共通デスクトップ環境 ToolTalk メッセージの概要

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

Part No: 816–4040–11
2002 年 12 月

原典: *Common Desktop Environment: ToolTalk Messaging Overview*

Part No: 816-0379-10

Revision A

020926@4660

目次

はじめに　**7**

1　ToolTalk サービスの紹介　**13**
ToolTalk サービスが解決できる業務上の問題　14
ツール互換性　14
制御統合　14
ネットワーク透過イベント　15
自動ツール起動機能　15
分散オブジェクト・システム　15
固定表示オブジェクト　15
ToolTalk サービスが業務上の問題を解決する方法を示すシナリオ　16
ToolTalk デスクトップ・サービス・メッセージ・セットの使用　16
ToolTalk ドキュメント・メディア交換メッセージ・セットの使用　18
アプリケーションの ToolTalk メッセージの使用方法　20
ToolTalk メッセージの送信　20
メッセージ・パターン　21
ToolTalk メッセージの受信　21
ToolTalk メッセージの配信　22
プロセス指向メッセージ方式　22
オブジェクト指向メッセージ方式　22
メッセージ配信の判別　23
ToolTalk サービスを使用するためのアプリケーションの変更　24

2　ToolTalk メッセージの使用方法　**25**
ToolTalk 機能のアプリケーションへの通知　25

メッセージ・ツールキットの使用と ToolTalk コマンドの組み込み　25
ToolTalk ライブラリの使用　26
コーディングを開始する前に　26
イベントとオペレーションとの違い　27
シナリオの開発　28
通信用アプリケーションの準備　29
ptype ファイルの作成　30
各 ToolTalk 対応アプリケーションが実行する必要があるタスク　31
ToolTalk 対応エディタ・アプリケーションが実行する必要があるタスク　33
ToolTalk 対応エディタ・アプリケーションが実行できるオプションのタスク 34

3　TTSnoop の使用によるメッセージおよびパターンのデバッグ　37
TTSnoop について　37
TTSnoop はどこにあるか　37
TTSnoop の起動　38
メッセージの作成および送信　39
パターンの作成および登録　39
メッセージ・コンポーネントの表示　39
作成済みメッセージの送信　40
メッセージの受信　40
メッセージ受信の停止　40

4　ToolTalk トレースの使い方　41
ToolTalk トレースへのアクセス　41
トレースのコントロール　42
libtt トレースのコントロール　42
クライアント側トレースのコントロール　42
ToolTalk セッション内のメッセージ・トラフィックのトレース　43
サーバによる ToolTalk 呼び出しとメッセージのトレース　45
トレースされた関数の形式　46
例　48
ToolTalk トレースのための設定　50

A　メッセージ・ツールキット　53
ToolTalk メッセージ・ツールキットの一般的な説明　53
ToolTalk の規約　55

アプリケーション記述時のメッセージ・ツールキットの使用　56
ToolTalk メッセージ・ツールキット　56
ttdt_close　57
ttdt_file_event　57
ttdt_file_join　58
ttdt_file_notice　60
ttdt_file_quit　61
ttdt_file_request　62
ttdt_Get_Modified　63
ttdt_message_accept　64
ttdt_open　67
ttdt_Revert　67
ttdt_Save　69
ttdt_sender_imprint_on　70
ttdt_session_join　71
ttdt_session_quit　74
ttdt_subcontract_manage　75
ttmedia_Deposit　76
ttmedia_load　77
ttmedia_load_reply　79
ttmedia_ptype_declare　80
tttk_block_while　82
tttk_message_abandon　82
tttk_message_create　83
tttk_message_destroy　84
tttk_message_fail　85
tttk_message_receive　85
tttk_message_reject　85
tttk_op_string　86
tttk_Xt_input_handler　86

B　CoEd デモンストレーション・プログラム　89
CoEd ptype ファイル　89
CoEd.C ファイル　90
Coeditor.C ファイル　93

C 新規の **ToolTalk** 関数 **113**
tt_error 113
tt_file_netfile 114
tt_host_file_netfile 114
tt_host_netfile_file 115
tt_message_print 117
tt_netfile_file 117
tt_pattern_print 118

D 例 **119**
Ttdt_contract_cb の例 119
Ttdt_file_cb の例 121
Ttmedia_load_msg_cb の例 122
Ttmedia_load_pat_cb の例 123
Ttmedia_ptype_declare 関数の ptype シグニチャーの例 125
Xt 入力処理関数の例 126

索引 **127**

はじめに

このマニュアルでは、共通デスクトップ環境のコンポーネント、コマンド、および ToolTalk™ サービスのエラー・メッセージについて記述しています。

注 – 一般的な ToolTalk サービスの機能についてのより詳細な情報は、このマニュアルの対象外です。つまり、ToolTalk API、コマンド、共通デスクトップ環境対応の ToolTalk サービスなど、このリリースには特に関連のない ToolTalk の機能については説明していません。これらの情報については、ToolTalk のマニュアルページおよび『*ToolTalk* ユーザーズガイド』を参照してください。

対象読者

このマニュアルは、ToolTalk サービスを使用して共通デスクトップ環境で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保守する開発者を対象としています。このマニュアルでは、読者が ToolTalk サービスとその機能、UNIX オペレーティング・システムのコマンド、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を持ってい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

ToolTalk サービスの動作、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、アプリケーションが ToolTalk サービスを使用する方法、およびアプリケーションと ToolTalk のコンポーネントについて説明します。

第 2 章

アプリケーションを作成するのに必要となる共通デスクトップ環境下での ToolTalk サービスの使用方法を説明します。他の ToolTalk 対応の共通デスクトップ環境準拠のアプリケーションと連携するためにアプリケーションに組み込む必要のある ToolTalk ツールキット・メッセージの種類についても説明します。

第 3 章

カスタム構築の ToolTalk メッセージを作成および送信する方法と、ToolTalk メッセージの一部またはすべてを選択して監視する方法についても説明します。

第 4 章

ToolTalk パターンを ttsession にある各メッセージと一致させ配信する方法について説明します。

付録 A

メッセージ・ツールキットの一部であるアプリケーション・プログラム・インタフェース (API 関数) について説明します。

付録 B

ToolTalk デモ・プログラム CoEd の ptype、ヘッダおよび `.c` ファイルの ToolTalk に関連のある部分を示します。

付録 C

ローカル・パスと正規パスの間にファイル名をマップする ToolTalk 関数について説明します。

関連マニュアル

このマニュアルには、ToolTalk とその機能についての詳細な情報は載っていません。ToolTalk プロダクトの基本マニュアル (『*ToolTalk* ユーザーズガイド』と ToolTalk のマニュアルページ) の他に、次の ToolTalk の関連マニュアルにも、このマニュアルには載っていない ToolTalk の機能についての詳細な情報が載っています。

- 『*The ToolTalk Service - An Inter-Operability Solution*』(SunSoft Press/PTR Prentice Hall, ISBN 013-088717-X)

このマニュアルでは、ToolTalk とその機能について詳細に説明しています。また、ToolTalk が移植されたすべてのプラットフォームで使用可能です。一般の書店または PTR Prentice Hall から直接お求めになれます。

このマニュアルは日本語版が出版されています。

『*The ToolTalk* サービス　リファレンスマニュアル』(サンソフト監修　データリンク訳　アスキー出版局)

- 『*ToolTalk and Open Protocols*』Astrid M. Julienne、Brian Holtz 共著 (SunSoft Press/PTR Prentice Hall, ISBN 013-031055-7)

  このマニュアルは、他のアプリケーションと通信するためにメッセージ・サービスを使用するアプリケーションのためにオープン・プロトコルを作成および開発する方法について説明しています。このマニュアルで説明している一般原理は、ツールを簡単に交換するためにユーザが必要とする柔軟性をアプリケーションに提供します。一般の書店または PTR Prentice Hall から直接お求めになれます。

- 『*ToolTalk Message Sets*』

  - 『*ToolTalk Desktop Services Message Sets*』

    これらの規格は、POSIX または X11 環境にあるツールに適用されます。これらの環境の標準メッセージの他に、デスクトップの規格は ToolTalk 内部クライアント規格のすべてに適用されるデータ型とエラー・コードを定義します。

  - 『*ToolTalk Document and Media Exchange Message Set*』

    ツールを任意のメディアのコンテナ、またはそのようなコンテナから駆動されるメディア・プレイヤー/エディタにします。

  - 『*CASE Inter-Operability Message Set*』

    Sun、DEC、および SGI による、CASE セットアップのための抽象的なフレームワーク・ニュートラル・メッセージ・インタフェースを定義するオープン規格です。HP の CASE Communique にマージされ、HP の SoftBench Broadcast Message Server フレームワーク用のメッセージ・インタフェースを定義し、ANSI X3H6 に共同ドラフトとして提出されたものです。 X3H6 標準のドラフトに関する情報については、/pub/X3H6 の *ftp.netcom.com* から取り出すことができます。または下記のところに連絡をとることもできます。

    X3 Secretariat
    Computer and Business Equipment Manufactures Assoc
    1250 Eye St NW
    Washington DC 20005-3922
    Telephone: (202) 737-8888 (press ‘1’ twice)
    Fax: (202) 638-4922 or (202) 628-2829

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 *filename* 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。

第 1 章

ToolTalk サービスの紹介

独自に開発されたアプリケーションが同時に動作することを要求するコンピュータ・ユーザが増えてきているため、相互運用はソフトウェア開発者にとって重要なテーマになってきています。お互いの機能を共同で使用することにより、相互運用アプリケーションは、単一のアプリケーションが提供するには難しい機能をユーザに提供します。ToolTalk サービスは、個人やワーク・グループで利用される相互運用アプリケーションの開発を簡単に行えるように設計されています。

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なくても通信できます。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、受信先を決定してから、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。

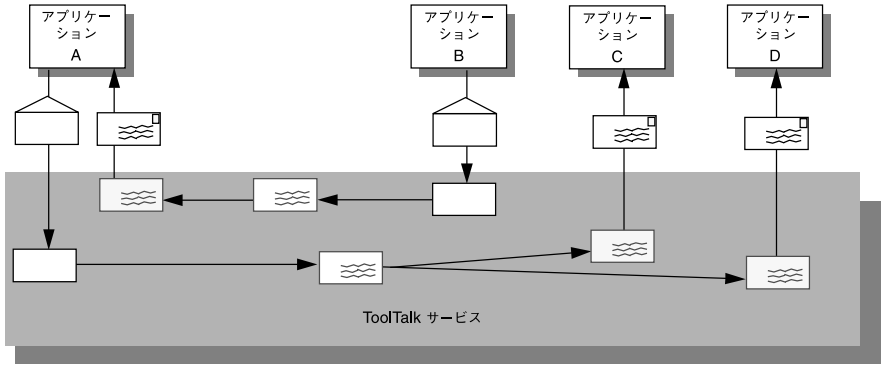

図 **1–1** ToolTalk サービスを使ったアプリケーション

ToolTalk サービスが解決できる業務上の問題

この節では、ToolTalk サービスが解決する相互運用に関する問題について説明します。ToolTalk サービスは、アプリケーションが次のようなことを必要とする場合に使用するのに適切な技術です。

- ツール互換性
- 制御統合
- ごく一般的なサーバ (たとえば X サーバ) では所有しておらず、予想可能なリスナの集合がないネットワーク透過イベント
- 自動ツール起動機能
- 広範囲で使用可能な分散オブジェクトシステム
- 固定表示オブジェクト

もちろん、相互運用の問題に対して ToolTalk サービスを使用するのは適切でない場合もあります。しかし、アプリケーションが両方の問題 (つまり、ToolTalk サービスが解決するようになっている相互運用に関する問題と ToolTalk サービスが解決するのではない問題) の解決を必要としている場合は、ToolTalk サービスを他の技術と組み合わせて使用できます。

ツール互換性

プラグ・アンド・プレイ機能を必要とする場合は、ToolTalk サービスを使用してください。プラグ・アンド・プレイという語は、ツールを同じプロトコルを提供する他のツールと交換可能であることを意味します。つまり、ToolTalk によって与えられるプロトコルをまねするツールをコンピューティング環境に配置 (プラグ) し、プロトコルが示す関数を実行 (プレイ) できます。ツールを変更することなく、お互いに特定の組み込み知識を持っていなくてもツールを併用できます。

制御統合

アプリケーションが制御統合を要求する場合、ToolTalk サービスを使用してください。制御統合という語は、ユーザの直接介入がなくても共通の目的に向かって一緒に動作するツールのグループを示します。ToolTalk サービスを使うと、簡単で柔軟性のある機能により特定のツール・インスタンスか不特定のサービス・プロバイダに対して任意の要求を発行する制御統合が可能になります。

ネットワーク透過イベント

アプリケーションがネットワーク透過イベントの生成や受信を必要とする場合は、ToolTalk サービスを使用してください。従来のイベント機構 (シグナルやウィンドウ・システム・イベントなど) を使用するには、特別な環境を必要とします。たとえば、プロセスやウィンドウ ID を認識していることなどです。ToolTalk サービスにより、イベントが参照するファイルや、イベントを適用できるネットワーク上のプロセスのグループに関してイベントは自然に記述されます。ToolTalk サービスは、ネットワーク上のいたる所にある配信対象プロセスにイベント (通知と呼ぶ) を配信します。

自動ツール起動機能

アプリケーションがネットワーク透過な自動起動機能を必要とする場合は、ToolTalk サービスを使用してください。ToolTalk サービスは、メッセージがネットワークから送信されると、ツールを起動させるメッセージを記述させます。ToolTalk 自動起動機能の使用は簡単で、従来の `inetd(1)` 機能ほどホストに固有のものではありません。

分散オブジェクト・システム

さまざまなプラットフォームにまたがって使用可能である分散オブジェクト・システムでアプリケーションを作成する必要がある場合は、ToolTalk を使用してください。ToolTalk のオブジェクト・システムは、一般的な UNIX プラットフォームにあるアプリケーションで使用できます。次のアプリケーションでも使用できます。

- シングル・スレッドまたはマルチ・スレッドである
- コマンド行またはグラフィカル・ユーザ・インタフェースを持っている
- 独自のイベント・ループ、またはウィンドウ・システムのツールキットのイベント・ループを使用している

注 – ToolTalk の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をしなければ CORBA 準拠のシステムに移植できません。

固定表示オブジェクト

アプリケーションが UNIX ファイル・システムでオブジェクトを目立たないように配置する必要がある場合は、ToolTalk サービスを使用してください。

ToolTalk サービスが業務上の問題を解決する方法を示すシナリオ

この節にあるシナリオは、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しています。これらのシナリオで使用するメッセージ・プロトコルは架空のものです。

ToolTalk デスクトップ・サービス・メッセージ・セットの使用

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、ユーザの介入がなくても他のアプリケーションを統合およびコントロールできます。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 (16 ページの「スマート・デスクトップ」と 17 ページの「統合ツールセット」) について説明します。

▼ スマート・デスクトップ

注 – この節のシナリオは、ユーザの要求を翻訳するアプリケーション・レベルのプログラムで ToolTalk サービスを使用する方法を示すためのものです。共通デスクトップ環境プロダクトにより ToolTalk サービスにユーザの要求を翻訳させる方法を示すためのものではありません。

グラフィカル・ユーザ・インタフェース (GUI) のフロント・エンドに対するユーザの共通した要求は、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」) ようにできることにあります。これを行うには、アプリケーション・レベルのプログラムがユーザ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・デスクトップという) には、アップル社の Macintosh ファインダ、マイクロソフト社の Windows ファイル・マネージャ、共通デスクトップ環境のファイル・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が増します。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. ダイアンが [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、ダイアンの現在のディレクトリ内のファイルを表示します。
2. ダイアンは、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てい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、ダイアンの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 をチェックし、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・マネージャに異常終了を返します。

      ---

   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. ダイアンは、ファイルを手動で編集します。

▼ 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 (複合ドキュメントなど) 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、総合的な解決法が 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から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。完成した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. ブルースはエディタを使用して複合ドキュメントを扱う作業をしています。
   ソースコード・テキストの一部を変更することにします。

2. ブルースは、ソースコード・テキストをダブルクリックします。

   a. ドキュメント・エディタは、まずソースコードが表すテキストを判別し、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別します。

   b. ドキュメント・エディタは、ファイル名をメッセージのパラメータとして使用し、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、ブルースの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 をチェックし、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、ドキュメント・エディタ・アプリケーションに異常終了を返します。

   ---

   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・ファイルが構成コントロールを受けていると判別し、ファイルをチェック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. ブルースは、ソースコード・テキストを編集します。

ToolTalk ドキュメント・メディア交換メッセージ・セットの使用

ToolTalk ドキュメント・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 ドキュメント・メディア交換メッセージ・セットの 3 つの使用方法について説明します。

- マルチメディアのオーサリング・アプリケーションへの統合
- 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
- メディア変換機能の追加による 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エンド・ユーザは、さまざまなメディアの型をそれらのドキュメント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、ドキュメントに埋め込まれます。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シナリオで説明するように処理されます。

1. ダニエルがマルチメディア・オブジェクトが入っているドキュメントを開きます。
2. ウィンドウがさまざまなメディアの種類 **(**音声、画像、グラフィックなど**)** を表す複数のアイコンでドキュメントを表示します。
3. ダニエルは、 **[**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 (プレイヤと呼ぶ) が起動され、録音済みの音声が再生されます。)
4. 録音状態を編集するために、ダニエルはアイコンを **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、ダニエルはメディア・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 ドキュメント・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、次のシナリオで示すように、カレンダ・マネージャを拡張し、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. シェルビーは、自分のカレンダ・マネージャを開き、アポイントメントを設定します。
2. シェルビーが **[**音声応答**]** ボタンをクリックすると、オーディオ・ツールが起動します。
3. シェルビーは、たとえば「レポートを持ってくる」といったようなメッセージを記録します。

   シェルビーがアポイントの覚え書きを実行すると、カレンダ・マネージャはオーディオ・ツールを起動し、シェルビーが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 ドキュメント・メディア変換メッセージ・セットは、拡張可能なオープン・エンドの変換機能をサポートできます。次のシナリオは、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. マリアがメディア型が異なる **2** つのドキュメントを開きます。
2. マリアがドキュメント **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、その部分をカットします。
3. マリアは、そのカットした部分をドキュメント **B** にペーストします。
   a. ドキュメント B は、カットしたデータの転送についてドキュメント A と交渉します。
   b. ドキュメント A が提供するデータのどの型もドキュメント B が認識しない場合、ドキュメント B はタグ付きメディア型を送信するようドキュメント A に要求します。ドキュメント B は、タグ付きメディア型を使用して、そのメディア型を理解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て、変換後のバージョンのメディア型をドキュメント B へ返します。
   d. 変換されたデータのドキュメント B へのペーストが実行されます。

アプリケーションの ToolTalk メッセージの使用方法

アプリケーションは、ToolTalk メッセージを作成、送信、および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、および送信します。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てメッセージ内の情報をチェックし、メッセージを破棄するか、オペレーション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、および配信情報のフィールドを含みます。ToolTalk メッセージを送信するため、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。

- 通知用メッセージか要求用メッセージか (つまり、受信側がメッセージに応答する必要があるかどうか)
- 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・セッションで実行されているものか、または特定のファイルを処理対象としているものか)

メッセージ配信の範囲を限定するために、送信側アプリケーションは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認識していなくてもかまわないことです。これは、メッセージを受信したいアプリケーション側が、受け取りたいメッセージの種類を明示的に示すからです。この情報は、メッセージ・パターンとして ToolTalk サービスに登録されます。

アプリケーションは、その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 サービスに指定できます。メッセージ・パターンは、メッセージと同じ方法で作成します。つまり、どちらの場合も同じ型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使用することにしたメッセージ・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ると、ToolTalk サービスは、受信側アプリケーションすべて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を希望する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どう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、要求されたオペレーションが確実に 1 回だけ実行されるようにするためです。ToolTalk サービスが要求に対するハンドラを見つけだせなかった場合は、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、ToolTalk サービスは、アプリケーションの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、その内容をチェックします。

- オペレーションが実行されたという情報がメッセージに含まれている場合、プロセスはその情報を読み取ってから、メッセージを破棄します。
- オペレーションの実行要求がメッセージに含まれている場合、プロセスはそのオペレーションを実行し、元のメッセージへの応答という形でオペレーションの結果を返します。応答が送信されると、プロセスは元のメッセージを破棄します。

ToolTalk メッセージの配信

ToolTalk サービスは、メッセージを配布する 2 つの方式を提供します。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方式

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、指定されたプロセスか特定の型のプロセスのどちらかにそのメッセージを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もあまり時間はかかりません。

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を作成するアプリケーションは、指定されたオブジェクトか特定の型のオブジェクトのどちらかにそのメッセージを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、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対して特に便利です。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 サービスを使うと、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ので、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します。配信範囲を指定することにより、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

セッションとは、同じ ToolTalk メッセージ・サーバ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 サービスとの通信を開始すると、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)、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 (「プロセス・ツリー・セッション」と呼ぶ)、または X ディスプレイ (「X セッション」と呼ぶ) によって配置されます。

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にできます。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。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は、アプリケーションは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コンテナ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にできます。また、ToolTalk サービスは、プロセスのデフォルト・セッションに関係なく、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。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は、アプリケーションはそのファイルを結合します。

また、1 つのセッション内に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、そのメッセージ・パターン内にあるファイルとセッションの両方を参照する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が使用できるのは、NFS™ ファイル・システムと UFS ファイル・システムだけです。

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オペレーションについて記述した ToolTalk メッセージの集合です。メッセージ・プロトコル仕様の内容は、メッセージの設定およびアプリケーションがメッセージを受信したときの動作です。

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) から 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、メッセージ情報をチェック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、まずプログラムに ToolTalk API のヘッダ・ファイルを組み込まなければなりません。また、次のことを実現するためにアプリケーションを変更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
- メッセージを送信および受信する
- メッセージ・パターンを登録解除し、ToolTalk セッションを終了する

第 2 章

ToolTalk メッセージの使用方法

注 – この章のコーディング例は、CoEd という名前の ToolTalk デモ・プログラムの一部です。このプログラムのヘッダ・ファイルと .c ファイルに ToolTalk 関連のコードを組み込む方法を示すソースコードのリストについては、付録 B を参照してください。

ToolTalk 機能のアプリケーションへの通知

アプリケーションは、ToolTalk サービスとメッセージ・ツールキットが提供する相互運用機能を利用する前に、ToolTalk ライブラリとツールキットがある場所を認識しておく必要があります。

メッセージ・ツールキットの使用と ToolTalk コマンドの組み込み

ToolTalk サービスを使用するために、アプリケーションは ToolTalk API から ToolTalk 関数を呼び出します。メッセージ・ツールキットは、ToolTalk サービスへの登録、メッセージ・パターンの作成、メッセージの送信、メッセージの受信、およびメッセージ情報のチェックを行うための関数を提供します。ToolTalk サービスとツールキットを使用できるようにアプリケーションを変更するには、適切なヘッダ・ファイルをアプリケーションの .h ファイルに組み込まなければなりません。

```
#include <Tt/tt_c.h>           // ToolTalk Header File
#include <Tt/tttk.h>           // Messaging Toolkit Header file
```

アプリケーションは、その .c ファイルにある新規の ToolTalk コマンドについても認識しておく必要があります。この情報はアプリケーションの .h ファイルにも格納してください。

例 2–1 は、CoEditor.h ファイルにヘッダ・ファイル情報を組み込む方法を示しています。

例 **2–1** メッセージ情報の組み込み

```
#ifndef CoEditor_h
#define CoEditor_h

#include <X11/Intrinsic.h>
#include <Tt/tt_c.h>                // ToolTalk Header
#include <Tt/tttk.h>                // Messaging Toolkit Header
```

