The ToolTalk Document and Media Exchange Message Set

Ronald Lee, Brian Holtz, and Astrid Julienne

What is the ToolTalk Document and Media Exchange Message Set?

Multimedia is an important emerging technology. While the base of multimedia-aware applications has expanded, no single vendor provides a completely integrated solution which meets the complex needs of today's market. The *ToolTalk Document and Media Exchange™ Message Set* is a genuine breakthrough in multimedia technologies. A powerful messaging protocol designed to benefit both developers and users of multimedia technologies, the ToolTalk Document and Media Exchange Message Set allows applications to easily share each others multimedia functionality. Using the ToolTalk Document and Media Exchange Message Set, multimedia applications can communicate with each other in a transparent manner, both locally and over networks, regardless of data formats, compression technology, and other technical issues which has previously confined the use of this technology.

Why the ToolTalk Document and Media Exchange Message Set was Developed

While a few vendors have established inter-operability alliances, the range of possible end-user solutions as been restricted. The ToolTalk Document and Media Exchange Message Set allows any application to share a set of multimedia functions with any other application in a transparent manner.

This document contains specifications that have been developed by an alliance of designers from key independent multimedia hardware and software vendors, SunSoft™, and the Sun Microsystems® Computer Corporation (SMCC)™. Applications that use these simple protocols can quickly and easily create a ToolTalk™ interface to an array of multimedia services without concern for a particular service provider. Entire groups of applications can now *plug-and-play* together, integrating sound, video, graphics, telephony, and other media sources into new and exciting applications. The term *plug-and-play* means that any tool can be replaced by any other tool that follows the same protocol. That is, any tool that follows a given ToolTalk protocol can be placed (plugged) into your computing environment and perform (play) those functions indicated by the protocol. Tools can be mixed and matched, without modification and without having any specific built-in knowledge of each other. For example, you could create a word processing application that integrates a piece of video into a composition and have the video played by another application. See "The ToolTalk Service" section for more information about the ToolTalk service.

The ToolTalk Document and Media Exchange Message Set is an efficient set of generic message definitions that provide media control and data exchange. The protocol consists of editor messages for media players, editors, and users. See the "The ToolTalk Media Exchange Protocol" section for a detailed description of these messages.

Using the ToolTalk Document and Media Exchange Message Set

The ToolTalk Document and Media Exchange Message Set is very flexible and robust. This section illustrates three applications of the ToolTalk Document and Media Exchange Message Set:

- Integrating multimedia into an authoring application
- Adding multimedia extensions to an existing application
- Extending the *cut and paste* facility of *X* with a media translation facility

Integrating Multimedia Functionality

Integrating multimedia functionality into an application allows end-users of the application to embed various media types in their documents.

Typically, an icon that represents the media object is embedded in the document. Upon selection of an embedded object, the ToolTalk service automatically invokes an appropriate external media application and the object is played as illustrated in the following scenario.

1. Daniel opens a document that contains multimedia objects.
2. The window shows the document with several icons representing various media types (such as sound, video, and graphics).

3. Daniel double-clicks on the sound icon.
   A sound application (called a *player*) is launched and the embedded recording is played.

4. To edit the recording, Daniel clicks once on the icon to select it and uses the third mouse button to bring up an Edit menu.
   An editing application is launched and Daniel edits the media object.

Adding Multimedia Extensions to Existing Applications

The ToolTalk Document and Media Exchange Message Set also allows an application to use other multimedia applications to extend its features or capabilities. For example, a calendar manager can be extended to use the audiotool to play a sound file as a reminder of an appointment, as illustrated in the following scenario:

1. Karin opens her calendar manager and sets an appointment.

2. Karin clicks on an audio response button, which causes the soundtool to pop up.

3. Karin records her message; for example, "Bring the report."

When Karin's appointment reminder is executed, the calendar manager will start the audiotool and play Karin's recorded reminder.

Extending the X Cut and Paste Facility

The ToolTalk Document and Media Exchange Message Set can support an extensible, open-ended translation facility. The following scenario illustrates how an extensible multimedia *cut and paste* facility could work:

1. Maria opens two documents that are different media types.

2. Maria selects a portion of *Document A* and cuts the portion using the standard X-windowing *cut* facility.

3. Maria then pastes the cut portion into *Document B*.

   a. *Document B* negotiates the transfer of the cut data with *Document A*.

   b. If *Document B* does not understand any of the types offered by *Document B*, it requests a *tagged media type*. *Document B* uses the tagged media type to broadcast a ToolTalk message requesting a translation of the media type to a media type it understands.

   c. A registered translation utility accepts the request and returns the translated version of the media type to *Document B*.

   d. The paste of the translated data into *Document B* is performed.

Key Benefits of the ToolTalk Document and Media Exchange Message Set

The ToolTalk Document and Media Exchange Message Set offers developers two key benefits:

1. Ease of multimedia integration to new and existing software.

   Adding multimedia functionality to any application is now vastly simplified. The ToolTalk Document and Media Exchange Message Set allows you to use other developers' multimedia technologies, thus reducing your development time and expenses while increasing your system functionality.

