信側アプリケーションは、このテキストを見ることができる

これらの要求の返される型は `Tt_status` です。

要求の拒否

要求の検査後、ユーザーのアプリケーションが現在その要求を処理できないが別のアプリケーションを使えば処理できる場合は、`tt_message_reject` を使ってその要求を拒否します。

ユーザーが要求を拒否すると、ToolTalk サービスはその要求を処理するための別の受信側を見つけようとします。ToolTalk サービスは、現在実行中のハンドラを発見できない場合、処置属性を検査し、メッセージを待ち行列に入れるか、適切なメッセージパターンが入った `ptype` でアプリケーションを起動しようとします。

要求の異常終了

要求の検査後、ユーザーまたはユーザーと同じ `p_type` の他のプロセスが要求された操作を実行できない場合は、`tt_message_fail` を使って、その操作が実行できないことを ToolTalk サービスに知らせます。ToolTalk サービスは、要求が異常終了したことを送信側に知らせます。

送信側に要求が異常終了した理由を知らせるには、`tt_message_fail` を呼び出す前に、`tt_message_status_set` または `tt_message_status_string_set` を使用します。

注 - `tt_message_status_set` によってユーザーが指定した状態コードは、`TT_ERR_LAST` より大きくなければなりません。

オフアの監視

使用プロセスで `TT_SENT` 状態のオフア (`class=TT_OFFER`) を受信したときは、次のいずれか 1 つを実行する必要があります。

1. メッセージの `tt_message_accept()` を呼び出してオフアを受信します。これにより、送信側の `procid` に対して受信側の `procid` がオフアを受信したことを知らせます。
2. メッセージの `tt_message_reject()` を呼び出してオフアを拒絶します。これにより、送信側の `procid` に対して受信側の `procid` がオフアを拒絶したことを知らせます。
3. メッセージの `tt_message_destroy()` を呼び出して、オフアを最初に受け入れたり拒絶したりせずに忌避します。これにより、送信側の `procid` に対して、受信側の `procid` がオフアを忌避したことを知らせます。
4. メッセージの `tt_message_receive()` をもう一度呼び出して、オフアを最初に受け入れたり拒絶したりせずに忌避します。この場合も、送信側の `procid` に対して受信側の `procid` がオフアを忌避したことを知らせます。
5. `tt_close()` を呼び出すか、(正常終了あるいは異常終了で) 終了させることによって ToolTalk サービスを切断します。この場合、クライアントプロセスが接続されている `ttsession` プロセスは、そのクライアントプロセスをオフアから忌避していると見なします。

ハンドラがある場合は、ハンドラと監視者のすべてがメッセージを受け入れるか、拒絶または忌避した場合は、メッセージ状態 (`Tt_state`) は `TT_RETURNED` に設定されます。他のメッセージクラスにないオフアの間状態は、次のように定義されます。

1. `TT_ACCEPTED` — 受信側がオフアに対して `tt_message_accept()` を実行すると、オフアはこの状態になります。

2. TT_REJECTED — 受信側がオフアに対して `tt_message_reject()` を実行すると、オフアはこの状態になります。
3. TT_ABSTAINED — 受信側が上記の 3、4、5 のいずれかを選択すると、オフアはこの状態になります。

メッセージの削除

メッセージの処理が完了し、メッセージの情報がなくなった場合は、`tt_message_destroy` を使ってメッセージを削除し、記憶領域の容量を解放します。

第 12 章

オブジェクト

この章では、アプリケーションが作成および管理するオブジェクトの ToolTalk 仕様 (スペック) の作成方法について説明します。オブジェクトの型を識別できるようになる前に、otype を定義して、その otype を ToolTalk 型データベースに格納する必要があります。otype の詳細は、第 10 章を参照してください。

ToolTalk サービスは、仕様と otype 情報を使用して、オブジェクト指向メッセージの受信側を決定します。

注 - ToolTalk オブジェクト指向メッセージインタフェース用にコーディングされたプログラムは、ソースコードを変更しなければ CORBA 準拠のシステムに移植できません。

オブジェクト指向メッセージ方式

オブジェクト指向メッセージは、アプリケーションが管理するオブジェクトにアドレス指定されます。オブジェクト指向メッセージ方式を使用するには、プロセス指向メッセージ方式と ToolTalk のオブジェクトの概念について詳しく知る必要があります。

オブジェクトデータ

オブジェクトデータは、図 12-1 に示すように 2 つの部分に分けて格納されます。

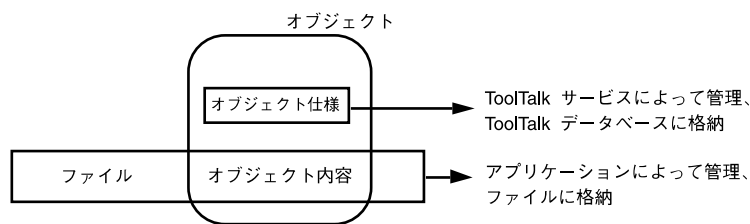


図 12-1 ToolTalk オブジェクトデータ

1つの部分を「オブジェクト内容」と呼びます。オブジェクト内容は、そのオブジェクトを作成または管理するアプリケーションが管理する通常ファイルの一部または複数の部分です。たとえば、パラグラフ、ソースコード関数、スプレッドシートのセルの集まりなどです。

オブジェクトのもう1つの部分は、「オブジェクト仕様」(「スペック」)と呼ばれます。スペックには、オブジェクトの型、オブジェクト内容が入っているファイル名、オブジェクトの所有者などの標準プロパティが入っています。また、アプリケーションはそれ自体のプロパティ(たとえば、ファイル内のオブジェクト内容の場所)をスペックに追加することもできます。アプリケーションがスペックに追加情報を格納できるので、ファイルのフォーマットを変更せずに、既存のファイル内のデータをオブジェクトとして識別できます。また、読み取り専用ファイルの一部からオブジェクトを作成できます。アプリケーションはスペックを作成し、`rpc.ttdbserverd`が管理する ToolTalk データベースに書き込みます。

注 - 読み取り専用ファイルシステムに存在しているファイルの中には、オブジェクトを作成できません。ToolTalk サービスは、オブジェクトが入っているファイルシステム内にデータベースを作成できなければなりません。

「ToolTalk オブジェクト」は、ToolTalk スペックが作成されるアプリケーションデータの一部分です。

オブジェクト仕様の作成

メッセージをオブジェクトに配信するよう ToolTalk サービスに指示するには、オブジェクトとその `otype` を識別する仕様を作成します。表 12-1 に、オブジェクト仕様の作成と書き込みに使用する ToolTalk 関数を示します。

表 12-1 作成用関数

ToolTalk 関数	説明
<code>tt_spec_create(const char *filepath)</code>	仕様を作成する。返される型は <i>char *</i>
<code>tt_spec_prop_set(const char *objid, const char *propname, const char *value)</code>	指定された文字列の値にプロパティを設定する。返される型は <i>Tt_status</i>
<code>tt_spec_prop_add(const char *objid, const char *propname, const char *value)</code>	文字列プロパティを追加する。返される型は <i>Tt_status</i>
<code>tt_spec_bprop_add(const char *objid, const char *propname, const unsigned char *value, int length)</code>	バイト配列プロパティを追加する。返される型は <i>Tt_status</i>
<code>tt_spec_bprop_set(const char *objid, const char *propname, const unsigned char *value, int length)</code>	指定されたバイト配列値にプロパティを設定する。返される型は <i>Tt_status</i>
<code>tt_spec_type_set(const char *objid, const char *otid)</code>	仕様のオブジェクト型を設定する。返される型は <i>Tt_status</i>
<code>tt_spec_write(const char *objid)</code>	データベースに仕様を書き込む。返される型は <i>Tt_status</i>

オブジェクト仕様をメモリーに作成し、そのオブジェクトの `objid` を取得するには、`tt_spec_create` を使用します。

otype の割り当て

オブジェクト仕様に `otype` を割り当てるには、`tt_spec_type_set` を使用します。仕様が最初に書き込まれる前に、その型を設定しなければなりません。この型は変更できません。

注 - `otype` を割り当てないでオブジェクト仕様を作成する場合、または ToolTalk 型データベースにとって不明の `otype` を持つオブジェクト仕様を作成する場合、そのオブジェクトにアドレス指定されたメッセージは配信できません。(ToolTalk サービスは、ユーザーが指定した `otype` が ToolTalk 型データベースにとって既知であるかどうかは検査しません。)

オブジェクト仕様プロパティの決定

ユーザーは、オブジェクトに関連付けたい「プロパティ」を決定できます。つまり、関連付けたいプロパティを仕様に追加します。ToolTalk サービスでは、ユーザー専用の内部データに、たとえば通常の ASCII 形式テキストファイル内のオブジェクト用

objid のような情報をいつでも格納できるわけではありません。まず、仕様プロパティ内の objid の場所を格納して、さらにこの場所を利用すれば、ツールの内部データ構造のどこにオブジェクトがあるかを識別できます。

仕様プロパティは、ユーザーにとっても便利なものです。ユーザーは、コメントまたはオブジェクト名などのプロパティを後で見ることができるオブジェクトに関連付けたいことがあります。ユーザーのアプリケーションまたは別の ToolTalk を使ったツールが、ユーザーに代わってオブジェクトに関連付けたプロパティの検索と表示を行います。

仕様プロパティの格納

仕様内にプロパティを格納するには、`tt_spec_prop_set` を使用します。

プロパティへの値の追加

プロパティに関連する値のリストに追加するには、`tt_spec_prop_add` を使用します。

オブジェクト仕様の書き込み

`otype` を設定してオブジェクト仕様にプロパティを追加した後に `tt_spec_write` を使うと、そのプロパティは永続的な ToolTalk 項目となり、他のアプリケーションから見えるようになります。`tt_spec_write` を呼び出すと、ToolTalk サービスは ToolTalk データベースにその仕様を書き込みます。

オブジェクト仕様の更新

既存のオブジェクト仕様プロパティを更新するには、既存の仕様の objid を指定し、`tt_spec_prop_set` と `tt_spec_prop_add` を使用します。仕様プロパティが更新された場合は、`tt_spec_write` を使って ToolTalk データベースにその変更を書き込みます。

既存の仕様を更新中に、`tt_spec_write` を呼び出した時点で ToolTalk サービスが `TT_WRN_STALE_OBJID` を返した場合、ToolTalk データベース内のオブジェクトを指す転送ポイントが見つかったこととなります。これは、オブジェクトが移動していることを示します。新しい objid を取得するには、古い objid が入ったオブジェクト

メッセージを作成し、このメッセージを送信します。ToolTalk サービスは、同じ状態コードの `TT_WRN_STALE_OBJID` を返しますが、新しい `objid` を入れるためにメッセージの `objid` 属性を更新します。`tt_message_object` を使って、メッセージから新しい `objid` を検索し、新しい `objid` を内部データ構造に配置します。

オブジェクト仕様の管理

ToolTalk サービスでは、オブジェクト仕様の検査、比較、照会、および移動を行う関数を提供しています。表 12-2 に、オブジェクト仕様を管理するために使用する ToolTalk 関数を示します。

表 12-2 オブジェクト仕様を管理する関数

返される型	ToolTalk 関数	説明
char *	<code>tt_spec_file(const char *objid)</code>	仕様があるファイル名
char *	<code>tt_spec_type(const char *objid)</code>	仕様のオブジェクト型
char *	<code>tt_spec_prop(const char *objid, const char *propname, int i)</code>	(ゼロから数えて)「i」番目のプロパティ値を文字列として検索する
int	<code>tt_spec_prop_count(const char *objid, const char *propname)</code>	このプロパティ名での値の数
Tt_status	<code>tt_spec_bprop(const char *objid, const char *propname, int i, unsigned char **value, int *length)</code>	このプロパティ名でのバイト配列の値の数
char *	<code>tt_spec_propname(const char *objid, int i)</code>	「i 番目」のプロパティ名
int	<code>tt_spec_propnames_count(const char *objid)</code>	この仕様のプロパティ数
char *	<code>tt_objid_objkey(const char *objid)</code>	仕様 ID に固有のキー
Tt_status	<code>tt_file_objects_query(const char *filepath, Tt_filter_function filter, void *context, void *accumulator)</code>	オブジェクト仕様用のデータベースを照会する
int	<code>tt_objid_equal(const char *objid1, const char *objid2)</code>	2つの仕様 ID が等しいかどうかを検査する
char *	<code>tt_spec_move(const char *objid, const char *newfilepath)</code>	オブジェクト仕様を新しいファイルに移動する

仕様情報の検査

指定された ToolTalk 関数によって、次の仕様情報を検査できます。

- オブジェクトが入ったファイルのパス名を調べる場合は、`tt_spec_file` を使用します。
- このオブジェクトの `otype` を調べる場合は、`tt_spec_type` を使用します。
- 仕様に格納されたプロパティを調べる場合は、`tt_spec_prop` または `tt_spec_bprop` を使用します。

オブジェクト仕様の比較

2つの `objid` を比較するには、`tt_objid_equal` を使用します。`tt_objid_equal` は、1つの `objid` が他の `objid` に対する転送ポインタの場合でも 1 を返します。

ファイル内の特定の仕様の照会

ファイル内の特定の仕様を照会するためにフィルタ関数を作成し、対象とする仕様を取得します。

`tt_file_objects_query` を使って、指定されたファイル内のすべてのオブジェクトを見つけます。ToolTalk サービスは、各オブジェクトを見つけるとフィルタ関数を呼び出し、そのオブジェクトの `objid` とアプリケーションが与えた2つのポインタをそのフィルタ関数に渡します。フィルタ関数は演算をいくつか行い、`Tt_filter_action` の値 (`TT_FILTER_CONTINUE` または `TT_FILTER_STOP`) を返します。これは照会を続行するか、検出を終了してすぐに復帰するかを示します。

例 12-1 は、仕様リストの取得方法を示します。

例 12-1 仕様リストの取得

```
/*
 * Called to update the scrolling list of objects for a file. Uses
 * tt_file_objects_query to find all the ToolTalk objects.
 */
int
cntl_update_obj_panel()
{
    static int list_item = 0;
    char *file;
    int i;

    cntl_objid = (char *)0;

    for (i = list_item; i >= 0; i--) {
        xv_set(cntl_ui_olist, PANEL_LIST_DELETE, i, NULL);
    }

    list_item = 0;
    file = (char *)xv_get(cntl_ui_file_field, PANEL_VALUE);
    if (tt_file_objects_query(file,
                              (Tt_filter_function)
cntl_gather_specs,
                              &list_item, NULL) != TT_OK) {
        xv_set(cntl_ui_base_window, FRAME_LEFT_FOOTER,
               "Couldn't query objects for file", NULL);
        return 0;
    }

    return 1;
}
```

tt_file_objects_query 関数内では、アプリケーションは、オブジェクトをスクロールリストに挿入するフィルタ関数 cntl_gather_specs を呼び出します。例 12-2 に objid の挿入方法を示します。

例 12-2 objid の挿入

```
/*
 * Function to insert the objid given into the scrolling lists of objects
 * for a file. Used inside tt_file_objects_query as it iterates through
 * all the ToolTalk objects in a file.
 */
Tt_filter_action
cntl_gather_specs(objid, list_count, acc)
    char *objid;
    void *list_count;
    void *acc;
{
    int *i = (int *)list_count;

    xv_set(cntl_ui_olist, PANEL_LIST_INSERT, *i,
          PANEL_LIST_STRING, *i, objid,
          NULL);

    *i = (*i + 1);

    /* continue processing */
    return TT_FILTER_CONTINUE;
}
```

オブジェクト仕様の移動

objid には、仕様情報が格納された特定のファイルシステムを指すポインタが入っています。仕様によって記述されたオブジェクトと同じように仕様情報を利用できるようにするため、ToolTalk サービスは仕様情報をそのオブジェクトと同じファイルシステムに格納します。そのため、オブジェクトが移動すると、仕様も移動しなければなりません。

tt_spec_move を使用して、オブジェクトが 1 つのファイルから別のファイルにいつ移動したか (たとえば、カット&ペースト操作などで) を ToolTalk サービスに知らせます。

- 新しい objid が不要な場合 (新旧のファイルが両方同じファイルシステム内にあるために)、ToolTalk サービスは TT_WRN_SAME_OBJID を返します。
- オブジェクトが別のファイルシステムに移動した場合、ToolTalk サービスはそのオブジェクトの新しい objid を返し、古い objid から新しい objid を指す転送ポインタを ToolTalk データベースに残します。

プロセスが古い objid にメッセージを送信する場合 (つまり、転送ポインタを持つ objid)、tt_message_send は特殊な状態コード TT_WRN_STALE_OBJID を返し、メッセージ内のオブジェクト属性を新しい場所の同じオブジェクトを指す新しい objid と置き換えます。

注 - オブジェクトを参照する内部データ構造体は、すべて新しい objid に更新してください。

オブジェクト仕様の削除

tt_spec_destroy を使用して、オブジェクト仕様をすぐに削除します。

オブジェクトおよびファイル情報の管理



注意 - ToolTalk サービスと統合アプリケーションがあるにも関わらず、rm や mv などの標準のオペレーティングシステムコマンドによるファイルの削除、移動、または名前の変更を行うことによって、オブジェクトの参照が削除されることがあります。参照が削除されると、メッセージが配信できなくなります。

オブジェクトデータが入ったファイルの管理

オブジェクトデータが入ったファイルが格納されているディスクパーティションのサービスを行う ToolTalk データベースを最新にしておくために、ファイルをコピー、移動、または削除する ToolTalk 関数を使用します。表 12-3 に、ユーザーがオブジェクトデータの入ったファイルの管理に使用する ToolTalk 関数を示します。

表 12-3 オブジェクトデータファイルをコピー、移動、および削除する関数

ToolTalk 関数	説明
tt_file_move(const char *oldfilepath, const char *newfilepath)	ファイルと ToolTalk オブジェクトデータを移動する
tt_file_copy(const char *oldfilepath, const char *newfilepath)	ファイルと ToolTalk オブジェクトデータをコピーする
tt_file_destroy(const char *filepath)	ファイルと ToolTalk オブジェクトデータを削除する

これらの関数から返される型は Tt_status です。

ToolTalk 情報が入ったファイルの管理

ToolTalk サービスは、ToolTalk オブジェクトとファイル情報をコピー、移動、および削除するための ToolTalk の拡張シェルコマンドを提供しています。アプリケーションのユーザーが、メッセージ内で参照されるファイルとオブジェクトの入ったファイルをコピー、移動、および削除するために使用する ToolTalk の拡張シェルコマンドを表 12-4 に示します。

表 12-4 ToolTalk の拡張シェルコマンド

コマンド	説明
ttcp	ファイルを新しい場所にコピーする。ToolTalk データベース内のファイルとオブジェクトの場所についての情報を更新する
ttmv	ディレクトリ名またはファイル名を変更する。ToolTalk データベース内のファイルとオブジェクトの場所についての情報を更新する
ttrm	指定されたファイルを削除する。ToolTalk データベース内のファイルとオブジェクトの情報を削除する
ttrmdir	ToolTalk オブジェクト仕様を関連付ける空のディレクトリ (ファイルが入っていないディレクトリ) を削除する (ディレクトリ用にオブジェクト仕様を作成できる。オブジェクト仕様を作成されると、ファイルまたはディレクトリのパス名が与えられる) ToolTalk データベースからオブジェクト情報を削除する
tttar	複数のファイルとオブジェクト情報を tarfile と呼ばれる単一のアーカイブに保存 (またはこのアーカイブから抽出) する。ToolTalk ファイルとオブジェクト情報を tarfile に保存 (または tarfile から抽出) するためにも使用できる

オブジェクト指向メッセージの例

edit_demo というプログラムを実行すれば、ToolTalk のオブジェクト指向メッセージのデモを見ることができます。このデモプログラムは、cnt1 と edit という 2 つのプログラムで構成されています。cnt1 の方は ToolTalk サービスを使用して、指定されたファイルを編集する編集プロセスを起動します。edit を使用すれば、ToolTalk オブジェクトを作成し、その指定されたファイル中のテキストと関連付けることができます。オブジェクトを作成してテキストと関連付けると、cnt1 を使用してそのファイルにオブジェクトを照会し、そのオブジェクトにメッセージを送信できます。

例 12-3 は、C 言語形式のコメントを前後につけてユーザーのオブジェクトを作成します。2 つのコード例から成り立ちます。オブジェクト仕様を作成して otype を設定し、その仕様を ToolTalk データベースに書き込むことにより、ユーザーの選択をラップします。さらに、アプリケーションは ToolTalk_EditDemo_new_object 操作を使って新しいオブジェクトを作成し、メッセージを監視する他のアプリケー

ションを更新した後、手続きアドレス指定された通知も送信します。他のアプリケーションが、ToolTalk_EditDemo が管理するファイル内のオブジェクトのリストを表示している場合、この通知を受信した後にリストを更新します。

例 12-3 オブジェクトの作成 -1

```
/*
 * Make a ToolTalk spec out of the selected text in this textpane. Once
 * the spec is successfully created and written to a database, wrap the
 * text with C-style comments in order to delimit the object and send out
 * a notification that an object has been created in this file.
 */
Menu_item
edit_ui_make_object(item, event)
Panel_item      item;
Event           *event;
{
    int          mark = tt_mark();
    char         *objid;
    char         *file;
    char         *sel;
    Textsw_index first, last;
    char         obj_start_text[100];
    char         obj_end_text[100];
    Tt_message   msg;

    if (! get_selection(edit_ui_xserver, edit_ui_textpane,
                        &sel, &first, &last)) {
        xv_set(edit_ui_base_window, FRAME_LEFT_FOOTER,
               "First select some text", NULL);
        tt_release(mark);
        return item;
    }
    file = tt_default_file();

    if (file == (char *)0) {
        xv_set(edit_ui_base_window, FRAME_LEFT_FOOTER,
               "Not editing any file", NULL);
        tt_release(mark);
        return item;
    }
}
```