ToolTalk ライブラリの使用

アプリケーションが ToolTalk ライブラリを使用するように、アプリケーションの makefile を変更する必要があります。このためには、次のように -ltt オプションを追加します。

```
LOCAL_LIBRARIES = -ltt $(XAWLIB) $(XMULIB) $(XTOOLLIB) $(XLIB)
```

コーディングを開始する前に

メッセージ・ツールキット機能をアプリケーションに組み込む前に、アプリケーションのツールが他のツールと一緒に動作する方法を決定する必要があります。考慮すべき基本的な内容は以下のとおりです。

1. これらのツールはどのように同時に動作するか
2. これらのツールはどのようなオペレーションを実行できるか
3. 他のツールにどのようなオペレーションを実行するように要求できるか
4. 他のツールを配信対象にできるこれらのツールはどのようなイベントを生成するか (これらのツールはどの型のメッセージを送信するか)
5. 他のツールによって生成されたイベントの中でこれらのツールの配信対象となるイベントはどれか (これらのツールはどの型のメッセージを受信するか)

これらの質問に的確に答えるには、イベントとオペレーションの間の違いと、ToolTalk サービスがお互いに関連のあるメッセージを処理する方法を理解する必要があります。

イベントとオペレーションとの違い

イベントは、何か起きたことを通知するものです。イベントは、簡単に言えばニュース速報です。送信側のプロセスは、他のプロセスがイベントが起きたことを聞くかどうかと、アクションがイベントの結果として起こるかどうかについて正式な予想を持っていません。プロセスが ToolTalk サービスを使用して、配信対象のプロセスにイベントが発生したことを知らせる場合、通知を送信します。送信側プロセスは応答を期待しないため、イベントは失敗できません。

オペレーションは、問い合わせまたはアクションです。要求側プロセスは、オペレーションが実行されるように問い合わせまたは要求を行います。要求側プロセスは結果が返されることを予想し、問い合わせまたはアクションのステータスを通知するように要求します。プロセスが ToolTalk サービスを使用して、他のツールにオペレーションを実行するように依頼する場合、要求を送信します。ToolTalk サービスは配信対象のプロセスに要求を配信し、送信側プロセスに要求のステータスを通知します。

通知の送信

アプリケーションが ToolTalk の通知を送信する場合、応答は受信せず、ツールがその通知に注意を払うかどうかについて通知を受けることもありません。通知をイベントの発生に応じて正確なレポートにすることが重要です。

たとえば、ツールがデスクトップ・サービス・メッセージの `Modified` を送信する場合、聞き手側のツールを指定の方法で反応させることができます。しかしツールは、他のツールがそのメッセージに反応するかどうか気にしたり、通知を受ける必要はありません。次のイベントをレポートするだけです。

```
THE_USER_HAS_MADE_CHANGES_TO_THIS.
```

要求の送信

アプリケーションが ToolTalk の要求を送信する場合、1 つのツールが指示されたオペレーションを実行する、または問い合わせに答え、応答メッセージを返すことを期待します。たとえば、ツールがデスクトップ・サービス・メッセージの `Get_Modified` を送信する場合、メッセージが配信され、アクションが実行されたという通知が期待できます。ToolTalk サービスでは必ず、プロセスの受信によって応答が返されるか、送信側に要求の失敗が通知されます。

次の 3 つの方法で要求を識別できます。

1. 異常終了できるツールによって要求されるオペレーションを識別する
2. 他のツール用に実行できるオペレーションを識別する
3. 他のツールに実行させるオペレーションを識別する

これらのオペレーションを識別するための良い方法は、ツールが実行するまたは実行を完了したイベントとオペレーションの順序について概説するシナリオを開発することです。

シナリオの開発

シナリオは、ツールが実行および実行を完了したイベントとオペレーションの順序について概説しています。たとえば、次のシナリオは、ToolTalk デモ・プログラム CoEd が実行および実行を完了したイベントについて概説しています。

1. ファイル・マネージャのドキュメント・アイコンをダブルクリックします。

   ファイルをエディタで開きます。エディタを実行中でない場合はシステムによって起動されます。

   ドキュメントに対して別のツールが保留中のテキストに変更を行うと、その別のツールが変更したテキストを保存するか、最後に保存されたバージョンのドキュメントに戻るか尋ねられます。
2. テキストを挿入します。
3. ドキュメントを保存します。

   別のツールがそのドキュメントに保留中の変更点を持っている場合、ドキュメントを変更するか尋ねられます。
4. エディタを終了します。

   テキストに保存されていない変更がある場合、ファイルを終了する前にその変更を保存するか、破棄するか尋ねられます。

シナリオがいったん実行されると、基本的な質問に答えることができます。

ツールを同時に動作させる方法

- ファイル・マネージャは、CoEd が編集用のドキュメントを開くように要求します。
- CoEd の各インスタンスは、その他の配信対象のインスタンスにドキュメントの状態に対して行われる変更を通知します。

これらのツールはどのようなオペレーションを実行できるか

- CoEd の各インスタンスは、インスタンス自身とその状態に関する質問 (「ステータスは ?」など) に答えることができます。
- CoEd の各インスタンスには、次のようなオペレーションを実行する機能があります。
  - アイコン化とアイコン化解除
  - 重なり順の変更
  - ドキュメントの編集
  - ドキュメントの表示
  - 終了

他のツールにどのようなオペレーションを実行するように要求できるか

- ファイル・マネージャは、CoEd が編集用のドキュメントを開くように要求する必要があります。
- CoEd のインスタンスは、CoEd の別のインスタンスに対し、開いているドキュメントの変更を保存するように要求できます。
- CoEd のインスタンスは、CoEd の別のインスタンスに対し、開いているドキュメントの最後に保存されたバージョンに戻るように要求できます。

他のツールを配信対象にできるこれらのツールはどのようなイベントを生成するか

- ドキュメントを開いています。
- ドキュメントを変更しています。
- ドキュメントは最後に保存されたバージョンの内容に戻っています。
- ドキュメントを保存しています。
- CoEd のインスタンスは終了しています。

他のツールによって生成されたイベントの中でこれらのツールの配信対象となるイベントはどれか

- ドキュメントを開いています。
- ドキュメントを変更しています。
- ドキュメントは最後に保存されたバージョンの内容に戻っています。
- ドキュメントを保存しています。
- CoEd のインスタンスは終了しています。

通信用アプリケーションの準備

ToolTalk サービスは、アプリケーション統合用の完全な関数セットを提供します。ToolTalk メッセージ・ツールキットが提供する機能を使用して、アプリケーションは ToolTalk 準拠の他のアプリケーションと「通信」できます。この節では、同じプロトコルに基づく他の ToolTalk 対応アプリケーションと通信できるように、アプリケーションに組み込む必要がある ToolTalk 関数の追加方法について説明します。

ptype ファイルの作成

ToolTalk 型機構は、ToolTalk サービス・ルート・メッセージに役立つように設計されています。ツールが ptype を宣言すると、そこにリストされているメッセージ・パターンが自動的に登録されます。ToolTalk サービスは、受信するメッセージとこれらの登録されたパターンを一致させます。これらの静的メッセージ・パターンは、ツールが ToolTalk サービスとの通信を終了するまで有効です。

ToolTalk 型データベースは、本リリースに付いているツール用の ptype をすでにインストールしてあります。次のようにして、ToolTalk 型データベースからインストールされた ptype のリストを取り出すことができます。

```
% 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。

他のツールすべて (つまり、本リリースには含まれていないツール) に対して、最初に ptype ファイルを作成してアプリケーションの ptype を定義し、ToolTalk 型コンパイラ `tt_type_comp` で、ptype をコンパイルする必要があります。ptype を定義するには、次の情報をファイルに入れる必要があります。

- プロセス型識別子 (*ptid*)
- オプションの開始文字列

  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開始します。
- シグニチャー

  プログラムが受信する `TT_PROCEDURE` アドレス指定メッセージを記述しています。監視されるメッセージは、処理されるメッセージとは別に記述します。

ptype ファイルを作成するには、どのテキスト・エディタ (`vi`、`emacs`、`dtpad` など) でも使用できます。例 2–2 は、CoEd アプリケーション用の ptype ファイルからの一部分です。

例 **2–2** CoEd ptype ファイル

```
ptype DT_CoEd {         /* Process type identifier */
    start "CoEd";       /* Start string */
    handle:             /* Receiving process */
    /*
     * Display ISO_Latin_1
     */
    session     Display(  in  ISO_Latin_1 contents) => start opnum = 1;  /* Signature */
        /* NOTE: 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.
         */

}
```

ptype ファイルを作成したら、ptype をインストールします。このためには、ToolTalk 型コンパイラを実行します。コマンド行に次のように入力してください。

```
% tt_type_comp CoEd.ptype
```

CoEd.ptype は、CoEd ptype ファイル名です。

現在のセッションにおける既存の ptype に対するテスト

ToolTalk サービスは、指定の ptype がすでに現在のセッションに登録されているかどうか、テストする単純関数を提供します。

```
// Test for existing ptype registered in current session
tt_ptype_exists(const char *ptid)
```

ptid は登録用にテストするセッションの識別子です。

コンパイルされた ptype ファイルと現在実行中の ttsession のマージ

ToolTalk サービスは、コンパイルされた ToolTalk 型ファイルを現在実行中の `ttsession` にマージする関数を提供します。

```
// Merge new compiled ptypes into currently running ttsession
tt_session_types_load(current_session, compiled_types_file)
```

上記の `current_session` は、現在のデフォルトの ToolTalk セッションです。`compiled_types_file` はコンパイルされた ToolTalk 型ファイル名です。この関数は、新しい型を追加し、同じ名前の既存の型と置換します。その他の既存の型は変更されません。

各 ToolTalk 対応アプリケーションが実行する必要があるタスク

各 ToolTalk 対応アプリケーションが実行する必要があるタスクは多数あります。たとえば次のとおりです。

- ツールキットの初期化
- ToolTalk セッションへの参加とパターンの登録
- ToolTalk サービスをそのイベント・ループへ追加

この節では、これらのタスクを実行できるようにアプリケーションに組み込む必要がある ToolTalk のコーディング例を取り上げます。

注 – この節で使用されているコーディングの一部分は、CoEd.C ファイルから引用しています。このファイルには、アプリケーションが実行しなければならない一般コマンドが入っています。このコマンドは、特定のアプリケーションに固有のものではありません。詳細なソースコードについては、付録 B を参照してください。

ツールキットの初期化

アプリケーションはツールキットを初期化し、ToolTalk の初期セッションに登録する必要があります。そのためには、最初にプロセス識別子 (procid) を取得します。次のコーディングの一部分は、procid の取得方法とツールキットの初期化方法を示します。

```
// Obtain process identifier
   int myTtFd;
// Initialize toolkit and create a ToolTalk communication endpoint
   char *myProcID = ttdt_open( &myTtFd, ToolName, "SunSoft", "%I", 1 );
```

注意 – アプリケーションは、他の呼び出しが行われる前に必ず ttdt_open を呼び出さなければなりません。そうしない場合、エラーが発生します。

ToolTalk セッションへの参加とメッセージ・パターンの登録

アプリケーションはメッセージを受信する前に、ToolTalk セッションに参加し、一致するメッセージ・パターンを登録する必要があります。

```
// Join a ToolTalk session and register patterns and default callbacks
    sessPats = ttdt_session_join( 0, 0, session_shell, this, 1 );
```

イベント・ループへの ToolTalk サービスの追加

アプリケーションは、ToolTalk サービスをそのイベント・ループに追加することも行わなければなりません。

```
// Process ToolTalk events for Xt Clients
    XtAppAddInput( myContext, myTtFd, (XtPointer)XtInputReadMask,
                   tk_Xt_input_handler, myProcID );
```

ToolTalk 対応エディタ・アプリケーションが実行する必要があるタスク

前述の 31 ページの「各 ToolTalk 対応アプリケーションが実行する必要があるタスク」で説明しているタスクの他に、ToolTalk 対応エディタ・アプリケーションは次のタスクも実行する必要があります。

- ptype の宣言
- 開始文字列メッセージの処理
- メディア・コールバックのパス
- メッセージの無視
- 要求完了時の応答

この節では、これらの追加タスクを実行できるようにエディタ・アプリケーションに組み込む必要がある ToolTalk のコーディング例を取り上げます。

注 – この節で使用しているコーディングの一部分は、`CoEditor.C` ファイルから引用しています。このファイルには、エディタ・アプリケーション用の特定コマンドが入っています。詳細なソースコードについては、付録 B を参照してください。

メディア・ロード・パターン・コールバックの記述

ToolTalk 関数を組み込むためにエディタ・アプリケーションをコーディングする前に、実行しなければならない手順が 1 つあります。この手順は、メディア・ロード・パターン・コールバック・ルーチンを記述する必要があるということです。たとえば、次のようになります。

```
Tt_message
CoEditor::loadISOLatin1_(
   Tt_message         msg,
   void               *pWidget,
   Ttttk_op           op,
   Tt_status          diagnosis,
   unsigned char      *contents,
   int                len,
   char               *file,
   char               *docname
)
```

このコールバックは、実行時にメディア・ロード関数に渡されます。

ptype の宣言

型情報は (アプリケーションのインストール時に) 1 回だけしか指定されないので、アプリケーションを起動するたびにその ptype を宣言する必要があります。

メディア・ロード・パターン・コールバックのパス

前述したメディア・ロード・パターン・コールバック・ルーチンは実行時に渡されます。コールバックは、アプリケーションがセッションに参加するときに登録されます。ツールが要求を処理することにすると、コールバック・メッセージが送信されます。ファイルが参加する、あるいはメッセージが無視される場合にも、コールバック・メッセージは送信されます。

```
// Join the session and register patterns and callbacks
sessPats = ttdt_session_join( 0, 0, session_shell, this, 1 );

// Accept responsibility to handle a request
    _contractPats = ttdt_message_accept(msg, CoEditor::_contractCB_,
                                        shell, this, 1, 1 );

// Optional task: Join a file (Can be called recursively)
    if (_filePats == 0) {_filePats = ttdt_file_join( _file, TT_SCOPE_NONE, 1,
                                                     CoEditor::_fileCB_, this );
    }

// Fail a message
tttk_message_fail( msg, TT_DESKTOP_ENODATA, 0, 1 );
```

要求完了時の応答

アプリケーションはオペレーションの要求を完了した後、送信側アプリケーションに応答しなければなりません。次のメッセージは、テキストの編集された内容を送信側に返します。

```
// Reply to media load pattern callback
// with edited contents of text
ttmedia_load_reply( _contract, (unsigned char *)contents, len, 1 );
```

ToolTalk 対応エディタ・アプリケーションが実行できるオプションのタスク

前述の 33 ページの「ToolTalk 対応エディタ・アプリケーションが実行する必要があるタスク」で説明しているタスクの他に、エディタ・アプリケーションはその他のオプションのタスクも実行できます。オプションのタスクには、他のエディタとの調整を取るためにデスクトップ・ファイル・インタフェースを使用するものなどがあります。この節では、これらのオプションのタスクを実行できるようにエディタ・アプリケーションに組み込む必要がある ToolTalk のコーディング例をいくつか取り上げています。

注 – この節で使用しているコーディングの一部分は、CoEditor.C ファイルから引用しています。このファイルには、エディタ・アプリケーション用の特定コマンドが入っています。詳細なソースコードについては、付録 B を参照してください。

変更、復元、または保存オペレーションの要求

次のコーディングの一部分は、保留中の変更があるかどうかファイルに尋ねているものです。

```
// Does the file have any changes pending?
_modifiedByOther = ttdt_Get_Modified( _contract, _file, TT_BOTH,
                                      10 * timeOutFactor );
```

次のコーディングの一部分は、ファイルを最後のバージョンの内容に復元します。

```
// Revert file to last version
status = ttdt_Revert(_contract, _file, TT_BOTH, 10 * timeOutFactor );
```

次のコーディングの一部分は、ファイルに対して保留中の変更を保存します。

```
// Save pending changes
status = ttdt_Save( _contract, _file, TT_BOTH, 10 * timeOutFactor );
```

ファイルの変更、復元、または保存時の通知

次のコーディングの一部分は、アプリケーションがファイルに対して保留中の変更を持っていることを配信対象のツールに通知します。

```
// File has been modified
ttdt_file_event( _contract, TTDT_MODIFIED, _filePats, 1 );
```

次のコーディングの一部分は、アプリケーションがファイルを最後に保存されたバージョンの内容に復元してあることを配信対象のツールに通知します。

```
// File has been reverted to last version
ttdt_file_event( _contract, TTDT_REVERTED, _filePats, 1 );
```

次のコーディングの一部分は、アプリケーションがファイルに対して保留中の変更を保存してあることを配信対象のツールに通知します。

```
// File has been saved
ttdt_file_event( _contract, TTDT_SAVED, _filePats, 1 );
```

ファイルの終了

次のコーディングの一部分は、ファイルに関する ToolTalk イベントの配信対象を登録解除し、パターンを破棄します。

```
// Unregister interest in ToolTalk events and destroy patterns
status = ttdt_file_quit( _filePats, 1 );
        _filePats = 0;
```

第 3 章

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。ToolTalk メッセージのどれか、またはすべてを選択的に監視するツールとして TTSnoop を使用することもできます。

TTSnoop について

TTSnoop は便利な対話型ツールで、これにより ToolTalk の概念と API 呼び出しをデモンストレーションの実行と同様よく知ることができます。さらに、TTSnoop はアプリケーションの開発中には貴重なデバッグ・ツールです。

TTSnoop を使用して、1 つ以上のパターンと一致するメッセージを監視できます。一致したメッセージが表示されると、エントリと一致したパターン名も表示されます。

スクロール可能なリストにメッセージおよびパターンを追加、編集、または削除できます。TTSnoop により、複数のパターンとメッセージの定義を保存したり、ファイルから読み込むことができます。また、アプリケーション (たとえば、DeskSet™ ツール) のカテゴリに特有のパターンとメッセージを定義、保存、および再び読み込んだり、メッセージとパターンをユーザ定義名に関連付けることもできます。

TTSnoop はどこにあるか

TTSnoop プログラムは、ディレクトリ `/usr/dt/bin/ttsnoop` にあります。

TTSnoop の起動

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。

```
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作成に使用する ToolTalk API 関数を表示するようにします。表 3–1 では、TTSnoop の起動時に表示されるボタンについて説明しています。

表 **3–1** TTSnoop ボタン

ボタン	説明
Start	このボタンは、メッセージの受信を起動する場合にクリックします。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。
Stop	このボタンは、メッセージの受信を停止する場合にクリックします。
Clear	このボタンは、ウィンドウを閉じる場合にクリックします。
About TTSnoop	このボタンは、TTSnoop についての一般的なヘルプを見る場合にクリックします。
Display	このボタンは、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・コンポーネントを強調表示する場合にクリックします。
Messages	このボタンは、ToolTalk メッセージの作成、格納、送信を可能にするパネルを表示する場合にクリックします。
Patterns	このボタンは、ToolTalk パターンの構成、登録を可能にするパネルを表示する場合にクリックします。
Send Messages	このボタンは、メッセージ・ディスプレイを使用して格納していたメッセージを送信する場合にクリックします。

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 `Help` キーを押します。

メッセージの作成および送信

メイン表示ウィンドウで [Messages] ボタンをクリックすると、表 3–2 に示す選択肢を含む表示パネルが表示されます。

表 **3–2** [Messages] ボタン表示ウィンドウのオプション

ボタン	説明
Add Message	このボタンは、現在のメッセージ設定を格納する場合にクリックします。メッセージが格納されてしまうと、メイン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および送信を実行できます。
Edit Contexts	このボタンは、送信メッセージ・コンテキストを追加、変更、または削除する場合にクリックします。表示される表示ウィンドウによって、メッセージと共に送信するコンテキストを編集できます。
Send Message	このボタンは、新規に作成したメッセージを送信する場合にクリックします。

パターンの作成および登録

メイン表示ウィンドウの [Patterns] ボタンをクリックすると、表示パネルが表示されます。

パターンを登録するには、[Apply] ボタンをクリックします。パターンが登録されると、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のかを監視できます。

パターン内の受信メッセージ・コンテキストを追加、変更、または削除するには、[Edit Receive Contexts] ボタンをクリックします。表示されたウィンドウにより、パターンと共に登録するコンテキストを編集できます。

メッセージ・コンポーネントの表示

メイン表示ウィンドウ上の [Display] ボタンをクリックすると、チェックボックスの表示パネルが表示されます。

チェックボックスを選択して [Apply] ボタンをクリックすると、指定された ToolTalk メッセージ・コンポーネントは、別の選択を行なったり変更を適用したりするまで表示されます。

作成済みメッセージの送信

メイン表示ウィンドウ上の [Send Message] ボタンをクリックすると、メッセージ・ディスプレイを使用して作成し格納したメッセージの 1 つを送信できます。

メッセージの受信

メイン表示ウィンドウ上の [Start] ボタンをクリックすると、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。

メッセージ受信の停止

メイン表示ウィンドウ上の [Stop] ボタンをクリックすると、TTSnoop はメッセージの受信を停止します。

第 4 章

ToolTalk トレースの使い方

ToolTalk ttsession トレースは、ToolTalk パターンを ttsession にある各メッセージと一致させ配信する方法を示します。本リリースの ToolTalk トレースは次のことを行います。

- 単一クライアントの ToolTalk との対話を表示します。この機能により、実行者は 1 つのクライアントだけをトレースできます。
- ttsession トレースを、たとえばメッセージ型、送信者、または受信者ごとといったフィルタに通します。

ToolTalk トレースへのアクセス

本リリースで新規のコマンドである `tttrace` は、主に ToolTalk トレースへのアクセスに使用します。このコマンドは、用途やコマンド行インタフェースの点で `truss` コマンドに似ています。このコマンドにより、3 種類の ToolTalk トレースをコントロールできます。`ttrace` コマンドには、サーバ・モードとクライアント・モードの 2 つの基本モードがあります。

- サーバ・モードでは、Session_Trace 要求を送信することによって、指定されたセッションのトレースを指示します。
- クライアント・モードでは、`ttrace` は環境変数を設定し、コマンド行に指定された ToolTalk クライアント・コマンドを実行します。実行されたクライアントの環境変数は、クライアント・メッセージとクライアント API 呼び出しをトレースするかどうかとその方法を `libtt` に指示します。

注 – ttrace は、旧バージョンのサーバや、libtt の旧バージョンを使用するクライアントとの下位互換性はありません。ttrace は、旧バージョンのサーバを検出して診断すると、libtt の旧バージョンを使用するクライアント上で何も通知せずに異常終了します。

トレースのコントロール

libtt トレースのコントロール

libtt トレースの動作をコントロールする方法の 1 つは、環境変数 $TT_TRACE_SCRIPT を設定することです。

注 – libtt のトレースは、変数の値が正確ではない場合や一貫性がない場合は異常終了します。

クライアント側トレースのコントロール

tt_trace_control 呼び出しは、内部フラグを設定または消去して、すべてのクライアント側トレースをコントロールします。この呼び出しを使用して、コード内の問題の領域をトレースします。この呼び出しの形式は次のとおりです。

```
int tt_trace_control(int option)
```

オプションの値として 0 を指定するとトレースをオフにし、1 を指定するとトレースをオンにします。-1 を指定すると、トレースのオンとオフを切り替えます。トレースがオンの場合は、トレースの範囲を TT_TRACE_SCRIPT 変数かトレースファイルでコントロールします。この呼び出しは、トレース・フラグの以前の設定を返します。

ToolTalk セッション内のメッセージ・トラフィックのトレース

Session_Trace 要求は、ttsession がそれ自身を処理するために登録する ToolTalk 要求です。つまり、ttsession は Session_Trace 要求のハンドラです。この要求は、どの ToolTalk クライアントでも送信できます。お勧めできる方法ではありませんが、この要求を処理するために他の ToolTalk クライアントを登録できます。