2. Creates a framework that extends the range of end-user solutions.

   By facilitating application inter-operability, the ToolTalk Document and Media Exchange Message Set allows end-users and other developers to create new vertical solutions. These solutions, in turn, create new opportunities for your products by opening markets that were previously beyond their scope.

General Description of the ToolTalk Document and Media Exchange Message Set

The ToolTalk Document and Media Exchange Message Set is composed of several request messages and one notification message, listed in Table 1.

Table 1 ToolTalk Document and Media Exchange Message Set

Requests
Abstract
Deposit
Display
Edit
Interpret
Print
Translate

These messages are oriented towards creating, editing, and using documents of a certain media type. Common parameters to these messages are *mediaType*, *component*, *locator*, and *region*, as described in Table 2.

Table 2 Common Parameters to ToolTalk Document and Media Exchange Message Set

Parameter	**Description**
mediaType	The name of a media format.

Table 2 Common Parameters to ToolTalk Document and Media Exchange Message Set

Parameter	Description
vector	A string vtype describing a distance and a direction in a document. The syntax of vectors varies by abstract mediaType.
locator	A string describing a location in a document. The syntax of locators varies by abstract mediaType, but should usually be a superset of vector syntax.
flat text	A family of mediaTypes (such as ISO_Latin_1) which consist of a sequence of characters from some character set. Legal vectors for flat text are lineVec ::= Line:[-][0-9]+ charVec ::= Character:[-][0-9]+ vector ::= <lineVec> vector ::= [<lineVec>,]<charVec> Legal locators for flat text are vectors.

See the "The ToolTalk Media Exchange Protocol" section for a complete description of these messages.

The ToolTalk Service

As computer users increasingly demand that independently developed applications work together, inter-operability is becoming an important theme for software developers. By cooperatively using each other's facilities, inter-operating applications offer users capabilities that would be difficult to provide in a single application. The ToolTalk service is designed to facilitate the development of inter-operating applications that serve individuals and their work groups.

The ToolTalk service, from SunSoft, enables independent applications to communicate with other applications without having direct knowledge of each other. Applications create and send ToolTalk messages to communicate. Applications pass these messages to the ToolTalk service, which then determines the recipients and delivers the messages to one or more appropriate applications, as shown in Figure 1.

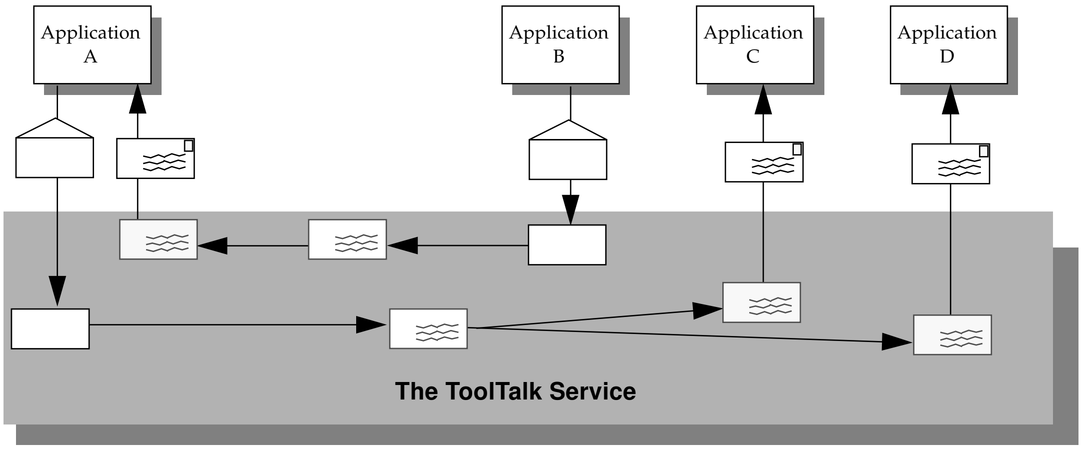

Figure 1 Applications Using the ToolTalk Service

Developers of inter-operating applications agree upon a set of messages and then use the ToolTalk service to deliver the messages. The set of messages exchanged by a group of inter-operating applications is called a *protocol*; a protocol forms a small, well-defined interface that maximizes application autonomy.

The message protocol specification includes the set of messages and how applications should behave when they receive the messages. If protocols are observed, cooperating applications can be modified, even replaced, without affecting one another. An application written or modified to observe a protocol is inherently compatible with any other applications that observe the same protocol.

The ToolTalk service supports several messaging styles. A sender can address a ToolTalk message to a particular process, to any interested process, to an object, or to an object type. Message senders are not concerned with the locations of processes and objects in a local area network; the ToolTalk service automatically finds the receiving processes and objects.