例 12-4 オブジェクトの作成 - 2

```
/*
/* create a new spec */

objid = tt_spec_create(tt_default_file());
if (tt_pointer_error(objid) != TT_OK) {
    xv_set(edit_ui_base_window, FRAME_LEFT_FOOTER,
           "Couldn't create object", NULL);
    tt_release(mark);
    return item;
}

/* set its otype */

tt_spec_type_set(objid, "Sun_EditDemo_object");
if (tt_spec_write(objid) != TT_OK) {
    xv_set(edit_ui_base_window, FRAME_LEFT_FOOTER,
           "Couldn't write out object", NULL);
    tt_release(mark);
    return item;
}

/* wrap spec's contents (the selected text) with C-style */
/* comments. */

sprintf(obj_start_text, " /* begin_object(%s) */", objid);
sprintf(obj_end_text, " /* end_object(%s) */", objid);
(void)wrap_selection(edit_ui_xserver, edit_ui_textpane,
                    obj_start_text, obj_end_text);

/* now send out a notification that we've added a new object */

msg = tt_pnotice_create(TT_FILE_IN_SESSION, "Sun_EditDemo_new_object");
tt_message_file_set(msg, file);
tt_message_send(msg);

tt_release(mark);
return item;
}
```


第 13 章

情報記憶領域の管理

アプリケーションの記憶領域管理を単純化するために、ToolTalk サービスは、アプリケーションが ToolTalk サービスに提供するすべての情報をコピーし、アプリケーションに返す情報のコピーをユーザーに提供します。

ToolTalk サービスに提供される情報

ToolTalk サービスにポインタを与えると、そのポインタによって参照される情報がコピーされます。その結果、ユーザーが与えた情報を処理できます。ToolTalk サービスは、その情報を検索するためにそのポインタを再度使用しません。

ToolTalk サービスが提供する情報

ToolTalk サービスは、ToolTalk サービスがユーザーに提供する情報を格納するために、ToolTalk API ライブラリに割り当てスタックを提供しています。たとえば、`tt_default_session` を使ってデフォルトセッションの `sessid` を要求すると、ToolTalk サービスは `sessid` が入っている割り当てスタック (`char * ポインタ`) に文字列のアドレスを返します。`sessid` を検索後、その文字列を処理し、割り当てスタックを再配置できます。

注 - API 割り当てスタックを、プログラムの実行時スタックと混同しないようにしてください。API スタックは、指示があるまで情報を破棄しません。

情報の記憶領域管理のための呼び出し

ToolTalk サービスでは、ToolTalk API 割り当てスタック内の情報の記憶領域を管理するために、表 13-1 に示した呼び出しを提供しています。

表 13-1 ToolTalk 記憶領域の管理

返される型	ToolTalk 関数	説明
int	tt_mark(void)	一連の関数が返す情報にマークを付ける
void	tt_release(int mark)	一連の関数が返す情報を解放する
caddr_t	tt_malloc(size_t s)	ユーザー用に、割り当てスタック内に指定された量の記憶領域を確保する
void	tt_free(caddr_t p)	tt_malloc が確保した記憶領域を解放する。この関数は、ToolTalk API が返したアドレスを使用し、関連付けられた記憶領域を解放する

情報のマーク付けと解放

tt_mark() 関数と tt_release() 関数は、情報の記憶領域を管理しやすくするための一般的な機構です。tt_mark() 関数と tt_release() 関数は、通常、ルーチンが終了して ToolTalk サービスによって返された情報がなくなった場合に、ルーチンの開始と終了時に使用されます。

記憶領域の情報のマーク付け

ToolTalk サービスに記憶領域の開始をマークするよう要求するには、tt_mark を使用します。ToolTalk サービスは、API スタック上の場所を表す整数のマークを 1 つ返します。ToolTalk サービスがそれ以降に返す情報はすべて、そのマークの後ろに格納されます。

不要になった情報の解放

記憶領域に格納されている情報が不要になった場合は、tt_release() を使用して、不要になった情報の開始位置を示すマークを指定します。

情報のマーク付けと解放の例

例 13-1 は、メッセージ内の情報を検査するルーチンの開始位置で tt_mark() を呼び出します。このルーチン内で検査された情報が不要になってメッセージが破棄されたとき、tt_release() がそのマークとともに呼び出され、スタック上の記憶領域を解放します。

例 13-1 記憶領域の情報のマーク付け

```
/*
 * Get a storage mark so we can easily free all the data
 * ToolTalk returns to us.
 */

mark = tt_mark();

if (0==strcmp("ttsample1_value", tt_message_op(msg_in))) {
    tt_message_arg_ival(msg_in, 0, &val_in);
    xv_set(gauge, PANEL_VALUE, val_in, NULL);
}

tt_message_destroy(msg_in);
tt_release(mark);
return;
```

記憶領域の割り当てと解放

tt_malloc() 関数と tt_free() 関数は、割り当てられた記憶領域の割り当てを管理しやすくするための一般的な機構です。

記憶領域の割り当て

tt_malloc() は、使用する割り当てスタック内に指定された量の記憶領域を確保します。たとえば、tt_malloc() を使用して、記憶領域の場所を作成し、デフォルトセッションの sessid をそこにコピーできます。

割り当て済み記憶領域の解放

ToolTalk サービスがポインタを与える個々のオブジェクトの記憶領域を解放するには、tt_free() を使用します。たとえば、sessid を検査した後で sessid を格納する API 割り当てスタック内の容量を解放できます。tt_free() は、割り当てスタック内のアドレス (tt_malloc() から返された char * ポインタまたはアドレス) を引数としてとります。

特殊な実装方法: コールバックルーチンとフィルタルーチン

フィルタ関数とコールバックに渡された情報によって ToolTalk サービスの動作を変える特殊な実装方法もあります。ToolTalk サービスが呼び出すコールバックルーチンとフィルタルーチンは、次に示す 2 種類の引数を伴って呼び出されます。

- コンテキスト引数 — コールバックを呼び出した API 呼び出しに渡した引数。これらの引数は、アプリケーションが所有する項目を指します。
- API オブジェクトを指すポインタ — 記憶領域内のメッセージまたはパターン属性のアドレス

コンテキスト引数は、ToolTalk サービスからアプリケーションに渡されます。ポインタが参照する API オブジェクトは、コールバックまたはフィルタ関数が戻るとすぐに、ToolTalk サービスによって解放されます。これらのオブジェクトのどれかを保持したい場合は、関数が戻る前にそのオブジェクトをコピーしなければなりません。

注 - フィルタ関数とコールバックに渡された情報に対する ToolTalk サービスの動作方法は特殊な場合です。他の場合はユーザーが解放するまでは、ToolTalk サービスが API 割り当てスタック内に情報を格納します。

コールバックルーチン

ToolTalk サービスの機能の 1 つとして、メッセージ、パターン、およびフィルタのコールバックサポートがあります。コールバックは、特定のメッセージが届いたとき(「メッセージコールバック」)、またはメッセージが登録済みの特定のパターンに一致するとき(「パターンコールバック」)に ToolTalk が呼び出すプログラム内のルーチンです。

これらのコールバックについて ToolTalk サービスに知らせるには、メッセージを送信する前またはパターンを登録する前に、メッセージまたはパターンにコールバックを追加します。

フィルタルーチン

`tt_file_objects_query()` などのファイル照会関数を呼び出す場合、照会から項目を返すときに、ToolTalk サービスが呼び出すフィルタルーチンを指します。たとえば、特定のオブジェクトを見つけるには、ToolTalk ファイル照会関数を使用するフィルタルーチンを使用できます。

`tt_file_objects_query()` 関数は、ファイルにすべてのオブジェクトを返し、ユーザーが与えるフィルタルーチンを介してそのオブジェクトを実行します。一度フィルタルーチンが指定されたオブジェクトを見つければ、`tt_malloc()` を使用して記憶場所を作成し、オブジェクトをそこにコピーできます。フィルタ関数が戻ると、ToolTalk サービスはファイル内でオブジェクトが使用する記憶領域をすべて解放しますが、`tt_malloc()` 呼び出しによって格納したオブジェクトは、それ以降も利用できます。

第 14 章

エラー処理

ToolTalk サービスはエラー状態をグローバル変数ではなく、関数の戻り値で返します。ToolTalk 関数は、次のいずれかのエラー値を返します。

- `Tt_status`
- `int`
- `char*` または隠されたハンドル

返される型はそれぞれエラーが発生したかどうかを判定するために、異なる処理をされます。たとえば、`tt_default_session_set` の戻り値は `Tt_status` コードです。ToolTalk サービスは、指定された `sessid` にデフォルトセッションを設定します。

- 問題がない場合 — 返される `Tt_status` コードは `TT_OK`
- 問題がある場合 — 返される `Tt_status` コードは `TT_ERR_SESSION`。この状態コードは、渡した `sessid` が有効ではなかったことを通知します。

ToolTalk エラー状態の検索

エラー値の検索には、表 14-1 に示す ToolTalk エラー処理関数を使用できます。

表 14-1 ToolTalk エラー状態の検索

ToolTalk 関数	説明
<code>tt_pointer_error(char * return_val)</code>	ポインタ内で符号化されたエラーを返す
<code>tt_pointer_error((void *) (p))</code>	<code>VOID *</code> にキャストされたポインタ内で符号化されたエラーを返す
<code>tt_int_error(int return_val)</code>	整数内で符号化されたエラーを返す

これらの関数の返される型は、`Tt_status` です。

ToolTalk エラー状態の確認

エラー値を確認するには、表 14-2 に示す ToolTalk エラーマクロを使用できます。

表 14-2 ToolTalk エラーマクロ

返される型	ToolTalk マクロ	展開
Tt_status	tt_is_err(status_code)	(TT_WRN_LAST < (status_code))

戻り値の状態

通常に戻り値を持つ関数と持たない関数の戻り値の状態について次に説明します。

通常に戻り値を伴う関数

ToolTalk 関数がポインタまたは整数などの通常に戻り値を持つ場合は、実際の値の代わりに特別な「エラー値」が返されます。

通常に戻り値を伴わない関数

ToolTalk 関数が通常に戻り値を持たない場合、戻り値は「列挙」型 Tt_status の要素です。

エラーがあるかどうかを確認するには、整数を返す ToolTalk マクロ tt_is_err を使用します。

- 戻り値が 0 の場合、「列挙」型 Tt_status は TT_OK か警告のどちらかです。
- 戻り値が 1 の場合、「列挙」型 Tt_status はエラーです。

エラーがある場合は、例 14-1 に示すように tt_status_message 関数を使用して、Tt_status コードを説明する文字列を取得できます。

例 14-1 エラー説明の取得

```
char *spec_id, my_application_name;
Tt_status tterr;

ttterr = tt_spec_write(spec_id);
if (tt_is_err(tterr)) {
    fprintf(stderr, "%s: %s\n", my_application_name,
            tt_status_message(tterr));
}
```

返されたポインタの状態

ポインタを返す ToolTalk 関数の実行中にエラーが発生すると、ToolTalk サービスは、正しい Tt_status コードを示す ToolTalk API ライブラリ内のアドレスを与えます。ポインタが有効かどうかを確認するには、ToolTalk マクロの tt_ptr_error を使用できます。ポインタがエラー値の場合は、tt_status_message を使用して Tt_status 文字列を取得できます。

例 14-2 では、エラー値が発見された場合、ポインタを確認し、Tt_status 文字列を検索して出力します。

例 14-2 返されたポインタ状態の検索

```
char *old_spec_id, new_file, new_spec_id, my_application_name;
Tt_status tterr;

new_spec_id = tt_spec_move(old_spec_id, new_file);
tterr = tt_ptr_error(new_spec_id);
switch (tterr) {
    case TT_OK:
        /*
         * Replace old_spec_id with new_spec_id in my internal
         * data structures.
         */
        update_my_spec_ids(old_spec_id, new_spec_id);
        break;
    case TT_WRN_SAME_OBJID:
        /*
         * The spec must have stayed in the same filesystem,
         * since ToolTalk is reusing the spec id. Do nothing.
         */
        break;
    case TT_ERR_FILE:
    case TT_ERR_ACCESS:
    default:
        fprintf(stderr, "%s: %s\n", my_application_name,
            tt_status_message(tterr));
        break;
}
```

返された整数の状態

整数を返す ToolTalk 関数の実行中にエラーが発生すると、戻り値は配信範囲外になります。値が配信範囲内にある場合、tt_int_error 関数は TT_OK の状態を返します。

値が配信範囲外かどうかを検査するには、tt_is_err マクロを使用して、エラーまたは警告が発生したかどうかを判定できます。

文字列を検索して Tt_status コードを探すには、tt_status_message を使用できます。

例 14-3 では、返された整数を確認しています。

例 14-3 返された整数の確認

```
Tt_message msg;
int num_args;
Tt_status tterr;
char *my_application_name;

num_args = tt_message_args_count(msg);
tterr = tt_int_error(num_args);
if (tt_is_err(tterr)) {
    fprintf(stderr, "%s: %s\n", my_application_name,
            tt_status_message(tterr));
}
```

接続の異常終了

ToolTalk サービスは、ツールが予期せず終了したことをプロセスに知らせる関数を提供します。tt_message_send_on_exit 呼び出しを組み込むとき、次の2つのイベントのいずれか1つが発生するまで、ToolTalk サービスは内部でメッセージを待ち行列に入れます。

1. プロセスが tt_close を呼び出す。
この場合 ToolTalk サービスは、その待ち行列からメッセージを削除します。
2. ttssession サーバーとプロセスとの接続が壊された。たとえば、アプリケーションがクラッシュした場合などです。
この場合 ToolTalk サービスは、待ち行列に入れられたメッセージとパターンを照合し、メッセージを終了前に通常送信するのと同じ方法で配信します。

プロセスは tt_close を呼び出す前に tt_message_send を呼び出すことによって、正常終了のメッセージを送信することもできます。この場合、プロセスがその正常終了メッセージを送信しても、tt_close を呼び出す前にクラッシュすると、ToolTalk サービスは、正常終了メッセージと tt_message_send_on_exit メッセージの両方を対象のプロセスに配信します。

エラーの伝達

ポインタを受け付ける ToolTalk 関数は、ポインタがエラー値の場合、渡されるポインタを常に確認してから TT_ERR_POINTER を返します。この確認によって、どの呼び出しに対してもポインタの値を確認しないで、妥当な方法で呼び出しを結合することが可能になります。

例 14-4 では、メッセージが作成され、埋め込まれ、送信されます。
tt_message_create が異常終了すると、エラーオブジェクトが「m」に代入され、
すべての tt_message_XXX_set 呼び出しと tt_message_send 呼び出しが異常終
了します。各呼び出しの間で検査を行わないでエラーを検出するには、
tt_message_send からのリターンコードを検査します。

例 14-4 エラーのチェック

```
Tt_message m;  
  
m=tt_message_create();  
tt_message_op_set(m,"OP");  
tt_message_address_set(m,TT_PROCEDURE);  
tt_message_scope_set(m,TT_SESSION);  
tt_message_class_set(m,TT_NOTICE);  
tt_rc=tt_message_send(m);  
if (tt_rc!=TT_OK)...
```

付録 A

分類機構データベースから ToolTalk 型データベースへの移行

注 - バージョン 1.1 およびその互換 ToolTalk サービスでは、`ttsession` が分類機構 (CE) データベースからその型を読み取りません。また、ToolTalk 型コンパイラの `tt_type_comp` は CE データベースに型を併合しません。

この付録では、既存の ToolTalk を利用したアプリケーションを、CE データベースから ToolTalk 型データベースへ移行する方法について説明します。

ttce2xdr スクリプト

ToolTalk サービスは、`ttce2xdr` というスクリプトを提供します。これによって、CE データベース (ToolTalk サービスのバージョン 1.0.x によって使用されたデフォルトのデータベース) に格納された ToolTalk の型を、XDR 形式のデータベースに変換します。これは、バージョン 1.1 およびその互換 ToolTalk サービスによって使用されるデータベースです。

ユーザーデータベースの変換

ToolTalk 1.1. およびその互換 `ttsession` が初めて起動されるときに、ユーザー型データベースは CE データベースから新しい ToolTalk 型データベースに自動的に変換されます。ただし、現在のユーザーデータベースをコマンドを使用して自分で変換することもできます。

```
ttce2xdr [ -xnh ] -d user
```

表 A-1 では、ttce2xdr スクリプトのオプションについて説明します。

表 A-1 ttce2xdr スクリプトのオプション

オプション	説明
-x	ttce2xdr によって実行された基本的なコマンドを表示する
-n	ttce2xdr が実行できる基本的なコマンドを表示する
-h	ttce2xdr のオプションについて説明する
-d	変換するデータベース(「user」、「system」、または「network」)を指定する

型は CE データベース ~/.cetables/cetables から読み取られ、新しい ToolTalk 型データベース ~/.tt/types.xdr に書き込まれます。

システムデータベースの変換

システム CE データベースは、マシン単位のデータベースです。ToolTalk 型の各マシン上で、ttce2xdr スクリプトを実行する必要があります。システム CE データベースに ToolTalk 型があるかどうかを判定するには、コマンド行に次のコマンドを入力します。

```
tt_type_comp -Epd system
```

システム CE データベースに ToolTalk 型がない場合、出力は生成されません。

注 – システム CE データベースの ttce2xdr スクリプトを実行するには、スーパーユーザーとしてログインする必要があります。

システム CE データベースの ttce2xdr スクリプトを実行するには、コマンド行に次のコマンドを入力します。

```
ttce2xdr [ -xnh ] -d system
```

表 A-2 では、ttce2xdr スクリプトのオプションについて説明します。

表 A-2 ttce2xdr スクリプトのオプション

オプション	説明
-x	ttce2xdr によって実行された基本的なコマンドを表示する
-n	ttce2xdr が実行できる基本的なコマンドを表示する
-h	ttce2xdr のオプションについて説明する
-d	変換するデータベース (ユーザー、システム、またはネットワーク) を指定する

型は、CE データベース /etc/cetables/cetables から読み取られ、新しい ToolTalk 型データベース /etc/tt/types.xdr に書き込まれます。

ネットワークデータベースの変換

ネットワーク CE データベースは、OpenWindows™ のインストール単位のデータベースです。OpenWindows 3.x 製品と一っしょに出荷されたものとは異なる ToolTalk 型を持つ各ネットワーク CE データベースは、変換する必要があります。

注 - ネットワーク CE データベースの ttce2xdr スクリプトを実行するには、スーパーユーザーとしてログインする必要があります。

ネットワーク単位のデータベースを変換するには、コマンド行に次のコマンドを入力します。

```
ttce2xdr [ -xnh ] -d network [ OPENWINHOME-from [ OPENWINHOME-to ] ]
```

表 A-3 では、ttce2xdr スクリプトのオプションについて説明します。

表 A-3 ttce2xdr スクリプトのオプション

オプション	説明
-x	ttce2xdr によって実行された基本的なコマンドを表示する
-n	ttce2xdr が実行できる基本的なコマンドを表示する
-h	ttce2xdr のオプションについて説明する
-d	変換するデータベース (ユーザー、システム、またはネットワーク) を指定する
OPENWINHOME-from	このディレクトリの中のデータベースから型を読み取る。OPENWINHOME-to が設定されている場合は、指定されたディレクトリの中のデータベース型に書き込まれる。設定されていない場合は、環境変数 OPENWINHOME の現在の値が使用され、型が書き込まれるデータベースを検索する
OPENWINHOME-to	このディレクトリの中のデータベースへ型を書き込む。OPENWINHOME-from が設定されている場合は、指定されたディレクトリの中のデータベースから型が読み取られる。設定されていない場合は、環境変数 OPENWINHOME の現在の値が使用され、型が読み取られるデータベースを検索する

型は CE データベース \$OPENWINHOME/lib/cetables/cetables から読み取られ、新しい ToolTalk 型データベース \$OPENWINHOME/etc/tt/types.xdr に書き込まれます。

OpenWindows 3.x 製品と一っしょに出荷されたものとは異なる ToolTalk 型を、ネットワーク CE データベースからネットワーク XDR データベースに移動するには、コマンド行に次のコマンドを入力します。

```
ttce2xdr -d network old_OPENWINHOME new_OPENWINHOME
```

old_OPENWINHOME は移動元のネットワーク CE データベースを保持している OpenWindows のインストール場所、new_OPENWINHOME は ToolTalk の XDR データベースを更新する OpenWindows のインストール場所です。

付録 B

ToolTalk サービスのデモンストレーション

この付録では、ToolTalk サービスが、アプリケーションと他のアプリケーションとの通信をどのように可能にしているのかを簡単なデモンストレーションによって示します。

簡易化されたアプリケーション間通信

ToolTalk サービスは、アプリケーション統合用関数の完全なセットを提供します。ToolTalk サービスによって提供された機能を使うことで、既存のアプリケーションは互いに「話す」ことができます。

ToolTalk のデモンストレーションは簡単です。簡単なテキストエディタを使いながら、ロードされたファイル内に表示されているフォントを変更するためにフォント名の選択をインタフェースに尋ねます。ToolTalk のデモンストレーションは、次のような X11R4 からの 2 つのアプリケーションから成ります。

- Xedit — X 用の簡単なテキストエディタ
- Xfontsel — X11 フォント名を選択するためのポイント&クリックインタフェース

この付録では、これら 2 つのアプリケーションが連携できるように変更する方法を概説します。173 ページの「アプリケーションへの ToolTalk コードの追加」では、ToolTalk コードがどのようにソースコードファイルに取り組みられているかを示します。

連携機能の追加

ツールが連携できるようになる前に、.c、個々のアプリケーションの Makefile、および Xedit アプリケーションのヘッダーファイルを変更する必要があります。さらに Xfontsel アプリケーションの ToolTalk プロセス型 (ptype) を宣言するための新しいファイルを作成する必要があります。