注 – この方法ではトレースは行われません。

この要求の構文は次のとおりです

```
[file] Session_Trace( in boolean on,
                      in boolean follow
                      [in attribute toPrint
                      |in state toTrace
                      |in op toTrace
                      |in handler_ptype toTrace
                      |in sender_ptype toTrace][...] );
```

Session_Trace 要求は、配信範囲指定されたセッションでのメッセージ・トレースをオンまたはオフにします。

- トレースがオンで、要求のファイル属性が設定されている場合、後続のトレース出力は属性が名前を付けたファイルに付け加えられます。
- トレースがオンで、ファイル属性が設定されていない場合、トレースは現在のトレースを続行します。

デフォルトの場合、デーモン・モードでは ttsession を実行中のホストのコンソールが出力先になります。ジョブ・コントロール・モードでは、ttsession の標準エラーが出力先になります。表 4–1 は、この要求の必須およびオプションの引き数を示しています。

表 **4–1** Session_Trace 引き数

引き数		説明
`boolean` *on*	必須	トレースをオンまたはオフにします。*toTrace* 引き数がなく *on* が true の場合は、前のトレース設定が復元されます。
`boolean` *follow*	必須	起動したクライアントのクライアント側トレースをオンにします。

表 **4–1** Session_Trace 引き数　*(続き)*

引き数		説明
`attribute` *toPrint*	オプション	トレースされた各メッセージの属性を出力します。有効な属性は次のとおりです。 ■ *none* – トレースされたメッセージの記述を 1 行だけ出力 (デフォルト) ■ *all* – トレースされたメッセージの属性全部を出力
`state` *toTrace*	オプション	メッセージをトレースするための状態です。`tt_c.h` に定義されている `Tt_states` の他に有効な状態は次のとおりです。 ■ *edge* – 初期の状態 (`TT_SENT`) と最後の状態 (`TT_HANDLED`、 `TT_FAILED`) を入力しているメッセージ ■ *deliver* – すべての状態の変更と、すべてのクライアントの配信 ■ *dispatch* – *deliver* 状態および一致するように考慮されたすべてのパターン (デフォルト)
`op` *toTrace* `sender_ptype` *toTrace* `handler_ptype` *toTrace*	オプション オプション オプション	表示されたメッセージ属性の値として *toTrace* を持っているトレース・メッセージです。 ■ 要求には、*toTrace* 引き数を任意の数だけ含めることができる ■ *toTrace* にはシェルのワイルドカード文字も指定できる ■ 指定のメッセージ属性に対して *toTrace* 引き数がない場合、トレースからメッセージを除外する属性の値も存在しない

現在のセッション・トレースの動作は、この要求が失敗しない場合にのみ変更されます。失敗した場合には、応答の `tt_message_status` は表 4–2 で説明されているエラーの 1 つに設定されます。

表 **4–2** Session_Trace 要求が返すエラー・メッセージ

エラー	説明
`TT_ERR_NO_MATCH`	要求に対するハンドラを見つけることができません。
`TT_ERR_APPFIRST + EACCES`	ttsession がトレース・ファイルを開くまたは作成するためのアクセス権を持っていません。
`TT_ERR_APPFIRST + EISDIR`	トレース・ファイルがディレクトリです。

表 **4–2** Session_Trace 要求が返すエラー・メッセージ　(続き)

エラー	説明
TT_ERR_APPFIRST + ENOSPC	ターゲット・ファイル・システムにトレース・ファイルの作成に十分なスペースがありません。
TT_ERR_APPFIRST + EEXIST	トレースがすでに別のファイルで行われています。ttsession は、応答のファイル属性を再設定して既存のトレース・ファイルの名前を付けます。別のファイルへのトレースを行うには、最初に現在のトレース・ファイルへのトレースをオフにします。

サーバによる ToolTalk 呼び出しとメッセージのトレース

ttrace 関数は、指定の ToolTalk セッションのためにサーバによりメッセージ・トラフィックをトレースするか、ToolTalk クライアント・トレースをオンにしてコマンドを実行します。セッションとコマンドの両方とも指定されない場合は、デフォルトのセッションがトレースされます。デフォルトでは、ttrace が終了するとトレースも終了します。この関数の構文は次のとおりです。

```
tttrace [-0FCa] [-o outfile ] -S session | command]
tttrace [-e script | -f scriptfile ] [-S session | command]
```

表 4–3 では、ttrace オプションについて説明します。

表 **4–3** tttrace オプション

オプション	説明
-0	セッション内のメッセージ・トレースをオフにするか、メッセージ・トレースを行わないで (つまり呼び出しのトレースだけを行なって) 指定のコマンドを実行します。
-F	指定のコマンドによってフォークされた、または ttsession によりセッション内で続けて起動されたすべての子を追跡します。通常、指定のコマンドだけか ttsession インスタンスがトレースされます。-F オプションが指定されている場合、プロセス ID がトレース出力の各行に表示され、どのプロセスが生成したかを示します。
-C	ToolTalk API へのクライアント呼び出しをトレースしないようにします。デフォルトでは呼び出しをトレースします。

表 **4–3** tttrace オプション　(続き)

オプション	説明
-a	トレースされたメッセージのすべての属性、引き数、およびコンテキスト・スロットを出力します。デフォルトでは、トレース出力へメッセージを出力するには単一行だけを使用します。
-o *outfile*	トレース出力に使用されるファイルです。セッション・トレースの場合、出力先は tttrace の標準出力です。
-S *session*	トレースするセッションです。デフォルトではデフォルト・セッション、つまり tt_open が通知するセッションです。
command	起動し、トレースを行う ToolTalk クライアント・コマンドです。
-e *script*	ttrace 設定として使用されるスクリプトです。
-f *scriptfile*	ttrace 設定を読み取るファイルです。

ttrace は、ttsession へのメッセージ・インタフェースと TT_TRACE_SCRIPT 環境変数を使用する ToolTalk クライアントとして実行できるようになっています。この変数が設定されると、トレース・スクリプトに指定されているようにクライアント側トレースをオンにすることを libtt に通知します。値の最初の文字が「.」か「/」の場合、その値は使用するトレース・スクリプトが入っているファイルのパス名として認識されます。その他の文字の場合は、インライン・トレース・スクリプトとして認識されます。

トレースされた関数の形式

次は、トレースされた ToolTalk 関数の例です。

[*pid*] *function_name*(*params*) = *return_value* (*Tt_status*)

メッセージ要約形式

-a オプションは、次のように 1 行のメッセージ要約の後にメッセージ属性を出力します。

Tt_state Tt_paradigm Tt_class (*Tt_disposition in Tt_scope*) : *status* == *Tt_status*

状態変更形式

状態の変更は次の形式で示されます。

old_state => *new_state*.

メッセージ配信形式

配信は次のように示されます。

`Tt_message => procid` *recipient_procid*

表 4–4 は、ディスパッチ・トレース中に受信するメッセージついて説明しています。

表 4–4 ディスパッチ・トレース用の理由

メッセージ	説明
`tt_message_send`	送信するメッセージです。
`tt_message_reject`	メッセージが拒否されました。
`tt_message_fail`	メッセージが無視されました。
`tt_message_reply`	メッセージへの応答です。
`tt_session_join`	参加するセッションです。
`tt_file_join`	参加するファイルです。
`tt_message_reply`	クライアントが指定の関数を呼び出しました。
`tt_message_send_on_exit`	ttsession は、`tt_close` を呼び出す前に切り離されたクライアントに対して on_exit メッセージをディスパッチしています。
`tt_message_accept`	ttsession は、ptype が起動されている間にブロックされたメッセージをディスパッチしています。起動されたクライアントは、ptype がブロック解除されることを示す `tt_message_accept` か `tt_message_reply` のどちらかを呼び出します。
`TT_ERR_PTYPE_START`	ptype インスタンスがメッセージの受信のために起動されましたが、起動コマンドは ttsession に接続される前に終了しました。
`TT_ERR_PROCID`	ttsession はこの要求に応じて動作中であったクライアントとの接続を切りました。
`ttsession -> ttsession`	別のセッションが、このセッションに対してメッセージの受信者を見つけるように求めています。
`ttsession <- ttsession`	別のセッションが、このセッションで発信したメッセージの更新 (たとえば失敗すること) を求めています。

形式の照合

ディスパッチがトレースされているとき、照合は次の形式のうちの 1 つで示されます。

```
Tt_message & Tt_pattern {
Tt_message & ptype ptid {
Tt_message & otype otid {
```

パターンまたはシグニチャーが次の形式で出力されます。

```
} == match_score; [/* mismatch_reason */]
```

例

この節では、tttrace 関数の使用例を取り上げます。

パターンの登録とパターンとの一致通知の送信

パターンを登録し、そのパターンに一致するという通知を送信するには、次のように入力します。

```
% tttrace -a myclientprogram
```

例 4–1 は、その結果を示します。

例 **4–1** パターンの登録と通知の送信

```
tt_open() = 0x51708=="7.jOHHM X 129.144.153.55 0" (TT_OK)
tt_fd() = 11 (TT_OK)
tt_pattern_create() = 0x50318 (TT_OK)
tt_pattern_category_set(0x50318, TT_OBSERVE) = 0 (TT_OK)
tt_pattern_scope_add(0x50318, TT_SESSION) = 0 (TT_OK)
tt_pattern_op_add(0x50318, 0x2f308=="Hello World") = 0 (TT_OK)
tt_default_session() = 0x519e0=="X 129.144.153.55 0" (TT_OK)
tt_pattern_session_add(0x50318, 0x519e0=="X 129.144.153.55 0") = 0 (TT_OK)
tt_pattern_register(0x50318) = 0 (TT_OK)
tt_message_create() = 0x51af0 (TT_OK)
tt_message_class_set(0x51af0, TT_NOTICE) = 0 (TT_OK)
tt_message_address_set(0x51af0, TT_PROCEDURE) = 0 (TT_OK)
tt_message_scope_set(0x51af0, TT_SESSION) = 0 (TT_OK)
tt_message_op_set(0x51af0, 0x2f308=="Hello World") = 0 (TT_OK)
tt_message_send(0x51af0)        ...
    TT_CREATED => TT_SENT:
    TT_SENT TT_PROCEDURE TT_NOTICE (TT_DISCARD in TT_SESSION): 0 == TT_OK
    id:             0 7.jOHHM X 129.144.153.55 0
    op:             Hello World
    session:        X 129.144.153.55 0
    sender:         7.jOHHM X 129.144.153.55 0
= 0 (TT_OK)
tt_message_receive()        ...
    Tt_message => procid <7.jOHHM X 129.144.153.55 0>
    TT_SENT TT_PROCEDURE TT_NOTICE (TT_DISCARD in TT_SESSION): 0 == TT_OK
    id:             0 7 jOHHM X 129.144.153.55 0
    op:             Hello World
    session:        X 129.144.153.55 0
```

例 **4–1** パターンの登録と通知の送信　*(続き)*

```
    sender:         7.jOHHM X 129.144.153.55 0
    pattern:        0:7.jOHHM X 129.144.153.55 0
= 0x51af0 (TT_OK)
```

ttsession によるメッセージ・フローの表示を見るには、次のように入力します。

```
% tttrace -a
```

ttsession による *mylientprogram* のメッセージ・フローの表示は、例 4–2 のとおりです。

例 **4–2** ttsession によるトレースの表示

```
tt_message_reply:
    TT_SENT => TT_HANDLED:
    TT_HANDLED TT_PROCEDURE TT_REQUEST (TT_DISCARD in TT_SESSION): 0 == TT_OK
    id:     0 2.jOHHM X 129.144.153.55 0
    op:     Session_Trace
    args:   TT_IN string: "> /tmp/traceAAAa002oL; version 1; states"[...]
    session:X 129.144.153.55 0
    sender: 2.jOHHM X 129.144.153.55 0
    pattern:0:X 129.144.153.55 0
    handler:0.jOHHM X 129.144.153.55 0
    Tt_message => procid <2.jOHHM X 129.144.153.55 0>
tt_message_send:
    TT_CREATED TT_PROCEDURE TT_NOTICE (TT_DISCARD in TT_SESSION): 0 == TT_OK
    id:     0 7.jOHHM X 129.144.153.55 0
    op:     Hello World
    session:X 129.144.153.55 0
    sender: 7.jOHHM X 129.144.153.55 0
    TT_CREATED => TT_SENT:
    TT_SENT TT_PROCEDURE TT_NOTICE (TT_DISCARD in TT_SESSION): 0 == TT_OK
    id:     0 7.jOHHM X 129.144.153.55 0
    op:     Hello World
    session:X 129.144.153.55 0
    sender: 7.j HHM X 129.144.153.55 0
    Tt_message & Tt_pattern {
    id:     0:7.jOHHM X 129.144.153.55 0
    category:TT_OBSERVE
    scopes: TT_SESSION
    sessions:X 129.144.153.55 0
    ops:    Hello World
    } == 3;
    Tt_message => procid <7.jOHHM X 129.144.153.55 0>
```

注 **–** トレースされた最初のメッセージは、ほとんどの場合 `ttrace` が送信した要求への ttsession による応答になります。

メッセージ・フローのトレース

特定の、デフォルトではないセッション内のメッセージ・フローをトレースするには、次のように入力します。

```
% tttrace -S "01 15303 1342177284 1 0 13691 129.144.153.55 2"
```

"01 15303 1342177284 1 0 13691 129.144.153.55 2" は、デフォルトではない特定のトレースされるセッションです。

ToolTalk トレースのための設定

`ttrace` スクリプトには、ToolTalk 呼び出しとメッセージをコントロールする設定があります。`ttrace` スクリプトは、セミコロンか復帰改行で区切られたコマンドから成ります。重複している値が設定に指定されると、最後の値が使用されます。

表 4–5 では、これらのコマンドについて説明します。

表 4–5 `tttrace` スクリプト・コマンド

コマンド	説明
`version` n	使用されている `tttracefile` コマンド構文のバージョンです。現在のバージョンは 1 です。
`follow [off | on]`	トレースされたクライアントによってフォークされた、またはトレースされたセッションで続けて起動されたすべての子についても追跡するかどうか設定します。デフォルトは `off` です。
`[> | >>]` *outfile*	トレース出力に使用されるファイルです。デフォルトでは、トレースの出力先は標準エラーです。> と >> の通常のシェル解釈が適用されます。
`functions [all | none |` *func...*`]`	トレースする ToolTalk API 関数です。*func* には、シェル・ワイルドカード文字が入ります。デフォルトは `all` です。
`attributes [all | none]`	none (デフォルト) は、トレース出力にメッセージを出力するときに単一行だけを使用することを意味します。`all` は、トレースされたメッセージの属性、引き数、コンテキスト・スロットすべてを出力することを意味します。

表 **4–5** tttrace スクリプト・コマンド　(続き)

コマンド	説明
states [none | edge | deliver | dispatch | Tt_states]...	メッセージをトレースするための状態です。tt_c.h に定義されている Tt_states の他に有効な状態は次のとおりです。 ■ none – メッセージのトレースをすべて使用不可能にする ■ edge – 最初の状態 (TT_SENT) と最後の状態 (TT_HANDLED、TT_FAILED) を入力しているメッセージ ■ deliver – すべての状態の変更と、すべてのクライアントの配信 ■ dispatch – 配信および 一致するように考慮されたすべてのパターン (デフォルト)
ops *toTrace*... sender_ptypes *toTrace*... handler_ptypes *toTrace*...	表示されたメッセージ属性の値として *toTrace* を持っているトレース・メッセージです。*toTrace* にはシェル・ワイルドカード文字が入ります。指定のメッセージ属性に対して指定されている *toTrace* 引き数がない場合、その属性の値でメッセージをトレースから除外するものもありません。

付録 **A**

メッセージ・ツールキット

ToolTalk メッセージ・ツールキットは、ToolTalk アプリケーション・プログラミング・インタフェース (API) のハイレベルなインタフェースです。同じメッセージ・プロトコルに準拠する他のアプリケーションとの最適な相互運用のための基本的な ToolTalk のメッセージと機能をアプリケーションへ簡単に統合できるよう共通の定義と規約を提供します。

ToolTalk メッセージ・ツールキットのメッセージの大部分は、標準 ToolTalk メッセージ・セットに含まれています。メッセージ・ツールキットの関数は別々にコーディングする必要があるいくつかのタスクを透過的に処理します。たとえば、`ttdt_file_join()` 関数はパターンを登録し、指定された配信範囲にある指定のファイルへの Deleted、Reverted、Moved、および Saved 通知を監視します。コールバック・メッセージも呼び出します。

ToolTalk メッセージ・ツールキットの一般的な説明

相互運用は、別々に開発されたアプリケーションを同時に実行する場合には重要なテーマです。相互運用アプリケーションの開発者は、ツールキットのメッセージを一致させています。つまり、プロトコルは小型の十分に定義されたインタフェースを形成します。このインタフェースは、アプリケーションの自律性を最大限にします。

ToolTalk メッセージ・ツールキットは、アプリケーションの相互運用において重要であり、メッセージに対する完全なサポートを提供します。メッセージ・プロトコル仕様は、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を含んでいます。これらのメッセージは、アプリケーションの機能を利用するために既存のアプリケーションに対して更新できます。共有している情報を送信、受信、および使用するために既存のアプリケーションにこれらのメッセージを簡単に追加できます。

ToolTalk メッセージ規約に準拠しているツールは、意味が異なる場合同じ ToolTalk 構文を使用しません。同じ意味の場合は異なる ToolTalk 構文を使用するのでツール間の通信に失敗することもありません。プロトコルが監視されている場合、互いに影響を及ぼすことなく連携するアプリケーションを変更したり置き換えたりすることもできます。

メッセージ・ツールキットのメッセージの大部分は、標準 ToolTalk メッセージ・セットに含まれています。標準 ToolTalk メッセージ・セットの詳細は、ToolTalk のマニュアルページを参照してください。表 A–1 に、この章で説明する関数の一覧を示します。これらの関数は、ToolTalk メッセージ・ツールキットの一部を構成します。

表 **A–1** ToolTalk メッセージ・ツールキット関数

関数	説明
`ttdt_close()`	ToolTalk 通信終端を破棄します。
`ttdt_file_event()`	ファイルに関するイベントを通知します。
`ttdt_file_join()`	ファイルに関する ToolTalk イベントを監視できるように登録します。
`ttdt_file_notice()`	ファイルに関する標準 ToolTalk 通知を作成して送信します。
`ttdt_file_quit()`	ファイルに関する ToolTalk イベントにおける配信対象を登録解除します。
`ttdt_file_request()`	ファイルに関する標準 ToolTalk 要求を作成して送信します。
`ttdt_Get_Modified()`	ファイルに変更内容を保留している ToolTalk クライアントがないかどうかを問い合わせます。
`ttdt_message_accept()`	ToolTalk 要求の処理を引き受けます。
`ttdt_open()`	ToolTalk 通信終端を作成します。
`ttdt_Revert()`	ファイルの内容を最後に保存した内容に戻すよう ToolTalk クライアントに要求します。
`ttdt_Save()`	ToolTalk クライアントがファイルを保存するよう要求します。
`ttdt_sender_imprint_on()`	ツールに指定の ToolTalk ツールの動作や特性をエミュレートさせます。
`ttdt_session_join()`	ToolTalk セッションに参加し、多くの標準デスクトップ・メッセージのパターンとデフォルトのコールバックを登録します。

表 **A–1** ToolTalk メッセージ・ツールキット関数　(続き)

関数	説明
`ttdt_session_quit()`	セッションに参加した時に登録したパターンとデフォルトのコールバックをすべて登録解除し、ToolTalk セッションを終了します。
`ttdt_subcontract_manage()`	未処理の要求を管理します。
`ttmedia_Deposit()`	ドキュメントにチェックポイントを設定するための Deposit 要求を送信します。
`ttmedia_load()`	ドキュメントの表示、編集、作成のための Media Exchange 要求を作成して送信します。
`ttmedia_load_reply()`	Display 要求、Edit 要求、または Compose 要求に返信します。
`ttmedia_ptype_declare()`	Media Exchange メディア・エディタの ptype を宣言します。
`tttk_block_while()`	返信などの待機中にプログラムをブロックします。
`tttk_message_abandon()`	メッセージを無視または拒否してから破棄します。
`tttk_message_create()`	メッセージ規約に準拠しているメッセージを作成します。
`tttk_message_fail()`	メッセージを無視します。
`tttk_message_receive()`	次の ToolTalk メッセージを取り出します。
`tttk_message_reject()`	メッセージを拒否します。
`tttk_op_string()`	オペレーションに対する文字列を返します。
`tttk_string_op()`	文字列に対するオペレーションを返します。
`tttk_Xt_input_handler()`	Xt クライアントのための ToolTalk イベントを処理します。

ToolTalk の規約

ツールキットのメッセージ規約の大部分は、標準 ToolTalk メッセージ・セットの記述で構成されます。この節では、特定の標準メッセージ・セットには関係ない規約について説明します。

表 **A–2** メッセージ・ツールキット規約

フィールド	説明
fileAttrib	メッセージのファイル属性を設定できるかどうか、または設定する必要があるかどうかを示します。ToolTalk サービスには、各メッセージがファイルを参照し、名前が付いているファイルでは「配信対象である」クライアントにメッセージを配信できる機能 (「ファイル配信範囲指定機能」と呼ぶ) があります。
opName	オペレーション名またはイベント名 (「op」ともいう) です。重要なのは、ツールが異なっても意味が同じものに対しては同じ opName を使用することです。メッセージが標準のものではない場合、その opName は一意でなければなりません。たとえば、opName に `Company_Product` (*Acme_HoarkTool_Hoark_My_Frammistat* など) という接頭辞を付けます。
requiredArgs	メッセージに必ず含まれていなければならない引き数です。
optionalArgs	メッセージに含まれることもある特別引き数です。メッセージ内のオプション引き数は、指定の順序で必須の引き数の後ろに指定しなければなりません。
vtype argumentName	特定の引き数についての記述です。vtype は、メッセージ引き数に含まれるデータの種類を表す文字列で、プログラマが定義します。ToolTalk サービスは、送信メッセージ・インスタンスと登録されているメッセージ・パターンとを照合する場合にだけ vtype を使用します。各 vtype は、規約により一般的な単一の既知のデータ型に対応づけなければなりません。

アプリケーション記述時のメッセージ・ツールキットの使用

ツールキットを使用するには、ToolTalk メッセージ・ツールキットのヘッダ・ファイルを組み込みます。

```
#include <Tt/tttk.h>
```

ToolTalk メッセージ・ツールキット

この節では、ToolTalk メッセージ・ツールキットの一部である関数について説明します。

ttdt_close

```
Tt_status   ttdt_close( const char *    procid,
                        const char *    new_procid,
                        int             sendStopped );
```

ttdt_close() 関数は、ToolTalk 通信終端を破棄します。この関数は、ToolTalk 関数 tt_close() を呼び出します。

- procid の値が != 0 の場合、この関数は次を呼び出します。

```
tt_default_procid_set(    procid    )
```

- new_procid の値が != 0 の場合、この関数は次を呼び出します。

```
tt_default_procid_set(    new_procid )
```

- sendStopped パラメータが設定されている場合、この関数は Stopped 通知を送信します。

ttdt_close() 関数は、ToolTalk 関数 tt_default_procid_set() および tt_close() が返すどのようなエラーも返す可能性があります。Sending 通知が失敗した場合、エラーは伝達されません。

ttdt_file_event

```
Tt_status   ttdt_file_event(    Tt_message      context,
                                Tttk_op         event,
                                Tt_pattern *    patterns,
                                int             send );
```

ttdt_file_event() 関数は、ToolTalk サービスを介してファイルに関するイベントを通知します。この関数は、指定されたファイルに関係のあるイベントを知らせる ToolTalk メッセージを作成し、必要に応じて送信します。このファイルは、patterns の作成時に ttdt_file_join() 関数に渡されたパス名で示されます。