General ToolTalk Development Guidelines and Conventions

Sun Microsystems, Inc. encourages *open protocols*. A protocol is open largely to the extent that it contains *anonymous message* (that is, messages that are sent without knowledge of who is to receive them). This section provides guidelines to help you independently develop applications that will successfully interact with any other application that supports the message protocol. These guideline and principles help ensure that two independently-developed applications will be able to initiate and maintain conventions; and, thus, interact with each other. By following these guidelines, you will enable users of your application to better control and customize their environment.

When you write a ToolTalk application, you need to follow these principles:

1. Always make requests anonymous.
2. Let tools be started as needed.
3. Reply to a request only when the requested operation has been completed.
4. Avoid statefulness whenever possible.
5. Declare one ptype for each role a tool can play.

Always Make Anonymous Requests

To design your application to be completely open, you want the requests to be completely anonymous. That is, the requesting process has no knowledge of which tool instance — or even which tool type — will perform the requested operation. If the requests are sent to a specific process, you unnecessarily restrict how users or potential message recipients can utilize their resources. If the requests are sent to a specific tool type, you unnecessarily restrict the other kinds of tools that can interact with your tool.

You want your message to describe the operation being requested or the event being reported. You do not want your message to describe the process that should receive the message. The less specific knowledge each tool encodes about the tools with which it will interact, the more flexible the overall system is for the user.

For more information about open protocols, see “Designing and Writing a ToolTalk Procedural Protocol” (SunSoft Part Number 801-3592-01).

Let Tools Be Started as Needed

To design your protocol to be completely open, you want the system to start tools only as needed. When you let a new tool instance be started only as needed, you provide the user with more flexibility and more efficient use of resources such as CPU, screen real estate, and swap space. The ToolTalk service has several features that assume the responsibility of determining when to start a new tool instance:

- The ToolTalk service allows messages and type signatures to have "start" reliability. Start reliability means that if no eligible recipient of a message is running (or willing to accept the request), the ToolTalk service will start an instance of the type of tool which is statically registered to handle or observe that message.
- The ToolTalk service allows each process type (*ptype*) to specify the maximum number of its instances that may be started in a given session.
- The ToolTalk service offers each request to all eligible running handlers before it starts a new tool instance. An eligible handler can accept or reject a request based on its own criteria (such as its ability to take on a new task; whether or not it has unsaved changes; idle time; iconic state; or whether or not the user has indicated that the tool is free to accept new work).

Reply When Operation has been Completed

To design your application to be completely open, you want to notify the sending process that its requested operation has been performed. However, the operation invoked by a request sometimes takes a relatively long time to complete compared to the very brief time it takes to send the message. Since the sending process is expecting a reply, your tool can respond in two ways:

1. It can reply immediately that it has received the request and then convey the actual results of the completed operation in a later message.
2. It can withhold the reply until the operation has been completed.

We recommend the second policy because ToolTalk messaging is entirely asynchronous: neither a tool (nor the session it is in) is blocked because it has one or more requests outstanding.

Avoid Statefulness Whenever Possible

To design your application to be open, you want each message to make sense by itself whenever possible. When a protocol is stateless, the messages in it avoid dependency on any previous messages or on some state in the assumed recipient.

Declare One Process Type per Role

A ToolTalk protocol is expressed in terms of the *roles* that each tool plays (that is, the kinds of tasks each tool is assigned to perform). A ToolTalk ptype essentially instructs the ToolTalk service how to handle any messages in which a tool is interested that are sent when that tool is not running. To design your protocol to be open, you want to declare one ptype for each role in your protocol. When you declare only one ptype per role in your protocol, you provide users with the flexibility to interchange tools as their needs require. For example, a user may want a sophisticated sound-authoring tool for recording but also prefers a simple audio tool to perform the playback.

Thus, you will sometimes want to include only one message signature per ptype. When you include more than one message signature in the same ptype, you are requiring that any program that can handle one message can handle the other messages. For example, a ptype “UWriteIt” can include the two message signatures “Display“ and “Edit” because it is expected that any tool that understands the UWriteIt document format can perform both of these operations.

Developing ToolTalk Applications

Developing ToolTalk aware-applications is a design process. You can enable your application to send and receive ToolTalk messages in a simple three-step process:

1. Determine how your application is to interact with other applications, and with users.
2. Select messages and define their use within the context of your application
3. Integrate ToolTalk calls and messages into your code.

Note – A demonstration of how your can easily add ToolTalk capability to your existing applications has been integrated with the ToolTalk software product. This demonstration is described in the paper entitled *Tool Inter-Operability: A Hands On Demonstration* (SunSoft Part Number 801-3593-01) and is part of the ToolTalk information pack available from SunSoft or your local Sun Sales office.

♦ **Define how the tools will work together and what operations must be performed.**

A clear understanding of what types of communications your application will require is a critical factor in successful application integration. The best approach to analyze this issue is to define scenarios that represent how your application will be used. From these scenarios you will be able to determine what interaction needs to take place and what information needs to be exchanged. Detailed scenarios that show exactly what information and status is being passed will greatly help you integrate messaging into your application.