「vi」などの標準エディタを使い、これらを変更して ptype ファイルを作成してください。

Xedit アプリケーションの変更

Xedit アプリケーションを変更し、Xfontsel アプリケーションと通信できるようにするには、次のファイルを変更する必要があります。

- xedit.h ファイル
- xedit.c ファイル
- commands.c ファイル
- Makefile

Xedit は、ToolTalk デモンストレーションのために、ToolTalk ヘッダーファイルと xedit.c ファイル内の新しい ToolTalk コマンドについて知る必要があります。xedit.h ファイルでの変更方法は、例 B-1 にコメントとともに示します。

次に、ToolTalk セッションを設定してフォント変更用のボタンを作成し、Xedit が ToolTalk メッセージを受信し処理できるように xedit.c ファイルにコードを追加する必要があります。これらのファイルでの変更方法は、例 B-2 にコメントとともに示されています。

commands.c ファイルにコードを追加して、フォント変更の完了時に応答を送る、または操作の異常終了時に通知するように、Xedit が Xfontsel アプリケーションに指示できるようにします。また、Xedit が何の操作を実行してほしいかを Xfontsel に知らせるコードを追加する必要があります。これらのファイルへの変更方法は、例 B-3 にコメントとともに示されています。

Xedit プログラムへの最後の変更は、ToolTalk ライブラリを使用するように Makefile を変更します。これを行うために、次のように -l`tt` オプションを追加します。

```
LOCAL_LIBRARIES = -ltt $(XAWLIB) $(XMULIB) $(XTOOLLIB) $(XLIB)
```

Xedit ファイルに対して指示された変更を行なった後、Xedit プログラムをコンパイルします。

Xfontsel アプリケーションの変更

Xfontsel アプリケーションを変更し、Xedit アプリケーションと通信できるようにするには、次のファイルを変更する必要があります。

- Xfontsel.c ファイル
- Makefile

また、Xfontsel アプリケーション用の ToolTalk ptype を宣言するための新しいファイルを作成する必要があります。

Xfontsel は、ToolTalk デモンストレーションのために、次のことを知る必要があります。

- ToolTalk ヘッダーファイルの見つけ方
- ToolTalk メッセージを受信したときの処理方法
- ToolTalk メッセージが原因となるエラーの処理方法
- 新しいフォント変更コマンド用の適用ボタンが有効になったときの動作

また Xfontsel は、フォントを変更するための適用ボタンとコマンドボックスを表示する必要があります。さらに、ToolTalk コールバックメッセージをいつ送信するかと、どのように ToolTalk セッションに参加するかを Xfontsel に知らせるために、コードを追加する必要があります。例 B-4 でコメントとともに示しているように、Xfontsel.c ファイルが変更されます。

次に、ToolTalk ライブラリを使用する Xfontsel プログラムの Makefile を変更します。これを行うために、次のように `-ltt` オプションを追加します。

```
LOCAL_LIBRARIES = -ltt $(XAWLIB) $(XMULIB) $(XTOOLLIB) $(XLIB)
```

ToolTalk 型機構は、ToolTalk サービスのルートメッセージを助けるように設計されています。最初にプロセス型 (ptype) を定義し、次にその ptype を ToolTalk 型コンパイラの `tt_type_comp` でコンパイルします。ToolTalk デモンストレーションについては、次に示すように Xfontsel アプリケーション用の ptype ファイルを作成する必要があります。

注 `-directory_name` は、変更された Xfontsel ファイルが入っているディレクトリへのパス名です。

```

ptype xfontsel {
    /* Process type identifier */
    start "/directory_name/xfontsel"; /* Start string */

handle:
    /* Receiving process */
    /* A signature is divided
    * into two parts by the => as follows:
    * Part 1 specifies how the message is to be matched;
    * Part 2 specifies what is to be taken when
    * a match occurs.
    */
    session GetFontName(out string fontname) => start;
}

```

ツールが ptype を宣言すると、その ptype 内で表示されたメッセージパターンが自動的に登録されます。ToolTalk は、受信するメッセージをそれらの登録されたパターンと照合します。これらの静的メッセージパターンは、そのツールが ToolTalk サービスとの通信を終了するまで有効です。

ptype ファイルを作成した後、その ptype をインストールする必要があります。これを行うために、次のように ToolTalk 型コンパイラを実行します。

```
machine_name% tt_type_comp xfontsel.ptype
```

xfontsel.ptype は、ユーザーの ptype ファイル名です。

Xfontsel ファイルに対して指示された変更を行い、ptype ファイルを作成し、その ptype をインストール後に Xfontsel プログラムをコンパイルします。

ツールコミュニケーション

ToolTalk 技術がどのように動作するかを見てみましょう。

1. **Xedit** アプリケーションを開始します。
次のコマンドを入力し、Xedit を開始します。

```
machine_name% xedit
```

図 B-1 のような画面が表示されます。

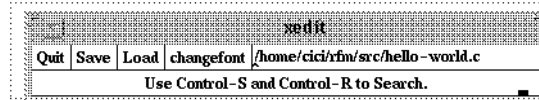


図 B-1 Xedit 画面

2. ファイルを読み込みます。
Xedit 画面に読み込んだファイルが表示されます。
3. 表示されたフォントを変更します。
 - a. Xedit 画面上で「**changefont**」ボタンをクリックします。
Xfontsel 画面が表示されます。
 - b. Xfontsel 画面上で新しいフォントを選択します。
 - c. Xfontsel 画面上で「**apply**」ボタンをクリックします。
フォントが変更された Xedit 画面が表示されます。

アプリケーションへの ToolTalk コードの追加

この節では、ToolTalk コードがどのように Xedit および Xfontsel アプリケーションに追加されたかを説明します。

- 省略記号 (...) は、コードがスキップされていることを表します。コード例は、挿入された ToolTalk コード行の前後数行だけを示します。
- ファイルに追加する必要がある実際の ToolTalk コードは、ボールド体で示します。次にその例を示します。

```
#include <desktop/tt_c.h>  
/* ToolTalk のヘッダー */
```

Xedit ファイルへの ToolTalk コードの追加

Xedit ファイルに対する変更は、170 ページの「Xedit アプリケーションの変更」で説明します。

例 B-1 Xedit.h ファイルの変更

```
/*
 *   rcs_id[] = "$XConsortium: xedit.h,v 1.18 89/07/21 19:52:58 kit Exp $";
 */

...

#include <X11/Xaw/Viewport.h>
#include <X11/Xaw/Cardinals.h>

/*
 * ToolTalk のデモンストレーション用に、以下のインクルード行を追加する
 */
#include <desktop/tt_c.h>
/* ToolTalk のヘッダ */

extern struct _app_resources {
Boolean enableBackups;
char *backupNamePrefix;
char *backupNameSuffix;

...

/* xedit.c に外部宣言 */

extern void Feep();
/*
 * ToolTalk のデモンストレーション用に、以下の外部宣言を追加する
 */
extern void processToolTalkMessage(); /* ToolTalk メッセージの処理 */
extern void dieFromToolTalkError();
        /* エラーが発生した場合は異常終了する */
extern Display *CurDpy;
        /* 表示 */

...

/* commands.c に外部宣言 */

...

extern void DoChangeFont();
        /* フォントを変更する */
```

例 B-2 変更された Xedit.c ファイル

```
#if (!defined(lint) && !defined(SABER)) \
static char Xrcsid[] = "$XConsortium: \
xedit.c,v 1.23 89/12/07 \
19:19:17 kit Exp $";
#endif          /* lint && SABER */

...
```

例 B-2 変更された Xedit.c ファイル (続き)

```
void main(argc, argv)
int argc;
char **argv;
{
Widget top;
String filename = NULL;
static void makeButtonsAndBoxes();

/*
 * ToolTalk のデモンストレーション用に、
 * 以下の行を追加する
 */
int ttmark;
/* ToolTalk のマーク */
int ttfd;
/* ToolTalk のファイル記述子 */
char *procid;
/* プロセス識別子 */
Tt_status ttrc;
/* ToolTalk の状態 */

top = XtInitialize( "xedit", \
"Xedit", NULL, 0, &argc, argv);

...

XtRealizeWidget(top);
XDefineCursor(XtDisplay(top), XtWindow(top), \
XCreateFontCursor( XtDisplay(top), \
XC_left_ptr));
/*
 * ToolTalk のデモンストレーション用に、
 * ToolTalk セッションをスタックの最初に
 * 作成して、デフォルトセッションとして
 * 設定するために、以下の行を追加する
 */
ttmark = tt_mark();
ttrc = tt_default_session_set(
/* 最上位ウィンドウを表示している .. */
tt_X_session(
/* .. X サーバーの .. */
DisplayString(
/* .. X セッションに ..*/
XtDisplay(top)))));
/* .. デフォルトセッションを設定する ... */
/*
 * デフォルトセッションがない場合は異常終了する
 */
dieFromToolTalkError( \
"tt_default_session_set", ttrc);
procid = tt_open();
/* ToolTalk の初期化 */
/*
```

例 B-2 変更された Xedit.c ファイル (続き)

```
        * プロセス識別子がない場合は異常終了する
        */
dieFromToolTalkError("tt_open" \
, tt_ptr_error(procid));
ttfd = tt_fd();
/* ToolTalk のファイル記述子 */
/*
 * ファイル記述子がない場合は異常終了する
 */
dieFromToolTalkError("tt_fd", \
tt_int_error(ttfd));
/*
 * ファイル記述子を起動する
 */
XtAddInput(ttfd, (XtPointer)XtInputReadMask, \
processToolTalkMessage, 0);

XtMainLoop();
}

...

MakeCommandButton(b_row, "load", DoLoad);
/*
 * ToolTalk のデモンストレーション用に、
 * フォント変更用コマンドのボタンを作成するために、*
 * 次の行を追加する
 */
MakeCommandButton(b_row, "changeFont", \
DoChangeFont);
filenamewindow = MakeStringBox(b_row, \
"filename", filename);
}
XtCreateManagedWidget("bc_label",
labelWidgetClass, outer, NULL, ZERO);

...

void Feep()
{
XBell(CurDpy, 0);
/*
 * ToolTalk のデモンストレーション用に、
 * 着信メッセージを受信して処理するために、
 * 以下の行を追加する
 */
}

void processToolTalkMessage()
/* ToolTalk メッセージを処理する */
{
    int ttmark;
```


例 B-2 変更された Xedit.c ファイル (続き)

```
/* ToolTalk のマーク */
Tt_message incoming;
/* 着信メッセージ */

ttmark = tt_mark();
/* ToolTalk のマーク */

incoming = tt_message_receive();
/* 着信メッセージを受信 */
/*
 * コールバックがメッセージを処理しなければならないので、
 * 返されたメッセージを取得してはならない
 */
if (incoming == 0) return;
/* 着信メッセージを戻す */

if (tt_is_err(tt_ptr_error(incoming))) {
    dieFromToolTalkError("tt_message_receive",
        tt_ptr_error(incoming));
}

/*
 * これは、認識していないメッセージである。
 * 開始の要求または通知があった場合に、異常終了する
 */

if (tt_message_class(incoming) == TT_REQUEST ||
    tt_message_status(incoming) ==
TT_WRN_START_MESSAGE) {
    tt_message_fail(incoming);
}
tt_message_destroy(incoming);
/* メッセージの削除 */
tt_release(ttmark);
/* 領域の解放 */
}

void dieFromToolTalkError(procname, errid)
char *procname;
Tt_status errid;
/* エラーが発生した場合は異常終了する */
{
    /*
     * 警告または TT_OK を無視してはならない
     */

    if (tt_is_err(errid)) {
        fprintf(stderr, "%s returned ToolTalk
error: %s\n",
            procname, tt_status_message(errid));
        exit(1);
    }
}

```

例 B-3 変更された commands.c ファイル

```
#if (!defined(lint) && !defined(SABER))
static char Xrcsid[] = "$XConsortium:
commands.c,v 1.27 89/12/10
      17:08:26 rws Exp $";
#endif /* lint && SABER */

...

#ifdef USG
int rename (from, to)
    char *from, *to;
{
    (void) unlink (to);
    if (link (from, to) == 0) {
        unlink (from);
        return 0;
    } else {
        return -1;
    }
}
#endif

/*
 * ToolTalk のデモンストレーション用に、
 * 操作が成功したか失敗したかについて
 * Xfontsel がコールバックを送信するよう
 * 以下の行を追加する
 */
static Tt_callback_action FinishChangeFont(m,p)
/* ToolTalk メッセージコールバック */
    Tt_message m;
    /* ToolTalk メッセージ */
    Tt_pattern p;
    /* ToolTalk パターン */
{
    static XFontStruct *fs;
    /* フォントの構造体 */
    int ttmark;
    /* ToolTalk のマーク */

    ttmark = tt_mark();
    /* ToolTalk のマーク */
    /*
     * 操作に失敗すると、ユーザーに通知する
     */
    if (TT_FAILED==tt_message_state(m)) {
        XeditPrintf("Font change failed\n");
        tt_message_destroy(m);
        /* メッセージの削除 */
    } else if (TT_HANDLED==tt_message_state(m)) {
        XFontStruct *newfs;
        /* 新しいフォントを読み込む */
        newfs =
```

例 B-3 変更された commands.c ファイル (続き)

```
XLoadQueryFont (CurDpy, tt_message_arg_val(m,0));
/* 新しいフォントが正常の場合、古いフォントがあれば
 * 読み込みを解除して、新しいフォントを使用する
 */
if (newfs) {
    if (fs) {
        XUnloadFont (CurDpy, fs->fid);
    }
    XtVaSetValues (textwindow, XtNfont, newfs, 0);
    fs = newfs;
}
tt_message_destroy(m);
/* メッセージの削除 */
}

tt_release(ttmark);
/* マークの解放 */
/*
 * 送信側に操作の終了を通知するために、
 * コールバックを処理する
 */
return TT_CALLBACK_PROCESSED;
}

void
DoChangeFont ()
/* フォントの変更 */
{
    Tt_message m;
    /* ToolTalk のメッセージ */
    Tt_status ttrc;
    /* ToolTalk の状態 */

    /*
     * 要求を作成
     */
    m = tt_prequest_create(TT_SESSION,
"GetFontName");
    /*
     * メッセージに引数を追加
     */
    tt_message_arg_add(m, TT_OUT, "string",
(char *)NULL);
    /*
     * 変更の終了を通知するため、コールバックを追加する
     */
    tt_message_callback_add(m, FinishChangeFont);
    /*
     * メッセージの送信
     */
    ttrc = tt_message_send(m);
    /*
     * エラーが発生した場合は異常終了する
     */
}

```

例 B-3 変更された `commands.c` ファイル (続き)

```
        dieFromToolTalkError("tt_message_send", ttrc);
    }
```

```
void DoSave()
{
```

Xfontsel ファイルへの ToolTalk コードの追加

Xfontsel ファイルに対する変更は、171 ページの「Xfontsel アプリケーションの変更」で説明しています。

例 B-4 変更された `Xfontsel.c` ファイル

```
#ifndef lint
static char Xrcsid[] = "$XConsortium:
  xfontsel.c,v 1.16 89/12/12 14:10:48 rws
  Exp $";
#endif

...

#include <X11/Xaw/Viewport.h>
#include <X11/Xmu/Atoms.h>

/*
 * ToolTalk のデモンストレーション用に、ToolTalk
 * のヘッダーファイルをインクルードするために
 * 以下の行を追加する
 */
#include <desktop/tt_c.h>
/* ToolTalk のヘッダーファイル */

#define MIN_APP_DEFAULTS_VERSION 1

...

void SetCurrentFont();
Boolean IsXLFDfontName();

/*
 * ToolTalk のデモンストレーション用に、
 * Xfontsel に ToolTalk メッセージの処理方法を
 * 通知するために、以下の行を追加する
 */
void dieFromToolTalkError();
/* エラーが発生した場合は異常終了する */
void processToolTalkMessage();
/* ToolTalk メッセージの処理 */
```

例 B-4 変更された Xfontsel.c ファイル (続き)

```
void ReplyToMessage();
    /* ToolTalk メッセージへの応答 */
Tt_message replymsg;

typedef void (*XtProc)();

...

int matchingFontCount;
static Boolean anyDisabled = False;
Widget ownButton;
/*
 * ToolTalk のデモンストレーション用に、
 * フォントを変更する適用ボタンを追加するために、
 * 次の行を追加する
 */
Widget applyButton;
    /* 適用ボタンを追加 */
Widget fieldBox;
/*
 * ToolTalk のデモンストレーション用に、
 * フォントを変更するコマンドボックスを追加するために、
 * 次の行を追加する
 */
Widget commandBox;
    /* commandBox を拡大 */
Widget countLabel;

...

void main(argc, argv)
    unsigned int argc;
    char **argv;
{
    Widget topLevel, pane;

/*
 * ToolTalk のデモンストレーション用に、
 * 以下の行を追加する
 */
int ttmark, ttfd;
    /* ToolTalk マークと ToolTalk ファイル記述子 */
char *procid;
    /* プロセス識別子 */
Tt_status ttrc;
    /* ToolTalk の状態 */

    topLevel = XtInitialize( NULL,
        "XFontSel", options, XtNumber(options),
        &argc, argv );

...

```

例 B-4 変更された Xfontsel.c ファイル (続き)

```
pane = XtCreateManagedWidget ("pane"
,panedWidgetClass,topLevel,NZ);
{
/*
 * ToolTalk のデモンストレーション用に、
 * コマンドボックスのウィジェットを拡大して、
 * Widget commandBox、fieldBox、
 * currentFontName、viewPort の行を
 * 以下のように変更する
 */
    Widget
/* commandBox, fieldBox, currentFontName,*/ viewPort;

    commandBox = XtCreateManagedWidget ("commandBox
",formWidgetClass,pane,NZ);
{
...

    ownButton =
        XtCreateManagedWidget ("ownButton"
,toggleWidgetClass,commandBox,NZ);
/*
 * ToolTalk のデモンストレーション用に、
 * フォントを変更する適用ボタンを以下の行に追加する
 */
    applyButton =
        XtVaCreateManagedWidget ("applyButton",
            commandWidgetClass,
            commandBox,
            XtNlabel, "apply",
            XtNfromHoriz, ownButton,
            XtNleft, XtChainLeft,
            XtNright, XtChainLeft,
            0);

    countLabel =
        XtCreateManagedWidget ("countLabel"
, labelWidgetClass,commandBox,NZ);

    XtAddCallback(quitButton, XtNcallback, Quit,
NULL);
    XtAddCallback(ownButton,XtNcallback,
OwnSelection,(XtPointer)True);
/*
 * ToolTalk のデモンストレーション用に、
 * 適用ボタンが押されたことを xedit に通知するために
 * 以下の行を追加する
 */
    XtAddCallback(applyButton,
XtNcallback,ReplyToMessage, NULL);
}
```

例 B-4 変更された Xfontsel.c ファイル (続き)

```
        fieldBox = XtCreateManagedWidget("fieldBox",
        boxWidgetClass, pane, NZ);

...

{
    int f;
    for (f = 0; f < FIELD_COUNT; f++)
        currentFont.value_index[f] = -1;
}

/*
 * ToolTalk のデモンストレーション用に、
 * ToolTalk セッションをスタックの最初に
 * 作成して、デフォルトセッションとして設定するために、
 * 以下の行を追加する
 */
ttmark = tt_mark();
ttrc = tt_default_session_set(
    /* 最上位ウィンドウを表示している */
    tt_X_session(
        /* X サーバーの */
        DisplayString(
            /* X セッションに */
            XtDisplay(top)));
    /* デフォルトセッションを設定 */
    /*
     * デフォルトセッションがない場合は異常終了する
     */
    dieFromToolTalkError("tt_default_session_set"
    ,ttrc);
    procid = tt_open();
    /*
     * プロセス識別子がいない場合は異常終了する
     */
    dieFromToolTalkError("tt_open"
    ,tt_ptr_error(procid));
    ttfd = tt_fd();
    /*
     * ToolTalk ファイル記述子がいない場合は異常終了する
     */
    dieFromToolTalkError("tt_fd"
    ,tt_int_error(ttfd));
    ttrc = tt_ptype_declare("xfontsel");
    /*
     * ptype が宣言されていない場合は異常終了する
     */
    dieFromToolTalkError("tt_ptype_declare"
    ,tt_int_error(ttfd));
    ttrc = tt_session_join(tt_default_session());
    /*
     * セッションを結合できない場合は異常終了する
     */

```

例 B-4 変更された Xfontsel.c ファイル (続き)

```
dieFromToolTalkError("tt_session_join", ttrc);
/*
 * 入力の追加
 */
XtAddInput(ttf, (XtPointer)XtInputReadMask,
processToolTalkMessage, 0);

XtAppMainLoop(appCtx);

tt_close();
/* ToolTalk セッションを終了する */
tt_release(ttmark);
/* 領域の解放 */
}

...
Boolean field_bits[FIELD_COUNT];
int max_field;
if (*fontName == DELIM) field++;
/*
 * ToolTalk のデモンストレーション用に、
 * BSD の代わりに標準ルーチンを使用して
 * bzero 以下のように読み込むために
 * bzero( field_bits, sizeof(field_bits) );
 * の行を変更する
 */
memset( field_bits, 0, sizeof(field_bits) );
if (Matches(pattern, fontName++, field_bits,
&max_field)) {

...

XtDisownSelection(w, XA_PRIMARY, time);
XtSetSensitive(currentFontName, False);
}

/*
 * ToolTalk のデモンストレーション用に、
 * 以下の行を追加する
 */
}

void dieFromToolTalkError(procname, errid)
/* エラーが発生した場合は異常終了する */
char *procname;
/* プロセス名 */
Tt_status errid;
/* エラー識別子 */
{
/*
 * 警告または TT_OK を無視してはならない
 */
}
```