- 表 A–3 では、event パラメータの値に対応する通知内容を示します。

表 **A–3** event パラメータの通知内容

通知される event	通知内容
TTDT_MODIFIED	ttdt_file_join() 関数に渡された配信範囲を登録し、Get_Modified 要求、Save 要求、Revert 要求を処理する配信対象ツールへイベントを通知します。

表 **A–3** event パラメータの通知内容　(続き)

通知される event	通知内容
TTDT_SAVED, TTDT_REVERTED	Get_Modified 要求、Save 要求、Revert 要求のハンドラ・パターンを登録解除します。 send パラメータが設定された場合、配信範囲に応じて Saved 通知か Reverted 通知を送信します。

- send パラメータを設定すると、配信範囲に Modified 通知を送信します。
- context パラメータがゼロ以外の値のとき、このルーチンによって作成されるメッセージはスロット名が ENV_ で始まるすべてのコンテキストを継承します。

表 A–4 では、この関数が返す可能性のあるエラーの一覧を示します。

表 **A–4** ttdt_file_event が返す可能性のあるエラー

エラーの値	説明
TT_DESKTOP_EINVAL	イベント通知が無効です。有効なイベント通知は、TTDT_MODIFIED、TTD_TSAVED、TTDT_REVERTED です。
TT_ERR_POINTER	patterns パラメータが NULL です。
TT_ERR_OVERFLOW	ToolTalk サービスが受信したメッセージの数が、正しく処理可能なアクティブ・メッセージの最大数 (2000) に達しました。
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか正しくインストールされていないかのどちらかを示します。

ttdt_file_join

```
Tt_message  ( *Ttdt_file_cb)(   Tt_message      msg,
                                Tttk_op         op,
                                char *          pathname,
                                void *          clientdata,
                                int             same_euid_egid,
                                int             same_procid );
Tt_pattern * ttdt_file_join(    const char *    pathname,
                                Tt_scope        the_scope,
                                int             join,
                                Ttdt_file_cb    cb,
                                void *          clientdata );
```

ttdt_file_join() 関数は、指定のファイルに関する ToolTalk イベントを監視できるように登録します。Deleted、Modified、Reverted、Moved、Saved の 5 種類の通知を監視するように配信範囲に登録します。

- コールバック・メッセージ引き数 Ttdt_file_cb は、表 A–5 に示すパラメータを取ります。

表 **A–5** Ttdt_file_cb が取るパラメータ

パラメータ	説明
message	送信中のメッセージ
op	要求されているオペレーション
pathname	メッセージに関連付けられたファイルのパス名
	このコピーは、ToolTalk 関数 tt_free() で解放できます。
clientdata	メッセージに含まれているクライアント・データ
same_euid_egid	送信側を識別するフラグ
	この値が true の場合は、送信側の信頼性は高いです。
same_procid	送信側を識別するフラグ
	この値が true の場合は、送信側の procid は受信側と同じです。

- the_scope パラメータの値がゼロ (つまり TT_SCOPE_NONE) の場合、ファイルの配信範囲はデフォルト (TT_BOTH) に設定されます。しかし、たとえば ToolTalk データベース・サーバ rpc.ttdbserver が pathname を所有するファイル・サーバにインストールされていない場合、ファイル配信範囲は TT_FILE_IN_SESSION に設定されます。

  ttdt_file_join() 関数は the_scope の値および pathname のコピーとを Tt_patterns 型の戻り値に関連付けることで、ttdt_file_quit() 関数がパターンにアクセスすることを可能にします。呼び出し側は、ttdt_file_join() 呼び出しが返ると、pathname を変更または解放できます。

- join パラメータの値が true の場合、この関数は次を呼び出します。

```
tt_file_join(    pathname  )
```

この関数は、NULL で終了する Tt_pattern 型の配列を返します。この配列を破棄するには、ttdt_file_quit() 関数を使用します。エラーが返される場合、返された配列は tt_ptr_error で解読できるエラー・ポインタです。表 A–6 は、ttdt_file_join() 関数が返す可能性のあるエラーの一覧です。

表 **A–6** ttdt_file_join が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスが存在しません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか正しくインストールされていないかのどちらかを示します。
TT_ERR_DBAVAIL	ToolTalk サービスが、このオペレーションに必要な ToolTalk データベースにアクセスできませんでした。
TT_ERR_DBEXIST	ToolTalk サービスが、指定された ToolTalk データベースを予期した場所で見つけることができませんでした。
TT_ERR_PATH	ToolTalk サービスが、指定されたファイル・パス名でディレクトリを読み取ることができませんでした。
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。

ttdt_file_notice

```
Tt_message  ttdt_file_notice(   Tt_message      context,
                                Tttk_op         op,
                                T_scope         scope,
                                const char *    pathname,
                                int             send_and_destroy );
```

ttdt_file_notice() 関数は、ファイルに関する標準 ToolTalk 通知を作成し、必要に応じて送信します。Created、Deleted、Moved、Reverted、Saved、Modified の 6 種類の標準ファイル通知を作成するには、この関数を使用します。

注 – ttdt_file_event() 関数は、ttdt_file_notice() 関数よりもハイレベルなインタフェースです。Moved 通知以外のすべての通知を送信する場合は、この関数を使用するようにしてください。

- context パラメータがゼロ以外の値のとき、このルーチンによって作成されるメッセージは、スロット名が ENV_ で始まるすべてのコンテキストを継承します。
- この関数は指定の op パラメータと scope パラメータで通知を作成し、そのファイル属性を pathname パラメータに設定します。
- send_and_destroy パラメータを設定すると、この関数はメッセージを送信してから破棄します。

  send_and_destroy パラメータの値が false の場合、作成されたメッセージが返されます。send_and_destroy パラメータの値が true の場合、ゼロが返されます。

エラーが発生すると、エラー・ポインタが返されます。Tt_status を調べるには、tt_ptr_error を使用します。表 A–7 では、この関数が返す可能性のあるエラーについて説明します。

表 **A–7** ttdt_file_notice が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスが存在しません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。
TT_ERR_OVERFLOW	ToolTalk サービスが受信したメッセージの数が、正しく処理可能なアクティブ・メッセージの最大数 (2000) に達しました。
TT_ERR_DBAVAIL	ToolTalk サービスが、このオペレーションに必要な ToolTalk データベースにアクセスできませんでした。
TT_ERR_DBEXIST	ToolTalk サービスが、指定された ToolTalk データベースを予期した場所で見つけることができませんでした。
TT_DESKTOP_EINVAL	オペレーションが移動され、send_and_destroy パラメータの値が true です。
TT_ERR_POINTER	パス名が NULL か、ToolTalk エラー・ポインタでした。

ttdt_file_quit

```
Tt_status   ttdt_file_quit( Tt_pattern *    patterns,
                            int             quit );
```

ttdt_file_quit() 関数は、ファイルの ToolTalk イベントにおける配信対象を登録解除します。この関数はパターンを破棄します。quit パラメータが設定されると、この関数は次を呼び出します。

```
    tt_file_quit( pathname )
```

patterns の作成時に ttdt_file_join() 関数に渡したパス名における配信対象を登録解除する場合にこの関数を使用します。表 A–8 に、この関数が返す可能性のあるエラーの一覧を示します。

表 **A–8** ttdt_file_quit が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_DBAVAIL	ToolTalk サービスが、このオペレーションに必要な ToolTalk データベースにアクセスできませんでした。
TT_ERR_DBEXIST	ToolTalk サービスが、指定された ToolTalk データベースを予期した場所で見つけることができませんでした。
TT_ERR_POINTER	パターンが NULL か、無効です。

ttdt_file_request

```
Tt_message      ttdt_file_request(  Tt_message        context,
                                    Tttk_op           op,
                                    Tt_scope          scope,
                                    const char        pathname,
                                    Ttdt_file_cb      cb,
                                    void              client_data,
                                    int               send_and_destroy  );
```

ttdt_file_request() 関数は、標準デスクトップのファイルに対する配信範囲指定要求 (Get_Modified、Save、Revert など) を作成し、必要に応じて送信します。

注 – この関数は、ttdt_Get_Modified() 関数、ttdt_Save() 関数、ttdt_Revert() 関数よりもローレベルのインタフェースです。要求を作成してから送信し、その応答に応じてブロックします。

---

ttdt_file_request() 関数は、指定の op と scope で要求を作成し、そのファイル属性を pathname に設定します。デスクトップ・メッセージ規約に従い、TT_IN と vtype File のまだ設定されていない Tt_mode 型引き数が要求に追加されます。指定のオペレーションが TTDT_GET_MODIFIED() の場合は、TT_OUT と vtype Boolean のまだ設定されていない Tt_mode 型引き数も要求に追加されます。

context パラメータがゼロ以外の値のとき、このルーチンによって作成される要求は、スロット名が ENV_ で始まるすべてのコンテキストを context から継承します。

この関数は、作成された要求のメッセージ・コールバックとして cb をインストールし、クライアント・データが確実にコールバックに渡されるようにします。send が true の場合、この関数はハンドルを返す前に要求を送信します。

この関数は正常終了時に、作成された Tt_message を返します。エラーが発生すると、エラー・ポインタが返されます。Tt_status を調べるには、tt_ptr_error を使用します。表 A–9 には、この関数が返す可能性のあるエラーの一覧を示します。

表 **A–9** ttdt_file_request が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。
TT_ERR_OVERFLOW	ToolTalk サービスが受信したメッセージの数が、正しく処理可能なアクティブ・メッセージの最大数 (2000) に達しました。
TT_ERR_DBAVAIL	ToolTalk サービスが、このオペレーションに必要な ToolTalk データベースにアクセスできませんでした。
TT_ERR_DBEXIST	ToolTalk サービスが、指定された ToolTalk データベースを予期した場所で見つけることができませんでした。
TT_ERR_POINTER	パス名が NULL か、無効です。

ttdt_Get_Modified

```
int     ttdt_Get_Modified(  Tt_message          context,
                            const char *        pathname,
                            Tt_scope            the_scope,
                            XtAppContext        app2run,
                            int                 ms_timeout );
```

ttdt_Get_Modified() 関数は、ファイルに変更内容を保留している ToolTalk クライアントがないかどうか問い合わせます。この関数は、Get_Modified 要求を送信して応答を待ちます。

- context パラメータがゼロ以外の値のとき、このルーチンによって作成されるメッセージは、スロット名が ENV_ で始まるすべてのコンテキストを継承します。
- Get_Modified 要求は、固定表示を行うつもりの pathname に変更内容を保留している ToolTalk クライアントがないかどうか問い合わせます。

- the_scope パラメータは、Get_Modified 要求が送信される配信範囲を示します。このパラメータの値がゼロ (つまり TT_SCOPE_NONE) の場合、ファイルの配信範囲はデフォルト (TT_BOTH) に設定されます。しかし、たとえば ToolTalk データベース・サーバ rpc.ttdbserver が pathname を所有するファイル・サーバにインストールされていない場合、ファイル配信範囲は TT_FILE_IN_SESSION に設定されます。
- app2run パラメータと ms_timeout パラメータは、この関数が送信する Get_Modified 要求への応答をブロックするために、tttk_block_while() 関数に渡されます。

Get_Modified 要求が指定のタイムアウト時間内に肯定応答を受信すると、ttdt_Get_Modified() 関数はゼロ以外を返します。そうでない場合は、ゼロを返します。この呼び出しはエラーを返しません。

ttdt_message_accept

```
Tt_pattern *    ttdt_message_accept(    Tt_message          contract,
                                        Ttdt_contract_cb    cb,
                                        void *              clientdata,
                                        Widget              shell,
                                        int                 accept,
                                        int                 sendStatus );
```

ttdt_message_accept() 関数は、ToolTalk 要求を処理することを受け入れます。ツールは、要求の処理 (つまり、無視したり拒否すること) を受け入れる場合にこの関数を呼び出します。

Ttdt_contract_cb 引き数は、表 A–10 に示すパラメータを取ります。

表 **A–10** Ttdt_contract_cb 引き数が取るパラメータ

パラメータ	説明
Tt_message msg	送信状態にある要求 クライアント・プログラムは、この要求を無視または拒否する、あるいはメッセージに応答します。
Tttk_op op	着信中の要求のオペレーション
Widget shell	ttdt_message_accept() 関数に渡すシェル
void *clientdata	ttdt_message_accept() 関数に渡すクライアント・データ
Tt_message contract	ttdt_message_accept() 関数に渡すコントラクト

メッセージ msg を正常に処理すると、コールバックはゼロを返します。そうでない場合は、Tt_message に送られた tt_error_pointer を返します。

メッセージ msg を処理しない場合、コールバックはメッセージを返し、TT_CALLBACK_CONTINUE ルーチンを呼び出しスタックに渡してメッセージを他のコールバックに提供するか、メッセージを tt_message_receive() 呼び出しに返します。

ttdt_message_accept() 関数は、ハンドラを宛先とする要求 (表 A–11 を参照してください) をデフォルト・セッションで登録します。

表 **A–11** ttdt_message_accept が登録する要求

要求	処理方法
Get_Geometry, Set_Geometry	shell パラメータが NULL でない場合、これらの要求は透過的に処理されます。shell パラメータが NULL で、cb パラメータが NULL でない場合は、これらの要求はコールバック・ルーチンに渡されます。それ以外の場合は、これらの要求はエラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Get_Iconified, Set_Iconified	shell パラメータが NULL でない場合、これらの要求は透過的に処理されます。shell パラメータが NULL で、cb パラメータが NULL でない場合は、これらの要求はコールバック・ルーチンに渡されます。それ以外の場合は、これらの要求はエラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Get_Mapped, Set_Mapped	shell パラメータが NULL でない場合、これらの要求は透過的に処理されます。shell パラメータが NULL で、cb パラメータが NULL でない場合は、これらの要求はコールバック・ルーチンに渡されます。それ以外の場合は、これらの要求はエラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Raise	shell パラメータが NULL でない場合、この要求は透過的に処理されます。shell パラメータが NULL で、cb パラメータが NULL でない場合は、この要求はコールバック・ルーチンに渡されます。それ以外の場合は、この要求はエラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Lower	shell パラメータが NULL でない場合、この要求は透過的に処理されます。shell パラメータが NULL で、cb パラメータが NULL でない場合は、この要求はコールバック・ルーチンに渡されます。それ以外の場合は、この要求はエラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Get_XInfo, Set_XInfo	shell パラメータが NULL でない場合、これらの要求は透過的に処理されます。shell パラメータが NULL で、cb パラメータが NULL でない場合は、これらの要求はコールバック・ルーチンに渡されます。それ以外の場合は、これらの要求はエラー TT_DESKTOP_ENOTSUP とともに異常終了します。

表 **A–11** ttdt_message_accept が登録する要求　(続き)

要求	処理方法
Pause	cb パラメータが NULL でない場合、この要求はコールバック・ルーチンに渡されます。cp パラメータが NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Resume	cb パラメータが NULL でない場合、この要求はコールバック・ルーチンに渡されます。cp パラメータが NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Quit	cb パラメータが NULL でない場合、この要求はコールバック・ルーチンに渡されます。cp パラメータが NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Get_Status	cb パラメータが NULL でない場合、この要求はコールバック・ルーチンに渡されます。cp パラメータが NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。

contract 引き数が TT_WRN_START_MESSAGE メッセージ・ステータスの時、メッセージはツールを起動します。

注 – 起動したツールは、その ptype にすでにディスパッチされている他のメッセージを受信するため、コントラクトを受け入れる前に使用したい配信範囲に参加する必要があります。そうしないと、ツールは動作中にその ptype の宣言を解除しなければなりません。ツールがどの配信範囲にも参加していないと、ディスパッチされたメッセージによって ptype の他のインスタンスが起動されます。

```
tt_message_accept(  contract    )
```

sendStatus 引き数が true の場合、ttdt_open() 関数に渡されたパラメータ (存在する場合) を使用して、ttdt_message_accept() 関数は要求側に Status 通知を送信します。

この関数は、NULL で終了する Tt_pattern 型の配列を返します。この配列を破棄する場合は、tttk_patterns_destroy() 関数を使用します。エラーが返される場合、返された配列は tt_ptr_error で解読されるエラー・ポインタになります。表 A–12 は、ttdt_message_accept() 関数が返す可能性のあるエラーの一覧です。

表 **A–12** ttdt_message_accept が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか正しくインストールされていないかのどちらかを示します。

表 **A–12** ttdt_message_accept が返す可能性のあるエラー　*(続き)*

エラーの値	説明
TT_ERR_POINTER	渡されたポインタが、このオペレーションに適した型のオブジェクトを指していません。たとえば、文字列が必要なときにポインタは整数を指している場合などです。
TT_ERR_UNIMP	デフォルト・セッションの ttsession は、tt_message_accept() 関数をサポートしていないバージョン (1.0 または 1.0.1) です。 注 – contract 引き数のメッセージ・ステータスが TT_WRN_START_MESSAGE の場合、ツールの ptype に対して送信されるメッセージは、contract が拒否、応答、または無視されるまでブロックされます。

ttdt_open

```
char *  ttdt_open(  int *           ttfd,
                    const char *    toolname,
                    const char *    vendor,
                    const char *    version,
                    int             sendStarted );
```

ttdt_open() 関数は、ToolTalk 通信終端を作成します。この関数は、tt_open() 関数と tt_fd() 関数を呼び出します。ttdt_open() 関数は、toolname、vendor、version と作成された procid とを関連付けます。新しい procid のデフォルト・コンテキストを environ(5) から初期化します。sendStarted 引き数が設定された場合は、Started 通知を送信します。

ttdt_open() 関数は、tt_free() 関数で解放できる文字列に、作成された procid を返します。

この関数は、tt_open() 関数および tt_fd() 関数で発生したどのようなエラーも返すことができます。Started 通知が失敗した場合、エラーは伝達されません。

ttdt_Revert

```
Tt_status       ttdt_Revert(    Tt_message      context,
                                const char *    pathname,
                                Tt_scope        the_scope,
                                XtAppContext    app2run,
                                int             ms_timeout );
```

ttdt_Revert() 関数は、ファイル内容を元に戻すように ToolTalk クライアントに要求します。この関数は、Revert 要求を the_scope に送信し、応答を待ちます。Revert 要求は、pathname に保留中の変更を破棄するように、処理中である ToolTalk クライアントに求めます。

- context パラメータがゼロ以外の値のとき、このルーチンによって作成されるメッセージは、スロット名が ENV_ で始まるすべてのコンテキストを継承します。
- the_scope パラメータの値がゼロ (つまり TT_SCOPE_NONE) の場合、ファイルの配信範囲はデフォルト (TT_BOTH) に設定されます。しかし、たとえば ToolTalk データベース・サーバ rpc.ttdbserver が pathname を所有するファイル・サーバにインストールされていない場合、ファイル配信範囲は TT_FILE_IN_SESSION に設定されます。
- app2run パラメータと ms_timeout パラメータは、この関数が送信する Revert 要求への応答をブロックするために、tttk_block_while() 関数に渡されます。

要求が指定のタイムアウト時間内に肯定応答を受信すると、ttdt_Revert() 関数は TT_OK を返します。そうでない場合は、失敗した応答に対する tt_message_status の戻り値、または表 A–13 に示すエラーのいずれかを返します。

表 **A–13** ttdt_Revert が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。
TT_ERR_OVERFLOW	ToolTalk サービスが受信したメッセージの数が、正しく処理可能なアクティブ・メッセージの最大数 (2000) に達しました。
TT_ERR_DBAVAIL	ToolTalk サービスが、このオペレーションに必要な ToolTalk データベースにアクセスできませんでした。
TT_ERR_DBEXIST	ToolTalk サービスが、指定された ToolTalk データベースを予期した場所で見つけることができませんでした。
TT_DESKTOP_ETIMEOUT	指定されたタイムアウト時間内に応答を受信しませんでした。
TT_DESKTOP_EPROTO	要求が無視されました。しかしハンドラは、特定のエラー・ステータスの代わりに失敗した応答に対する tt_message_status の戻り値を TT_OK に設定します。

表 **A–13** ttdt_Revert が返す可能性のあるエラー　(続き)

エラーの値	説明
TT_ERR_POINTER	パス名が NULL か、ToolTalk エラー・ポインタです。

ttdt_Save

```
Tt_status   ttdt_Save(  Tt_message      context,
                        const char *    pathname,
                        Tt_scope        the_scope,
                        XtAppContext    app2run,
                        int             ms_timeout );
```

ttdt_Save() 関数はファイルを保存するように ToolTalk クライアントに要求します。この関数は、Save 要求を the_scope に送信し、応答を待ちます。Save 要求は、pathname に保留中の変更を破棄するように、処理中である ToolTalk クライアントに求めます。

- context パラメータがゼロ以外の値のとき、このルーチンによって作成されるメッセージは、スロット名が ENV_ で始まるすべてのコンテキストを継承します。
- the_scope パラメータの値がゼロ (つまり TT_SCOPE_NONE) の場合、ファイルの配信範囲はデフォルト (TT_BOTH) に設定されます。しかし、たとえば ToolTalk データベース・サーバ rpc.ttdbserver が pathname を所有するファイル・サーバにインストールされていない場合、ファイル配信範囲は TT_FILE_IN_SESSION に設定されます。
- app2run パラメータと ms_timeout パラメータは、この関数が送信する Save 要求への応答をブロックするために、tttk_block_while() 関数に渡されます。

要求が指定のタイムアウト時間内に肯定応答を受信すると、ttdt_Save() 関数は TT_OK を返します。そうでない場合は、失敗した応答に対する tt_message_status の戻り値、または表 A–14 に示すエラーのいずれかを返します。

表 **A–14** ttdt_Save が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。

表 **A–14** ttdt_Save が返す可能性のあるエラー　(続き)

エラーの値	説明
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。
TT_ERR_OVERFLOW	ToolTalk サービスが受信したメッセージの数が、正しく処理可能なアクティブ・メッセージの最大数 (2000) に達しました。
TT_ERR_DBAVAIL	ToolTalk サービスが、このオペレーションに必要な ToolTalk データベースにアクセスできませんでした。
TT_ERR_DBEXIST	ToolTalk サービスが、指定された ToolTalk データベースを予期した場所で見つけることができませんでした。
TT_DESKTOP_ETIMEOUT	指定されたタイムアウト時間内に応答を受信しませんでした。
TT_DESKTOP_EPROTO	要求が無視されました。しかし、ハンドラは、特定のエラー・ステータスの代わりに失敗した応答に対する tt_message_status の戻り値を TT_OK に設定します。
TT_ERR_POINTER	パス名が NULL か、ToolTalk エラー・ポインタです。

ttdt_sender_imprint_on

```
Tt_status   ttdt_sender_imprint_on(     const char *    handler,
                                        Tt_message      contract,
                                        char **         display,
                                        int *           width,
                                        int *           height,
                                        int *           xoffset,
                                        int *           yoffset,
                                        XtAppContext    app2run,
                                        int             ms_timeout);
```

ttdt_sender_imprint_on() 関数を呼び出すと、呼び出し側ツール (以下、ツール B とします) で別のツール (以下ツール A とします) の動作と特定の特性が採用されます。ツール B は、ツール A の X11 ディスプレイ、ロケール、現在の作業ディレクトリを採用します。さらに、自身を正しい位置に置くためにツール A の X11 ジオメトリを取得します。

display パラメータが NULL の場合、環境変数 $DISPLAY はツール A のディスプレイに設定されます。display パラメータが NULL でない場合、ツール A のディスプレイがこのパラメータに返されます。戻り値は、ToolTalk の tt_free() 関数で解放できる文字列です。