♦ **Select the appropriate messages that accomplish these tasks.**

Once you have determined how your applications will interact with other applications and users, you must determine the specific messages needed to accomplish the required tasks.

First, look at the standard message sets available from industry groups such as SunSoft, ANSI, X3H6, and CFI. Use of these messages is strongly recommended for two reasons.

1. The standard messages provide your application with a well-known and documented interface. This interface allows other developers to independently develop applications that can interface with your work. In addition, it provides an interface around which your customers can build integrated systems.

2. The standard message sets provide your application with the “universal plug-and-play” capability. This capability allows you to provide your customers with the flexibility to use multiple applications to provide a service. By giving your customers a choice of applications to use, they can pick the best tool for a particular job and you are not forced to offer features that you feel your product does not need.

If the standard message sets do not support your design, then you will need to develop custom messages.

If you use non-standard message, please contact the Document and Media Exchange Messaging Alliance at *media_exchange@Sun.Com* so we can consider adding your new messages to the standard message sets.

♦ **Integrate ToolTalk calls and messages into your application.**

Once you have completed the design aspect, you are ready to add the ToolTalk capabilities into your application.

First, you need to include the ToolTalk header file in all files that will use ToolTalk API calls. You will also need to register and initialize the patterns that control the sending and receiving functions. For detail information about registering and initializing patterns, see the book entitled "The ToolTalk Service: An Inter-Operability Solution." (The book is available in bookstores, through SunSoft Press, and directly from Prentice Hall.)

Next, add the ability to send ToolTalk messages to your code. Based on the knowledge gained from designing the scenarios, it is very straight forward to determine what routines need to send what messages, and what the arguments for each message should be.

Once the ToolTalk service is initialized, your application uses the ToolTalk API calls to create and fill in messages to be sent to other applications.

- If your applications uses a windowing system, you only need to add the calls to activate the ToolTalk service in the event polling loop.
- If your application does not already use a polling loop, you need to create a simple loop that periodically checks for messages. For detailed information, see the book entitled "The ToolTalk Service: An Inter-Operability Solution." (The book is available in bookstores, through SunSoft Press, and directly from Prentice Hall.)

The ToolTalk Document and Media Exchange Message Set

This section contains a description of each of the messages which constitute the ToolTalk Document and Media Exchange Message Set. Each message described in this section contains the information described in Table 3. In addition the next sections describe common information to all messages. These sections describe ToolTalk unique definitions and error messages common to all messages.

General Tooltalk Message Definitions and Conventions

In the ToolTalk messages there are terms used with specific ToolTalk definitions. This section defines these terms and conventions used in the ToolTalk message man pages.

Table 3 ToolTalk Document and Media Exchange Message Set Descriptions

Type of Information	**Description**
header	A single line that describes the message in the following format: *MsgName*(*Tt_class*) where *MsgName* is the name of the message and *Tt_class* is either Request or Notice.
name	The name of the message and a one-line description of the message.
description	An explanation of the operation (event) that the message requests (announces).

Table 3 ToolTalk Document and Media Exchange Message Set Descriptions *(Continued)*

Type of Information	**Description**
synopsis	A representation of the message in the ToolTalk types-file syntax (similar to the syntax understood by the ToolTalk type compiler `tt_type_comp`) in the following format: <fileAttrib> <opName> (<requiredArgs> [<optionalArgs>]); A synopsis entry is given for each interesting variant of the message. <fileAttrib> - An indication of whether the file attribute of the message can/should be set. <opName> - The name of the operation or event is called the "op name" (or "op"). It is important that different tools not use the same opName to mean different things. Therefore, unless a message is a standard one, its opName should be made unique. A good way to do this is to prefix it with: <Company><Product> e.g., "Acme_Hoarktool_My_Frammistat". <requiredArgs>, <optionalArgs> - The arguments that must always be included in the message. A particular argument is described in the following format: <mode> <vtype> <argument name> where *mode* is one of "in","out", or "inout", *vtype* is a programmer-defined string that describes what kind of data a message argument contains; and *argument name* is the name of the argument. The ToolTalk service uses vtypes to match sent message instances with registered message patterns. By convention, a vtype maps to a single, well-known data type.

Table 3 ToolTalk Document and Media Exchange Message Set Descriptions *(Continued)*

Type of Information	Description
required arguments	The arguments that must always be in the message. <vtype> <argumentName> A description of a particular argument. A ‘vtype’ is a programmer-defined string that describes what kind of data a message argument contains. ToolTalk uses vtypes for the sole purpose of matching sent message instances with registered message patterns. Every vtype should by convention map to a single, well-known data type. The data type of a ToolTalk argument is either integer, string, or bytes. The data type of a message or pattern argument is determined by which ToolTalk API function is used to set its value. The argument name is merely a comment hinting to human readers at the semantics of the argument, much like a parameter name in a C typedef.
optional arguments	The extra arguments that may be included in a message. Unless otherwise noted, any combination of the optional arguments, in any order, may be appended to the message after the required arguments.
description	An explanation of the operation that the request entreats, or the event that the notice announces.
errors	A list of the error codes that can be set by the handler of the request (or the sender of the notice).