例 B-4 変更された Xfontsel.c ファイル (続き)

```
    if (tt_is_err(errid)) {
        fprintf(stderr, "%s returned ToolTalk error:
%s\n",
            procname, tt_status_message(errid));
        exit(1);
    }
}

void processToolTalkMessage()
/* プロセスメッセージ */
{
    int ttmark;
    /* ToolTalk マーク */
    Tt_message incoming;
    /* 着信メッセージ */

    ttmark = tt_mark();

    incoming = tt_message_receive();
    /* メッセージを受信 */
/*
 * ファイル記述子が使用可能であっても、
 * 実際には ToolTalk のメッセージがない場合も
 * あり得る
 */
    if (incoming == 0) return;

    if (tt_is_err(tt_ptr_error(incoming))) {
        dieFromToolTalkError("tt_message_receive",
            tt_ptr_error(incoming));
    }

    if (0==strcmp(tt_message_op(incoming), "
GetFontName")) {
        /*
         * これは、予想していたメッセージである。
         * すでに使用中の場合はそのメッセージを拒否する。
         * そうでない場合は、適用ボタンを有効にする
         */
        if (replymsg) {
            tt_message_reject(incoming);
            tt_message_destroy(incoming);
            tt_release(ttmark);
            return;
        }
        XtVaSetValues(applyButton, XtNsensitive,
TRUE, 0);
        replymsg = incoming;
        tt_release(ttmark);
        return;
    }

    /*
```

例 B-4 変更された Xfontsel.c ファイル (続き)

```
* これは認識していないメッセージである。
* 開始の要求または通知があった場合に異常終了する
*/

    if (tt_message_class(incoming) == TT_REQUEST ||
        tt_message_status(incoming) ==
TT_WRN_START_MESSAGE) {
        tt_message_fail(incoming);
    }
    tt_message_destroy(incoming);
    tt_release(ttmark);
}

/*
* 適用ボタンが押されると呼び出される。
* 未応答のメッセージに回答して、適用ボタンをオフにする
*/

/* 使用済み引数 */
void ReplyToMessage(w, msg, wdata)
Widget w;
caddr_t msg;
caddr_t wdata;
{
    tt_message_arg_val_set(replymsg, 0,
currentFontNameString);
    tt_message_reply(replymsg);
    tt_message_destroy(replymsg);
    replymsg = 0;
    XtVaSetValues(applyButton, XtNsensitive, FALSE,
0);
}
}
```

ToolTalk の標準メッセージセット

標準のメッセージセットは、同じメッセージプロトコルに準拠して自分以外の人が開発したアプリケーションを自動的に統合するようなアプリケーションを開発するためのものです。標準のメッセージセットを定義するために、主要なソフトウェアベンダやエンドユーザーの協力を得て、さまざまな作業が行なわれてきました。ToolTalk の標準メッセージセットは、ToolTalk API の高水準インタフェースで、複数のアプリケーションの制御とアプリケーション間でのデータの統合を簡単に行うための共通の定義と規約を提供します。

この付録で示す標準の ToolTalk メッセージセットについては、『*ToolTalk* リファレンスマニュアル』を参照してください。

ToolTalk デスクトップサービスメッセージセット

デスクトップを根本的に統合するには、アプリケーション間を制御する基本メッセージセットをアプリケーションでサポートする必要があります。「ToolTalk デスクトップサービスメッセージセット」は、すべてのアプリケーションにこの機能を提供する共通のメッセージセットです。このメッセージセットは、デスクトップアプリケーションの開発者とユーザーのどちらにも役立つ強力なメッセージ処理用プロトコルです。これを使用するとアプリケーションは、他のデスクトップアプリケーションと容易にやり取りできます。また、ローカルでもネットワーク経由でも、透過的な方法で相互に通信できます。

開発目的

アプリケーションを統合的に制御するには、起動、停止、表示制御、入出力データ情報の受け渡しを行うために一定の基本機能が必要です。この機能は、ツールセット中の他のアプリケーションと基本的な制御情報を交換できるよう、すべてのアプリケーションでサポートする必要があります。このような機能を使用すれば、スマートデスクトップや統合スマートツールセットを開発できます。アプリケーション同士でグループを組むことにより、互いに呼び合ってタスクを処理したりやり取りしたりできるため、1つの統合環境をエンドユーザーに提供できます。

特長

ToolTalk デスクトップサービスメッセージセットは、開発者にとって次の2つの利点があります。

1. ユーザーが直接的に介入しなくても、アプリケーションの基本的な制御を行うことができる。ユーザーの便宜を図るため、ルーチンや共通プロシージャを自動化できる。
2. 共通の情報交換セットを使用するため、ツールを限定できる。ToolTalk 対応のアプリケーションはすべて、これらの機能を実行できる。

ToolTalk 文書メディア交換メッセージセット

マルチメディアは、今後重要になる技術です。マルチメディア対応のアプリケーション数が増えているにもかかわらず、今日のマーケットの複雑なニーズに応える完全統合型ソリューションを提供しているベンダは1つもありません。この「ToolTalk 文書メディア交換メッセージセット」は、マルチメディア技術を真に打開するものです。

このメッセージセットは、マルチメディア技術の開発者とユーザーのどちらにも役立つよう設計した強力なメッセージ処理プロトコルです。これを使用すれば、アプリケーションでお互いのマルチメディア機能を容易に共有できます。また、マルチメディアアプリケーションは、データフォーマットや圧縮方法、従来は制約となっていた技術上の問題点などに関係なく、ローカルでもネットワーク経由でも、透過的な方法でお互いに通信できます。

開発目的

いくつかのベンダがアプリケーション間通信のために協力関係を結んでいます。しかし、エンドユーザーで解決できる範囲は限られています。しかし、ToolTalk 文書メディア交換メッセージセットを使用すれば、どのようなアプリケーションでも一連のマルチメディア機能を、他のどのようなアプリケーションとも透過的な方法で共有できます。

このマニュアルでは、Sun Microsystems® と、マルチメディアハードウェアやマルチメディアソフトウェアを開発している主要な独立ベンダーの協力によって開発した規格を紹介しています。

アプリケーションは、この簡単なプロトコルを使用することにより、個々のサービス提供者を気にすることなく、数多くのマルチメディアサービス用の ToolTalk インタフェースをすぐに、しかも簡単に作成できます。アプリケーションのグループ全体で互いに「プラグアンドプレイ」できるため、音声、ビデオ、グラフィックス、電話、その他のメディアソースを、新規および既存のアプリケーションに統合できます。

「プラグアンドプレイ」とは、同じプロトコルに準拠するツールであれば、他のどのツールとでも交換できることを意味します。つまり、一定の ToolTalk プロトコルに準拠したツールであれば、ユーザーのコンピュータ環境に搭載 (plug) でき、プロトコルが指示する機能を実行 (play) できます。それぞれのツールは変更する必要はなく、互いに個々の機能をあらかじめ知らなくても、一緒に協力して動作できます。たとえば、文書の一部にビデオを入れ、そのビデオを他のアプリケーションに再生させるような文書処理アプリケーションも作成できます。

ToolTalk 文書メディア交換メッセージセットは、効率のよい汎用メッセージセットを定義したもので、メディア制御とデータ交換の機能があります。このプロトコルは、メディアプレイヤー用、エディタ用、ユーザー用のエディタメッセージから構成されています。

特長

ToolTalk 文書メディア交換メッセージセットは、開発者にとって次の 2 つの利点があります。

1. 新規および既存ソフトウェアへのマルチメディアの組み込みが簡単

マルチメディアの機能をどのアプリケーションにも簡単に追加できます。ToolTalk 文書メディア交換メッセージセットを使用すれば、他の開発者のマルチメディア技術を利用できるため、システムの機能を向上させながら、開発に要する時間やコストを低減できます。

2. エンドユーザーの戦略範囲を拡大するフレームワークの作成

ToolTalk 文書メディア交換メッセージセットを使用すれば、アプリケーションの連携が促進されるため、エンドユーザーや開発者は得意分野に的を絞った戦略を新しく立てることができます。これにより、以前は対象外であったマーケットを開拓できるため、ユーザーの製品を新たに展開できます。

ToolTalk メッセージの一般的な定義と表記法

ToolTalk メッセージには、ToolTalk 独自の定義で使用する用語があります。この節では、これらの用語と ToolTalk メッセージのマニュアルページで使用している表記法を定義します。

表 C-1 文書メディア交換メッセージセットの用語

情報の種類	説明
ヘッダー	次のフォーマットでメッセージを 1 行で説明したもの。 <i>MsgName</i> (<i>Tt_class</i>) <i>MsgName</i> はメッセージ名、 <i>Tt_class</i> は要求か通知を表す。
名前	メッセージ名と 1 行のメッセージの説明
説明	メッセージの要求操作または通知イベントについての説明
フォーマット	メッセージを次のようなフォーマットの ToolTalk 型ファイル構文 (ToolTalk 型コンパイラ <i>tt_type_comp</i> が処理する構文に似ている) で表記したもの。 <fileAttrib> <opName> (<requiredArgs> [<optionalArgs>]); フォーマットは、メッセージの種類ごとに示される。 <fileAttrib>: メッセージのファイル属性。必須、オプション、指定不可のいずれかを示す。 <opName>: 操作やイベント名を「op name」または「op」と呼ぶ。ツールが異なっても、同じ opName で異なるものを表さないことが重要である。このため、標準以外のメッセージの opName は一意にしなければならない。たとえば、「Acme_Hoarktool_My_Frammistat」のように、<会社名><製品名> という接頭辞を付けるのは良い方法である。 <requiredArgs>、<optionalArgs>: メッセージに常に指定しなければならない引数。個々の引数は、次のフォーマットで表される。 <mode> <vtype> <argument name> <i>mode</i> には in、out、inout のどれか 1 つを指定する。vtype はメッセージ引数のデータの種類を記述する文字列であり、プログラマが定義する。argument name は引数名を表す。 ToolTalk サービスは vtype を使用して、送信メッセージのインスタンスと登録済みメッセージパターンを照合する。通常、vtype は単一の決まったデータ型に対応する。

表 C-1 文書メディア交換メッセージセットの用語 (続き)

情報の種類	説明
必須引数	<p>メッセージに常に指定しなければならない引数。</p> <p><vtype> <argumentName></p> <p>「vtype」は、メッセージ引数のデータの種別を記述する文字列であり、プログラマが定義する。ToolTalk が vtype を使用するの、送信メッセージのインスタンスと登録済みメッセージパターンの照合を行うためだけである。</p> <p>通常、vtype はすべて単一の決まったデータ型に対応させる。ToolTalk 引数のデータ型は、整数、文字列、バイトのいずれかである。メッセージ引数やパターン引数のデータ型は、その値を設定する ToolTalk API 関数で決まる。</p> <p>引数名は、C の typedef でのパラメタ名と同様、引数の構文の意味を分かりやすくするコメントである。</p>
オプション引数	<p>メッセージに指定してもよい引数。特に断らないかぎり、必須引数の後であれば、オプション引数をどのように組み合わせても、どのような順序で追加してもよい。</p>
説明	<p>要求が意味する操作や、通知が知らせるイベントについての説明</p>
エラー	<p>要求のハンドラや通知の送信側が設定できるエラーコードの一覧</p>

通達 — 要求に似た通知です。要求がデータを返さない (または、データが返されても送信側が見ない) 場合、その要求を一連のツールに送る方が便利ことがあります。通達は一種の通知であるため、どのツールがそのメッセージを入手してもデータや応答は得られず、送信側は何も知らされません。

ハンドラ — 要求を受け取る者の procid です。この procid のプロセスは、指示された操作を完了する義務があります。

通知 — イベントを知らせるメッセージです。通知を受信するツールがない場合も複数ある場合もあります。送信側は、ツールが通知を受信したかどうかは分かりません。通知に対して応答できません。

procid — ToolTalk メッセージを送受信する主体を表します。procid は ToolTalk サービスが (tt_open 時に) 作成して渡す身元です。プロセスはメッセージを送受信する際に procid を指定しなければなりません。1 つのプロセスで複数個の procid を使用することも、連携プロセスの 1 グループで 1 つの procid を使用することもできます。)

要求 — 操作を実行するよう要求するメッセージです。要求は、指示された操作を完了する義務のある、ハンドラという受け取る者を 1 つ持ちます。要求に対してハンドラは、異常終了、拒否、または応答が可能です。要求を拒否するハンドラはいくつあってもかまいません。最終的に要求に対して、異常終了させるか応答できるのは 1

つのハンドラだけです。要求を受け取るハンドラが動作していない場合、ToolTalk サービスはハンドラを1つ自動的に起動できます。要求を受け取るハンドラが見つからなかった場合やハンドラが要求を異常終了させた場合、その要求は「失敗」という状態で送信側に戻されます。

エラー

`tt_message_status` を使用すると、応答から `Tt_status` コードを読み取ることができます。この状態のデフォルトは `TT_OK` ですが、`tt_message_status_set` を使用すればハンドラが設定することもできます。異常な状況下 (対応するハンドラがない場合など) では、ToolTalk サービス自身がこのメッセージ状態を設定します。

ToolTalk API が定義する `Tt_status` 値以外に、メッセージセットごとに定義されるエラー条件があります。それらについては、各メッセージセットの概要を記述した個所に一覧を掲載してあります。ここでは、各エラー条件について次の項目を記述してあります。

- エラー名
- 整数値
- エラー条件を説明した文字列 (C ロケールで表現)

ToolTalk Inter-Client Conventions (TICC) はバイナリ形式のメッセージインタフェースであるため、整数と文字列はそのバイナリインタフェースの一部ですが名前は含まれません。

- エラー文字列を `SUNW_TOOLTALK_INTERCLIENTCONVENTIONS` ドメインでキーとして使用すれば、エラー条件のローカルな説明を検索できます。詳細は、`dgettext(3C)` を参照してください。
- 状態コードを表す整数値は、1537 (`TT_ERR_APPFIRST + 1`) から始まります。最初の151個のコードは、`intro(2)` で定義されているシステムエラーのリストに対応しています。

この基準には、エラー名を整数値に対応させるような標準のプログラミングインタフェースはまだありません。

ToolTalk サービスを使用すれば、任意の状態文字列を任意の応答に組み込むことができます。各状態コードから標準のローカルな文字列を派生させることができるため、エラー状態を表す自由形式の説明にこの状態文字列を使用できます。たとえば、ある要求が `TT_DESKTOP_EPROTO` で異常終了した場合、その状態文字列に「The vtype of argument 2 was 'string'; expected 'integer'」を設定することもできます。ツールを扱う場合、状態文字列は要求側のロケールで作成するようにしてください。詳細は、`Get_Locale` 要求を参照してください。

ToolTalk 開発の一般的な指針と表記法

Sun では、「開放型プロトコル」を推奨しています。プロトコルは一般に開放的なものであり、「相手を特定しないメッセージ」(つまり、誰がそれを受け取るかを知らずに送信するメッセージ)をサポートしています。この節では、この開放型メッセージプロトコルをサポートするものであれば、どのようなアプリケーションともうまくやり取りできるアプリケーションを独自に開発する際の指針を示します。ここで示す指針と原則に従えば、独自に開発した別々のアプリケーションでもさまざまな基準を考案して運用でき、互いにやり取りできます。また、アプリケーションのユーザーはその環境をより細かく制御でき、カスタム化できます。

ToolTalk のアプリケーションを作成する際は、次の原則に従ってください。

1. 要求相手を特定しない
2. ツールは必要なときだけ起動する
3. 要求した操作が完了してから要求に応答する
4. できるだけ内部状態を持たないようにする
5. ツールの役割ごとに ptype を 1 つ宣言する

要求相手を特定しない

アプリケーションを完全に開放的な設計にするには、要求の受け取り者を指定しないでください。つまり、要求側のプロセスでは、要求した操作を処理するツールのインスタンスやツールの型も指定しないでください。要求を特定のプロセスに送信すると、ユーザーのリソースや、メッセージを受け取る可能性のあるプロセスのリソースの利用を不要に制限することになります。また、要求を特定の型のツールに送信すると、やり取りできるツールを必要ないのに制限することになります。

メッセージには、要求する操作が通知するイベントを指定してください。メッセージを受け取るプロセスを、メッセージに記述してはいけません。やり取りする相手を各ツールで限定しないほど、ユーザーにとって柔軟なシステムになります。

開放型プロトコルの詳細は、『*Designing and Writing a ToolTalk Procedural Protocol*』(Sun Part No. 801-3592-01)を参照してください。

ツールは必要なときだけ起動する

プロトコルを完全に開放型の設計にするには、必要なときだけツールを起動してください。ツールのインスタンスを必要なときだけ起動すると、ユーザーは CPU、画面領域、スワップ空間などをより柔軟に効率よく使用できます。ToolTalk サービスには、ツールのインスタンスの起動時期を決定するための機能として、次のようなものがあります。

- メッセージと型シングニチャで、確実な「起動」を指定できます。これを指定すると、メッセージの適切な受け取り者が動作していない (または、要求を受け取る適切な者がいない) 場合、メッセージの処理または監視を行うよう静的に型に登録してあるツールインスタンスを ToolTalk サービスが起動します。
- 各プロセス型 (ptype) は、1 つのセッションで起動できるインスタンスの最大数を指定できます。
- ToolTalk サービスでは、ツールのインスタンスを新しく起動する前に、動作中の適切なハンドラすべてに各要求を配布します。これを受け取ったハンドラは、自分自身の判断基準 (タスクを新しく引き受けることができるか、保存していない変更があるか、アイドル時間、アイコン状態、または新しい仕事を自由に受け入れるようユーザーが指定してあるかなど) に従って、要求を受け取るか拒否できます。

操作が完了したら応答する

アプリケーションを完全に開放的な設計にするには、送信側プロセスの要求した操作を完了したら、送信側プロセスにその旨を通知してください。ただし、要求メッセージの送信が非常に短時間で終わるのに比べ、その操作の完了には長時間かかることがあります。送信側プロセスへは、次のどちらかの方法で応答できます。

- 要求を受け取ったことは即座に応答するが、実際に操作を完了した結果は後で別のメッセージで送信する
- 操作が完了するまで応答を保留する

ToolTalk メッセージはまったく非同期であるため、後者の方法を推奨します。ツールやツールのセッションは、未処理の要求を少なくとも1つかかえることになるため、ブロックしてはいけません。

できるだけ内部状態を持たないようにする

アプリケーションを開放的な設計にするには、各メッセージをできるだけそれ自身で完結させてください。プロトコルの状態の数を少なくすると、メッセージは、以前のメッセージや指定した受信側の状態に頼らなくても済みます。

役割ごとにプロセス型を1つ宣言する

ToolTalk プロトコルは、各ツールが果たす「役割」(つまり、各ツールが実行するタスクの種類) で表現されます。ToolTalk の ptype は、ツールが動作していないときに処理対象のメッセージを受け取ったらどう処理するかを ToolTalk サービスに指示します。プロトコルを開放的にするには、プロトコルの中の役割ごとに ptype を1つ宣言してください。こうすると、ユーザーはそのニーズに応じて、ツールを自由に交換できます。たとえば、録音用には洗練されたサウンドオーサリングツールを使用したいが、再生用には簡単なオーディオツールでかまわないというユーザーもいます。

ptype ごとにメッセージシグニチャを1つだけ指定する必要もあります。同一の ptype にメッセージシグニチャを2つ以上指定すると、片方のメッセージを処理できるプログラムならず、もう一方のメッセージも処理できなければならないと要求することになります。たとえば、「UWriteIt」という ptype には「Display」と「Edit」という2つのメッセージシグニチャを指定できます。これは UWriteIt のドキュメントフォーマットを理解するものであれば、どのようなツールでも両方の操作を実行できると思われるからです。

ToolTalk アプリケーションの開発

この節では、設計プロセスを説明します。アプリケーションでは、次の3つの手順で簡単に ToolTalk メッセージを送受信できます。

1. 他のアプリケーションやユーザーとやり取りする方法を決定する
2. メッセージを選択し、アプリケーションのコンテキスト内での使用方法を定義する
3. ToolTalk の呼び出しとメッセージをアプリケーションのコードに組み込む

注 - ToolTalk ソフトウェア製品には、既存のアプリケーションに ToolTalk 機能を簡単に追加する方法を示すデモンストレーションが入っています。このデモンストレーションは、『*Tool Inter-Operability: A Hands On Demonstration*』 (Sun Part No. 801-3593-01) に掲載されています。

- 各ツールが連携動作する方法と実行する必要がある操作を定義する

アプリケーションにどのような種類のやり取りが必要か明確に理解することは、アプリケーションの統合を成功させるための必須条件です。これを解析するのに最もよい方法は、アプリケーションの使用法を示すシナリオを定義することです。このシナリオから、実行する必要のあるやり取りや交換する必要のある情報を定義できます。どのような情報や状態をやり取りするのかを詳細にシナリオに記述しておく、アプリケーションにメッセージを組み込む際に役立ちます。

- タスクを実現するのに適切なメッセージを選択する

アプリケーションが他のアプリケーションやユーザーとやり取りする方法が決まると、必要なタスクを実現するための各メッセージを決定しなければなりません。

まず、Sun、ANSI、X3H6、CFI などの産業グループから入手できる標準メッセージセットを参照してください。これらのメッセージを使用するよう強く推奨するのは、次の2つの理由からです。

1. 標準メッセージにより標準のインタフェースを文書として入手できます。このインタフェースを使用すれば、他の開発者も連携するアプリケーションを独自に開発できます。カスタマが統合システムを構築する際の関連インタフェースも入手できます。

2. この標準メッセージセットは、アプリケーションに「汎用プラグアンドプレイ」機能を提供します。この機能を使用すれば、カスタマは複数のアプリケーションを自由に使用してサービスを受けられます。カスタマが自由に使用するアプリケーションを選択して個々のジョブに最適なツールを使用できるため、アプリケーションの開発者は不要な機能を製品でサポートするよう強要されません。

この標準メッセージセットが設計に役立たない場合は、カスタムメッセージを開発する必要があります。

標準メッセージにないものを使用する場合は、「media_exchange@sun.com」の文書メディア交換メッセージ協会にお問い合わせください。そのメッセージを標準メッセージセットに追加するかどうかを協会で検討いたします。