この関数は、Get_Geometry 要求をツール A に送信します。ツール A がジオメトリ・パラメータに対して値を返さない場合は、次のようになります。

- width パラメータに対して値が返されない場合、-1 を設定します。
- height パラメータに対して値が返されない場合、-1 を設定します。
- xoffset パラメータに対して値が返されない場合、INT_MAX を設定します。
- yoffset パラメータに対して値が返されない場合、INT_MAX を設定します。

ttdt_sender_imprint_on() 関数の width パラメータ、height パラメータ、xoffset パラメータ、および yoffset パラメータすべてに NULL を設定すると、Get_Geometry 要求はツール A に送信されません。

app2run パラメータと ms_timeout パラメータは、この関数が送信する Get_Geometry 要求への応答をブロックするために、tttk_block_while() 関数に渡されます。

表 A–15 は、この関数が返す可能性のあるエラーの一覧です。

表 A–15 ttdt_sender_imprint_on が返す可能性のあるエラー

エラーの値	説明
TT_DESKTOP_ETIMEDOUT	指定されたタイムアウト時間内に送信された要求の一部が完了しませんでした。
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。
TT_ERR_OVERFLOW	ToolTalk サービスが受信したメッセージの数が、正しく処理可能なアクティブ・メッセージの最大数 (2000) に達しました。

ttdt_session_join

```
Tt_message  ( *Ttdt_contract_cb) (  Tt_message          msg,
                                    void *              clientdata
                                    Tt_message          contract );
Tt_pattern  * ttdt_session_join (   const char *        sessid,
                                    Ttdt_session_cb     cb,
                                    Widget              shell,
                                    void *              clientdata,
                                    int                 join);
```

ttdt_session_join() 関数は、「デスクトップの良き市民」として ToolTalk セッションに参加します。つまり、セッション sessid への参加時に多くの標準デスクトップ・メッセージ・インタフェースのパターンとデフォルト・コールバックを登録します。表 A–16 は、この関数が現在登録しているメッセージ・インタフェースの一覧です。

表 **A–16** ttdt_session_join が登録している標準メッセージ

要求	処理方法
Get_Environment, Set_Environment	これらのメッセージは透過的に処理されます。
Get_Locale, Set_Locale	これらのメッセージは透過的に処理されます。
Get_Situation, Set_Situation	これらのメッセージは透過的に処理されます。
Signal	このメッセージは透過的に処理されます。
Get_Sysinfo	このメッセージは透過的に処理されます。
Get_Geometry, Set_Geometry	shell パラメータの値が NULL でなく、シェルが実体化された *mappedWhenManaged applicationShellWidget* の場合は、これらのメッセージは透過的に処理されます。シェルが *mappedWhenManaged applicationShellWidget* でない場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Get_Iconified, Get_Iconified	shell パラメータの値が NULL でなく、シェルが実体化された *mappedWhenManaged applicationShellWidget* の場合は、これらのメッセージは透過的に処理されます。シェルが *mappedWhenManaged applicationShellWidget* でない場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Get_Mapped, Set_Mapped	shell パラメータの値が NULL でなく、シェルが実体化された *mappedWhenManaged applicationShellWidget* の場合は、これらのメッセージは透過的に処理されます。シェルが *mappedWhenManaged applicationShellWidget* でない場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Raise	shell パラメータの値が NULL でなく、シェルが実体化された *mappedWhenManaged applicationShellWidget* の場合は、これらのメッセージは透過的に処理されます。シェルが *mappedWhenManaged applicationShellWidget* でない場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Lower	shell パラメータの値が NULL でなく、シェルが実体化された *mappedWhenManaged applicationShellWidget* の場合は、これらのメッセージは透過的に処理されます。シェルが *mappedWhenManaged applicationShellWidget* でない場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。

表 **A–16** ttdt_session_join が登録している標準メッセージ　(続き)

要求	処理方法
Get_XInfo	shell パラメータの値が NULL でない場合、これらのメッセージは透過的に処理されます。shell パラメータの値が NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Set_XInfo	shell パラメータの値が NULL でなく、シェルが実体化された *mappedWhenManaged applicationShellWidget* の場合は、これらのメッセージは透過的に処理されます。シェルが *mappedWhenManaged applicationShellWidget* でない場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Pause	cb パラメータが NULL でない場合は、このメッセージはコールバックに渡されます。cb パラメータが NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Resume	cb パラメータが NULL でない場合は、このメッセージはコールバックに渡されます。cb パラメータが NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Quit	cb パラメータが NULL でない場合は、このメッセージはコールバックに渡されます。cb パラメータが NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Get_Status	cb パラメータが NULL でない場合は、このメッセージはコールバックに渡されます。cb パラメータが NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。
Do_Command	cb パラメータが NULL でない場合は、このメッセージはコールバックに渡されます。cb パラメータが NULL の場合は、エラー TT_DESKTOP_ENOTSUP とともに異常終了します。

sessid パラメータが NULL の場合は、デフォルト・セッションに参加します。

join パラメータが設定されている場合は、指定のセッションに参加します。

Ttdt_contract_cb メッセージは、表 A–17 に示すパラメータを取ります。コールバックはメッセージを処理しない場合は、メッセージを返します。メッセージを処理する場合は、ゼロあるいは Tt_message に送られたエラー・ポインタを返します。

表 **A–17** Ttdt_session_cb が取るパラメータ

パラメータ	説明
Tt_message msg	送信状態にある要求。クライアント・プログラムは、この要求を無視または拒否するか、メッセージに応答します。 注 – 処理後にメッセージ msg を破棄してください。
void *clientdata	ttdt_session_join() または ttdt_message_accept() 関数のどちらかに渡されるクライアント・データ

表 **A–17** Ttdt_session_cb が取るパラメータ　(続き)

パラメータ	説明
Tt_messagecontract	ttdt_message_accept() 関数に渡されるコントラクト。コールバックが ttdt_session_join() 関数によってインストールされると、contract パラメータの値は必ずゼロになります。

ttdt_session_join() 関数は、NULL で終了する Tt_pattern 型の配列を返します。この配列は、ttdt_session_quit() 関数に渡され破棄できます。エラーが発生すると、返された配列はエラー・ポインタになります。Tt_status を調べるには、tt_ptr_error を使用します。表 A–18 は、返される可能性のあるエラーの一覧です。

表 **A–18** ttdt_session_join が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_SESSION	旧式、または無効な ToolTalk セッションが指定されました。
TT_ERR_POINTER	渡されたポインタが、このオペレーションに適した型のオブジェクトを指していません。たとえば、文字列が必要なときにポインタは整数を指している場合などです。
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。

ttdt_session_quit

```
Tt_status   ttdt_session_quit(    const char *      sessid,
                                  Tt_pattern *      sess_pats,
                                  int               quit );
```

ttdt_session_quit() 関数は、「デスクトップの良き市民」として ToolTalk セッションを終了します。つまり、セッションへの参加時に登録したすべてのパターンとデフォルト・コールバックを登録解除します。

この関数は、sess_pats で指定されたすべてのパターンを破棄します。quit パラメータが設定されると、セッション sessid を終了します。sessid パラメータが NULL の場合は、デフォルト・セッションを終了します。

表 A–19 は、この関数が返す可能性のあるエラーの一覧です。

表 **A–19** ttdt_session_quit が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_SESSION	旧式、または無効な ToolTalk セッションが指定されました。
TT_ERR_POINTER	渡されたポインタが、このオペレーションに適した型のオブジェクトを指していません。たとえば、文字列が必要なときにポインタは整数を指している場合などです。

ttdt_subcontract_manage

```
Tt_pattern *ttdt_subcontract_manage(    Tt_message              contract,
                                        Ttdt_contract_cb        cb,
                                        Widget                  shell,
                                        void *                  clientdata);
```

ttdt_subcontract_manage() 関数は、未処理の要求を管理します。この関数が呼び出されると、要求側ツールは要求を処理しているツールとの間で行われるデスクトップ上の標準的な対話を管理できます。この関数は、TT_HANDLER を宛先とする Get_Geometry 要求と Get_XInfo 要求、および Status 通知のための登録をデフォルト・セッションで行います。

shell パラメータが NULL の場合は、要求または通知が cb パラメータに渡されます。shell パラメータが NULL でない場合は、要求は透過的に処理されます。

ttdt_subcontract_manage() 関数は、NULL で終了する Tt_pattern 型の配列を返します。この配列は、ttdt_session_quit() 関数に渡して破棄できます。エラーが発生すると、返された配列はエラー・ポインタになります。Tt_status を調べるには、tt_ptr_error を使用します。表 A–20 は、返される可能性のあるエラーの一覧です。

表 **A–20** ttdt_subcontract_manage が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。

表 **A–20** ttdt_subcontract_manage が返す可能性のあるエラー　(続き)

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_POINTER	subcontract パラメータが有効な Tt_message ではありません。
TT_ERR_EINVAL	shell パラメータと cb パラメータの両方が NULL です。

ttmedia_Deposit

```
Tt_status   ttmedia_Deposit(    Tt_message              load_contract,
                                const char *            buffer_id,
                                const char *            media_type,
                                const unsigned char *   new_contents,
                                int                     new_len,
                                const char *            pathname,
                                XtAppContext            app2run,
                                int                     ms_timeout );
```

ttmedia_Deposit 関数は Deposit 要求を送信して、Edit、Compose、Open などの Media Exchange load_contract 要求の変換対象であったドキュメントにチェックポイントを設定します。

この関数は Deposit 要求を作成して送信し、その要求が成功したか失敗したかを返します。

- load_contract は、このエディタがドキュメントを読み込むことを要求します。
- buffer_id は、ドキュメントが Open 要求により読み込まれた場合に、このエディタが作成したバッファの ID です。
- media_type は、送信された要求の contents 引き数の vtype です。
- new_contents と new_len は、contents 引き数の値です。

要求が送信されると、app2run と ms_timeout は、応答を待つために、tttk_block_while() 関数に渡されます。

表 **A–21** ttmedia_Deposit が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。
TT_ERR_OVERFLOW	ToolTalk サービスが受信したメッセージの数が、正しく処理可能なアクティブ・メッセージの最大数 (2000) に達しました。
TT_ERR_DBAVAIL	ToolTalk サービスが、このオペレーションに必要な ToolTalk データベースにアクセスできませんでした。
TT_ERR_DBEXIST	ToolTalk サービスが、指定された ToolTalk データベースを予期した場所で見つけることができませんでした。
TT_DESKTOP_ETIMEOUT	指定されたタイムアウト時間内に応答を受信しませんでした。
TT_ERR_POINTER	パス名が NULL、または ToolTalk エラー・ポインタです。

ttmedia_load

```
Tt_message  (*Ttmedia_load_msg_cb)( Tt_message              msg,
                                    void *                  clientdata,
                                    Tttk_op                 op,
                                    unsigned char *         contents,
                                    int                     len,
                                    char *                  file  );
Tt_message  ttmedia_load(   Tt_message              context,
                            Ttmedia_load_msg_cb     cb,
                            void *                  clientdata,
                            Tttk_op                 op,
                            const char *            media_type,
                            const unsigned char*    contents,
                            int                     len,
                            const char *            file,
                            const char *            docname,
                            int                     send );
```

ttmedia_load() 関数は、ドキュメントの表示、編集、作成を行う Media Exchange 要求を作成し、必要に応じて送信します。この関数は、Display、Edit、Compose のいずれかの要求を作成して送信します。

注 – 要求のハンドラとの標準的な対話を管理するには、このメッセージで作成した要求の送信後すぐに ttdt_subcontract_manage() 関数を使用してください。

context 引き数の値がゼロ以外のとき、このルーチンによって作成されるメッセージは、スロット名が ENV_ で始まるすべてのコンテキストを継承します。

clientdata 引き数は、応答を受信する場合、またはドキュメントの中間のバージョンが Deposit 要求によってチェックポイントを設定される場合に cb 引き数に渡されます。

op 引き数は、TTME_DISPLAY、TTME_EDIT、TTME_COMPOSE のいずれかでなければなりません。

media_type 引き数は、ドキュメントのデータ形式に名前を付けます。通常、この引き数により、要求を処理するためにどのアプリケーションを選択するか決定します。

contents 引き数と len 引き数はドキュメントを指定します。これら両方の引き数の値がゼロで、file 引き数の値がゼロでない場合、ドキュメントは指定のファイルに格納されていると想定されます。

docname 引き数が NULL でないとき、その値はドキュメントのタイトルとして使用されます。

send 引き数が true のとき、メッセージは返される前に送信されます。

表 A–22 は、Ttmedia_load_msg_cb メッセージが取るパラメータの一覧です。

表 **A–22** Ttmedia_load_msg_cb が取るパラメータ

パラメータ	説明
Tt_message msg	要求に対する応答、または読み込み要求の tt_message_id を指定する messageID 引き数が指定された Deposit 要求 このパラメータの値が Deposit 要求である場合、クライアント・プログラムは要求に応答するか、無視しなければなりません。 注 – メッセージ msg は処理後に破棄してください。
Tttk_op op	メッセージのオペレーション (TTME_DEPOSIT あるいは ttmedia_load() メッセージに渡されたオペレーション)
unsigned char * contents int len char *file	到着中のドキュメントの内容 len 引き数がゼロのとき、ドキュメントは指定のファイルに格納されています。contents 引き数または file 引き数が NULL でない場合は、ToolTalk 関数 tt_free() を使用してそれらの引き数を解放してください。

表 **A–22** Ttmedia_load_msg_cb が取るパラメータ　*(続き)*

パラメータ	説明
void *clientdata	ttmedia_load() メッセージに渡されたクライアント・データ

メッセージが正常に処理されると、コールバックはゼロを返します。処理中にエラーが発生した場合は、コールバックは Tt_message に伝えられたエラー・ポインタを返します。

コールバックがメッセージ msg を処理しない場合、コールバックはメッセージを返し、TT_CALLBACK_CONTINUE ルーチンを呼び出しスタックに渡してメッセージを他のコールバックに提供するか、メッセージを tt_message_receive() 関数に返します。

終了時に、ttmedia_load() 関数は、構築するように要求された要求を返します。エラーが発生した場合、この関数はエラー・ポインタを返します。Tt_status を調べるには、tt_ptr_error を使用します。表 A–23 は、返される可能性のあるエラーの一覧です。

表 **A–23** ttmedia_load が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。
TT_ERR_OVERFLOW	ToolTalk サービスが受信したメッセージの数が、正しく処理可能なアクティブ・メッセージの最大数 (2000) に達しました。

ttmedia_load_reply

```
Tt_message      ttmedia_load_reply (Tt_message              contract,
          const unsigned char *   new_contents,
          int                     new_len,
          int                     reply_and_destroy);
```

ドキュメントの表示、編集、作成を行う Media Exchange 要求に対して応答するには、ttmedia_load_reply() 関数を使用します。

new_contents 引き数と new_len 引き数の両方がゼロでない場合、それらの値は contract 引き数の適切な出力引き数にドキュメントの新しい内容を設定するのに使用されます。reply_and_destroy 引き数が true の場合、contract 引き数に対して応答が行われ、その後メッセージは破棄されます。

表 A–24 は、返される可能性のあるエラーの一覧です。

表 **A–24** ttmedia_load_reply が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_NUM	
TT_ERR_NOTHANDLER	

ttmedia_ptype_declare

```
Tt_message      (*Ttmedia_load_pat_cb)( Tt_message          msg,
                                        void *              clientdata,
                                        Tttk_op             op,
                                        Tt_status           diagnosis,
                                        unsigned char *     contents,
                                        int                 len,
                                        char *              file,
                                        char *              docname );
Tt_status       ttmedia_ptype_declare(  const char *        ptype,
                                        int                 base_opnum,
                                        Ttmedia_load_pat_cb cb,
                                        void *              clientdata,
                                        int                 declare );
```

ttmedia_ptype_declare() 関数は、Media Exchange メディア・エディタの ptype を宣言します。この関数は、特定のメディア型用の Media Exchange メッセージ・インタフェースを実装するエディタを初期化します。

- エディタが ptype でサポートされる種類のドキュメントを編集するように要求される場合、この関数は cb 引き数を呼び出します。
- この関数は、ptype が含まれていると想定される一連のシグニチャーに、ツールキット内部のオペレーション番号 (opnum) コールバックをインストールします。ツールキット内部の opnum コールバックは、これらのシグニチャーのいずれかに一致する要求を受信すると、clientdata を cb 引き数に渡します。opnum は、

base_opnum から始まり、ゼロか 1000 の倍数でなければなりません。

- declare 引き数が true のとき、この関数は次を呼び出します。

```
tt_ptype_declare(    ptype )
```

ptype が複数の異なるメディア型を実現する場合、ttmedia_ptype_declare() 関数を 2 回以上呼び出すことができます。base_opnum には、各呼び出しで異なる値を指定しなければなりません。

注 – ttmedia_ptype_declare() 関数は何回も呼び出すことができますが、declare 引き数を「true」に設定できるのは 1 回だけです。

表 A–25 は、Ttmedia_load_pat_cb メッセージが取るパラメータの一覧です。

表 **A–25** Ttmedia_load_pat_cb メッセージが取るパラメータ

パラメータ	説明
Tt_message msg	送信された要求 クライアント・プログラムは、この要求を無視または拒否する、あるいは要求に応答しなければなりません。
Tttk_op op	着信中の要求のオペレーション (TTME_COMPOSE、TTME_EDIT、TTME_DISPLAY のいずれか)
Tt_status diagnosis	ツールキットが要求を無視するように勧告するときに使用するエラー・コード (たとえば、TT_DESKTOP_ENODATA) diagnosis が TT_OK ではなく、コールバック・ルーチンがメッセージ msg を返す場合、ツールキットはメッセージ msg を無視し破棄します。
unsigned char * contents int len char *file	到着中のドキュメントの内容 len 引き数がゼロのとき、ドキュメントは指定のファイルに格納されています。contents 引き数または file 引き数が NULL でない場合は、ToolTalk 関数 tt_free を使用して、それらの引き数を解放します。
char * docname	ドキュメントに名前が付けられている場合はその名前
void * clientdata	ttmedia_ptype_declare() メッセージに渡すクライアント・データ

メッセージが正常に処理されると、コールバックはゼロを返します。処理中にエラーが発生した場合は、コールバックは Tt_message に伝えられたエラー・ポインタを返します。

コールバックがメッセージ msg を処理せず、diagnosis 引き数の値が TT_OK でない場合、コールバックはメッセージを返し、ツールキットは TT_CALLBACK_CONTINUE ルーチンを呼び出しスタックに渡してメッセージを他のコールバックに提供するか、メッセージを tt_message_receive() 呼び出しに返します。

エラーが発生すると、この関数は表 A–26 にあるエラーのうちのいずれかを返します。

表 **A–26** ttmedia_ptype_declare が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_PTYPE	ToolTalk サービスが、指定された ptype を検出することができません。
TT_ERR_POINTER	渡されたポインタが、このオペレーションに適した型のオブジェクトを指していません。たとえば、文字列が必要なときにポインタは整数を指している場合などです。

tttk_block_while

```
Tt_status       tttk_block_while(   const int       *blocked,
                                    int             ms_timeout);
```

tttk_block_while 関数は、ms_timout で指定されたタイムアウト時間中にプログラムが応答を待っている間プログラムをブロックします。

tttk_message_abandon

```
Tt_status       tttk_message_abandon (  Tt_message      msg );
```

tttk_message_abandon() 関数は、要求を放棄した後に破棄します。

注 – プログラムは、メッセージを理解できずに処分したいときはメッセージを放棄しなければなりません。

エラーが発生すると、この関数は表 A–27 にあるエラーのうちのいずれかを返します。

表 **A–27** tttk_message_abandon が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_POINTER	渡されたポインタが、このオペレーションに適した型のオブジェクトを指していません。たとえば、文字列が必要なときにポインタは整数を指している場合などです。
TT_ERR_NOTHANDLER	

tttk_message_create

```
Tt_message  tttk_message_create(    Tt_message                  context,
                                    Tt_class                    the_class,
                                    Tt_scope                    the_scope,
                                    const char *                handler,
                                    const char *                op,
                                    Tt_message_callback         callback);
```

tttk_message_create() 関数は、規約に従ったメッセージを作成します。この関数を使用すれば、継承されたコンテキストをあるメッセージから別のメッセージに伝達するメッセージを簡単に作成できます。

tttk_message_create() 関数はメッセージを作成し、スロット名が ENV_ で始まる context からすべてのコンテキスト・スロットを新しいメッセージ上にコピーします。作成されたメッセージには、the_class パラメータに指定された Tt_class 値と the_scopee パラメータに指定された Tt_scope 値が設定されます。

handler パラメータが NULL の場合、メッセージには TT_PROCEDURE の Tt_address が設定されます。handler パラメータが NULL でない場合は、メッセージは TT_HANDLER を経由して procid に送信されます。

op 引き数が NULL でない場合、その値にはメッセージの op 引き数が設定されます。

コールバック引き数が NULL でない場合、その値がメッセージ・コールバックとしてメッセージに追加されます。

正常終了時には、tttk_message_create() 関数は作成した Tt_message を返します。これは、他の Tt_message と同じ方法で変更、送信、および破棄できます。

エラーが発生すると、エラー・ポインタが返されます。Tt_status を調べるには、tt_ptr_error を使用します。表 A–28 は、返される可能性のあるエラーの一覧です。

表 **A–28** tttk_message_create が返す可能性のあるエラー

エラーの値	説明
TT_ERR_NOMP	ttsession プロセスを使用できません。ToolTalk サービスは、ttsession が実行されていない場合はその再起動を試みます。このエラーは、ToolTalk サービスがインストールされていないか、正しくインストールされていないかのどちらかを示します。
TT_ERR_PROCID	指定されたプロセス識別子が旧式か、無効です。
TT_ERR_NOMEM	オペレーションを実行するのに十分なメモリがありません。

tttk_message_destroy

```
Tt_status  tttk_message_destroy    (    Tt_message      msg );
```

tttk_message_destroy() 関数は、規約に従うすべてのメッセージを破棄します。

注 – このメッセージは、tt_message_destroy() メッセージの代わりに使用できます。

---

tttk_message_destroy() 関数は、ttdt_message_accept() 関数あるいは ttdt_subcontract_manage() 関数によってメッセージに格納されたパターンを破棄し、その後メッセージ msg を tt_message_destroy() 関数に渡します。

この関数は、tt_message_destroy() 関数によって返された値を返します。

tttk_message_fail

```
Tt_status   tttk_message_fail(      Tt_message              msg,
                                    Tt_status               status,
                                    const char              *status_string,
                                    int                     destroy );
```

tttk_mesage_fail() 関数は、メッセージ msg を無視した後で破棄します。

注 – プログラムは、メッセージを理解できずに処分したいときは、メッセージを放棄しなければなりません。

状態が TT_SENT であるメッセージは無視できます。メッセージが、ハンドラが宛先指定されているメッセージ、あるいは TT_WRN_START_MESSAGE の tt_message_status を持つ場合、そのメッセージを無視できます。

この関数は、TT_DESKTOP_ENOTSUP を返します。

tttk_message_receive

```
Tt_status   tttk_message_receive(   const char*     procid );
```

tttk_message_receive() 関数は、tt_message_receive() 関数を呼び出し、次の ToolTalk メッセージを取り出します。

procid がゼロでない場合、この関数は次を呼び出します。

```
    tt_default_procid_set( procid )
```

tttk_message_reject

```
Tt_status   tttk_message_reject(    Tt_message              msg,
                                    Tt_status               status,
                                    const char*             status_string,
                                    int                     destroy );
```

tttk_message_reject() 関数は、メッセージ msg を拒否した後で破棄します。

注 – プログラムは、メッセージを理解できずに処分したいときは、メッセージを放棄しなければなりません。

状態が TT_SENT であるメッセージは拒否できます。メッセージが、ハンドラが宛先指定されているメッセージでない、あるいは TT_WRN_START_MESSAGE 以外の tt_message_status を持つ場合、そのメッセージを拒否できます。