Edict

An *edict* is a notice that looks like a request. If a request returns no data (or if the sender does not care about the returned data), it can sometimes be useful to broadcast that request to a set of tools. Since the message is a notice, no data is returned, no replies are received, and the sender is not told if any tool gets the message.

Handler

The *handler* is the distinguished recipient procid of a request. This procid is responsible for completing the indicated operation.

Notice

A *notice* is a message that announces an event. Zero or more tools may receive a given notice. The sender does not know whether any tools receive its notice. A notice cannot be replied to.

Procid

A *procid* is a principal that can send and receive ToolTalk messages. A procid is an identity, created and handed over by the ToolTalk service on demand (via tt_open()), that a process must assume in order to send and receive messages. A single process can use multiple procids; and a single procid can be used by a group of cooperating processes.

Request

A request is a message that asks an operation to be performed. A request has a distinguished recipient, called a handler, who is responsible for completing the indicated operation. A handler may fail, reject, or reply to a request. Any number of handlers may reject a request but ultimately only one handler can fail it or reply to it. If no running handler can be found to accept a request, the ToolTalk service can automatically start a handler. If no willing handler can be found, or if a handler fails the request, then the request is returned to the sender in the 'failed' state.

Errors

A `Tt_status` code can be read from a reply via `tt_message_status()`. This status defaults to `TT_OK`, or can be set by the handler via `tt_message_status_set()`. In extraordinary circumstances (such as no matching handler) the ToolTalk service itself sets the message status.

In addition to the `Tt_status` values defined by the ToolTalk API, the overview reference page for each set of messages lists the error conditions defined for that set of messages. For each error condition, the overview reference page provides

- Its name
- Its integer value
- A string in the "C" locale that explains the error condition

Since the ToolTalk Inter-Client Conventions (TICC) are a binary message interface, the integer and string are part of that binary interface; the name is not.

- The string may be used as a key in the `SUNW_TOOLTALK_INTERCLIENTCONVENTIONS` domain to retrieve a localized explanation of the error condition. See `dgettext(3)`.
- The integer values of these status codes begin at 1537 (`TT_ERR_APPFIRST + 1`). The first 151 codes correspond to the system error list defined in `intro(2)`.

A standard programming interface for these conventions that binds the name to the integer value does not yet exist.

The ToolTalk service allows an arbitrary status string to be included in any reply. Since a standard localized string can be derived for each status code, this status string may be used as a free-form elucidation of the status. For example, if a request is failed with `TT_DESKTOP_EPROTO`, the status string could be set to `"The vtype of argument 2 was 'string'; expected 'integer'"`. Handling tools should try to compose the status string in the locale of the requestor. See the `Get_Locale()` request.

Media Exchange Definitions and Conventions

Specific to the media exchange messages there are values associated with fields. The following paragraphs define those fields.

Media exchange messages are sent and received by tools that display or edit some kind of media. The parts of a Media Exchange message is defined as follows:

<document>
A vector of bytes with an associated mediaType.

<mediaType>
The name of a media format. The mediaType allows messages about that document to be dispatched to the right editor. Standard mediaTypes include:

• EPS		
• GIF	Graphics Interchange Format	Compuserve
• ISO_Latin_1	ISO 8859-1 (+tab+newline)	ISO

•	JPEG		ISO/CCITT
•	JPEG_Movie		Parallax Graphics
•	PostScript	Postscript Lang Ref. Manual	Adobe
•	RTF	MS Word Technical Ref	Microsoft
•	Sun_Audio	audio_intro(3),audio_hdr(3)	Sun Microsystems
•	Sun_CM_Appointment		Sun Microsystems
•	Sun_Raster	Rasterfile(5)	Sun
•	TIFF	"TIFF Rev. 5" Technical Memo	Aldus/Microsoft
•	RFC_822_Message	RFC 822	NIC
•	Unix_Mail_Folder		
•	XPM	XPM --The X PixMap Format	Groupe Bull

Note – The mediaType list will be extended as required. You can extract a list of the installed mediaTypes from the ToolTalk Types Database.

abstract mediaType
: A family of similar mediaTypes, such as flat text or structured graphics.

vector
: A string vtype describing a distance and a direction in a document. The syntax of vectors varies by abstract mediaType.

locator
: A string describing a location in a document. The syntax of locators varies by abstract mediaType, but should usually be a superset of vector syntax.

flat text
: A family of mediaTypes (such as ISO_Latin_1) which consist of a sequence of characters from some character set.