- **ToolTalk** 呼び出しとメッセージをアプリケーションに組み込む

設計段階が完了すると、ToolTalk 機能をアプリケーションに組み込む段階です。

まず、ToolTalk API 呼び出しを使用するすべてのファイルに、ToolTalk ヘッダーファイルをインクルードする必要があります。また、送受信機能を制御するためのパターンの登録と初期設定を行う必要もあります。パターンの登録と初期設定の詳細は、『*The ToolTalk Service: An Inter-Operability Solution*』(SunSoft Press / Prentice Hall 社発行)を参照してください。

次に、ToolTalk メッセージを送信する機能をアプリケーションに追加します。設計時のシナリオを基にすれば、どのルーチンがどのメッセージを送信するか、各メッセージの引数をどのようにするかは簡単に決定できます。

ToolTalk サービスの初期設定が完了すると、アプリケーションは ToolTalk API でメッセージを作成し、その内容を書き込んで、他のアプリケーションに送信できます。

- アプリケーションがウィンドウシステムを使用する場合は、ToolTalk サービスを有効にするための呼び出しを、イベントポーリンググループの中に追加するだけです。
- アプリケーションがまだポーリンググループを使用していない場合は、メッセージを定期的に検査するための簡単なループを作成する必要があります。詳細は、『*The ToolTalk Service: An Inter-Operability Solution*』(SunSoft Press / Prentice Hall 社発行)を参照してください。

メッセージ方式に関する問い合わせ先

文書メディア交換メッセージセットに関する質問、コメント、情報の請求は、media_exchange@sun.com の文書メディア交換メッセージ協会までご連絡ください。

Q & A

この付録では、ToolTalk サービスについて、次のような質問にお答えします。

- 199 ページの「ToolTalk サービスとは何ですか」
- 199 ページの「ToolTalk サービスは、Common Object Request Broker Architecture (CORBA) の Sun 版ですか」
- 200 ページの「ToolTalk サービスに入っているファイルの種類を教えてください」
- 201 ページの「X ベースの `ttsession` が最初に起動されるのはいつですか」
- 201 ページの「`rpc.ttdbserverd` はどこで起動されますか」
- 201 ページの「ToolTalk 型データベースはどこにありますか」
- 202 ページの「ToolTalk サービスを使用するには、X Window System が必要ですか」
- 202 ページの「MIT X で ToolTalk サービスを使用できますか」
- 203 ページの「X セッションのセッション ID はどこにありますか」
- 203 ページの「`tt_open` はどのように `ttsession` に接続しますか」
- 204 ページの「`tt_open` を呼び出した後、セッションが実際に始まるのはいつですか」
- 204 ページの「別のセッションが接続されると最初のセッションは終了しますか」
- 204 ページの「動作するマシンが異なるプロセス同士は、ToolTalk サービスをどのように使用すれば通信できますか」
- 206 ページの「`tt_default_session_set` の目的は何ですか」
- 206 ページの「1 つのプロセスで 2 つ以上のセッションに接続するには、どうすればいいですか」
- 207 ページの「あらかじめ決めたセッション ID で `ttsession` を起動できますか」
- 207 ページの「セッション ID には、どのような情報が入っていますか」
- 207 ページの「プログラムが新しくセッションに参加したことを通知する標準的な方法はありますか」
- 207 ページの「メッセージの行き先を教えてください」
- 208 ページの「メッセージの基本的なフローを教えてください」
- 209 ページの「アプリケーションにメッセージが到着するとどうなりますか」
- 210 ページの「メッセージを区別する方法を教えてください」
- 211 ページの「プロセスは自分自身に要求を送信できますか」
- 212 ページの「`tt_message_callback_add` で登録した関数に、自分自身のデータを渡せますか」

- 212 ページの「任意のデータをメッセージで送信する方法を教えてください」
- 212 ページの「ToolTalk サービスでファイルを転送できますか」
- 213 ページの「ToolTalk サービスは、メモリー (バイト) の順序の問題をどのように処理しますか」
- 213 ページの「メッセージは再使用できますか」
- 213 ページの「メッセージを破棄するとどうなりますか」
- 213 ページの「1つのメッセージを2つ以上のハンドラで処理できますか」
- 214 ページの「1つの ptype のハンドラを2つ以上実行できますか」
- 215 ページの「メッセージの処置の値とは何ですか」
- 215 ページの「メッセージ状態要素とは何ですか」
- 215 ページの「tt_free はいつ使用しますか」
- 216 ページの「ptype とは何ですか」
- 216 ページの「新しく作成した型が認識されない理由を教えてください」
- 216 ページの「ptype のプロセスが既に存在する場合、ptype 情報は使用されますか」
- 217 ページの「(インスタンスが既に動作中かどうかにかかわらず) インスタンスを常に起動するよう ptype の定義を変更できますか」
- 217 ページの「tt_ptype_declare は何を行いますか」
- 217 ページの「TT_TOKEN とは何ですか」
- 217 ページの「パターンはいつ有効になりますか」
- 218 ページの「応答を入手するにはパターンの登録が必要ですか」
- 218 ページの「要求を監視するにはどうすればいいですか」
- 218 ページの「静的パターンの属性値と照合させる方法を教えてください」
- 218 ページの「TT_HANDLER に対してワイルドカードでパターンを指定できない理由を教えてください」
- 218 ページの「ファイルを配信範囲とする任意のメッセージを監視するようなパターンを設定できますか」
- 219 ページの「静的パターンのファイル配信範囲は file_in_session と同じですか」
- 219 ページの「arg_add、barg_add、iarg_add の違いを教えてください」
- 219 ページの「メッセージ引数の type や vtype とは何ですか」
- 220 ページの「コンテキストの使用方法を教えてください」
- 220 ページの「ttsession はどのように照合をチェックしますか」
- 221 ページの「ToolTalk サービスには、配信範囲が何種類ありますか」
- 222 ページの「TT_DB ディレクトリとは何ですか、また、型データベースと TT_DB ディレクトリは何が違いますか」
- 222 ページの「tt_db データベースには何を入れればいいですか」
- 222 ページの「rpc.ttdbserverd は何を実行しますか」
- 223 ページの「ttsession と rpc.ttdbserverd は通信しますか」
- 223 ページの「どのような帯域幅のメッセージをサポートできますか」
- 223 ページの「メッセージサイズや引数の個数に制限はありますか」
- 223 ページの「メッセージを送信する際、最も時間効率のよい方法は何ですか」
- 224 ページの「ネットワークのオーバーヘッドの種類を教えてください」
- 224 ページの「ToolTalk サービスは、要求を処理するのに負荷分散を使用しますか」
- 224 ページの「ToolTalk アプリケーションに必要なリソースには何がありますか」
- 224 ページの「ttsession が異常終了するとどうなりますか」
- 225 ページの「rpc.ttdbserverd が異常終了するとどうなりますか」
- 225 ページの「ホストやリンクが停止するとどうなりますか」

- 226 ページの「tt_close は何をしますか」
- 226 ページの「メッセージの配信は、ネットワーク上でも保証されますか」
- 226 ページの「メッセージの配信には、時間的な順序がありますか」
- 226 ページの「unix、xauth、des とは何ですか」
- 227 ページの「アプリケーションが互いにメッセージを隠すことはできますか」
- 227 ページの「横取りや模造に対して保護機構はありますか」
- 227 ページの「待ち行列に入ったメッセージはどこに格納されていますか、また、その記憶領域はどのくらい安全ですか」
- 227 ページの「ToolTalk サービスは C2 保証されていますか」
- 228 ページの「メッセージの行方を追跡するには、どうすればいいですか」
- 228 ページの「ToolTalk サービスを使用して、他のすべてのツールから自分のデバッグツールを隔離するには、どうすればいいですか」
- 229 ページの「C++ で ToolTalk サービスを使用できますか」
- 229 ページの「ファイル名を修飾する必要はありますか」
- 229 ページの「ToolTalk オブジェクトとは何ですか」
- 230 ページの「ToolTalk ニュースグループはありますか」

Q & A

次に示す頻繁に寄せられる質問は、ToolTalk サービスに関する追加情報の役割も果たしています。

ToolTalk サービスとは何ですか

ToolTalk サービスを使用すれば、個々のアプリケーションは直接相手を知らなくても、互いに通信できます。アプリケーションは ToolTalk メッセージを作成して送信することにより、互いに通信します。ToolTalk サービスはこのメッセージを受信すると、受信側を判別し、適切なアプリケーションに配信します。

ToolTalk サービスは、Common Object Request Broker Architecture (CORBA) の Sun 版ですか

違います。ToolTalk サービスは、CORBA に準拠した Sun オブジェクト要求仲介 (ORB) ではありません。ToolTalk サービスは、オブジェクト管理グループ (OMG) の CORBA 仕様が定義される以前の 1991 年に設計出荷されました。

CORBA に準拠した Sun ORB は、分散オブジェクト管理機能 (DOMF) であり、これは Sun のプロジェクト DOE 製品の一部です。Sun は、DOMF が Solaris の一部として一般利用できるようになった時点で、DOMF 上で動作する ToolTalk API をサポートすると公約しています。現在 ToolTalk メッセージサービスを使用しているアプリケーションは、将来は分散オブジェクト環境に移行します。

ToolTalk サービスに入っているファイルの種類を教えてください

ToolTalk ファイルは、`/usr/openwin/bin`、`lib`、`include/desktop`、および `man` ディレクトリの他に `/usr/dt/bin`、`lib`、および `include/Tt` ディレクトリにもあります。これには、過去のいきさつがあります。ToolTalk は、共通デスクトップ環境 (CDE) の前から存在していて、Solaris とともに `/usr/openwin` ディレクトリ構造で出荷されていました。CDE がリリースされると、ToolTalk は、シンボリックリンクを使って `/usr/dt` ディレクトリから見えるようになりましたが、実際には、依然として `/usr/openwin` にインストールされていました。CDE がインストールされた Solaris 2.6 オペレーティング環境およびその互換バージョンのシステムでは、ToolTalk の 2 つの完全版がインストールされています。1 つは `/usr/dt` に、もう 1 つは `/usr/openwin` にインストールされていますが、`/usr/dt` にインストールされているものだけが CDE で有効です。

表 D-1 にこれらのファイルを示します。

表 D-1 ToolTalk のファイル

ファイル名	説明
<code>ttsession</code>	ネットワーク上で通信してメッセージを配信する
<code>rpc.ttdbserverd</code>	ToolTalk オブジェクト仕様や、ToolTalk メッセージで参照するファイル情報を格納し管理する
<code>ttcp</code> , <code>ttmv</code> , <code>ttrm</code> , <code>ttrmdir</code> , <code>tttar</code>	標準オペレーティングシステムのシェルコマンド ToolTalk オブジェクトの入ったファイルや、ToolTalk メッセージのサブジェクトであるファイルがコピー、移動、または削除されると、これらのコマンドは ToolTalk サービスに連絡する
<code>tttrace</code> , <code>ttsnoop</code>	<code>tttrace</code> は、 <code>truss (1)</code> に類似している。これにより、特定の <code>ttsession</code> で発生するメッセージパッシングとパターンマッチングをトレースできる。また、これを使用することにより、すべての呼び出しのプログラムごとのトレースを ToolTalk API に提供できる。 <code>ttsnoop</code> は Motif をベースにしたプログラムで、 <code>tttrace</code> のメッセージとパターントレース機能に、デバッグとチュータ機能として、メッセージを簡単に作成または送信したり、パターンを登録したりできる付加機能
<code>ttddbck</code>	ToolTalk データベースの検査と修復を行うツール
<code>tt_type_comp</code>	<code>pptype</code> と <code>otype</code> のファイルをコンパイルし、ToolTalk 型データベースに自動的にインストールする
<code>ttce2xdr</code>	ToolTalk 型データを、分類機構データベースフォーマットから XDR データベースフォーマットに変換する
<code>libtt.a</code> , <code>libtt.so</code> , <code>tt_c.h</code> , <code>tttk.h</code>	アプリケーションが使用してメッセージを送受信する ToolTalk 関数のアプリケーションプログラミングインタフェース (API) ライブラリとヘッダーファイル

X ベースの ttsession が最初に起動されるのはいつですか

ttsession が1つも動作していない場合は、tt_open を最初に呼び出した時点で tttsession が自動的に起動されます。ただし、/usr/dt/bin/Xsession ファイルには下記のようなエントリがあり、それによって usr/dt/bin/dtlogin を使用している際に、ログイン時に自動的に tttsession を起動します。

```
# Start tttsession here.  
dtstart_tttsession="$DT_BINPATH/tttsession"
```

rpc.ttdbserverd はどこで起動されますか

/etc/inet/inetd.conf ファイルに、次のようなエントリが入っています。

```
# Sun ToolTalk Database Server  
100083/1 tli rpc/tcp wait root /usr/dt/bin/rpc.ttdbserverd  
/usr/dt/binrpc.ttdbserverd
```

ToolTalk 型データベースはどこにありますか

ToolTalk 型データベースの場所は、環境変数 TTPATH で ToolTalk サービスに教えます。この環境変数のフォーマットを次に示します。

```
userDB[:systemDB[:networkDB]]
```

注 - 型ファイルは、TTPATH に指定した順序と逆の順序で読み込まれます。

この環境変数は、データベースサーバーのリダイレクトファイルの場所も ToolTalk サービスに教えます。そのデフォルトの位置を表 D-2 に示します。

表 D-2 ToolTalk 型データベースのデフォルト位置

データベース	位置
ユーザー	~/.tt
システム	/etc/tt
ネットワーク	\$OPENWINHOME/etc/tt または /usr/dt/appconfig/tttypes

ToolTalk サービスを使用するには、X Window System が必要です

ToolTalk サービスは、メッセージの配信に X のメッセージやプロトコルを使用しません。ToolTalk サービスが X Window System と関連付けられるのは、X セッションを動作させた場合だけです。

X セッションを起動すると、そのセッション名が X サーバーのルートウィンドウ上にプロパティ (名称は `TT_SESSION`) として表示されます。この X サーバーをディスプレイとして指定するすべてのプロセスは、その X セッションをデフォルトセッションとして取得します。X セッションは個々の X ディスプレイ上で表示を行うプロセスのグループとして定義されているため、定義に従えば X Window System を起動する必要がありますが、ToolTalk サービスから要求があるまで X Window System を起動する必要はありません。

動作中の X サーバーが 1 つもない (たとえば、ダム端末上で動作する文字モードアプリケーションだけのセッションを実行する) 場合は、「プロセスツリーセッション」を使用してください。プロセスツリーセッションを実行すると、そのセッション名は環境変数 `TT_SESSION` に表示されます。プロセスツリーセッションは、プロセスツリー上で、自分より下のすべてのプロセスに対してデフォルトセッションになります。

MIT X で ToolTalk サービスを使用できますか

使用できます。ただし、`libtt.so` ファイル用に、`LD_LIBRARY_PATH` で `/usr/dt/lib` を指定しなければなりません。

XセッションのセッションIDはどこにありますか

この識別子を入力するには、次のコマンドを入力します。

```
xprop -root | grep TT_SESSION
```

注 - Xセッションは、ルートウィンドウの TT_SESSION プロパティ上にそのセッションIDを表示します。

tt_open はどのように ttsession に接続しますか

内部的な初期設定を行なった後、tt_open は ttsession の検索を開始します。

1. tt_open は、環境変数 TT_SESSION が設定されているか検査します。
設定されている場合は、その値を ttsession の ID として使用します。
設定されていない場合は、環境変数 DISPLAY が設定されているか検査します。
 - 設定されている場合は、その値を ttsession の ID として使用します。
 - 設定されていない場合は、(そのディスプレイを実行しているマシンの) ルート X Window System 上の TT_SESSION プロパティが設定されているか検査します。

以上の結果、これらの環境変数が1つも設定されていない場合は、tt_open は自分で ttsession を起動します。

2. tt_open は、ttsession が動作中か確認します。
3. tt_open は環境変数 TT_TOKEN を検査し、そのクライアントが ptype の「Start」コマンドから起動されたかどうかを決定します。
ptype の「Start」コマンドから起動された場合、tt は procid を作成します。
4. tt_open は、ttsession が接続するクライアント側の TCP/IP ソケットを作成します。

ソケット上の動作は、関連付けられたファイル記述子で通知されます。
ttsession はこのチャンネルだけを使用して、着信メッセージをクライアントに通知します。

注 - このファイル記述子には、tt_close を使用してください。close 関数を使用しないでください。tt_fd が返すファイル記述子に close 関数を使用すると、後で tt_open や close を呼び出すたびに、ファイル記述子のカウント値が増えます。

5. `tt_open` は、データベースのホスト名リダイレクト用マップを読み取ります。

tt_open を呼び出した後、セッションが実際に始まるのはいつですか

デフォルトセッションが X セッションであり、動作中の `ttsession` がない場合は、`libtt` がセッションを 1 つ起動します。そうでない場合は、セッション名を入手するために、`ttsession` を最初に起動しなければなりません。

別のセッションが接続されると最初のセッションは終了しますか

終了しません。最初のセッションはまだ動作しています。

動作するマシンが異なるプロセス同士は、ToolTalk サービスをどのように使用すれば通信できますか

動作するマシンが異なるプロセス同士が ToolTalk サービスを使用して通信するには、次の 2 つの方法があります。

1. 同一セッションに接続する。
2. それぞれのマシン上で NFS にファイルをマウントし、そのファイルを配信範囲とする。

同一セッションへの接続

複数のプロセスを同一セッションに接続するには、まずプロセス共通の処理対象 (セッション名など) を決定します。次に、それらの全プロセスにセッション名を伝達する方法を決定します。ToolTalk サービスには、セッションアドレスを配布する手段がありません (できるのは、X サーバルートウィンドウの `TT_SESSION` プロパティにセッション ID を表示することだけです)。

セッション名を入手するには、次のコマンドを使用できます。

```
ttsession -p
```

これはセッションを新規に生成し、標準出力にそのセッション名を出力します。また、次のコマンドも使用できます。

```
ttsession -c
```

これは環境変数 `$TT_SESSION` にセッション ID を設定します。

次に、他のプロセスが見つけれられる場所に、セッション名を設定する必要があります。セッション名を設定する場所の例としては、次のものがあります。

- 共有ファイル
- .plan ファイル
- メールメッセージ
- 独自に設計した別の RPC 呼び出し
- NIS

NFS™ 公開ファイルシステムで、公共ファイルを使用する場合の例を次に示します。

1. 次のコマンドで `ttsession` を起動します。

```
ttsession -p >/home/foo/sessionaddress
```

2. この `/home/foo/sessionaddress` ファイルに入っているセッションアドレスをクライアントで確実に使用するには、たとえば、次のようなシェルスクリプトでセッションアドレスを読み取り、`_SUN_TT_SESSION` を設定してからクライアントを起動します。

```
#!/bin/csh
setenv TT_SESSION `cat /home/foo/sessionaddress`
exec client-program
```

プロセスからこのセッションに接続するには、`tt_default_session_set` を呼び出す際にセッション名を指定します。

また、個々の X サーバーに関連付けられた `ttsession` からメッセージを送信すると、新規に作成した `ttsession` を表示できます。

NFS にマウントしたファイルを配信範囲とする

ファイルが配信範囲になるのは、ファイルを配信範囲とするパターンをプロセスが登録したときです。登録したプロセスのセッション名は、登録したファイルに関連付けられた `rpc.ttdbserverd` のセッションリスト上に格納されます。ファイルを配信範囲とするメッセージが送信されると、ToolTalk サービスは、この該当ファイルのセッションリストを検索し、そのリスト上の各セッションにメッセージを配布します。

注 - NFS にマウントしたファイルを配信範囲とするには、すべてのシステム上で1つのファイルシステムを NFS にマウントし、rpc.ttdbserverd を NFS サーバー上で実行する必要があります。

tt_default_session_set の目的は何ですか

tt_default_session_set は、tt_open 呼び出しで接続する ttsession を指定します。

1つのプロセスで2つ以上のセッションに接続するには、どうすればいいですか

ToolTalk サービスと通信する際に使用されるデフォルト変数を表 D-3 に示します。

表 D-3 デフォルト変数

変数	説明
procid	tt_open が設定する。この変数により ttsession はクライアントを識別する
ptype	tt_ptype_declare が設定する
file	ファイルを結合すると設定される。メッセージ中にファイルが1つも設定されていない場合は、デフォルトファイルが設定される

API 関数を使用して procid の取得と設定を行うと、アプリケーションから複数セッション間の切り替えができます。次に例を示します。

```
connect to session 1
store the default procid in filename
connect to session 2,
store the default procid filename
restore associated default procid
interact with particular_session
```

注 - デフォルトのファイルと ptype は、現在のデフォルト procid の一部です。デフォルトの procid を変更すると、デフォルトのファイルと ptype も、その procid に関連付けられたものに変更されます。

あらかじめ決めたセッション ID で ttsession を起動できますか

起動できません。ttsession のセッション ID は、ToolTalk サービスから入手しなければなりません。

セッション ID には、どのような情報が入っていますか

セッション ID はたくさんのフィールドから構成されていますが、その中には次のものがあります。

- アドレスフォーマットのバージョン
- プロセスの UNIX pid
- RPC 非常駐プログラム番号
- 未使用バージョン (互換性のために使用)
- 認証レベル
- ユーザー ID
- ホストの IP アドレス
- RPC のバージョン



注意 - セッション ID のフォーマットは、公開されていないインタフェースです。セッション ID のフォーマットに依存するような ToolTalk クライアントを作成しないでください。

プログラムが新しくセッションに参加したことを通知する標準的な方法がありますか

プロセスが新しくセッションに参加する際は、通知メッセージを送って、処理対象のプロセスに通知してください。また、プロセスが新しくセッションに参加した際に通知をもらいたいプロセスは、その通知メッセージを監視するためのパターンを登録しておく必要があります。