この関数は、TT_DESKTOP_ENOTSUP を返します。

tttk_op_string

```
char            *tttk_op_string(    Tttk_op                     op);
```

tttk_op_string() 関数は、正常終了する場合はオペレーション op に対する文字列を返します。そうでない場合、この関数はゼロを返します。

注 – 返された文字列を解除するには、tt_free() 関数を使用してください。

```
Tttk_op         tttk_string_op( const char *        opstring );
```

tttk_string_op() 関数は、指定の文字列に対するオペレーションを保持している文字列を返します。エラーが発生すると、この関数は TTDT_OP_NONE を返します。

tttk_Xt_input_handler

```
void        tttk_Xt_input_handler(  XtPointer       procid,
                                    int *           source,
                                    XtInputId *     id );
```

tttk_Xt_input_handler() 関数は、Xt クライアントのための ToolTalk イベントを処理します。一部のメッセージがコールバックで処理されないと予期される場合を除き、この関数を Xt 入力ハンドラとして使用します。

この関数は、procid 引き数を tttk_message_receive() 関数に渡し、すべての返されたメッセージ (つまり、コールバックで処理されないメッセージ) を tttk_message_abandon() 関数に渡します。

この関数がエラー TT_ERR_NOMP を返すと、tttk_Xt_input_handler() 関数は id パラメータを XtRemoveInput() 関数に渡します。

付録 B

CoEd デモンストレーション・プログラム

この付録には、CoEd と呼ばれる ToolTalk デモンストレーション・プログラムのための ToolTalk 関連コードを示すファイルとソースコードの一覧があります。CoEd デモ・プログラムは、ToolTalk デスクトップ・サービス・メッセージ・セットを使用します。エディタの複数のインスタンスが同時に同じファイルを編集しているときに、エディタが ToolTalk サービスを使用して、ユーザが加えたすべての変更の同期を取る方法について説明します。

CoEd ptype ファイル

CoEd ptype ファイルについて、例 B–1 に示します。

例 **B–1** CoEd ptype ファイル

```
ptype DT_CoEd {                                    /* Process type identifier */
    start   "CoEd;                                 /* Start string */
    handle:                                        /* Receiving process */
    /*
     * Display ISO_Latin_1
     */
    session     Display( in     ISO_Latin_1 contents) => start opnum = 1;
                                                   /* Signature */
    session     Display( in     ISO_Latin_1 contents,
                         in     messageID   counterfoil) => start opnum = 2;
    session     Display( in     ISO_Latin_1 contents,
                         in     title       docName) => start opnum = 3;
    session     Display( in     ISO_Latin_1 contents,
                         in     messageID   counterfoil,
                         in     title       docName) => start opnum = 4;
    /*
     * Edit ISO_Latin_1
     */
    session     Edit(    inout  ISO_Latin_1   contents) => start opnum = 101;
```

例 **B–1** CoEd ptype ファイル　(続き)

```
    session     Edit(    inout  ISO_Latin_1   contents,
                         in     messageID     counterfoil) => start opnum = 102;
    session     Edit(    inout  ISO_Latin_1   contents,
                         in     title         docName) => start opnum = 103;
    session     Edit(    inout  ISO_Latin_1   contents,
                         in     messageID     counterfoil,
                         in     title         docName) => start opnum = 104;
    /*
     * Compose ISO_Latin_1
     */
    session     Edit(    out    ISO_Latin_1   contents) => start opnum = 201;
    session     Edit(    out    ISO_Latin_1   contents,
                         in     messageID     counterfoil) => start opnum = 202;
    session     Edit(    out    ISO_Latin_1   contents,
                         in     title         docName) => start opnum = 203;
    session     Edit(    out    ISO_Latin_1   contents,
                         in     messageID     counterfoil,
                         in     title         docName) => start opnum = 204;
    /*
     * Open an ISO_Latin_1 buffer
     */
    session     Open(    in     ISO_Latin_1   contents,
                         out    bufferID      docBuf,
                         in     boolean       readOnly  ) => start opnum = 400;
    session     Open(    in     ISO_Latin_1   contents,
                         out    bufferID      docBuf,
                         in     boolean       readOnly,
                         in     boolean       mapped    ) => start opnum = 401;
    session     Open(    in     ISO_Latin_1   contents,
                         out    bufferID      docBuf,
                         in     boolean       readOnly,
                         in     boolean       mapped,
                         in     integer       shareLevel) => start opnum = 402;
    session     Open(    in     ISO_Latin_1   contents,
                         out    bufferID      docBuf,
                         in     boolean       readOnly,
                         in     boolean       mapped,
                         in     integer       shareLevel,
                         in     locator       initialPos) => start opnum = 403;
};
```

CoEd.C ファイル

例 B–2 に示す `CoEd.C` ファイルは、各アプリケーションに組み込む必要がある ToolTalk コードを示します。これにより、ツールキットの初期化、ToolTalk セッションへの参加、パターンの登録、ToolTalk サービスのイベント・ループへの追加を実行できます。

注 – このファイルには、エディタ・アプリケーションとして機能する場合の CoEd に固有の ToolTalk コードも入っています。このコードには、ptype の宣言と起動メッセージの処理も含まれています。

例 **B–2** CoEd.C ファイル

```
/*
 * CoEd.cc
 *
 * Copyright (c) 1991,1993 by Sun Microsystems.
 */

#include <stdlib.h>
#include <desktop/tttk.h>          // Include the ToolTalk messaging toolkit
#include <CoEd.h>
#include "CoEditor.h"
#include "CoEdTextBuffer.h"

XtAppContext    myContext;
Widget          myTopWidget = 0;
Display         *myDpy;
int             abortCode   = 0;
Tt_pattern      *sessPats   = 0;         // Patterns returned when session joined
int             timeOutFactor   = 1000;
int             maxBuffers      = 1000;
int             *pArgc;
char            **globalArgv;

const char      *ToolName = "CoEd";
const char      *usage    =
"Usage: CoEd [-p01] [-w n] [-t n] [file]\n"
" -p        print ToolTalk procid\n"
" -0        do not open an initial composition window\n"
" -1        be a single-buffer editor\n"
" -w        sleep for n seconds before coming up\n"
" -t        use n as timeout factor, in milliseconds (default: 1000)\n"
;

void main(  int     argc,
            char    **argv
)
{
    static const char *here = "main()";

    int     delay   = 0;
    int     printid = 0;
    int     compose = 1;
    char    *file   = 0;

    OlToolkitInitialize( 0 );
    XtToolkitInitialize();
    myContext = XtCreateApplicationContext();
```

例 **B–2** CoEd.C ファイル　(続き)

```
    //
    // This display may get closed, and another opened, inside
    // CoEditor::_init(), if e.g. our parent is on a different screen
    //
    pArgc = &argc;
    globalArgv = argv;
    myDpy = XtOpenDisplay( myContext, 0, 0, "CoEd", 0, 0, &argc, argv );
    int c;
    while ((c = getopt( argc, argv, "p01w:t:" )) != -1) {
        switch (c) {
        case 'p':
          printid = 1;
          break;
        case '0':
          compose = 0;
          break;
        case '1':
           maxBuffers = 1;
           break;
        case 'w':
           delay = atoi( optarg );
           break;
        case 't':
           timeOutFactor = atoi( optarg );
           break;
        default:
           fputs( usage, stderr );
           exit( 1 );
        }
}
if (optind < argc) {
    file = argv[ optind ];
}
while (delay > 0) {
    sleep( 1 );
    delay--;
}
int myTtFd;                                 // Obtain process identifier
// Initialize toolkit and create a ToolTalk communication endpoint
char *myProcID = ttdt_open( &myTtFd, ToolName, "SunSoft", "%I", 1 );

// Declare ptype
ttmedia_ptype_declare( "DT_CoEd", 0, CoEditor::loadISOLatin1_,
                       (void *)&myTopWidget, 1 );

// Process the message that started us, if any
tttk_Xt_input_handler( 0, 0, 0 );
if (abortCode != 0) {
    // Error in message that caused us to start.
    exit( abortCode );
}

if (CoEditor::numEditors == 0) {
```

例 **B–2** CoEd.C ファイル　(続き)

```
    // started by hand, not by ToolTalk
    if (file == 0) {
      if (compose) {
          new CoEditor( &myTopWidget );
      }
    } else {
      new CoEditor( &myTopWidget, file );
    }
}
//
// If sessPats is unset, then we have not joined the desktop
// session yet.     So join it.
//
if (sessPats == 0) {
    Widget session_shell = CoEditor::editors[0]->shell;
    if (maxBuffers > 1) {
        //
        // In multi-window mode, no single window is the
        // distinguished window.
        //
        session_shell = myTopWidget;
    }
    sessPats = ttdt_session_join( 0, 0, session_shell, 0, 1 );
}

    XtAppAddInput( myContext, myTtFd,(XtPointer)XtInputReadMask,
                   tttk_Xt_input_handler, myProcID );
    XtAppMainLoop( myContext );
}
```

Coeditor.C ファイル

例 B–3 に示す `Coeditor.C` ファイルは、各エディタ・アプリケーションに組み込む必要がある ToolTalk コードを示します。これにより、メディア・コールバックを渡し、要求の完了時に応答できます。エディタ・アプリケーションに組み込むことができるその他のオプションの ToolTalk コードも示します。

注 **–** 省略符号 (...) は、省略されたコードを示します。

例 **B–3** CoEditor.C ファイル

```
...

CoEditor::CoEditor(
```

例 **B–3** CoEditor.C ファイル　*(続き)*

```
    Widget *parent
)
{
    _init();
    _init( parent );
}

CoEditor::CoEditor(
    Widget          *parent,
    const char      *file
)
{
    _init();
    _init( parent );
    _load( file );
}

CoEditor::CoEditor(
    Widget          *parent,
    Tt_message      msg,
    const char      *           /*docname*/,
    Tt_status       &status
)
{
    _init();
    status = _init( msg );
    if (status != TT_OK) {
            return;
    }
    _init( parent );
    status = _acceptContract( msg );
}

CoEditor::CoEditor(
    Widget          *parent,
    Tt_message      msg,
    int                         /*readOnly*/,
    const char      *file,
    const char      *           /*docname*/,
    Tt_status       &status
)
{
    _init();
    status = _init( msg );
    if (status != TT_OK) {
            return;
    }
    _init( parent );
    status = _load( file );
    if (status != TT_OK) {
            return;
    }
    status = _acceptContract( msg );
```

例 **B–3** CoEditor.C ファイル　（続き）

```
}

CoEditor::CoEditor(
    Widget          *parent,
    Tt_message      msg,
    int                         /*readOnly*/,
    unsigned char   *contents,
    int                         /*len*/,
    const char      *         /*docname*/,
    Tt_status       &status
)
{
    _init();
    status = _init( msg );
    if (status != TT_OK) {
            return;
    }
    _init( parent );
    XtVaSetValues( (Widget)_text,
                   XtNsourceType,       (XtArgVal)OL_STRING_SOURCE,
                   XtNsource,           (XtArgVal)contents,
                   NULL );
    _textBuf  = OlTextEditTextBuffer( _text );
    RegisterTextBufferUpdate( _textBuf, CoEditor::_textUpdateCB_,
                              (caddr_t)this );
    status = _acceptContract( msg );
}

CoEditor::~CoEditor()
{
    //
    // No need for a separate save if we are sending the document
    // back in a reply.
    //
    if (_contract == 0) {
        if (_modifiedByMe) {
            // we revert before quitting if we don't want to save
            _save();
        }
    } else {
        int     len;
        char *contents = _contents( &len );

        // Reply to media load callback with edited contents of text
        ttmedia_load_reply( _contract, (unsigned char *)contents,
                            len, 1 );
        if (contents != 0) {
            XtFree( contents );
        }
        _contract = 0;
   }
   numEditors--; // XXX assumes user destroys windows LIFO!
}
```

例 **B–3** CoEditor.C ファイル　*(続き)*

```
Tt_message
CoEditor::loadISOLatin1_(
    Tt_message          msg,
    Tttk_op             op,
    Tt_status           diagnosis,
    unsigned char       *contents,
    int                 len,
    char                *file,
    char                *docname,
    void                *pWidget
)
{
    static const char *here = "CoEditor::loadISOLatin1_()";

    Tt_status status    = TT_OK;
    CoEditor *coEditor  = 0;
    if (diagnosis != TT_OK) {
       // toolkit detected an error
       if (tt_message_status( msg ) == TT_WRN_START_MESSAGE) {
       //
       // Error is in start message!  We now have no
       // reason to live, so tell main() to exit().
       //
           abortCode = 2;
       }

       // let toolkit handle the error
       return msg;
    }
    if ((op == TTME_COMPOSE) & (file == 0)) {
        oEditor = new CoEditor( (Widget *)pWidget, msg, docname, status );

    } else if (len > 0) {
        coEditor = new CoEditor( (Widget *)pWidget, msg,
                                 (op == TTME_DISPLAY),
                                 contents, len, docname, status );
    } else if (file != 0) {
        coEditor = new CoEditor( (Widget *)pWidget, msg,
                                 (op == TTME_DISPLAY),
                            file, docname, status );
    } else {
        // Fail a message
        tttk_message_fail( msg, TT_DESKTOP_ENODATA, 0, 1 );
    }
    tt_free( (caddr_t)contents );
    tt_free( file );
    tt_free( docname );
    return 0;
}

void
CoEditor::_init()
```

例 **B–3** CoEditor.C ファイル　(続き)

```
{
    _baseFrame          = 0;
    _controls           = 0;
    _fileBut            = 0;
    _editBut            = 0;
    _scrolledWin        = 0;
    _text               = 0;
    _textBuf            = 0;
    _modifiedByMe       = FALSE;
    _modifiedByOther    = 0;
    _contract           = 0;
    _contractPats       = 0;
    _filePats           = 0;
    _file               = 0;
    _x                  = INT_MAX;
    _y                  = INT_MAX;
    _w                  = INT_MAX;
    _h                  = INT_MAX;
}

Tt_status
CoEditor::_init(
    Tt_message msg
)
{
    int width, height, xOffset, yOffset;

    width = height = xOffset = yOffset = INT_MAX;
    _contract = msg;
    ttdt_sender_imprint_on( 0, msg, 0, &_w, &_h, &_x, &_y,
                            10 * timeOutFactor );
    return TT_OK;
}

typedef enum {
    Open,
    Save,
    SaveAs,
    Revert
} FileOp;

static const char *fileButs[] = {
    "Open...",
    "Save",
    "Save as...",
    "Revert",
};

const int numFileButs = sizeof( fileButs ) / sizeof( const char * );

typedef enum {
    Undo,
    Cut,
```

例 **B–3** CoEditor.C ファイル　*(続き)*

```
    Copy,
    Paste,
    Delete,
    SelText,
    SelAppt
} EditOp;

static const char *editButs[] = {
    "Undo",
    "Cut",
    "Copy",
    "Paste",
    "Delete",
    "Text as ISO_Latin_1",
    "Text as Appointment";
};

const int numEditButs = sizeof( editButs ) / sizeof( const char * );

void
CoEditor::_init(
    Widget *parent
)
{
    if (*parent == 0) {
        if (_contract != 0) {
        //
        // Re-open display, since $DISPLAY may have changed by
        // ttdt_sender_imprint_on().
        //
          XtCloseDisplay( myDpy );
          myDpy = XtOpenDisplay( myContext, 0, 0, "CoEd", 0, 0,
                                 pArgc, globalArgv );
        }
        *parent = XtAppCreateShell( 0, "CoEd",
                                    applicationShellWidgetClass, myDpy, 0, 0 );
        XtVaSetValues( *parent,
                       XtNmappedWhenManaged, False,
                       XtNheight, 1,
                       XtNwidth, 1,
                       0 );
        XtRealizeWidget( *parent );
    }
    shell = XtCreatePopupShell( "CoEd",
                                applicationShellWidgetClass, *parent, 0, 0 );
    XtVaSetValues( shell, XtNuserData, this, 0 );
    // Pop up next to our parent
    if ((_x != INT_MAX) && (_y != INT_MAX) && (_w != INT_MAX)) {
        // XXX Be smarter about picking a geometry
        Dimension x       = _x + _w;
        Dimension y       = _y;
        XtVaSetValues( shell, XtNx, x, XtNy, y, 0 );
    }
```

例 **B–3** CoEditor.C ファイル　(続き)

```
    XtAddCallback( shell, XtNdestroyCallback, CoEditor::_destroyCB_, this );

    OlAddCallback( shell, XtNwmProtocol, CoEditor::_wmProtocolCB_, this );
    _baseFrame = XtVaCreateManagedWidget( "baseFrame", rubberTileWidgetClass,
                                          shell, 0 );

    _controls = XtVaCreateManagedWidget( "controls",
                                         controlAreaWidgetClass, _baseFrame,
                                         XtNweight, (XtArgVal)0, 0 );
    _fileBut = XtVaCreateManagedWidget( "File", menuButtonWidgetClass,
                                        _controls, 0 );
    Widget menuPane;
    XtVaGetValues( _fileBut, XtNmenuPane, &menuPane, 0 );
    for (int i = 0; i < numFileButs; i++) {
        Widget but = XtVaCreateManagedWidget( fileButs[i],
                                              oblongButtonWidgetClass,
                                              menuPane, XtNuserData, i, 0 );
        XtAddCallback( but, XtNselect, CoEditor::_fileButsCB_, this );
    }
    _editBut = XtVaCreateManagedWidget( "Edit", menuButtonWidgetClass,
                                        _controls, 0 );
    XtVaGetValues( _editBut, XtNmenuPane, &menuPane, 0 );
    for (i = 0; i < numEditButs; i++) {
        Widget but = XtVaCreateManagedWidget( editButs[i],
                                              oblongButtonWidgetClass,
                                              menuPane, XtNuserData, i, 0 );
         XtAddCallback( but, XtNselect, CoEditor::_editButsCB_, this );
    }
    _scrolledWin = XtVaCreateManagedWidget(
                   "scrolledWin", scrolledWindowWidgetClass,
                   _baseFrame,
                   XtNforceVerticalSB,(XtArgVal)True,
                   0 );
    _text = (TextEditWidget)XtVaCreateManagedWidget(
                   "text", textEditWidgetClass, _scrolledWin,
                   0 );
    XtVaSetValues( (Widget)_text, XtNuserData, this, 0 );
    XtRealizeWidget( shell );
    XtPopup( shell, XtGrabNone );
    if (numEditors < MaxEditors) {
        editors[ numEditors ] = this;
        numEditors++;
    }
    if (numEditors >= maxBuffers) {
        tt_ptype_undeclare( "DT_CoEd" );
    }
}
Tt_status
CoEditor::_unload()
{
    Tt_status status = TT_OK;
    if (_filePats != 0) {
        // Unregister interest in ToolTalk events and destroy patterns
```

例 **B–3** CoEditor.C ファイル　*(続き)*

```
        status = ttdt_file_quit( _filePats, 1 );
        _filePats = 0;
    }
    if (_file != 0) {
        free( _file );
        _file = 0;
    }

    return status;
}

Tt_status
CoEditor::_load(
    const char *file
)
{
    int reloading = 1;
    if (file != 0) {
        if ((_file != 0) && (strcmp( file, _file ) != 0)) {
            reloading = 0;
            _unload();
        } else {
            _file = strdup( file );
        }
    }
        // Join a file Can be called recursively, below
    if (_filePats == 0) {
        _filePats = ttdt_file_join( _file, TT_SCOPE_NONE, 1,
                                    CoEditor::_fileCB_, this );
    }
    XtVaSetValues(  (Widget)_text,
                    XtNsourceType,          (XtArgVal)OL_DISK_SOURCE,
                    XtNsource,              (XtArgVal)_file,
                    NULL );
    _textBuf  = OlTextEditTextBuffer( _text );
    RegisterTextBufferUpdate( _textBuf, CoEditor::_textUpdateCB_,
                              (caddr_t)this );
    if (_modifiedByMe && reloading) {
           ttdt_file_event( _contract, TTDT_REVERTED, _filePats, 1 );
    }
    _modifiedByMe = 0;
    // Does the file have any changes pending?
    _modifiedByOther = ttdt_Get_Modified( _contract, _file, TT_BOTH,
                                          10 * timeOutFactor );
    if (_modifiedByOther) {
        int choice = userChoice( myContext, _baseFrame,
                                 "Another tool has modifications pending for ",
                                 "this file.\nDo you want to ask it to save ",
                                 "or revert the file?", 3, "Save",
                                 "Revert", "Ignore" );
        Tt_status status = TT_OK;
        switch (choice) {
               case 0:
```

例 **B–3** CoEditor.C ファイル　(続き)

```
                // Save pending changes
                status = ttdt_Save( _contract, _file, TT_BOTH,
                                    10 * timeOutFactor );
                break;
            case 1:
                // Revert file to last version
                status = ttdt_Revert( _contract, _file, TT_BOTH,
                                      10 * timeOutFactor );
                break;
        }
        if (status != TT_OK) {
            char *s = tt_status_message( status );
            userChoice( myContext, _baseFrame, s, 1, "Okay" );
            tt_free( s );
        } else if (choice == 0) {
            // file was saved, so reload<
            return _load( 0 );
        } else if (choice == 1) {
            // file was reverted
            _modifiedByOther = 0;
        }
    }

    return TT_OK;
}

Tt_status
CoEditor::_load(
    unsigned char   *contents,
    int                         //len
)
{
    _unload();
    XtVaSetValues( (Widget)_text,
                   XtNsourceType,                   (XtArgVal)OL_DISK_SOURCE,
                   XtNsource,                       (XtArgVal)contents,
                   NULL );
    _textBuf      = OlTextEditTextBuffer( _text );
    RegisterTextBufferUpdate( _textBuf, CoEditor::_textUpdateCB_,
                              (caddr_t)this );
    _modifiedByMe = 0;
    _modifiedByOther = 0;
    return TT_OK;
}

//
// Caller responsible for reporting any errors to user
//
Tt_status
CoEditor::_save()
{
    Tt_status status;
    if (_file != 0) {
```

例 **B–3** CoEditor.C ファイル　*(続き)*

```
        if (SaveTextBuffer( _textBuf, _file ) != SAVE_SUCCESS) {
            return TT_DESKTOP_EIO;
        }
        _modifiedByMe = 0;
        _modifiedByOther = 0;
        // File has been saved
        ttdt_file_event( _contract, TTDT_SAVED, _filePats, 1 );
    }
    if (_contract != 0) {
        int     len      = 0;
        char *  contents = 0;
        if (_file == 0) {
        // If you worry that the buffer might be big,
        // you could instead try a a temp file to
        // transfer the data "out of band".
            contents = _contents( &len );
        }
        status = ttmedia_Deposit( _contract, 0, "ISO_Latin_1",
                                  (unsigned char *)contents,
                                  len, _file, 10 * timeOutFactor );
        if (status != TT_OK) {
            return status;
        }
        _modifiedByMe = 0;
        _modifiedByOther = 0;
        if (contents != 0) {
           XtFree( contents );
        }
    }
    return status;
}

Tt_status
CoEditor::_revert() // XXX how about we always just send Revert? :-)
{
    if (! _modifiedByMe) {
        return TT_OK;
    }
    return _load( 0 ); // XXX what if it's not a file? keep last deposit
}

void
CoEditor::_destroyCB_(
    Widget      w,
    XtPointer   coEditor,
    XtPointer   call_data
)
{
    ((CoEditor *)coEditor)->_destroyCB( w, call_data );
}

void
CoEditor::_destroyCB(
```