Legal vectors for flat text are:

```
lineVec ::= Line:[-][0-9]+
charVec ::= Character:[-][0-9]+
vector ::= <lineVec>
vector ::= [<lineVec>,]<charVec>
```

Legal locators for flat text are vectors.

Errors

These definitions are common to all messages. Any differences or additions will be noted in the man pages.

`1700 TT_MEDIA_ERR_SIZE`

The specified size was too big or too small.

`1701 TT_MEDIA_ERR_FORMAT`

The data do not conform to their alleged format.

`1702 TT_MEDIA_NO_CONTENTS`

The message neither contains nor refers to any document.

Abstract(Request)

Requests a summary representation of a document.

Synopsis

```
[file] Abstract (in     <mediaType>    contents,
                  out      <mediaType>   output
                  in       boolean       inquisitive,
                  in       boolean       covert
                 [in       messageID    counterfoil]
                 [inout    vector       size] );
```

Description

The Abstract message requests that a summary representation of a document (for example, an icon or a video frame raster) be returned. The abstraction is the best possible representation of the document within the size constraints of the sending tool.

Note – You can extract a list of the installed mediaType-to-mediaType mappings from the ToolTalk Types Database.

Required Arguments

<mediaType> contents
The contents of the document.

If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

<mediaType> output
The abstracted document.

boolean inquisitive
The boolean value that indicates whether the recipient is allowed to seek user input about interpretation options.

Note – However, even if this value is *true*, the recipient is not required to seek the input.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

boolean covert
The boolean value that indicates whether the recipient is allowed to make itself apparent to the user as it performs the interpretation.

Note – However, even if the value is *false*, the recipient is not required to make itself apparent.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

Optional Arguments

messageID counterfoil
A unique string created by the message sender, typically by concatenating a procid and a counter. The sending application includes this argument if it anticipates a need to communicate with the handler about this request before the request is completed; for example, you could include this argument to cancel the request.

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

vector size

- On input, the maximum size of the abstraction. The recipient returns an abstraction as close to this size as possible without exceeding this size.
- On output, the actual size of the abstraction to be returned; or, if the error `TT_MEDIA_ERR_SIZE` is returned, the smallest possible size the recipient is capable of returning.

Examples

In this scenario, a container application requires a representation of some video data. To abstract a representation frame of the video tool, you could send an Abstract request such as:

```
Abstract (in Acme_Video, out Sun_Raster output, ...);
```

to obtain a custom raster representation; or

```
Abstract (in Acme_Video, out Sun_XView_Icon output, ...);
```

to obtain a generic icon representation. In either case, the container application does not need to understand the Acme_Video format.

Errors

`1700 TT_MEDIA_ERR_SIZE`

The specified size was too big or too small

`1701 TT_MEDIA_ERR_FORMAT`

The data do not conform to their alleged format

`1702 TT_MEDIA_NO_CONTENTS`

The message neither contains nor refers to any document

Deposit(Request)

Saves the document to its backing store.

Synopsis

```
[file] Deposit(      in        <mediaType>  contents
                    {in        bufferID      beingDeposited
                     in        messageID     commission} );

[file] Deposit(      in        <mediaType>  contents,
                     out       bufferID      beingDeposited
                    [in        title          docName] );
```

Description

Save this document to its backing store. This request is different from the Save() request, because here the requestor (and not the handler) has the data that needs to be written. Deposit() should almost never be file-scoped, because if the sending tool knows what file the document belongs in, that tool should be able to perform the save itself.

Required Arguments

<mediaType> contents

The contents of the document.

If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

bufferID beingDeposited
messageID commission

The Identifier of the buffer to be deposited to backing store. The identifier is either a bufferID returned or the messageID of the edit request that created this buffer.

If the `beingDeposited` argument is an `out` parameter, a new document is created and the handling container application must save the document and return a new bufferID for it.

Optional Arguments

title docName
The name of the document.

Example

This request is especially useful for when the user checkpoints (e.g., via a "Save" menu item) her modifications to a document that is the subject of a purely-session-scoped Edit request in progress.

The second variant of this request can be issued by editors that allow the user to create, as an afterthought, extra documents 'near' the document that was just edited. This can be useful if the each document in the series can serve as the template or starting point for the next document. Of course, if the handling container application does not support the notion of accomodating uninvited documents, it should reject the request.

Errors

`TT_DESKTOP_ENOENT`

The file that was alleged to contain the document does not exist.

`TT_MEDIA_NO_CONTENTS`

The in-mode contents arg had no value and the file attribute of the message was not set.

`TT_MEDIA_ERR_FORMAT`

See general info for description.

Display(Request)

Displays a document.

Synopsis

```
[file] Display( in <mediaType>   contents
                [in   messageID   counterfoil]
                [in   title       docName   ] );
```

Description

The Display message requests that a document be displayed. *Display* is a generic term for the operation the player performs; for example, an audiotool displays sound. The Display request invokes the requested playback mechanism (such as a video tool, or an audio tool). The receiving tool decides:

- when the display operation is complete.
- what user gesture signals that the display is completed (that is, what determines that the user has signaled "I have completed the display.").
- the action it takes after it has replied to the request.