注 - デスクトップサービスの「Started」メッセージは、このために開発されました。

メッセージの行き先を教えてください

送信した各メッセージを ttsession がどのように処理するかを監視するには、起動時に `-t` (トレースモード) を指定します。次のように `USR1` シグナルを ttsession に送信すると、トレースモードのオンとオフを切り替えることができます。

```
kill -USR1 <ttsession_pid>
```

また、`ttsnoop` ユーティリティ (または `tttrace` ユーティリティ) を使用すると、削除メッセージを監視できます。

メッセージの基本的なフローを教えてください

メッセージフローには、次の2種類があります。

- セッションを配信範囲とするもの
- ファイルを配信範囲とするもの

セッションを配信範囲とするメッセージフロー

セッションを配信範囲とするメッセージの基本的なフローは、次のとおりです。

1. クライアントが要求メッセージを作成し、`tt_message_send` を呼び出します。
2. `ttsession` がハンドラを見つけます。
`ttsession` はハンドラを起動する際に、環境変数 `TT_TOKEN` を設定します。
3. ハンドラが動作を開始し、`tt_open` と `tt_fd` を呼び出して、`ttsession` への通信を確立します。
4. ハンドラは自分の `pctype` を `ttsession` に宣言します。
5. `ttsession` は `pctype` の静的パターンをすべて動的パターンに変更します。
この時点では、ハンドラがまだセッションに参加していないため、そのパターンは無効です。
6. ハンドラがセッションに参加して、パターンが有効になります。
7. `ttsession` は、メッセージが待ち行列に入っていることをハンドラに通知します。
8. ハンドラはファイル記述子で通知を受け、`tt_message_receive` を呼び出してそのメッセージを検索します。
`tt_message_receive` が返したメッセージ状態が `TT_WRN_START_MESSAGE` の場合、ToolTalk サービスはすでにそのプロセスを起動してメッセージを配信済みです。この場合 `pctype` 用のメッセージは、プロセスがメッセージ (通知の場合でも) に対して応答、拒否、異常終了のいずれかの処理を行うか、`tt_message_accept` を呼び出すまでブロックされます。
9. ハンドラは要求された操作を実行します。
10. ハンドラは要求に対する応答を返します。
11. `ttsession` は、(応答) メッセージが待ち行列に入っていることをクライアントに通知します。

クライアントのファイル記述子が有効になります。

注 – 実際には、要求メッセージの状態が変化するたびに、クライアントはメッセージを受信します。

12. クライアントは `tt_message_receive` を呼び出し、結果を検索します。

ファイルを配信範囲とするメッセージフロー

ファイルを配信範囲とするメッセージの基本的なフローは、次のとおりです。

1. ファイルを配信範囲とするパターンが登録されます。
ファイルと、パターンを登録しようとしているセッションを、`libtt` がデータベースサーバーに通知します。
2. `libtt` はデータベースサーバーに照会して、指定されたファイルを処理対象として登録しているクライアントのセッションをすべて検索します。
 - 通知の場合、`libtt` はこれらのセッションすべてと直接通信します。
 - 要求の場合、`libtt` はメッセージと、関連する他のセッションリストをそのセッションに通知します。
3. これらのセッションは互いに通信し合い、ハンドラを探します。

アプリケーションにメッセージが到着するとどうなりますか

アプリケーションにメッセージが到着すると、次のようになります。

1. ファイル記述子が有効になります。
2. Xt メインループが `select` から抜け出し、`XtAppAddInput` を呼び出しで登録してある関数を呼び出します。
3. この登録関数は、`tt_message_receive` を呼び出します。
メッセージが読み込まれ、メッセージに関連付けられているコールバックがすべて起動されます。
4. メッセージのコールバックから戻ります。
 - メッセージコールバックの戻り値が `TT_CALLBACK_PROCESSED` の場合、`tt_message_receive` は値として `NULL` を入力コールバックに返します。
 - メッセージコールバックの戻り値が `TT_CALLBACK_CONTINUE` の場合、メッセージの `Tt_message` ハンドルが返されます。
5. 入力コールバックは他の処理を継続します。

たとえば、次のような入力コールバックに対して、

```
input_callback(...)
{
    Tt_message m;
    printf ("input callback entered\n");
    m = tt_message_receive();
    printf ("input callback exiting, message handle is %d\n",
           (int)m);
}
```

メッセージコールバックが次のようになっている場合、

```
message_callback(...)
{
    printf("message callback entered\n");
    return TT_CALLBACK_PROCESSED;
}
```

次のように出力されます。

```
input callback entered
message callback entered
input callback exiting, message handle is 0
```

メッセージを区別する方法を教えてください

メッセージの区別は、次のように行います。

- 動作中のすべての `ttsession` では、メッセージを一意に識別する識別子がメッセージごとに1つあります。
- `tt_message_user` 呼び出しを使用すれば、ユーザーセルに関する情報を読み込み、メッセージをアプリケーションの内部状態に関連付けることができます。
- メッセージハンドルは同じです。たとえば、例 D-1 では、受信メッセージと送信メッセージが同じかどうかを調べています。

例 D-1 メッセージの区別

```
Tt_message m, n;
m = tt_message_create();
...
tt_message_send(m);

... wait around for tt_fd to become active

n = tt_message_receive();
if (m == n) {
// this is a reply to the message we sent
    if (TT_HANDLED == tt_message_state(m)) {
        // the receiver has handled the message, so we can go on
        ....
    }
} else {
    // this is some new message coming in
}
```

プロセスは自分自身に要求を送信できますか

送信できます。プロセスは、自分で処理する要求を送信できます。このような要求は通常、次のように処理します。

```
{
    ...
    tt_message_arg_val_set(m, 1, "answer");
    tt_message_reply(m);
    tt_message_destroy(m);
    return TT_CALLBACK_PROCESSED;
}
```

しかし、ハンドラと送信者が同じプロセスの場合、(同じプロセスに) 応答が返って来た時点では、メッセージは既に削除されています。送信側がメッセージに添付したメッセージ(コールバックやユーザーデータなど)もすべて削除されています。これを防ぐには、メッセージを削除しないでください。次に例を示します。

```
{
    ...
    if (0 != strcmp(tt_message_sender(m), tt_default_procid())) {
        tt_message_destroy(m);
    }
}
```

tt_message_callback_add で登録した関数に、自分自身のデータを渡せますか

tt_message_callback_add で登録した関数に自分自身のデータを渡すには、メッセージのユーザーデータセルを使用します。次に例を示します。

```
x = tt_message_create();
tt_message_callback_add(x, my_callback);
tt_message_user_set(x, 1, (void *)my_data);

....

Tt_callback_action
Tt_message_callback(Tt_message m, Tt_pattern p)
{
    struct my_data_t *my_data;
    my_data = (struct my_data_t *)tt_message_user(m, 1);

    ...
}
```

注 - ユーザーデータは、送信先クライアントでしか見ることができません。

任意のデータをメッセージで送信する方法を教えてください

ToolTalk サービスには構造体を送信する方法が組み込まれていないため、送信できるのは文字列、整数、バイト配列だけです。構造体を送信するには、XDR ルーチンを使用して構造体をバイト配列に変換した後、メッセージに設定します。このシリアル化を解除するにも、同じ XDR ルーチンを使用します。

ToolTalk サービスでファイルを転送できますか

直接は転送できません。ただし、次のことは可能です。

- ファイルデータをメッセージの引数に設定する。
ToolTalk サービスはメッセージデータを、アプリケーションからライブラリに、ライブラリから ttsession に、ttsession から受信側のライブラリへとコピーし、受信側が引数値を取得する際に、この受信側のライブラリから取り出します。データが大きいと、この方法では非常に遅くなり、使用するメモリーも膨大になり

- ます。
- ファイル名をメッセージの引数に設定する。
この方法では、すべての受信側がファイルを同じ場所にマウントすることが前提です。
 - ファイル名を `tt_message_file` 属性に設定する。
この方法でもすべての受信側がファイルをマウントすることが前提ですが、マウント場所が異なる場合は、ToolTalk サービスがすべて解決します。

ToolTalk サービスは、メモリー (バイト) の順序の問題をどのように処理しますか

ToolTalk サービスでは、整数、文字列、バイト配列をメッセージに格納できます。XDR ルーチンにより、これらのデータ型はどのクライアントでも保証されています。これら 3 つの型以外のデータの場合は、メッセージに設定する前にシリアル化して、バイト配列にしなければなりません。

メッセージは再使用できますか

再使用できません。引数を変えて同じメッセージを複数回は送信できません。メッセージはそれぞれ、作成、送信、破棄のサイクルを繰り返さなければなりません。

メッセージを破棄するとどうなりますか

メッセージを破棄する際、そのハンドルは破棄できますが、メッセージ本体は破棄できません。メッセージ本体は、ToolTalk がそのメッセージの処理を完了し、外部のハンドルがすべて破棄されたときにだけ破棄されます。たとえば、メッセージを送信した直後にそのハンドルを破棄しても、応答が返って来た時点で新しいハンドルが渡されます。

メッセージをいったん破棄してしまうと、二度とそのメッセージを見ることはできません。たとえば、自分が送信する要求を監視するパターンを登録し、パターンが一致した時点でそのメッセージを破棄すると、そのメッセージの状態が「handled」(応答)になってもそのメッセージを見ることはできません。

1 つのメッセージを 2 つ以上のハンドラで処理できますか

現在は処理できません。1 つのメッセージを複数のプロセスで処理したい場合は、通知を使用できます。また、メッセージの拒否を利用して ToolTalk サービスに、使用可能なハンドラすべてに要求を配信させることもできます。ただし、これらのハンドラはそれぞれ、実際には何らかの操作を処理する必要があります。

1 つの ptype のハンドラを 2 つ以上実行できますか

実行できます。ただし、ToolTalk サービスには負荷分散の概念がないため、ToolTalk はいくつかのハンドラから 1 つだけ選択し、一致するものであれば、その他のメッセージもそのハンドラだけに配信します。メッセージをその他のハンドラにも配信させるには、次のような方法があります。

1. `tt_message_reject` を使用する。

メッセージを受信してもビジーなのでそのメッセージを処理したくないプロセスは、メッセージを拒否できます。この場合、ToolTalk サービスは使用可能な次のハンドラを探します。(登録されているハンドラがすべて拒否した場合は、処置オプションが適用されます。)

この方法では、`tt_fd` が有効になったら `tt_message_receive` を呼び出すというイベントループを、プロセスが実行中でなければなりません。ただし、プロセスが複雑な計算ループを実行していると、この方法ではうまくいきません。

2. ビジーになるようなメッセージの場合は、そのパターンを登録解除する。次に例を示します。

```
m = tt_message_receive();
if (m is the message that causes us to go busy) {
    tt_pattern_unregister(p);
}
```

ToolTalk サービスは、パターンが登録されていないと、一致するメッセージをプロセスに渡しません。プロセスでもう一度メッセージを受信したい場合は、パターンを再登録してください。

注 - この方法を使用すると、競争条件が生じます。たとえば、`tt_message_receive` と `tt_pattern_unregister` 呼び出しの間に次のメッセージが送信され、このプロセスに渡される可能性があります。

3. 方法 1 と方法 2 を組み合わせる。

方法 1 と方法 2 を組み合わせて、次のようにすることもできます。

```
get the message
unregister the pattern
loop, calling tt_message_receive until it returns 0; reject
all the returned messages
handle the message
re-register the pattern
repeat
```

注 - この方法では、プロセスで登録するパターンは1つであることが前提です。

メッセージの処置の値とは何ですか

メッセージの処置を指定すると、静的型定義に指定された処置の値を上書きできます。ハンドラに `pctype` を指定したメッセージが静的シグニチャに一致しないと、メッセージで指定した処置に従って処理されます。たとえば、メッセージの処置に `TT_START`、`pctype` に開始文字列を指定すると、インスタンスが1つ起動されます。

メッセージ状態要素とは何ですか

ToolTalk サービスでは、`message_status_string` を使用しません。このメッセージ要素は、アプリケーションが使用するためのものです。ToolTalk サービスがこのメッセージ状態を設定するのは、メッセージの配信に問題が発生したときだけです。その他の場合、このメッセージ要素はアプリケーション独自の方法で設定または読み取りを行います。

tt_free はいつ使用しますか

アプリケーションがデータバッファとして受け取る内部記憶領域のスタックは、`libtt` が管理します。アプリケーションが ToolTalk API ルーチンから返される `char *` や `void *` はすべて、そのコピー領域を指していて、これらの領域はアプリケーションで解放する必要があります。

割り当てられたバッファは、マーク関数と解放関数を使用して、一連の操作で解放してください。ただし、解放関数を呼び出すと、対応するマーク関数を呼び出した以降に割り当てられたすべての領域が解放されます。ToolTalk サービスが返したデータを一部残しておきたい場合は、解放操作を実行する前にデータのコピーを取っておいてください。

ptype とは何ですか

ptype はツールの種類を指定する文字列で、プログラマが定義します。(「プロセス型」と呼ぶこともあります。) 各 ptype は一連のパターンに関連付けることができます。そのパターンには、ptype が処理対象とするメッセージと、その ptype のインスタンスを起動する際に ToolTalk サービスが使用する文字列を指定します。

ptype の主な目的は、ツールのインスタンスがメッセージの配信範囲で1つも動作していないときも、メッセージをツールの処理対象にできるようにすることです。メッセージが要求する操作を実行できる場合や、メッセージが送信された際に通知を受けたい場合は、ツールの ptype でその旨を指示しておく、ToolTalk は必要に応じてそのツールを起動します。ptype データベースはシステム管理者やユーザーが変更できるため、現場やユーザーの好みに応じて、個々のメッセージを処理するツールを指定できます。

新しく作成した型が認識されない理由を教えてください

ttsession がデータベースタイプを読み込むのは、起動時、USR1 信号受信時、またはデータベースタイプが変更されたときに特別な ToolTalk メッセージによってその旨を通知されたときです。通常、手作業で tttsession を更新してファイルタイプをもう一度読み込む必要はありません。ただし、強制的に実行中の tttsession にデータベースタイプを再読み込みさせたいときは、下記のような USR2 信号を送信することによって実行できます。たとえば、次のように送信します。

```
kill -USR2 <ttsession pid>
```

ptype のプロセスが既に存在する場合、ptype 情報は使われますか

ToolTalk サービスは、すべてのメッセージに対して常に1つのハンドラと任意の数のオブザーバを探します。この場合、ToolTalk サービスは動作中のハンドラを1つ見つけても、メッセージに一致する任意の監視パターンを ptype の中から探します。一致する監視パターンの ptype が1つ存在するが、その ptype のプロセスが動作していない場合、ToolTalk サービスは (ptype パターンまたはメッセージの指定に従って) プロセスを新しく起動するか、メッセージを待ち行列に入れます。

(インスタンスが既に動作中かどうかにかかわらず) インスタンスを常に起動するよう ptype の定義を変更できますか

変更できません。起動時の ptype へのメッセージは、ptype がそのメッセージに応答するか、tt_message_accept 呼び出しを発行するまでブロックされます。しかし ptype は、TT_WRN_START_MESSAGE 状態以外のすべての要求に対して、tt_message_reject を実行できます。したがって、その ptype を持つ動作中のすべてのインスタンスにすべての要求が配信され (そして拒否され) てから、インスタンスが新しく起動されます。この方法は、同時に動作している ptype が多い場合や、メッセージに大量のデータが入っている場合は遅くなります。また、tt_message_accept を使用することもできます。これは、ptype へのメッセージを基本的にはブロックしません。

tt_ptype_declare は何を行いますか

ptype を宣言した時点では、静的パターンは ttsession のメモリーに存在します。アプリケーションが ptype を登録すると、ToolTalk サービスはその ptype を指定している otype をチェックし、otype 中のパターンも登録します。静的パターンを有効にするには、アプリケーションから適切な join 関数を呼び出す必要があります。

注 - 1 つのアプリケーションで同じ ptype を複数回宣言しても無視されます。

TT_TOKEN とは何ですか

アプリケーションの起動を要求するメッセージを処理する際、ToolTalk サービスは、その子プロセスの _SUN_TT_TOKEN 環境変数を設定します。アプリケーションが動作を開始して tt_open を実行すると、この情報は ToolTalk サービスに戻され、メッセージを処理するのに起動、つまり任命されたのはそのアプリケーションであることを ToolTalk に知らせます。

パターンはいつ有効になりますか

パターンは、そのパターンを有効にしたいセッションに登録しなければなりません。パターンは (1 つの procid で) 2 つ以上のファイルに対して有効にできるため、パターンのファイル部はリストされたすべてのファイルに一致します。

注 - コンテキストは配信範囲ではありません。コンテキストに結合してもファイルやセッションに結合していないパターンは、どのメッセージにも一致できません。

応答を入手するにはパターンの登録が必要ですか

必要ありません。ただし、応答に一致するパターンを登録すると、その応答はイベントループに2度表われます。1度目はパターンに一致したため、2度目はそれが応答であるためです。

要求を監視するにはどうすればいいですか

パターンに一致し、しかもポイントツーポイント (つまり TT_HANDLER) ではない要求メッセージは監視できます。監視パターンがどの要求にも一致しない場合は、ttsession をトレースモードで動作させれば原因が分かります。

静的パターンの属性値と照合させる方法を教えてください

ToolTalk の静的パターン (つまり、型データベース) 機構では、属性値でパターンと照合させることはできません。ファイルの配信範囲や引数の vtype で照合させることはできますが、ファイル名や引数値で照合させることはできません。

注 - この制限は、静的パターンのコンテキストの照合にも当てはまります。

TT_HANDLER に対してワイルドカードでパターンを指定できない理由を教えてください

TT_HANDLER にアドレス指定されたメッセージは、パターンと照合されないため、ワイルドカードでパターンを指定できません。

ファイルを配信範囲とする任意のメッセージを監視するようなパターンを設定できますか

設定できません。ファイルを配信範囲とする際にファイル名を指定しないのは、すべてのファイルについてファイルを配信範囲とするメッセージと照合するよう指定するのと実質的には同じです。

注 - ファイルを配信範囲とするパターンにセッション属性を設定して、セッション中のファイルを配信範囲とするようエミュレートしてもかまいません。ただし、配信範囲が TT_FILE になっているパターンのセッション属性は、tt_session_join を呼び出しても更新されません。

静的パターンのファイル配信範囲は file_in_session と同じですか

違います。これらの配信範囲は使用目的が違います。

たとえば、すべてのセッションが同じ静的パターンを持ち、(アプリケーションが送信しようとしている) メッセージ M に一致するパターン P を少なくとも 1 つ持つとします。そして、ファイル foo.bar を処理対象とするクライアントのセッションが 1 つもないとします。

アプリケーションはセッション A に接続しており、ファイル foo.bar を配信範囲とするメッセージ M を発行します。このファイルを処理対象とするクライアントのセッションは 1 つもないため、このメッセージを受け取るファイルはセッション A だけです。(このメッセージは、セッション A の静的パターン P と一致します。) ptype が動作を開始すると、そのパターンは実際には (そのセッション中の) ファイルを配信範囲とするようになり、セッション A がすべて処理します。

セッションの静的パターンがすべて同じとは限らない場合は、結果は異なります。たとえばセッション B が、ファイル配信範囲指定でメッセージ M に一致する P' というパターンを持つことも考えられます。メッセージ M がセッション A に送信されると、ファイル foo.bar を処理対象にしたクライアントがセッション B に 1 つもない場合、dbserver はそのメッセージをセッション B に送信しません。しかし、セッション B のクライアントの 1 つがファイル foo.bar を処理対象としていれば、セッション B の少なくとも 1 つのクライアントがファイル foo.bar を処理対象としていることを知り、dbserver はセッション B にもそのメッセージを送信します。

arg_add、barg_add、iarg_add の違いを教えてください

barg_add と iarg_add の呼び出しも、基本的には一組の値が後ろに付いた arg_add 呼び出しです。

メッセージ引数の type や vtype とは何ですか

メッセージ引数の type や vtype (value type の短縮形) は、その引数値が意味を持つ領域を示し、アプリケーションが指定します。

vtype は C の typedef に似ています。すべての vtype は通常、その引数に設定できる 3 種類のデータ型のどれか 1 つに対応します。

vtype 機構を使用すれば、2 つの値を同じ型として宣言できます。たとえば、messageID と bufferID という 2 つの vtype を C の文字列と同様に、それぞれ異なる意味を持つように定義し、操作によって messageID にだけ有効、bufferID にだけ有効、または両方の vtype に有効とすることもできます。このパターン照合機構を使用すれば、bufferID 文字列を指定した要求は、messageID 文字列にだけ有効な操作のパターンには一致しません。

コンテキストの使用方を教えてください

コンテキストは、照合を制限するのに使用できます。照合を制限するには、照合する確率を大きくするために、メッセージは同じコンテキスト、つまりコンテキストのスーパーセットを持たなければなりません。また、コンテキストスロット名がドル記号 (\$) で始まり (たとえば \$ISV)、そのメッセージによってアプリケーションが起動される場合、コンテキストスロットにどのような値が設定されていても、その値がアプリケーションの環境変数に設定されます。

ttsession はどのように照合をチェックしますか

ttsession が照合を検査するさまざまな方法を表 D-4 に示します。

表 D-4 tttsession の照合の検査方法

方法	説明	一致するか
TT_HANDLER	このようなアドレス指定は「ポイントツーポイント」配信であり、メッセージは受信側に直接渡される。登録されたパターンは検査されないため、ポイントツーポイントのメッセージは監視できない	照合は不要

表 D-4 ttsession の照合の検査方法 (続き)