例 **B–3** CoEditor.C ファイル　(続き)

```
    Widget      ,
    XtPointer //call_data
)
{
    delete this;
}

void
CoEditor::_wmProtocolCB_(
    Widget      w,
    XtPointer   coEditor,
    XtPointer   wmMsg
)
{
    ((CoEditor *)coEditor)->_wmProtocolCB( w, (OlWMProtocolVerify*)wmMsg );
}

void
CoEditor::_wmProtocolCB(
    Widget               w,
    OlWMProtocolVerify   *wmMsg
)
{
    switch (wmMsg->msgtype) {
        case OL_WM_DELETE_WINDOW:
            if (_modifiedByMe) {
                int choice =
                    userChoice( myContext, _baseFrame,
                                "The text has unsaved changes.",
                                3, "Save, then Quit",
                                "Discard, then Quit",
                                "Cancel" );
                switch (choice) {
                  case 0:
                      break;
                  case 1:
                     _revert();
                     break;
                 case 2:
                     return;
                }
           }
           if  umEditors > 1) {
               XtDestroyWidget( shell );
           } else {
               // XXX OlWmProtocolAction() doesn't call destructor?!
               delete this;
               OlWMProtocolAction( w, wmMsg, OL_DEFAULTACTION );
           }
           break;
        default:
           OlWMProtocolAction( w, wmMsg, OL_DEFAULTACTION );
           break;
```

例 **B–3** CoEditor.C ファイル　*(続き)*

```
    }
}

void
CoEditor::_fileButsCB_(
    Widget      button,
    XtPointer   coEditor,
    XtPointer   call_data
)
{
    ((CoEditor *)coEditor)->_fileButsCB( button, call_data );
}

void
CoEditor::_fileButsCB(
    Widget      button,
    XtPointer   //call_data
)
{
    FileOp op;

    XtVaGetValues( button, XtNuserData, &op, 0 );
    Tt_status status = TT_OK;
    switch (op) {
       case Open:
          break;
       case Revert:
          status =_revert();
          break;
       case Save:
          status =_save();
          break;
       case SaveAs:
          break;
    }
    if (status != TT_OK) {
        _adviseUser( status );
    }
}

void
CoEditor::_editButsCB_(
    Widget      button,
    XtPointer   coEditor,
    XtPointer   call_data
)
{
    ((CoEditor *)coEditor)->_editButsCB( button, call_data );
}

void
CoEditor::_editButsCB(
    Widget      button,
```

例 **B–3** CoEditor.C ファイル　（続き）

```
    XtPointer   //call_data
)
{
    EditOp op;
    XtVaGetValues( button, XtNuserData, &op, 0 );
    Tt_status status = TT_OK;
    switch (op) {
          int         len;
          char        *contents;
          const char  *mediaType;
          Tt_messagemsg;
          Tt_pattern  *pats;
        case SelText:
        case SelAppt:
          if (op == SelText) {
              mediaType = "ISO_Latin_1";
          } else {
              mediaType = "DT_CM_Appointment";
          }
          //contents = _selection( &len );
          contents = _contents( &len );
          if (len <= 0) {
              return;
          }
          // Media load callback
          msg = ttmedia_load( _contract, CoEditor::_mediaLoadMsgCB_,
                              this, TTME_EDIT, mediaType,
                              (unsigned char *)contents, len, 0, 0, 1 );
          if (contents != 0) {
              XtFree( contents );
          }
          status = tt_ptr_error( msg );
          if (status != TT_OK) {
              break;
          }
          pats = ttdt_subcontract_manage( msg, 0, shell, this );
          status = tt_ptr_error( pats );
          if (status != TT_OK) {
              break;
          }
          break;
    }
    if (status != TT_OK) {
        char *s = tt_status_message( status );
        char buf[ 1024 ];
        sprintf( buf, "%d: %s", status, s );
                 tt_free( s );
                 userChoice( myContext, _baseFrame, buf, 1, "Okay" );
    }
}

char *
CoEditor::_contents(
```

例 **B–3** CoEditor.C ファイル　*(続き)*

```
    int *len
)
{
    _textBuf  = OlTextEditTextBuffer( _text );
    TextLocation  start     = { 0, 0, 0 };
    TextLocation  end       = LastTextBufferLocation( _textBuf );
    char         *contents  = GetTextBufferBlock( _textBuf, start, end );
    *len = 0;
    if (contents != 0) {
        *len = strlen( contents );
    }
    return contents;
}

Tt_status
CoEditor::_acceptContract(
    Tt_message msg
)
{
    static const char *here = "CoEditor::_acceptContract()";
    _contract = msg;
    if (tt_message_status( msg ) == TT_WRN_START_MESSAGE) {
                    //
                    // Join session before accepting start message,
                    // to prevent unnecessary starts of our ptype
                    //
                    Widget session_shell = shell;
                    if (maxBuffers > 1) {
                    //
                    // If we are in multi-window mode, just use
                    // our unmapped toplevel shell as our session
                    // shell, since we do not know if any particular
                    // window will exist the whole time we are in
                    // the session.
                    //
                      session_shell = XtParent(shell );
                    }
                    // Join the session and register patterns and callbacks
                    sessPats = ttdt_session_join( 0, 0, session_shell,
                                                  this, 1 );
    }
    // Accept responsibility to handle a request
    _contractPats = ttdt_message_accept(
              msg, CoEditor::_contractCB_, shell, this,
              1, 1 );
Tt_status status = tt_ptr_error( _contractPats );
    if (status != TT_OK) {
           return status;
    }     return status;
}

Tt_message
CoEditor::_contractCB_(
```

例 **B–3** CoEditor.C ファイル　（続き）

```
    Tt_message,                         //msg,
    Tttk_op,                            //op,
    Widget,                             //shell,
    void*,                              //coEditor,
    Tt_message                          //Contract
)
{
    return 0;
}

void
CoEditor::_editButCB_(
    Widget      w,
    XtPointer   coEditor,
    XtPointer   call_data
)
{
    ((CoEditor *)coEditor)->_editButCB( w, call_data );
}

void
CoEditor::_editButCB(
    Widget      ,
    XtPointer //call_data
)
{
    int     len;
    char    *contents = _contents( &len );

    // Media Load Callback
    Tt_message msg = ttmedia_load( _contract, CoEditor::_mediaLoadMsgCB_,
                                   this, TTME_EDIT, "ISO_Latin_1",
                                   (unsigned char *)contents,
                                   len, 0, 0, 1 );
    if (contents != 0) {
        XtFree( contents );
    }
    Tt_pattern *pats = ttdt_subcontract_manage( msg, 0, shell, this );
}

Tt_message
CoEditor::_mediaLoadMsgCB_(
    Tt_message      msg,
    Tttk_op         op,
    unsigned char   *contents,
    int             len,
    char            *file,
    void            *clientData
)
{
    return ((CoEditor *)clientData)->_mediaLoadMsgCB( msg, op,
            contents, len, file ); }
```

例 **B–3** CoEditor.C ファイル　*(続き)*

```
Tt_message
CoEditor::_mediaLoadMsgCB(
    Tt_message      msg,
    Tttk_op,
    unsigned char  *contents,
    int            len,
    char           *file
)
{
    if (len > 0) {
        XtVaSetValues( (Widget)_text,
                       XtNsourceType,  (XtArgVal)OL_STRING_SOURCE,
                       XtNsource,      (XtArgVal)contents,
                       NULL );
        _textBuf     = OlTextEditTextBuffer( _text );
        RegisterTextBufferUpdate( _textBuf, CoEditor::_textUpdateCB_,
                                  (caddr_t)this );
       // ReplaceBlockInTextBuffer
    } else if (file != 0) {
    }
    tt_message_destroy( msg );
    return 0;
}

void
CoEditor::_textUpdateCB_(
    XtPointer       coEditor,
    XtPointer       pTextBuffer,
    EditResult      status
)
{
    if (coEditor == 0) {
        return;
    }
    ((CoEditor *)coEditor)->_textUpdateCB( (TextBuffer *)pTextBuffer,
                                           status );
}

void
CoEditor::_textUpdateCB(
    TextBuffer       *textBuf,
    EditResult              //editStatus
)
{
    //Tt_status status;
    if (_textBuf != textBuf) {
        fprintf( stderr, "_textBuf != textBuf" );
    }
    if ((! _modifiedByMe) && TextBufferModified( _textBuf )) {
        _modifiedByMe = TRUE;
        // File has changes pending
        ttdt_file_event( _contract, TTDT_MODIFIED, _filePats, 1 );
    }
```

例 **B–3** CoEditor.C ファイル　（続き）

```
}

Tt_message
CoEditor::_fileCB_(
    Tt_message        msg,
    Tttk_op           op,
    char              *pathname,
    void              *coEditor,
    int               trust,
    int               me
)
{
    tt_free( pathname );
    if (coEditor == 0) {
        return msg;
    }
    return ((CoEditor *)coEditor)->_fileCB( msg, op, pathname,
                                            trust, me );
}

Tt_message
CoEditor::_fileCB(
    Tt_message         msg,
    Tttk_op            op,
    char               *pathname,
    int,               //trust
    int                //me
)
{
    tt_free( pathname );
    Tt_status status = TT_OK;
    switch (op) {
        case TTDT_MODIFIED:
          if (_modifiedByMe) {
          // Hmm, the other editor either doesn't know or
          // doesn't care that we are already modifying the
          // file, so the last saver will win.
          // XXX Or: a race condition has arisen!
          } else {
          // Interrogate user if she ever modifies the buffer
            _modifiedByOther = 1;
             XtAddCallback( (Widget)_text, XtNmodifyVerification,
                            (XtCallbackProc)CoEditor::_textModifyCB_, 0 );
          }
          break;
        case TTDT_GET_MODIFIED:
          tt_message_arg_ival_set( msg, 1, _modifiedByMe );
          tt_message_reply( msg );
          break;
        case TTDT_SAVE:
          status = _save();
          if (status == TT_OK) {
              tt_message_reply( msg );
```

例 **B–3** CoEditor.C ファイル　(続き)

```
          } else {
          // Fail message
              tttk_message_fail( msg, status, 0, 0 );
          }
          break;
        case TTDT_REVERT:
          status = _revert();
          if (status == TT_OK) {
              tt_message_reply( msg );
          } else {
          // Fail message
              tttk_message_fail( msg, status, 0, 0 );
          }
          break;
        case TTDT_REVERTED:
        case TTDT_SAVED:
        case TTDT_MOVED:
        case TTDT_DELETED:
          printf( "CoEditor::_fileCB(): %s\n", tttk_op_string( op ));
          break;
    }
    tt_message_destroy( msg );
    return 0;
}

void CoEditor::_textModifyCB_(
    TextEditWidget        text,
    XtPointer             ,
    OlTextModifyCallData  *mod
)
{
    CoEditor *coEditor = 0;
    XtVaGetValues( (Widget)text, XtNuserData, &coEditor, 0 );
    if (coEditor == 0) {
        return;
    }
    coEditor->_textModifyCB( mod );
}

void
CoEditor::_textModifyCB(
    OlTextModifyCallData *mod
)
{
    if (_modifiedByOther != 1) {
          return;
    }
    int cancel = userChoice( myContext, _baseFrame,
                  "Another tool has modifications pending for this file.\n",
                  "Are you sure you want to start modifying the file?",
                  2, "Modify", "Cancel" );
    if (cancel) {
        mod->ok = FALSE;
```

例 **B–3** CoEditor.C ファイル　（続き）

```
    }
   _modifiedByOther = 2;
}

void
CoEditor::_adviseUser(
    Tt_status status
)
{
    char *s = tt_status_message( status );
    char buf[ 1024 ];
    sprintf( buf,  "%d: %s", status, s );
    tt_free( s );
    userChoice( myContext, _baseFrame, buf, 1, "Okay" );
}
```

付録 C

新規の ToolTalk 関数

この章では、今回のリリースで新規に作成された ToolTalk 関数について説明します。これらの関数を使用するには、次のような ToolTalk ヘッダ・ファイルを組み込む必要があります。

```
#include <Tt/tt_c.h>
```

tt_error

```
void tt_error(const char *funcname, Tt_status status)
```

tt_error() 関数は、一般的によく知られている NULL 関数です。この関数は、TT_OK 以外のステータスを持っている ToolTalk API 呼び出しから返される直前に ToolTalk ライブラリによって呼び出されます。返されようとしている関数名とステータス・コードが渡されます。この呼び出しを使って、tt_error() にデバッガのブレークポイントを設定し、ToolTalk エラーをすばやくキャッチして追跡できます。また、たとえば stderr に ToolTalk エラーのログを取るために、この関数を割り込み処理することもできます。次のコード・サンプルは、アプリケーションが上記の動作を行う方法を示しています。

```
void
tt_error(const char *funcname, Tt_status status)
{
    fprintf(stderr, "ToolTalk function %s returned %s.\n",
            funcname, tt_status_message(status));
}
```

tt_file_netfile

```
char *      tt_file_netfile( const char *  filename );
```

tt_file_netfile() 関数は、ローカル・パス名と正規のパス名とを対応づけます。filename で指定したファイルを、ネットワーク上の他のホストに渡すことができる netfilename に変換します。filename は、ローカル・ホストで有効な絶対パス名または相対パス名です。filename の最後のコンポーネントは必須ではありませんが、filename のその他の各コンポーネントは必ず存在しなければなりません。

注 – この関数を使用する前に tt_open 関数を呼び出す必要はありません。

この関数は、エラー・ポインタを返すか、正常終了した場合は未指定の書式の新たに割り当てられた NULL で終了する文字列を返します。この文字列は tt_netfile_file 関数に渡されます。

エラー・ポインタからステータスを取り出すには、tt_ptr_error を使用します。返される可能性のあるエラーを表 C–1 に示します。

表 **C–1** tt_file_netfile が返す可能性のあるエラー

エラー	説明
TT_ERR_PATH	filename はこのホスト上では有効でないパスです。
TT_ERR_DBAVAIL	ホスト上の rpc.ttdbserverd に到達できませんでした。
TT_ERR_DBEXIST	rpc.ttdbserverd はホストに適切にインストールされていない可能性があります。

割り当てられた文字列を解除するには、tt_free または tt_release 呼び出しを使用します。

ファイル名を同じファイルのローカル・ファイル名に戻すには、tt_netfile_file 関数を使用します。

tt_host_file_netfile

```
char *  tt_host_file_netfile(   const char *    host,
                                const char *    filename );
```

tt_host_file_netfile() 関数は、リモート・ホスト上のローカル・パス名と正規のパス名とを対応づけます。host で指定したファイルを、ネットワーク上の他のホストに渡すことができる netfilename に変換します。filename は、リモート・ホストで有効な絶対パス名または相対パス名です。filename の最後のコンポーネントは必須ではありませんが、その他の各コンポーネントは必ず存在しなければなりません。

注 – この関数を使用する前に tt_open 関数を呼び出す必要はありません。

この関数は、エラー・ポインタを返すか、正常終了した場合は未指定の書式の新たに割り当てられた NULL で終了する文字列を返します。この文字列は tt_netfile_file 関数に渡されます。

エラー・ポインタからステータスを取り出すには、tt_ptr_error を使用します。返される可能性のあるエラーを表 C–2 に示します。

表 **C–2** tt_host_file_netfile が返す可能性のあるエラー

エラー	説明
TT_ERR_PATH	filename はリモート・ホスト上では有効でないパスです。
TT_ERR_DBAVAIL	ホスト上の rpc.ttdbserverd に到達できませんでした。
TT_ERR_DBEXIST	rpc.ttdbserverd はホストに適切にインストールされていない可能性があります。
TT_ERR_UNIMP	rpc.ttdbserverd バージョンは、tt_host_file_netfile() 関数をサポートしていません。

- 割り当てられた文字列を解除するには、tt_free または tt_release 呼び出しを使用します。

ファイル名を同じファイルのローカル・ファイル名に戻すには、tt_host_netfile_file() 関数を使用します。

tt_host_netfile_file

```
char * tt_host_netfile_file(  const char *      host,
                              const char *      netfilename );
```

tt_host_file_netfile() 関数は、リモート・ホスト上のローカル・パス名と正規のパス名とを対応づけます。netfilename で指定したファイルを、リモート・ホスト上で有効なパス名に変換します。netfilename は、tt_netfile_file 関数によって返される NULL で終了する文字列のコピーです。

注 – この関数を使用する前に tt_open 関数を呼び出す必要はありません。

指定されたファイルが現在ローカル・ホストにマウントされていない場合、次のような書式のパス名が作成されます。

/DTMOUNTPOINT/host/filepath

DTMOUNTPOINT は、オートマウントのホスト・マップのために作成されたマウント・ポイントです。環境変数 DTMOUNTPOINT で、このマウント・ポイントを指定することもできます。

host は、ファイルを格納しているホストです。

filepath は、ホストに格納されているファイルに対するパスです。

この関数は、エラー・ポインタを返すか、正常終了した場合は新たに割り当てられた NULL で終了するローカル・ファイル名を返します。

エラー・ポインタからステータスを取り出すには、tt_ptr_error を使用します。返される可能性のあるエラーを表 C–3 に示します。

表 **C–3** tt_host_netfile_file が返す可能性のあるエラー

エラー	説明
TT_ERR_PATH	netfilename は有効なネットファイル名ではありません。
TT_ERR_DBAVAIL	ホスト上の rpc.ttdbserverd に到達できませんでした。
TT_ERR_DBEXIST	rpc.ttdbserverd はホストに適切にインストールされていない可能性があります。
TT_ERR_UNIMP	rpc.ttdbserverd バージョンは、tt_host_netfile_file() 関数をサポートしていません。

割り当てられた文字列を解除するには、tt_free または tt_release 呼び出しを使用します。

ファイル名を同じファイルのローカル・ファイル名に戻すには、tt_host_file_netfile() 関数を使用します。

tt_message_print

```
char *    tt_message_print(Tt_message m);
```

tt_message_print 関数により、理解しないまま受信したメッセージを出力できます。

割り当てられた文字列を解除するには、tt_free または tt_release 呼び出しを使用します。

この関数は、エラー TT_ERR_POINTER を返すか、正常終了した場合は (tt_X_session などの他の ToolTalk API 呼び出しで行われているのと同様の方法で) ToolTalk によって割り当てられたバッファにあるメッセージ m を返します。

tt_netfile_file

```
char *  tt_netfile_file( const char *  netfilename );
```

tt_netfile_file 関数は、ローカル・パス名と正規のパス名とを対応づけます。netfilename で指定したファイルを、ローカル・ホスト上で有効なパス名に変換します。netfilename は、tt_netfile_file 関数によって返される NULL で終了する文字列のコピーです。

注 – この関数を使用する前に tt_open 関数を呼び出す必要はありません。

指定されたファイルが現在ローカル・ホストにマウントされていない場合、次のような形式のパス名が作成されます。

/DTMOUNTPOINT/host/filepath

DTMOUNTPOINT は、オートマウントのホスト・マップのために作成されたマウント・ポイントです。環境変数 DTMOUNTPOINT でこのマウント・ポイントを指定することもできます。

host は、ファイルを格納しているホストです。

filepath は、ホストに格納されているファイルに対するパスです。

この関数は、エラー・ポインタを返すか、正常終了した場合は新たに割り当てられた NULL で終了するローカル・ファイル名を返します。

エラー・ポインタからステータスを取り出すには、tt_ptr_error を使用します。返される可能性のあるエラーを表 C–4 に示します。

表 **C–4** tt_netfile_file が返す可能性のあるエラー

エラー	説明
TT_ERR_PATH	netfilename は有効なネットファイル名ではありません。
TT_ERR_DBAVAIL	ホスト上の rpc.ttdbserverd に到達できませんでした。
TT_ERR_DBEXIST	rpc.ttdbserverd はホストに適切にインストールされていない可能性があります。

割り当てられた文字列を解除するには、tt_free または tt_release 呼び出しを使用します。

ファイル名を同じファイルのネット・ファイル名に戻すには、tt_file_netfile() 関数を使用します。

tt_pattern_print

```
char *  tt_message_print(Tt_pattern p);
```

tt_pattern_print() 関数により、パターンを出力できます。

割り当てられた文字列を解除するには、tt_free または tt_release 呼び出しを使用します。

この関数は、エラー TT_ERR_POINTER を返すか、正常終了した場合は (tt_X_session などの他の ToolTalk API 呼び出しで行われているのと同様の方法で) ToolTalk によって割り当てられたバッファにあるパターン p を返します。

付録 D

例

Ttdt_contract_cb の例

例 D–1 は、アプリケーションのための Ttdt_contract_cb コールバックの典型的なアルゴリズムの例です。このアプリケーションは、Pause 要求、 Resume 要求、 Quit 要求を処理しますが、ツールキットには X11 関連の要求しか処理させません。

注 – この例にあるコールバックは、contract パラメータがゼロ以外の値を持っている場合に処理します。したがって、ttdt_message_accept に渡される Ttdt_contract_cb コールバックとして使用することもできます。

例 D–1 Ttdt_contract_cb の典型的なアルゴリズム

```
Tt_message myContractCB(
    Tt_message  msg,
    void        *clientdata,
    Tt_message  contract
)
{
    char *opString = tt_message_op( msg );
    Tttk_op op = tttk_string_op( opString );
    tt_free( opString );
    int silent = 0;
    int force    = 0;
    Boolean cancel = False;
    Boolean sensitive = True;
    char *status, command;
    switch (op) {
      case TTDT_QUIT:
        tt_message_arg_ival( msg, 0, &silent );
        tt_message_arg_ival( msg, 1, &force );
        if (contract == 0) {
          /* Quit entire application */
```

例 **D–1** Ttdt_contract_cb の典型的なアルゴリズム　*(続き)*

```
        cancel = ! myQuitWholeApp( silent, force );
      } else {
        /* Quit just the specified request being worked on */
        cancel = ! myCancelThisRequest(contract, silent, force);
      }
      if (cancel) {
        /* User canceled Quit; fail the Quit request */
        tttk_message_fail( msg, TT_DESKTOP_ECANCELED, 0, 1 );
      } else {
        tt_message_reply( msg );
        tttk_message_destroy( msg );
      }
      return 0;
    case TTDT_PAUSE:
      sensitive = False;
    case TTDT_RESUME:
      if (contract == 0) {
        int already = 1;
        if (XtIsSensitive( myTopShell ) != sensitive) {
          already = 0;
          XtSetSensitive( myTopShell, sensitive );
        }
        if (already) {
          tt_message_status_set(msg,TT_DESKTOP_EALREADY);
        }
      } else {
        if (XtIsSensitive( thisShell ) == sensitive) {
          tt_message_status_set(msg,TT_DESKTOP_EALREADY);
        } else {
          XtSetSensitive( thisShell, sensitive );
      }
      tt_message_reply( msg );
      tttk_message_destroy( msg );
      return 0;
    case TTDT_GET_STATUS:
      if (contract == 0) {
        status = "Message about status of entire app";
      } else {
        status = "Message about status of this request";
      }
      tt_message_arg_val_set( msg, 0, status );
      tt_message_reply( msg );
      tttk_message_destroy( msg );
      return 0;
    case TTDT_DO_COMMAND:
      if (! haveExtensionLanguage) {
        tttk_message_fail( msg, TT_DESKTOP_ENOTSUP, 0, 1 );
        return 0;
      }
      command = tt_message_arg_val( msg, 0 );
      result = myEval( command );
      tt_free( command );
      tt_message_status_set( msg, result );
```

例 **D–1** Ttdt_contract_cb の典型的なアルゴリズム　(続き)

```
        if (tt_is_err( result )) {
        tttk_message_fail( msg, result, 0, 1 );
        } else {
        tt_message_reply( msg );
        tttk_message_destroy( msg );
        }
        return 0;
    }
    /* Unrecognized message; do not consume it */
    return msg;
}
```