Note – The display request does not allow changes to be saved back to the source data; however, a tool that supports a "save as" operation may allow edits to be saved back to the document.

Required Arguments

<mediaType> contents
The contents of the document. If this argument is empty (i.e., has a value of (char *)0), then the contents of the document are in the file named in the message's file attribute. The data type of the contents argument shall be string, unless nulls are legal in the given mediaType, in which case the data type shall be bytes.

Optional Arguments

messageID counterfoil

The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

---

title docName

The name of the document.

Examples

To display a PostScript document, send a Display request with a first argument whose vtype is "PostScript", and whose value is a vector of bytes such as "%!^J/inch {72 mul} def...". (By "^J" here we mean the newline character, octal 12.)

To display a PostScript document contained in a file, send a Display request, scoped to that file, with a first argument whose vtype is "PostScript", and whose value is not set.

Errors

`TT_DESKTOP_ENOENT`

The file that was alleged to contain the document does not exist.

`TT_MEDIA_NO_CONTENTS`

The in-mode contents arg had no value and the file attribute of the message was not set.

`TT_MEDIA_ERR_FORMAT`

See general info for description.

Edit(Request)

Edits or composes a document.

Synopsis

```
[file] Edit ( [in]out <mediaType>  contents
                [in  messageID  counterfoil]
                [in  title      docName  ] );
```

Description

The Edit message requests that a document be edited and a reply containing the new contents be returned when the editing is completed. The receiving tool decides:

- when the edit operation is complete.
- what user gesture signals that the edit is completed (that is, what determines that the user has signaled "I have completed the edit.").
- the action it takes after it has replied to the request.

If a tool supports a "save" or "checkpoint" operation during editing, it can send a Deposit request back to the tool that requested the edit.

Required Arguments

<mediaType> contents
: The contents of the document. If the message is file-scoped, the contents argument has no value, and the document is contained in the scoped file. The data type of the contents argument is *string* unless nulls are legal in the given mediaType; if nulls are legal, the data type is *bytes*. If the contents argument is mode `out`, a new document is to be composed and its contents to be returned in this argument.

Optional Arguments

messageID counterfoil

The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

---

title docName

The name of the document.

Examples

To edit an X11 “xbm” bitmap, send an Edit request with a first argument whose vtype is “XBM”, and whose value is a a string such as “#define foo_width 44^J#define foo_height 94^J...”. (By “^J” here we mean the newline character, octal 12.)

To edit an X11 “xbm” bitmap contained in a file, send an Edit request, scoped to that file, with a first argument whose vtype is “XBM”, and whose value is not set.

Errors

`TT_DESKTOP_ENOENT`

The file that was alleged to contain the document does not exist.

`TT_MEDIA_NO_CONTENTS`

The in-mode contents arg had no value and the file attribute of the message was not set.

`TT_MEDIA_ERR_FORMAT`

Interpret(Request)

Translates a document and displays the translation.

Synopsis

```
[file] Interpret(  in        <mediaType>  contents,
                   in        <mediaType>  targetMedium,
                   in        boolean      inquisitive,
                   in        boolean      covert
                  [in        messageID    counterfoil]
                  [in        title        docName ] );
```

Description

The Interpret message translates a document from one media type to another and displays the translation.

Note – The translation is the best possible representation of the document in the target media type; however, it is possible that the resulting representation cannot be perfectly translated back into the original document.

The Interpret request is equivalent to issuing a Translate request followed by a Display request. The Interpret message is a useful optimization when the sender has no interest in retaining the translation.

Note – It is possible to extract from the ToolTalk types database a list of the installed Translate() mediaType-to-mediaType mappings.

Required Arguments

<mediaType> contents
The contents of the document.

If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

<mediaType> targetMedium
An empty argument whose vtype indicates the mediaType into which the document is to be translated before it is displayed.

boolean inquisitive
The boolean value that indicates whether the recipient is allowed to seek user input about interpretation options.

Note – However, even if this value is *true*, the recipient is not required to seek the input.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

boolean covert
The boolean value that indicates whether the recipient is allowed to make itself apparent to the user as it performs the interpretation.

Note – However, even if the value is *false*, the recipient is not required to make itself apparent.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

Optional Arguments

messageID counterfoil
The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

---

title docName
The name of the document.

Examples

Text-to-Speech Translation

To request a string to be spoken, send an Interpret request such as the following:

```
Interpret( in ISO_Latin_1 contents, in Sun_Audio targetMedium )
```

ToolTalk will then pass this request to the appropriate third party server in your environment.

Errors

`TT_DESKTOP_ENOENT`

The file that was alleged to contain the document does not exist.

`TT_MEDIA_NO_CONTENTS`

The in-mode contents arg had no value and the file attribute of the message was not set.

`TT_MEDIA_ERR_FORMAT`

See general description for definition.