方法	説明	一致するか
TT_PROCEDURE	操作 (op) の同じ静的シグニチャ (sig) のリストを検索し、オブザーバと見込みのあるハンドラのリストを収集する。	=> 一致する
	sig が引数もコンテキストも持たない場合	=> 一致しない
	sig プロトタイプ (引数の個数、型、モード) の値が異なる場合	=> 一致する
	sig コンテキストがメッセージのコンテキストのサブセットである場合	
	待ち行列に入れる必要のあるすべての静的オブザーバ用情報を保存する。	
	動的パターンを検索し、オブザーバと使用可能なハンドラのリストに追加する。このリストを作成するのに、ttsession はまず操作付きのパターンを使用し、次に操作なしのパターンを使用する。	
TT_OBJECT & TT_OTYPE	信頼性、状態、クラス、アドレス、ハンドラ、ハンドラの ptype、配信範囲、オブジェクト、otype、送信側、送信側の ptype、引数、コンテキストを検査する。	
	まずオブザーバに配信する (ハンドラは状態が変わる可能性があるため)。	
	最もよく一致したハンドラに配信する。2 つ以上のハンドラが等しく「最適」な場合は、任意に選択する。	
	type 引数が設定されているかどうかを検査する。	=> 一致しない
	sig が異なる otype を持つ場合	=> 一致しない
	sig が otype を 1 つも持たず、配信範囲が異なる場合	
	その他の場合は、TT_PROCEDURE	

ToolTalk サービスには、配信範囲が何種類ありますか

現在は、セッションとファイルの 2 種類だけです。

注 - X セッションも 1 つの配信範囲と言われることがありますが、実際にはセッション配信範囲です。

TT_DB ディレクトリとは何ですか、また、型データベースと TT_DB ディレクトリは何が違いますか

ToolTalk 型データベースには、静的な `p_type` と `o_type` の定義が入っています。これらの定義は、アプリケーションやオブジェクトが応答するメッセージを宣言しています。静的型定義を追加または変更すると、ToolTalk 型コンパイラがその型データベースを更新します。`tt_session` は起動時、これらの型ファイルを読み込みます。

TT_DB データベースは、`rpc.ttdbserverd` が作成します。`tt_db` ディレクトリには、このパーティション内のファイルと、そのファイルを処理対象とするパターンのセッションの関連付けが入っています。また、このパーティション内のファイルのオブジェクト仕様情報もすべて入っています。

tt_db データベースには何を入れればいいですか

`tt_db` データベースには現在、次の 10 個のファイルが入っています。

```
access_table.ind
access_table.rec
file_object_map.ind
file_object_map.rec
file_table.ind
file_table.rec
file_table.var
property_table.ind
property_table.rec
property_table.var
```

これらのファイルのアクセス権は、`-rw-r--r--` に設定されています。

rpc.ttdbserverd は何を実行しますか

ToolTalk データベースサーバーのデーモンは、次の主な機能を実行します。

1. `tt_file_join` 呼び出しでファイルを結合したクライアントのセッション ID を格納する。
2. メッセージの処置が `TT_QUEUED` で、しかもそのメッセージを処理できるハンドラがまだ起動されていないために待ち行列に入っているファイル配信範囲のメッセージを格納する。
3. ToolTalk のオブジェクト仕様を格納する。
4. ToolTalk ファイル名フレームマッピング API への要求に応答する。

ttsession と rpc.ttdserverd は通信しますか

通信しません。

どのような帯域幅のメッセージをサポートできますか

小さなメッセージなら 1 秒あたり約 100 個です。性能は主に、各メッセージの受信側の数で決まります。つまり、どのパターンにも一致しない通知が最も安く、多くのオペレータに一致するメッセージが最も高くなります。

メッセージサイズや引数の個数に制限はありますか

制限はありません。ToolTalk メッセージのサイズや引数の個数に設計上の制限はありませんが、ToolTalk はメッセージデータを (クライアントのアドレス空間内の領域間、および RPC 接続を経由したサーバーとの間で) 何回かコピーします。たとえば、ToolTalk メッセージで 1M バイトのデータを送信すると、次のように少なくとも 4 回コピーされます。

- アプリケーション領域から ToolTalk ライブラリ領域へ
- ToolTalk ライブラリ領域から ToolTalk サーバーへ
- ToolTalk サーバーから受信側のライブラリへ
- 受信側のライブラリから最終目的地へ

このメッセージを監視しているプロセスがあると、さらにコピー回数が増えます。また、ttsession プロセスはシングルスレッドであるため、コピー期間中は、このセッションに他のメッセージは配信されません。このため、非常に大きなデータを頻繁に送信する場合は、ToolTalk 以外の方法でデータを渡すよう検討する必要があります。

メッセージを送信する際、最も時間効率のよい方法は何ですか

手続き的なメッセージで 1 つの受信者だけに一致させるより、直接処理する (つまり、TT_HANDLER を使用してメッセージをアドレス指定する) 方が高速です。

ネットワークのオーバーヘッドの種類を教えてください

ToolTalk サービスでは、ハードウェアによるブロードキャストやマルチキャストを使用しません。メッセージは、(ネットワークを経由するかどうかに関係なく) 該当セッションの `ttsession` プロセスに直接送信されます。パターンを登録すると、そのパターンも `ttsession` プロセスに直接送信されます。`ttsession` プロセスは、すべてのパターンに対してメッセージを照合し、メッセージに一致するパターンを登録してあるプロセスにだけメッセージを直接送信します。メッセージを処理対象とするプロセスがマシン上にない場合、そのマシンは起動されません。

ToolTalk サービスは、要求を処理するのに負荷分散を使用しますか

使用しません。ToolTalk サービスは、負荷分散機構ではありません。同じパターンを持つプロセスを2つ登録してある場合、ToolTalk サービスはどちらか1つのプロセスを任意に選択して、一致するメッセージをすべてそのプロセスに配信します。プロセスがビジーの間はパターンを登録解除して、同時にパターンを登録解除する前に受信していた可能性のあるメッセージをすべて拒否すると、負荷を分散できます。

ToolTalk アプリケーションに必要なリソースには何がありますか

メッセージを処理するには、送信側クライアント、`ttsession`、受信側クライアントで約数百キロバイトのワーキングセットが必要です。クライアントがメッセージをタイムリに処理している限り、ToolTalk のメモリー条件は、時間が経過しても変わりません。

`ttsession` が異常終了するとどうなりますか

`ttsession` に障害が発生すると `tt_fd` が有効になり、ToolTalk API 呼び出しのほとんどが次のような `TT_ERR_NOMP` エラーメッセージを返します。

```
No Message Passer
```

このメッセージが返されると、ほとんどのアプリケーションは `ttsession` に何か問題が発生したもとして、ToolTalk メッセージの送受信を停止します。この状況から回復する方法を次に示します。

- TT_ERR_NOMP という状況を認識する。
- tt_close を呼び出し、相手との接続を切断する。
- ToolTalk サービスを再度初期設定する。
- 次の順序で呼び出す。

```
tt_open, tt_default_session_join, tt_fd
```

- パターンをすべて再登録し、ptype を再度宣言する。

注 - 障害の発生した ttsession を再起動して障害発生箇所から継続する際、環境変数 TT_SESSION の設定と、ルートの X Window System がある場合は TT_SESSION プロパティ値を操作する必要がある場合があります。また、障害の発生したセッションの他の参加者も回復できるように、再起動したセッションと新しいセッションの ID を連絡する必要があります。

ttsession に障害が発生すると、次の項目は回復できません。

- 障害が発生したセッションの procid で登録したパターン
- 障害が発生したセッションの procid から出した要求で保留されているもの
- 障害が発生したセッションの procid で tt_message_send_on_exit 呼び出しに渡したメッセージ
- セッションのプロパティ
- セッションの待ち行列に入っているメッセージ

rpc.ttdbserverd が異常終了するとどうなりますか

rpc.ttdbserverd が不意に終了すると、inetd は代わりに新しい rpc.ttdbserverd を起動します。データは一時的に使用できなくなりますが消失しません。ただし、API 呼び出しが TT_ERR_DBAVAIL を返すことがあります。API 呼び出しが TT_OK を返すと、dbserver は即座に、または障害回復記録を新しい dbserver が読んだ時点で、ToolTalk データベースを更新します。

ホストやリンクが停止するとどうなりますか

ホストやリンクが停止したことを TCP が検知すると、TCP 接続は破棄されます。プロセスと ttsession の接続が破壊されると、ttsession はそのプロセスが終了したときと同じように動作します。パターンがすべて消去されるため、メッセージを送受信しようとする、プロセスは TT_ERR_NOMP というエラーメッセージを受信します。

tt_close は何をしますか

tt_close を呼び出すと、ttsession は現在の procid だけを閉じます。現在の procid が最後の procid の場合は、tt_open 呼び出し以降に作成された ToolTalk の構造体をすべて消去します。tt_close は tt_fd が返したファイル記述子を使って呼び出さなければなりません。そうしないと、後で tt_open や close を呼び出すたびにファイル記述子のカウンタ値が増えます。

メッセージの配信は、ネットワーク上でも保証されますか

保証されます。メッセージは TCP/IP の RPC を使用して送信するため、配信の信頼性があります。

メッセージの配信には、時間的な順序がありますか

送信側と受信側の間では、メッセージの順序は決まっています。まず、プロセス A がメッセージ M1 を送り、その後でメッセージ M2 を送信し、これらのメッセージをプロセス B が受信する場合、プロセス B はメッセージ M2 を受信する前にメッセージ M1 を受信します。ただし、次の 2 つの例外があります。

1. プロセス B がメッセージ M1 を受信して拒否すると、メッセージ M1 は再度ディスパッチされてプロセス C に行きます。この間 (プロセス B がメッセージ M1 に対して応答するか拒否するかを決定している間)、ToolTalk サービスはメッセージの配信を継続します。このような場合は、後のメッセージが最初のメッセージを「追い越した」ように見えることがあります。
2. プロセス B へのメッセージが待ち行列に入っている場合、待ち行列に入る原因となったパターンの ptype をプロセス B が宣言した時点で、プロセス B 待ち行列に入っていたメッセージを受信します。しかし、プロセス B は実際は、プロセス A から次のメッセージをすべて受信するまで、待ち行列に入ったメッセージ (この場合はメッセージ M1) を受信しない場合があります。

unix、xauth、des とは何ですか

これらは認証方法の種類です。

- unix は、RPC 呼び出しでアプリケーションを呼び出しているエンティティの uid を教えます。dbserver は各 RPC 呼び出しに機密保護を実施し、デフォルト時はこの認証方法を使用します。
- xauth はホームディレクトリ中の読み出し保護ファイルを使用して、X ディスプレイへのアクセスを制御します (これにより、ttsession へのアクセスも制御されます)。

- `des` はデータ暗号化規格 (DES) を使用して、`ttsession` にアクセスするプロセスの身元をチェックします。

アプリケーションが互いにメッセージを隠すことはできますか

できません。あるアプリケーションへのメッセージを他のアプリケーションが隠す機構を、ToolTalk サービスでは意図的に提供していません。

横取りや模造に対して保護機構はありますか

ありません。ToolTalk サービスの「プラグアンドプレイ」概念によれば、アプリケーションは個々のタスクに最適なツールを選択して、インストールしたりインストールを解除したりできます。プロトコル X に対して、アプリケーション A よりアプリケーション B の方が応答が良い場合、アプリケーション A のインストールを解除して、アプリケーション B をインストールできるようにプロトコル X を設計してください。

待ち行列に入ったメッセージはどこに格納されていますか、また、その記憶領域はどのくらい安全ですか

ファイルを配信範囲とするメッセージで待ち行列に入ったものは、配信範囲となるファイルと同じファイルシステム上のデータベースに格納されます。このデータベースを読むことができるのはスーパーユーザーだけであり、(ルートとして動作中の) ToolTalk データベースサーバーは、そのファイルの読み取り権を持つユーザーのプロセスだけにそのメッセージを渡します。

セッションを配信範囲とするメッセージで待ち行列に入ったものは、そのセッションを管理する `ttsession` のアドレス空間に格納されます。`ttsession` は、自分が動作している認証モードを満たすプロセスだけにそのメッセージを渡します。

ToolTalk サービスは C2 保証されていますか

保証されていません。

メッセージの行方を追跡するには、どうすればいいですか

メッセージの行方を追跡するには、使用する `ttsession` のトレース出力をオンにしてください。これを行う最も簡単な方法は、関連する `ttsession` のトレース出力をオンにすることですが、次のコマンドを使って `SIGUSR1` 信号を `ttsession` の実行プロセスに送信する方法もあります。

```
kill -USR1 <unix_pid_of_the_ttsession_process>
```

ToolTalk サービスを使用して、他のすべてのツールから自分のデバッグツールを隔離するには、どうすればいいですか

デバッグツールを隔離するには、「プロセスツリーセッション」モードを使用してください。このモードはセッション名を環境変数に設定して、該当する `ttsession` プロセスを見つけます。このモードを使用するには、次のように実行してください。

1. トレースモードをオンにして、プロセスツリーセッションを新しく起動します。

```
% ttsession -t -c $SHELL
*
* ttsession (version 1.3, library 1.3)
*
ttsession: starting
%
```

`ttsession` が動作を開始して該当する環境変数を設定し、指定されたコマンド (`$SHELL`) を生成します。この段階では、サブシェルを実行しています。このサブシェルから起動したコマンドはすべて、上記ので起動した `ttsession` を使用します。この新しい `ttsession` のセッション ID は、環境変数 `TT_SESSION` に入っています。

2. サブシェルの中で次のテストプログラムを実行します。

```
% ./my_receiver &
[1] 4532
% ./my_sender &

.. and look at the output of the ttsession trace.
```

3. テストが完了したら、サブシェルを終了します。

サブシェルの中で ToolTalk サービスを使用するツールを起動すると、Xセッションの `ttsession` ではなく、プロセスツリーの `ttsession` が使用されるため未定義となります。

C++ で ToolTalk サービスを使用できますか

使用できます。ToolTalk API ヘッダーファイルは、C++ を扱えるようになっていました。C++ を使用する際、`tt_c.h` はすべての API 呼び出しを `extern C` として宣言します。

ファイル名を修飾する必要はありますか

ありません。ToolTalk サービスでは、パス名の明示的なホスト名修飾を許可していません。ファイル名の中にコロン (:) を使用すると、ToolTalk サービスはコロンの入ったファイル名を検索します。`tt_message_file` や `tt_default_file` を呼び出すと、使用中のマシンの画面に表示されるのと同じような形式で、指定したファイルの「実パス」が返されます。ToolTalk サービスでは、次のことを保証しています。

- 2つのクライアントが、異なるマシン上にある同じ名前のファイルを配信範囲に指定した場合、各マシン上にファイルが実際にマウントされているかどうかにかかわらず、これらのクライアントは互いに通信できる。
- ローカルで有効な、標準的なパス名が返される。

ToolTalk オブジェクトとは何ですか

ToolTalk オブジェクトは、通常のオブジェクト指向言語によく出てくるオブジェクトとは少し異なります。

`otype` と継承は実行専用です。2つの仕様が同じ `otype` を持つことはできますが、プロパティが異なります。これらが共有するのは、`otype` の宣言のシグニチャで定義した操作だけです。`otype` の宣言の各シグニチャに対して、`potype` を1つ指定しなければなりません。指定した `potype` (プロセス型) は、この `otype` を持つオブジェクトに対する操作の「実行エンジン」です。仕様のファイル部は要求されたプロパティと同様、すべての仕様がファイル名を持たなければなりません。ただし、そのファイルが存在する必要はありません。仕様のファイル名部分にはいくつかの機能がありますが、その中には次のようなものがあります。

- 仕様を格納するホストやパーティションを指定する。
- オブジェクトに対してグループ化機構を提供する。
- ToolTalk で拡張した標準オペレーティングコマンド (`ttmv` など) で、データベース内外の整合性をとる。

ToolTalk ニュースグループはありますか

あります。ToolTalk news グループは、`alt.soft-sys.tooltalk` です。共通デスクトップ環境 (CDE) は、新規のアプリケーションの統合やアプリケーションプログラムの起動時に ToolTalk を利用しているため、`comp.unix.cde` グループも役立つでしょう。

用語集

CAD	計算機設計支援 (Computer Aided Design)
CASE	計算機ソフトウェア設計支援 (Computer Aided Software Engineering)
fd	ファイル記述子
libtt	ToolTalk アプリケーションプログラミングインタフェース (API) ライブラリ
procid	プロセス識別子
ptid	プロセス型識別子
ptype	プロセス型
rpc.ttdbserverd	ToolTalk データベースのサーバプロセス
sessid	セッションを識別します。
ToolTalk 型データベース	ToolTalk 型情報を記憶するデータベース
ttdbck	ToolTalk データベースの検査および修復ユーティリティ
ttsession	ToolTalk 通信プロセス
tt_type_comp	ToolTalk 型コンパイラ
xdr フォーマットテーブル	ttsession が呼び出されたときに読み取られる型データベース
オブジェクト型 (otype)	アプリケーションのオブジェクト型 (otype) は、オブジェクト指向メッセージの配信時に ToolTalk サービスが使用するアドレス指定情報を持っています。
オブジェクト型識別子 (otid)	オブジェクト型を識別します。
オブジェクト指向メッセージ	アプリケーションが管理するオブジェクトにアドレス指定されたメッセージ

オブジェクト仕様 (スペック)	オブジェクト仕様 (スペックと呼ばれる) には、オブジェクト型、オブジェクト内容が配置されているファイル名、およびその所有者などの標準プロパティがあります。
オブジェクト内容	オブジェクト内容は、オブジェクトを作成または管理するアプリケーションによって管理される。一般的に通常ファイルの一部であり、パラグラフ、ソースコード関数、一連のスプレッドシートセルなどがあります。
オブジェクトファイル	オブジェクト情報が入っているファイル。アプリケーションはファイル内のオブジェクトを照会し、オブジェクトに対してバッチ単位で動作を実行できます。
隠されたポインタ (オペークポインタ)	特定のインタフェースを介して引き渡されたときだけ意味を持つ値
カテゴリ	アプリケーションがパターンに一致する要求を処理したいのか、または要求を監視したいだけなのかを示すパターン属性
コンテキスト	任意のペア (つまり、<名前、値> のペア) を ToolTalk のメッセージとパターンに関連付けること
シグニチャ	ptype または otype におけるパターン。シグニチャには、処置の値および操作番号を入れることができます。 <ul style="list-style-type: none"> ■ ptype シグニチャ (psignatures) は、プログラムが受信したい手続き型メッセージを記述します。 ■ otype シグニチャ (osignatures) は、その型のオブジェクトにアドレス指定可能なメッセージを定義します。
初期セッション	アプリケーションを起動した ToolTalk セッション
スペック	「オブジェクト仕様 (スペック)」を参照してください。
静的なメッセージパターン	定義済みの一連のメッセージを受信したい場合に、メッセージパターン情報を指定する簡単な方法です。
セッション	同一のデスクトップまたは同一のプロセスツリーにより関連付けられたプロセスのグループ
通知	通知とは、情報を伝えるものであり、アプリケーションがイベントのことを連絡するための方法
ツールマネージャ	開発ツールの環境を調整するためのプログラム
動的なメッセージパターン	アプリケーションの実行中にメッセージパターン情報を提供すること
配信範囲	メッセージまたはパターンの属性であり、ToolTalk が一致するメッセージまたはパターンを探す範囲を決定します。
パターンコールバック	クライアント関数。ToolTalk サービスは、指定されたパターンに一致するメッセージを受信したときにこの関数を呼び出します。

パッケージ	共同で数種類のソフトウェアを作成する構成要素のグループ。1つのパッケージは、ソフトウェアを構成する実行可能ファイルだけでなく、情報ファイルとスクリプトも持っています。ソフトウェアは、パッケージの形式に従ってインストールされます。
ファイル	アプリケーションの処理対象であるデータの入れ物
プロセス	ToolTalk サービスを使用するアプリケーション、ツール、またはプログラムを1回実行すること
プロセス指向メッセージ	プロセスにアドレス指定されたメッセージ
分類機構 (CE: Classing Engine)	デスクトップオブジェクトの特性を識別すること。すなわち、デスクトップオブジェクトの印字方法、アイコン、およびファイルのオープン処理コマンドなどの属性を格納します。
分類機構テーブル	OpenWindows の分類機構が読み取る型データベース
マーク	API スタック上の場所を表す整数
メッセージ	ToolTalk サービスがプロセスに配信する構造体。ToolTalk メッセージは、操作名、型引数のベクトル、ステータス値またはステータス文字列ペア、および付属アドレス指定情報から構成されています。
メッセージコールバック	クライアント関数。ToolTalk サービスはこの関数を呼び出し、指定されたメッセージに関する情報を送信側アプリケーションに報告します。この情報とは、たとえば、メッセージが失敗した、メッセージによってツールが起動されたなどです。
メッセージの監視	要求された動作を実行せずにメッセージだけを表示すること
メッセージの処理	送信側アプリケーションから要求のあった操作を実行することであり、要求に対して ToolTalk 応答を送信すること
メッセージパターン	アプリケーションが受信したい情報を定義すること
メッセージプロトコル	メッセージプロトコルとは、アプリケーション間で実行することが同意されている操作を記述した、一連の ToolTalk メッセージ
約束監視	開始処置または待ち行列処置のオブザーバのシグニチャを持つ ptype に一致する各メッセージのコピーを、ToolTalk サービスが配信することを保証します。ToolTalk サービスは、インスタンスを起動して ptype の実行インスタンスにメッセージを配信するか、または ptype のメッセージを待ち行列に入れてメッセージを配信します。
要求	処理を呼び出すこと。処理の結果はメッセージに記録され、メッセージは応答として送信側に返されます。
要求の拒否	要求された操作を受信側アプリケーションが実行できず、別のツールにメッセージを渡す必要があることを ToolTalk サービスに通知します。
要求の失敗	要求された操作を実行できないことを送信側アプリケーションに通知すること

ラップシェルコマンド ToolTalk の拡張シェルコマンド。これらのコマンドを使用すると、ToolTalk ファイルに対する共通ファイル操作を安全に実行できます。