Ttdt_file_cb の例

例 D–2 は、このコールバックの典型的なアルゴリズムの例です。

例 **D–2** Ttdt_file_cb の典型的なアルゴリズム

```
Tt_message myFileCB(
    Tt_message      msg,
    Tttk_op         op,
    char            *pathname,
    int             trust,
    int             isMe
)
{
    tt_free( pathname );
    Tt_status status = TT_OK;
    switch (op) {
      case TTDT_MODIFIED:
        if ((_modifiedByMe) && (! isMe)) {
          // Hmm, the other editor either does not know or
          // does not care that we are already modifying the
          // file, so the last saver will win.
        } else {
          // Interrogate user if she ever modifies the buffer
          _modifiedByOther = 1;
          XtAddCallback( myTextWidget, XmNmodifyVerifyCallback,
                         myTextModifyCB, 0 );
        }
        break;
      case TTDT_GET_MODIFIED:
        tt_message_arg_ival_set( msg, 1, _modifiedByMe );
        tt_message_reply( msg );
        break;
      case TTDT_SAVE:
        status = mySave( trust );
```

例 **D–2** Ttdt_file_cb の典型的なアルゴリズム　(続き)

```
        if (status == TT_OK) {
          tt_message_reply( msg );
        } else {
          tttk_message_fail( msg, status, 0, 0 );
        }                   break;
      case TTDT_REVERT:
        status = myRevert( trust );
        if (status == TT_OK) {
          tt_message_reply( msg );
        } else {
          tttk_message_fail( msg, status, 0, 0 );
        }
        break;
      case TTDT_REVERTED:
        if (! isMe) {
          _modifiedByOther = 0;
        }
        break;
      case TTDT_SAVED:
        if (! isMe) {
          _modifiedByOther = 0;
          int choice = myUserChoice( myContext, myBaseFrame,
                                     "Another tool has saved "
                                     "this file.", 2, "Ignore",
                                     "Revert" );
          switch (choice) {
            case 1:
              myRevert( 1 );
                break;
          }
        }
        break;
      case TTDT_MOVED:
      case TTDT_DELETED:
        // Do something appropriate
        break;
    }
    tttk_message_destroy( msg );
    return 0;
}
```

Ttmedia_load_msg_cb の例

例 D–3 は、このコールバックの典型的なアルゴリズムの例です。

例 **D–3** Ttmedia_load_msg_cb の典型的なアルゴリズム

```
Tt_message
myLoadMsgCB(
```

例 **D–3** Ttmedia_load_msg_cb の典型的なアルゴリズム　(続き)

```
    Tt_message      msg,
    void            *clientData,
    Tttk_op         op,
    unsigned char   *contents,
    int             len,
    char            *file
)
{
    if (len > 0) {
      // Replace data with len bytes in contents
    } else if (file != 0) {
      // Replace data with data read from file
    }
    if (op == TTME_DEPOSIT) {
      tt_message_reply( msg );
    }
    tttk_message_destroy( msg );
    return 0;
}
```

Ttmedia_load_pat_cb の例

例 D–4 は、このコールバックの典型的なアルゴリズムの例です。

例 **D–4** TTtmedia_load_pat_cb の典型的なアルゴリズム

```
Tt_message
myAcmeSheetLoadCB(
    Tt_message      msg,
    void            *client_data,
    Tttk_op         op,
    Tt_status       diagnosis,
    unsigned char   *contents,
    int             len,
    char            *file,
    char            *docname
)
{
    Tt_status status = TT_OK;
    if (diagnosis != TT_OK) {
      // toolkit detected an error
      if (tt_message_status( msg ) == TT_WRN_START_MESSAGE) {
        //
        // Error is in start message!    We now have no
        // reason to live, so tell main() to exit().
        //
        myAbortCode = 2;
```

例 **D–4** TTtmedia_load_pat_cb の典型的なアルゴリズム　(続き)

```
      }
      // let toolkit handle the error
      return msg;
    }
    if ((op == TTME_COMPOSE) && (file == 0)) {
      // open empty new buffer
    } else if (len > 0) {
      // load contents into new buffer
    } else if (file != 0) {
      if (ttdt_Get_Modified( msg, file, TT_BOTH, myCntxt, 5000 )) {
        switch (myUserChoice( "Save, Revert, Ignore?" )) {
          case 0:
            ttdt_Save( msg, file, TT_BOTH, myCntxt, 5000 );
            break;
          case 1:
            ttdt_Revert( msg, file, TT_BOTH, myCntxt, 5000);
            break;
        }
      }
        // load file into new buffer
    } else {
      tttk_message_fail( msg, TT_DESKTOP_ENODATA, 0, 1 );
      tt_free( contents ); tt_free( file ); tt_free( docname );
      return 0;
    }
    int w, h, x, y = INT_MAX;
    ttdt_sender_imprint_on( 0, msg, 0, &w, &h, &x, &y, myCntxt, 5000 );
    positionMyWindowRelativeTo( w, h, x, y );
    if (maxBuffersAreNowOpen) {
      // Un-volunteer to handle future requests until less busy
      tt_ptype_undeclare( "Acme_Calc" );
    }
    if (tt_message_status( msg ) == TT_WRN_START_MESSAGE) {
      //
      // Join session before accepting start message,
      // to prevent unnecessary starts of our ptype
      //
      ttdt_session_join( 0, myContractCB, myShell, 0, 1 );
    }
    ttdt_message_accept( msg, myContractCB, myShell, 0, 1, 1 );
    tt_free( contents ); tt_free( file ); tt_free( docname );
    return 0;
}
```

Ttmedia_ptype_declare 関数の ptype シグニチャーの例

例 D–5 は、メディア ptype のシグニチャーレイアウトの例です。

例 **D–5** メディア ptype のシグニチャーレイアウトの例

```
ptype Acme_Calc {
    start "acalc";
    handle:
    /*
     * Display Acme_Sheet
     * Include in tool's ptype if tool can display a document.
     */
    session Display( in    Acme_Sheet    contents     ) => start opnum = 1;
    session Display( in    Acme_Sheet    contents,
                     in    messageID     counterfoil  ) => start opnum = 2;
    session Display( in    Acme_Sheet    contents,
                     in    title         docName      ) => start opnum = 3;
    session Display( in    Acme_Sheet    contents,
                     in    messageID     counterfoil,
                     in    title         docName      ) => start opnum = 4;
    /*
     * Edit Acme_Sheet
     * Include in tool's ptype if tool can edit a document.
     */
    session Edit(    inout Acme_Sheet    contents     ) => start opnum = 101;
    session Edit(    inout Acme_Sheet    contents,
                     in    messageID     counterfoil  ) => start opnum = 102;
    session Edit(    inout Acme_Sheet    contents,
                     in    title         docName      ) => start opnum = 103;
    session Edit(    inout Acme_Sheet    contents,
                     in    messageID     counterfoil,
                     in    title         docName      ) => start opnum = 104;
    /*
     * Compose Acme_Sheet
     * Include in tool's ptype if tool can compose a document from scratch.
     */
    session Edit(    out   Acme_Sheet    contents     ) => start opnum = 201;
    session Edit(    out   Acme_Sheet    contents,
                     in    messageID     counterfoil  ) => start opnum = 202;
    session Edit(    out   Acme_Sheet    contents,
                     in    title         docName      ) => start opnum = 203;
    session Edit(    out   Acme_Sheet    contents,
                     in    messageID     counterfoil,
                     in    title         docName      ) => start opnum = 204;
    /*
     * Mail Acme_Sheet
     * Include in tool's ptype if tool can mail a document.
     */
```

例 **D–5** メディア ptype のシグニチャーレイアウトの例　*(続き)*

```
    session Mail(    in    Acme_Sheet    contents     ) => start opnum = 301;
    session Mail(    inout Acme_Sheet    contents     ) => start opnum = 311;
    session Mail(    inout Acme_Sheet    contents,
                     in    title         docName      ) => start opnum = 313;
    session Mail(    out   Acme_Sheet    contents     ) => start opnum = 321;
    session Mail(    out   Acme_Sheet    contents,
                     in    messageID     counterfoil  ) => start opnum = 323;
};
```

Xt 入力処理関数の例

例 D–6 は、Xt 入力処理関数の例です。

例 **D–6** Xt 入力処理関数の例

```
int  myTtFd;
char *myProcID;
myProcID = ttdt_open( &myTtFd, "WhizzyCalc", "Acme", "1.0", 1 );
/* ... */
/* Process the message that started us, if any */
tttk_Xt_input_handler( myProcID, 0, 0 );
/* ... */
XtAppAddInput( myContext, myTtFd, (XtPointer)XtInputReadMask,
               tttk_Xt_input_handler, myProcID );
```

索引

C
CoEd.C ファイル，89, 90
Coeditor.C ファイル，93
CoEditor.h ファイル，26
CoEd デモ・プログラム，89
CoEd デモンストレーション・プログラム，89
Compose 要求，77
CORBA 準拠のシステム，15, 22
Created 通知，60

D
Deleted 通知，59, 60
Deposit 要求，78
$DISPLAY，70
Display 要求，77
Do_Command 要求，73
DTMOUNTPOINT，116, 117

E
Edit 要求，77
ENV_，58, 78
environ (5)，67

F
filepath，116, 117

G
Get_Environment 要求，72
Get_Geometry 要求，65, 70, 72, 75
Get_Iconified 要求，65, 72
Get_Locale 要求，72
Get_Mapped 要求，65, 72
Get_Modified 要求，57, 63
Get_Situation 要求，72
Get_Status 要求，66, 73
Get_Sysinfo 要求，72
Get_XInfo 要求，65, 73, 75

L
libtt，41
Lower 要求，65, 72
-ltt オプション，26

M
makefile，26
Modified 通知，58, 59, 60
Moved 通知，59, 60

N
netfilename，114, 115, 117

P

Pause 要求，66, 73
process type (ptype)，17
procid，32
ptype のインストール，31
ptype のマージ，31
ptype ファイル，30

Q

Quit 要求，66, 73

R

Raise 要求，65, 72
Resume 要求，66, 73
Reverted 通知，58, 59, 60
Revert 要求，57, 68
rpc.ttdbserverd，114

S

Saved 通知，58, 59, 60
Save 要求，57, 69
Session_Trace 要求，43
Set_Environment 要求，72
Set_Geometry 要求，65, 71, 72
Set_Iconified 要求，65, 72
Set_Locale 要求，72
Set_Mapped 要求，65, 72
Set_Situation 要求，72
Set_XInfo 要求，65, 73
Signal 要求，72
Started 通知，67
Status 通知，75
Stopped 通知，57

T

ToolTalk 型コンパイラ，30, 31
ToolTalk 型データベース，30
ToolTalk 関数
  tt_close，57
  tt_default_procid_set，57
ToolTalk 関数 (続き)
  tt_file_netfile，118
  tt_free，59, 67, 70, 78, 81, 114, 115, 116, 117, 118
  tt_netfile_file，115
  tt_open，67, 114, 115, 116, 117
  tt_ptr_error，114, 115, 116, 118
  tt_release，114, 115, 116, 117, 118
  tt_X_session，117, 118
  tttrace，45
ToolTalk コマンド, tttrace，41
ToolTalk サービス，13
ToolTalk サービスを使用するためのアプリケーションの変更，24
ToolTalk のセッションの概念，23
ToolTalk の特徴，21
ToolTalk ヘッダ・ファイル，25, 113
ToolTalk メッセージ，20
ToolTalk メッセージ・セット
  デスクトップ，16
  ドキュメント・メディア交換，18
ToolTalk メッセージ・ツールキットのヘッダ・ファイル，56
ToolTalk メッセージの受信，21
ToolTalk メッセージの送信，20
ToolTalk ライブラリ，26
truss コマンド，41
TT_BOTH，59, 64, 68, 69
tt_close 関数，57
tt_default_procid_set (new_procid)，57
tt_default_procid_set (procid)，57
tt_default_procid_set 関数，57
TT_DESKTOP_EINVAL，58, 61
TT_DESKTOP_ENODATA，81
TT_DESKTOP_ENOTSUP，65, 66, 72, 73
TT_DESKTOP_EPROTO，68, 70
TT_DESKTOP_ETIMEOUT，68, 70, 71, 77
TT_ERR_APPFIRST + EACCES，44
TT_ERR_APPFIRST + EEXIST，45
TT_ERR_APPFIRST + EISDIR，44
TT_ERR_APPFIRST + ENOSPC，45
TT_ERR_DBAVAIL，60, 61, 62, 63, 68, 70, 77, 114, 115, 116, 118
TT_ERR_DBEXIST，60, 61, 62, 63, 68, 70, 77, 114, 115, 116, 118
TT_ERR_EINVAL，76
TT_ERR_NO_MATCH，44

TT_ERR_NOMEM, 60, 61, 63, 68, 70, 71, 74, 75, 77, 79, 84
TT_ERR_NOMP, 58, 60, 61, 62, 63, 66, 68, 69, 71, 74, 75, 76, 77, 79, 80, 82, 83, 84, 87
TT_ERR_NOTHANDLER, 80, 83
TT_ERR_NUM, 80
TT_ERR_OVERFLOW, 58, 61, 63, 68, 70, 71, 77, 79
TT_ERR_PATH, 60, 114, 115, 116, 118
TT_ERR_POINTER, 58, 61, 62, 63, 67, 69, 70, 74, 75, 76, 77, 82, 83
TT_ERR_PROCID, 61, 62, 63, 68, 69, 71, 74, 75, 76, 77, 79, 80, 82, 84
TT_ERR_PTYPE, 82
TT_ERR_SESSION, 74, 75
TT_ERR_UNIMP, 67, 115, 116
tt_error 関数, 113
tt_fd, 67
TT_FILE_IN_SESSION, 59, 64, 68, 69
tt_file_join (pathname), 59
tt_file_netfile 関数, 114, 118
tt_file_quit (pathname), 61
tt_free, 59
tt_free 関数, 67, 70, 78, 81, 114, 115, 116, 117, 118
tt_host_file_netfile 関数, 114
tt_host_netfile_file 関数, 115
tt_message_accept (contract), 66
tt_message_destroy メッセージ, 84
tt_message_id, 78
tt_message_print, 117
tt_message_receive, 85
tt_message_receive 関数, 86
tt_message_status, 68, 69
tt_netfile_file 関数, 114, 115, 116, 117
tt_open, 46
tt_open 関数, 67, 114, 115, 116, 117
tt_pattern_print, 118
TT_PROCEDURE, 83
tt_ptr_error, 59, 66, 74, 75, 79, 84
tt_ptr_error 関数, 114, 115, 116, 118
tt_ptype_declare (ptype), 81
tt_release 関数, 114, 115, 116, 117, 118
TT_SCOPE_NONE, 59, 64, 68, 69
tt_trace_control 関数, 42
$TT_TRACE_SCRIPT, 42
TT_TRACE_SCRIPT 環境変数, 46
tt_type_comp, 30
TT_WRN_START_MESSAGE, 66, 85, 86
tt_X_session, 117, 118
ttdt_close, 57
Ttdt_contract_cb, 73, 119
Ttdt_contract_cb 引き数, 64
Ttdt_file_cb, 59, 121
ttdt_file_event, 57, 60
ttdt_file_join, 53, 57, 58, 61
ttdt_file_notice, 60
ttdt_file_quit, 59, 61
ttdt_file_request, 62
ttdt_Get_Modified, 62, 63
TTDT_GET_MODIFIED, 62
ttdt_message_accept, 64
TTDT_MODIFIED, 57
ttdt_open, 66, 67
ttdt_Revert, 62, 67
TTDT_REVERTED, 58
ttdt_Save, 62, 69
TTDT_SAVED, 58
ttdt_sender_imprint_on, 70
ttdt_session_join, 71
ttdt_session_quit, 74, 75
ttdt_subcontract_manage, 75, 78
TTME_COMPOSE, 78, 81
TTME_DEPOSIT, 78
TTME_DISPLAY, 78, 81
TTME_EDIT, 78, 81
ttmedia_Deposit, 76
ttmedia_load, 77, 78, 79
Ttmedia_load_msg_cb, 122
Ttmedia_load_msg_cb メッセージ, 78
Ttmedia_load_pat_cb, 123
Ttmedia_load_pat_cb メッセージ, 81
ttmedia_load_reply, 79
Ttmedia_ptype_declare, 125
ttmedia_ptype_declare, 80
ttsession トレース, 41
TTSnoop, 37
ttsnoop コマンドの -t オプション, 38
tttk_block_while, 64, 68, 69, 71, 82
tttk_message_abandon, 82, 86
tttk_message_create, 83
tttk_message_destroy, 84
tttk_message_fail, 85
tttk_message_receive, 85
tttk_message_reject, 85
tttk_op_string, 86
tttk_patterns_destroy, 66

tttk_Xt_input_handler，86, 126
tttrace，41
tttrace 関数，42, 45
tttrace コマンド，41

U

/usr/dt/bin/ttsnoop，37

X

XtRemoveInput 関数，87

あ

アプリケーション統合，29
アプリケーションの ToolTalk メッセージの使用方法，20
アプリケーション・プログラミング・インタフェース (API)，24

い

イベント，27

え

エラー
  TT_ERR_APPFIRST + EACCES，44
  TT_ERR_APPFIRST + EEXIST，45
  TT_ERR_APPFIRST + EISDIR，44
  TT_ERR_APPFIRST + ENOSPC，45
  TT_ERR_DBAVAIL，114, 115, 116, 118
  TT_ERR_DBEXIST，114, 115, 116, 118
  TT_ERR_NO_MATCH，44
  TT_ERR_PATH，114, 115, 116, 118
  TT_ERR_UNIMP，115, 116
エラーの値
  TT_DESKTOP，81
  TT_DESKTOP_EINVAL，58, 61
  TT_DESKTOP_ENOTSUP，65, 66, 72, 73
  TT_DESKTOP_EPROTO，68, 70
  TT_DESKTOP_ETIMEOUT，68, 70, 71, 77
エラーの値 (続き)
  TT_ERR_DBAVAIL，60, 61, 62, 63, 68, 70, 77
  TT_ERR_DBEXIST，60, 61, 62, 63, 68, 70, 77
  TT_ERR_EINVAL，76
  TT_ERR_NOMEM，60, 61, 63, 68, 70, 71, 74, 75, 77, 79, 84
  TT_ERR_NOMP，58, 60, 61, 62, 63, 66, 68, 69, 71, 74, 75, 76, 77, 79, 80, 82, 83, 84, 87
  TT_ERR_NOTHANDLER，80, 83
  TT_ERR_NUM，80
  TT_ERR_OVERFLOW，58, 61, 63, 68, 70, 71, 77, 79
  TT_ERR_PATH，60
  TT_ERR_POINTER，58, 61, 62, 63, 67, 69, 70, 74, 75, 76, 82
  TT_ERR_PROCID，61, 62, 63, 68, 69, 71, 74, 75, 76, 77, 79, 80, 82, 84
  TT_ERR_PTYPE，82
  TT_ERR_SESSION，74, 75
  TT_ERR_UNIMP，67

お

オブジェクト指向メッセージ，22
オペレーション，27

か

開始文字列，30
型機構，30
型情報, マージ，31
型情報のマージ，31
環境変数
  $DISPLAY，70
  DTMOUNTPOINT，116, 117
  $TT_TRACE_SCRIPT，42

き

既存の ptype に対するテスト，31

く

クライアント・モード，41

こ
固定表示オブジェクト，15
コンパイルされた ToolTalk 型ファイルを実行中の ttsession にマージする，31

さ
サーバ・モード，41

し
シグニチャ，30
自動起動，15
受信側，20
受信側の判別，21
使用中の ToolTalk サービスを示すシナリオ，16
新規の ToolTalk 関数
tt_error，113
tt_file_netfile，114
tt_host_file_netfile，114
tt_host_netfile_file，115
tt_message_print，117
tt_netfile_file，114, 115, 116, 117
tt_pattern_print，118

せ
正規のパス名とローカル・パス名との対応づけ，117
制御統合，14
静的メッセージ・パターン，30
セッション識別子 (sessid)，23

そ
相互運用の問題，14
送信側，20

つ
通知，27

ツールキット・メッセージ
ttdt_close，57
ttdt_contract_cb，73
ttdt_file_event，57, 60
ttdt_file_join，57, 58, 61
ttdt_file_notice，60
ttdt_file_quit，59, 61
ttdt_file_request，62
ttdt_Get_Modified，63
ttdt_message_accept，64
ttdt_open，66, 67
ttdt_Revert，67
ttdt_Save，69
ttdt_sender_imprint_on，70
ttdt_session_join，71
ttdt_session_quit，74, 75
ttdt_subcontract_manage，75, 78
ttmedia_Deposit，76
ttmedia_load，77, 78, 79
ttmedia_load_reply，79
ttmedia_ptype_declare，80
tttk_block_while，64, 68, 69, 71, 82
tttk_message_abandon，82
tttk_message_create，83
tttk_message_destroy，84
tttk_message_fail，85
tttk_message_receive，85
tttk_message_reject，85
tttk_op_string，86
tttk_patterns_destroy，66
tttk_Xt_input_handler，86

て
デスクトップ・サービス・メッセージ・セット，16

と
ドキュメント・メディア交換メッセージ・セット，18

ね
ネットワーク透過イベント，15

は
パス名，115, 117

ふ
ファイル
- CoEd.C ファイル，89, 90
- Coeditor.C ファイル，93
- CoEditor.h ファイル，26
- ToolTalk のファイルの概念，23
- ToolTalk ヘッダ，25, 113
- ToolTalk メッセージ・ツールキットのヘッダ，56
- メッセージ・ツールキットのヘッダ，25

ファイルの配信範囲指定機能の制限，23
プラグ・アンド・プレイ，14
プロセス型 (ptype)，30
プロセス型識別子 (ptid)，30
プロセス指向メッセージ，22
分散オブジェクト・システム，15

ほ
ホスト，116, 117
本リリースに付いているツール用の ptype，30
本リリースには含まれていないツール用の ptype，30

め
メッセージ
- オブジェクト指向メッセージ，22
- 監視，21
- 受信，21
- 処理，21
- 送信，20
- プロセス指向メッセージ，22
- メッセージの受信者の判別，23
- メッセージを配布する方式，22

メッセージ・セット
- ツールキット
  - ttdt_Revert，67
  - ttdt_Save，69
  - ttdt_file_quit，59, 61
  - ttdt_close，57

ツールキット (続き)
- ttdt_contract_cb，73
- ttdt_file_event，57, 60
- ttdt_file_join，57, 58, 61
- ttdt_file_notice，60
- ttdt_file_request，62
- ttdt_Get_Modified，63
- ttdt_message_accept，64
- ttdt_open，66, 67
- ttdt_sender_impr int_on，70
- ttdt_session_join，71
- ttdt_session_quit，74, 75
- ttdt_subco ntract_manage，78
- ttdt_subcontract_manage，75
- ttmedia_Deposit，76
- ttmedia_load，77, 78, 79
- ttmedia_load_reply，79
- ttmedia_ptype_declare，80
- tttk_block_while，64, 68, 69, 71, 82
- tttk_message_abandon，82
- tttk_message_create，83
- tttk_message_destroy，84
- tttk_message_fail，85
- tttk_message_receive，85
- tttk_message_reject，85
- tttk_op_string，86
- tttk_pattern s_destroy，66
- tttk_Xt_input_hand ler，86

メッセージ・ツールキットの組み込み，26
メッセージ・ツールキットのヘッダ・ファイル，25
メッセージ・パターン，21
メッセージ・プロトコル，24
メッセージを配布する方式，22

よ
要求，27
要求の識別，27

ら
ライブラリ, ToolTalk，26

ろ
ローカル・パス名と正規のパス名との対応づけ，114, 115