Print(Request)

Prints a document.

Synopsis

```
[file] Print( in       <mediaType>  contents,
              in       boolean       inquisitive,
              in       boolean       covert
             [in       messageID    counterfoil]
             [in       title         docName ] );
```

Description

The Print message prints a document. In effect, the recipient assumes the user issued a “print...” command via the recipient's user interface. The recipient tool decides issues such as what it should do with itself after replying.

Required Arguments

<mediaType> contents
The contents of the document.

If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

boolean inquisitive
The boolean value that indicates whether the recipient is allowed to seek user input about interpretation options.

Note – However, even if this value is *true*, the recipient is not required to seek the input.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

boolean covert

The boolean value that indicates whether the recipient is allowed to make itself apparent to the user as it performs the interpretation.

Note – However, even if the value is *false*, the recipient is not required to make itself apparent.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

Optional Arguments

messageID counterfoil

The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

---

title docName

The name of the document.

Examples

Printing a PostScript Document

To print a PostScript document,

```
Print( in PostScript  contents,
       in boolean     inquisitive,
       in boolean     covert)
```

where the first argument is vtype `PostScript` whose value is a a vector of bytes.

Printing a PostScript Document Contained in a File

To print a PostScript document contained in a file,

```
Print( in PostScript  contents,
       in boolean     inquisitive,
       in boolean     covert)
```

where the file attribute is set to filename, and the first argument is vtype `PostScript` whose value is not set.

Errors

`TT_DESKTOP_ENOENT`

The file that was alleged to contain the document does not exist.

`TT_MEDIA_NO_CONTENTS`

The in-mode contents arg had no value and the file attribute of the message was not set.

`TT_MEDIA_ERR_FORMAT`

Translate(Request)

Translates a document from one media type to another media type.

Synopsis

```
[file] Translate(       in       <mediaType>  contents,
                         out      <mediaType>  output,
                         in       boolean      inquisitive,
                         in       boolean      covert
                        [in       messageID    counterfoil] );
```

Description

The Translate message requests that a document be translated from one media type to another media type and that a reply containing the translation be returned. The translation is the best possible representation of the document in the target media type; however, it is not guaranteed that the resulting translation can be perfectly translated back into the original document.

Note – You can extract a list of the installed mediaType-to-mediaType mappings from the ToolTalk Types Database.

Required Arguments

<mediaType> contents
The contents of the document.

If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

<mediaType> output
The translated document.

boolean inquisitive

The boolean value that indicates whether the recipient is allowed to seek user input about interpretation options.

Note – However, even if this value is *true*, the recipient is not required to seek the input.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

boolean covert

The boolean value that indicates whether the recipient is allowed to make itself apparent to the user as it performs the interpretation.

Note – However, even if the value is *false*, the recipient is not required to make itself apparent.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

Optional Arguments

messageID counterfoil

The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

Examples

Speech-to-Text Translation

To translate speech to text, send a Translate request such as the following:

```
Translate (in Sun_Audio contents, out ISO_Latin_1 output);
```

Optical Character Recognition (OCR)

To translate a GIF format bit image to text, send a Translate request such as the following:

```
Translate (in GIF contents, out ISO_Latin_1 output);
```

Errors

`TT_DESKTOP_ENOENT`

The file that was alleged to contain the document does not exist.

`TT_MEDIA_NO_CONTENTS`

The in-mode contents arg had no value and the file attribute of the message was not set.

`TT_MEDIA_ERR_FORMAT`

Future Direction

The ToolTalk Document and Media Exchange Message Set will evolve as new requirements arise. It leverages all of the existing media formats because it allows an application to specify the format of the data within the message. The ToolTalk Document and Media Exchange Message Set, therefore, will allow improved, or new, file formats — such as those under development by the Interactive Media Association (IMA) — to be used in the future as easily as a currently existing file format.

The ToolTalk Document and Media Exchange Message Set will create new business opportunities for developers and users of multimedia technologies by bringing multimedia technologies to more traditional UNIX markets. Developers of non-multimedia technologies will benefit from the ability to add multimedia to their products quickly without having to become multimedia technology experts.

The ToolTalk Document and Media Exchange Message Set is an *open* specification with no royalty or license fees. It will be continuously revised and modified to meet the expanding needs of multimedia developers and users. Send questions, comments, and requests for information to the Document and Media Exchange Messaging Alliance at *media_exchange@Sun.Com*.

The ToolTalk Document and Media Exchange Message Set: Release 1.1.1

A White Paper

Media Exchange

SunSoft

A Sun Microsystems, Inc. Business

Revision History

Revision	Dash	Date	Comments
92204	–001	July 1992	Initial Release
92204	-002	February 1993	Add new messages : • Interpret • Print Updated all Rev 1.0 messages

Table of Contents

SunSoft, Inc.
2550 Garcia Avenue
Mountain View, CA 94043

For more information, call 1 800 227-9227.

Printed in USA 2/93 92204-002 .5K