索引

A

alt.soft-sys.tooltalk, 230
API オブジェクトを指すポインタ, 156
API ファイルをプログラムへ組み込む, 75
arg_add 呼びだし, 219

B

barg_add 呼びだし, 219

C

C2 保証, 227
CASE 連携メッセージセット, 23
CEPATH, 50
close 関数, 203
contextdcl, 119, 123
CORBA 準拠のシステム, 141
cpp コマンド, 59
cp コマンド, 35

D

des, 227
DISPLAY, 50, 203

E

edit_demo, 36, 150

/etc/inet/inetd.conf, 201

F

file, 206

H

hostname_map ファイル, 56

I

iarg_add 呼びだし, 219

K

kill コマンド, 62

L

LD_LIBRARY_PATH, 202
lib, 200
libtt, 31, 204, 215
libtt.a, 200
libtt.so, 200, 202

M

makefile の変更, 171
message_status_string, 215
MIT X, 202
mv コマンド, 35, 71

O

objid
取得, 143
新規検索, 145
新規取得, 145
比較, 146
objid の取得, 143
objid の比較, 146
OMG 準拠のシステム, 27
\$OPENWINHOME/lib/openwin-sys, 201
otype, 仕様への割り当て, 143
otype シグニチャ, 121
otype のアドレス指定, 94
otype の割り当て, 143
otype ファイル, 120
作成, 122
シグニチャ情報, 122
ヘッダー情報, 122
otype ファイルのインストール, 59
otype ファイルの作成, 122

P

partition_map ファイル, 57
procid, 76, 206
デフォルトの設定, 77
デフォルトを閉じる, 80
ps コマンド, 62
ptype, 206
ptype のインストール, 172
ptype ファイル
property 情報, 118
ToolTalk で登録解除する, 127
ToolTalk で登録する, 125
作成, 117
シグニチャ情報, 119
登録, 116
ptype ファイルのインストール, 59
ptype ファイルの作成, 117, 171

ptype を宣言解除する, 127
ptype を登録, 116

R

rm コマンド, 35, 72
rpc.ttdbserverd, 31, 142, 200

S

share/include/desktop, 200
SIGUSR1 シグナル, 34
SIGUSR2 シグナル, 34
_SUN_TT_ARG_TRACE_WIDTH, 50
_SUN_TT_FILE, 50
_SUN_TT_HOSTNAME_MAP, 50, 56
_SUN_TT_PARTITION_MAP, 50, 57
_SUN_TT_SESSION, 34, 50, 80, 202, 203, 205
_SUN_TT_TOKEN, 50, 203, 217
_SUN_TTSESSION_CMD, 50

T

TMPDIR 環境変数, 51
ToolTalk API スタック上のマークを取得する, 132
ToolTalk エラー状態の確認, 160
ToolTalk エラー状態の検索, 159
ToolTalk オブジェクト, 142, 229
ToolTalk 型 コンパイラ tt_type_comp, 121
ToolTalk 型情報の検査, 61
ToolTalk 型データベースの移動, 168
ToolTalk 型データを変換, 48
ToolTalk 技術の動作, 172
ToolTalk コードの追加, 173
Xedit ファイル, 173
Xfontsel ファイル, 180
ToolTalk サービス, 19
ToolTalk サービスが提供する情報, 153
ToolTalk サービスに提供される情報, 153
ToolTalk サービスの更新, 62
ToolTalk サービスの構成要素, 31
ToolTalk サービスを使用するためのアプリケーションの変更, 29
ToolTalk セッションの開始, 32

ToolTalk データベース, 73
 ToolTalk データベースサーバー
 以前のバージョンに戻る, 55
 新規に実行, 55
 ToolTalk データベースサーバーの以前のバージョンに戻る, 55
 ToolTalk データベースサーバーのインストー
 ル, 53
 Solaris 2.6 オペレーティング環境およびその
 互換バージョン CD-ROM から, 55
 リモートマシンからの, 54
 ToolTalk データベースサーバーのリダイレク
 ト, 55
 ToolTalk データベースの検査, 35
 ToolTalk データベースの修復, 35
 ToolTalk データベースへの仕様の書き込み, 144
 ToolTalk データベースを実行していないマシン
 から ToolTalk データにアクセスする, 55
 ToolTalk データを読み取り専用ファイルシステ
 ムのパーティションから読み取る, 56
 ToolTalk データを読み取り専用ファイルシステ
 ムのパーティションに書き込む, 56
 ToolTalk の拡張シェルコマンド, 150
 ttcp, 71, 150
 ttmv, 71, 150
 ttrm, 71, 150
 ttrmdir, 72, 150
 tttar, 72, 150
 ToolTalk の拡張シェルコマンドを変更する, 72
 ToolTalk の型を変換するためのスクリプト, 165
 ToolTalk の機能, 156
 ToolTalk の重要な特徴, 26
 ToolTalk ファイルとデータベースの管理, 35
 ToolTalk メッセージ, 25
 ToolTalk メッセージセット
 デスクトップ, 20
 文書メディア交換, 22
 ToolTalk メッセージの受信, 26
 ToolTalk メッセージの送信, 25
 TT_BOTH, 43
 tt_c.h, 200
 TT_CALLBACK_CONTINUE, 209
 TT_CALLBACK_PROCESSED, 209
 tt_close, 80, 111, 203
 tt_default_file, 229
 tt_default_session_set, 77, 78, 206
 tt_fd, 77, 78
 TT_FILE, 41
 TT_FILE_IN_SESSION, 41
 TT_FILE_IN_SESSION と TT_SESSION の論理
 和, 87
 tt_file_join, 113
 tt_file_objects_query, 146, 157
 tt_file_quit, 113
 TT_HANDLED, 130
 tt_int_error, 162
 tt_is_err, 160, 162
 tt_message_accept, 131, 217
 tt_message_callback_add, 102, 131, 212
 tt_message_create, 98
 tt_message_destroy, 102, 104, 139
 tt_message_fail, 138
 tt_message_file, 88, 229
 tt_message_file_set, 89
 tt_message_file 属性, 213
 tt_message_object, 104, 145
 tt_message_receive, 129, 134, 209, 214
 tt_message_reject, 137, 214, 217
 tt_message_send, 148
 tt_message_send_on_exit, 163
 tt_message_set, 98
 tt_message_status_set, 138
 tt_message_status_string_set, 138
 tt_message_user, 79
 tt_message_user_set, 131
 tt_message_user 呼び出し, 210
 tt_objid_equal, 146
 tt_onotice_create, 101
 tt_open, 77, 78, 201, 217
 tt_orequest_create, 101
 tt_pattern_add, 109
 tt_pattern_callback_add, 109, 131
 tt_pattern_create, 109
 tt_pattern_destroy, 110
 tt_pattern_register, 110
 tt_pattern_set, 109
 tt_pattern_unregister, 110, 126, 214
 tt_pnotice_create, 100
 tt_pointer_error, 161
 tt_prequest_create, 100
 tt_ptype_declare, 125
 tt_ptype_undeclare, 126, 127
 TT_SESSION, 40
 tt_session_join, 111
 tt_session_quit, 112
 tt_spec_bprop, 146

- tt_spec_create, 143
- tt_spec_destroy, 149
- tt_spec_file, 146
- tt_spec_move, 148
- tt_spec_prop, 146
- tt_spec_prop_add, 144
- tt_spec_prop_set, 144
- tt_spec_type, 146
- tt_spec_type_set, 143
- tt_spec_write, 144
- Tt_status, 80
- tt_status_message, 160
- tt_type_comp, 59, 116, 121, 200
- TT_WRN_STALE_OBJID, 104
- TT_WRN_START_MESSAGE, 131, 217
- ttce2xdr, 165, 200
- ttcp, 71, 150, 200
- ttdbck, 35, 73, 200
- ttmv, 71, 150, 200
- ttmv コマンド, 229
- TTPATH, 50, 201
- ttrm, 71, 150, 200
- ttrmdir, 72, 150, 200
- ttsample1, 35
- ttsession, 31, 47, 200
- ttsession が応答するシグナル, 34
- ttsession セッションパラメタ, 32
- ttsession の呼び出し方法, 32
- ttsnoop の -t オプション, 63
- ttsnoop の使用によるデバッグ, 63
- ttsnoop ユーティリティ, 208
- tttar, 72, 150, 200
- tttk.h, 200
- t オプション, 207

U

- unix, 226
- /usr/dt/lib, 202
- /usr/openwin/bin, 200
- USR1 シグナル, 207

V

- v オプション, 49

X

- xauth, 226
- XDR 形式のファイル, 32
- Xedit, 169
- Xfontsel, 169
- XtAppAddInput 呼びだし, 209
- X Window System のもとでセッションを確立する, 35

あ

- アーキテクチャ, 31
- 新しい objid の検索, 104, 145
- 新しい objid の取得, 145
- 新しい ToolTalk データベースサーバーの実行, 55
- アドレス指定, otype, 94
- アドレス属性, 86, 98
- アドレス方式, 27
- アプリケーション型のインストール, 59
- アプリケーションの ToolTalk メッセージの使用法, 25
- アプリケーションの変更, 170
 - Xedit, 170
 - xfontsel, 171
- アプリケーションプログラミングインタフェース (API), 29
- アルゴリズム
 - オブジェクト指向メッセージ配信, 92
 - プロセス指向メッセージ配信, 90

い

- イベントループ, 214

え

- エラー状態, 159
 - 確認, 160
 - 検索, 159
- エラー処理関数, 159
- エラー値, 160
- エラーの伝播, 163
- エラーマクロ, 160

お

- 応答, 容易な認識方法と処理方法, 131
- 応答の容易な処理方法, 131
- 応答の容易な認識方法, 131
- オブジェクト
 - ToolTalk, 142
 - ファイル間の移動, 148
 - ファイルシステム間のオブジェクトの移動, 148
- オブジェクト型 (otype), 120
- オブジェクト指向, 27
- オブジェクト指向アルゴリズム, 92
- オブジェクト指向メッセージ, 141
 - 作成, 101
- オブジェクト指向メッセージの作成, 101
- オブジェクト仕様 (スペック), 142
- オブジェクト仕様の削除, 149
- オブジェクト情報, 管理, 150
- オブジェクトデータ, 141
- オブジェクトデータが入ったファイルの管理, 149
- オブジェクトとファイル情報のコピー, 150
- オブジェクト内容, 142

か

- 返された整数の状態, 162
- 返されたポインタの状態, 161
- 格納
 - hostname_map ファイル, 56
 - partition_map ファイル, 57
- 型コンパイラ, 48
- 型情報
 - インストール, 124
 - 検査, 61
 - 削除, 61
 - すべての型情報を検査する, 61
 - 併合, 124
- 型情報のインストール, 124
- 型情報の削除, 61
- 型情報の併合, 124
- 型情報を再読み取りする, 62
- 型ファイルを再読み取り, 34
- 環境変数, 49
 - CEPATH, 50
 - DISPLAY, 50
 - _SUN_TT_ARG_TRACE_WIDTH, 50

環境変数 (続き)

- _SUN_TT_FILE, 50
- _SUN_TT_HOSTNAME_MAP, 50, 56
- _SUN_TT_PARTITION_MAP, 50
- _SUN_TT_PARTITION_MAP, 57
- _SUN_TT_SESSION, 34, 50, 80
- _SUN_TT_TOKEN, 50
- _SUN_TTSESSION_CMD, 50
- TMPDIR, 51
- TTPATH, 50
 - メッセージコンテキストから作成された, 53
- 関数, tt_message_user, 79

き

- 記憶領域の解放, 104, 155
- 記憶領域の情報のマーク付け, 154
- 記憶領域の割り当て, 155
- 既存の ptype の確認, 125
- 既存の仕様の更新, 145
- 既存の仕様プロパティの更新, 144
- 既存のファイル内のデータを識別する, 142

く

- クラスの属性, 98

け

- 計算ループ, 214
- 現在動作中の ttsession にコンパイル済み ToolTalk 型ファイルを併合する, 124

こ

- 構造化データのシリアル化, 89
- 構造化データのシリアル化を解除する, 89
- コールバックルーチン, 156
 - 起動, 134
- コールバックルーチンの起動, 134
- コールバックルーチンのメッセージパターンへの追加, 109
- コンテキストで照合を制限する, 220
- コンテキストスロット, 53

コンテキスト引数, 156
コンパイラ tt_type_comp, 116

さ

サーバー認証レベル, 32
サイレント操作, 33
削除された参照, 149

し

シェルコマンド
ToolTalk の拡張, 35, 48, 71, 150
ttmv, 71
変更, 72
標準
cp, 35
mv, 35, 71
rm, 35, 72
シェルスクリプト, 51
ttrsh, 51
シグニチャ, 116
otype, 121
ptype, 116
システムデータベースの変換, 166
実行時スタック, 153
実バス, 229
自動的にメッセージパターンを削除する, 111
自動的にメッセージパターンを登録解除する, 111
受信側, 25
受信メッセージの判別, 27
仕様
ToolTalk データベースへの書き込み, 144
otype の割り当て, 143
オブジェクトの移動, 148
オブジェクトの照会, 146
管理, 145
既存のプロパティの更新, 144
更新, 145
削除, 149
作成, 143
仕様プロパティの決定, 144
情報の検査, 145
プロパティの格納, 144
プロパティへの値の追加, 144

仕様情報の検査, 145
使用する ToolTalk のシナリオ, 20
状態の変化を知らせるメッセージ, 85
仕様の管理, 145
仕様の削除, 149
仕様プロパティの格納, 144
仕様プロパティの決定, 144
仕様プロパティへの値の追加, 144
情報の記憶領域管理のための呼び出し, 154
情報を検査する, 61
仕様を作成する, 143
初期セッション, 76
処置属性, 45
処理対象とするファイル
結合, 113
終了, 113
処理対象とするファイルの結合, 113
処理対象とするファイルの終了, 113
新規の ToolTalk グループ, 230

す

スペック, 142

せ

静的な方法, 37
静的パターン
コールバックの付加, 135
コールバックの追加, 104
静的パターンに付加されたコールバック, 135
静的パターンへのコールバックの追加, 104
静的メッセージパターン, 115
セッション識別子 (sessid), 28
セッションだけの配信範囲指定, 40
セッション中のファイル配信範囲, 89
セッション内のファイルへの配信範囲指定, 41
セッションについての ToolTalk での概念, 28
セッション配信範囲, 88
セッションを手入力で開始, 32
セッションを配信範囲とするメッセージフロー
208
接続の異常終了の原因, 53

そ

- 操作属性, 99
- 送信側, 25
- 送信側に要求の異常終了を知らせる, 138
- 属性
 - アドレス, 86, 98
 - クラス, 98
 - 設定, 98
 - 操作, 99
 - 配信範囲, 86, 99
 - 引数, 99
- 属性の設定, 98

つ

- 通常の戻り値を伴う関数, 160
- 通常の戻り値を伴わない関数, 160
- 通信プロセス, 31
- 通知, 83
- 通知の送信, 83
- ツールが予期せず終了したことをプロセスに知らせる, 163

て

- 定義されたコンテキスト, 44
- データベース
 - ToolTalk の管理, 35
 - ToolTalk へのアクセス, 73
 - 検査および修復のユーティリティ `ttdbck`, 35
 - チェックおよびリカバリツール, 48
 - データベースの表示、検査、および修復, 73
 - レコード, 31
- データベースから型を読み取る, 33
- データベースサーバー
 - ToolTalkのインストール, 53
 - ファイルシステムのパーティションのリダイレクト, 56
 - プロセス, 31
 - ホストマシンのリダイレクト, 56
 - リダイレクト, 55
- データベースサーバーのリダイレクトファイル, 201
- データベースユーティリティの `ttdbc`, 73
- デスクトップサービス, 起動した, 207
- デスクトップサービスメッセージセット, 20

デフォルトセッション

- 結合, 111
- 終了, 112
- デフォルトセッションの結合, 111
- デフォルトセッションの終了, 112
- デモプログラム, `edit_demo`, 150
- デモンストレーション, 169
- デモンストレーションプログラム
 - `edit_demo`, 36
 - `ttsample1`, 35

と

- 同一セッションにプロセスを追加する, 204
- 同一プロセスでのメッセージの送受信, 79
- 動的な方法, 37
- 動的メッセージの作成, 109
- 動的メッセージパターン, 107
- 登録
 - ToolTalk サービスへの, 76
 - 指定セッションへの, 77
 - 初期セッションへの, 76
- トレースモード, 33, 207, 218, 228
- トレースモードの切り替え, 34

ね

- ネットワーク環境, 80
- ネットワークデータベースの変換, 167

は

- バージョン番号, 33
- バージョン文字列, 49
- 配信範囲指定したパターンへのファイルの追加, 43
- 配信範囲属性, 86
 - セッション, 88
 - セッション中のファイル, 89
 - ファイル, 87
- 配信範囲の型, 40
- 配信範囲の属性, 99
- 配信範囲を処理の対象として登録しているすべてのクライアントに指定する, 89
- パス名のホスト名修飾, 229

パターンコールバック, 134, 156
パターン属性をメッセージ属性と比較, 39
パターン引数, 44
バックグラウンドジョブ, 34
バッチセッション, 34
ハンドラに対するコールバック, 135
汎用メッセージの作成, 98

ひ

引数の属性, 99

ふ

ファイル

hostname_map, 56
partition_map, 57
ToolTalk の管理, 35
XDR 形式, 32
オブジェクト型, 120
オブジェクトデータが入ったファイルの管理, 149
～についての ToolTalk 概念, 28
ファイル間のオブジェクトの移動, 148
ファイルシステム間のオブジェクトの移動, 148
ファイルシステムのパーティションのリダイレクト, 56
ファイル照会関数, 157
ファイル情報, 管理, 150
ファイルだけの配信範囲指定, 41
ファイルの配信範囲指定機能, 28
ファイルの配信範囲の制限, 40
ファイル配信範囲, 87
ファイル配信範囲のパターンの登録, 205
ファイルまたはセッションへの配信範囲指定, 43
ファイル名マッピング機能, 25
ファイルを使用する配信範囲, 87
ファイルを配信範囲とするメッセージフロー, 209
フィルタルーチン, 156
負荷分散, 214
複数セッション, セッションのセッション ID を格納する, 113
複数セッション間の切り替え, 206
複数セッションの結合, 112

複数の ptype, 127
複数のセッション識別子, 32
複数のセッションへの登録, 78
複数のプロセス, 213
プロジェクト DOE, 199
プロセス
通信, 31
データベースサーバー, 31
プロセス型 (ptype), 115
プロセス型エラー, 62
プロセス型を宣言する, 125
プロセス識別子 (procid), 76
プロセス指向アルゴリズム, 90
プロセス指向メッセージ, 27
作成, 100
プロセス指向メッセージの作成, 100
プロセスツリーセッションを開始する, 34
プロセスを初期化する, 76
文書メディア交換メッセージセット, 22

へ

ヘッダーファイル, 48

ほ

ポイントツーポイント (PTP) メッセージ渡し機能, 99
ポイントツーポイントのメッセージ, 135
ホストマシンのリダイレクト, 56

ま

待ち行列に入ったセッションを配信範囲とするメッセージ, 227
待ち行列に入ったファイルを配信範囲とするメッセージ, 227
マッピング機能, ファイル名, 25
マルチスレッド環境, 25

め

メッセージ
アドレス方式, 27

- メッセージ (続き)
 - オブジェクト指向, 27
 - 監視, 26
 - 完了, 95
 - 検査, 132
 - 削除, 104, 139
 - 作成, 95
 - 受信, 26
 - 処理, 26
 - 送信, 25, 104
 - ～の受信側の決定, 26
 - 汎用メッセージの作成, 98
 - プロセス指向, 27
 - 容易な識別方法と処理方法, 131
- メッセージコールバック, 134, 156
- メッセージコールバックの追加, 101
- メッセージ受信の設定, 78
- メッセージ属性をパターン属性と比較, 39
- メッセージの検査, 132
- メッセージの検索, 129
- メッセージの削除, 104, 139
- メッセージの作成, 95
- メッセージの送信, 104
 - アプリケーションの変更, 95
- メッセージの属性, 85
- メッセージの容易な識別方法, 131
- メッセージの容易な処理方法, 131
- メッセージ配信
 - オブジェクト指向アルゴリズム, 92
 - プロセス指向アルゴリズム, 90
- メッセージパターン, 26, 37
 - 更新, 111
 - 最小限の指定, 38
 - 自動的に登録解除して削除する, 111
 - 静的, 115
 - 登録解除, 110
 - ～へのコールバックの追加, 109
- メッセージパターンコールバックの追加, 109
- メッセージパターンの更新, 111
- メッセージパターンの削除
 - メッセージパターン
 - 削除, 110
- メッセージパターンの作成, 作成, 109
- メッセージパターンの属性, 37
- メッセージパターンの登録解除, 110
- メッセージプロトコル, 29
- メッセージを送信するためのアプリケーションの変更, 95

- も
- 文字端末に確立したセッション, 34
- 戻り値
 - 通常, 160
 - 通常の～を持たない, 160
- 戻り値の状態, 160

- ゆ
- ユーザーデータセル, 212
- ユーザーデータベースの変換, 165

- よ
- 要求, 83
 - 異常終了, 138
 - 拒否, 137
 - 処理, 136, 138
 - 送信側に～の異常終了を知らせる, 138
 - ～への応答, 136
- 要求の異常終了, 138
- 要求の拒否, 137
- 要求の処理, 136, 138
- 要求の送信, 84
- 要求への応答, 136
- 読み取り
 - hostname_map ファイル, 56
 - partition_map ファイル, 57
- 読み取り専用ファイルからオブジェクトを作成する, 142
- 読み取り専用ファイルシステム, 142

- り
- リモートホスト上のプログラムの起動, 51

- る
- ルーチン
 - コールバック, 156
 - フィルタ, 156

れ

レコードデータベース, 31

連携機能の追加, 170

わ

ワイルドカードでパターンを指定する, 218

割り当てスタック, 153