The ToolTalk Document and Media Exchange Message Set

Dan Clark, 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 Microcsystems® 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" 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 four request messages and one notification message, listed in Table 1.

Table 1 ToolTalk Document and Media Exchange Message Set

Requests	**Notices**
Abstract	
Deposit	
Display	
Edit	
Translate	

These five 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. The *mediaType* of a document or its components allows messages to be dispatched to the correct editor.
component	The label given to a unit of the document used to navigate around the document; for example, *chapter* or *page*.
locator	The description of a location in a document. The *locator* consists of pairs of components and component identifiers.
region	The portion of a document bounded by a series of locators.

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.

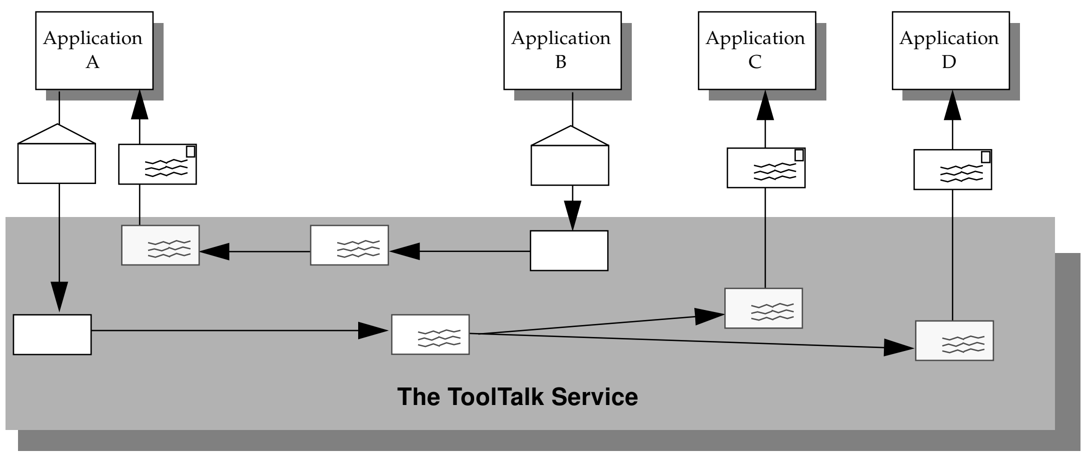

Figure 1 Applications Using the ToolTalk Service

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 92199-001).

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.

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.

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*(*MsgKind*) where *MsgName* is the name of the message and *MsgKind* is either Request or Notice.
name	A one-line description of the message.
description	An explanation of the operation (event) that the message requests (announces).
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: `<scope> <msgName>( <requiredArgs> /* optionalArgs */);`
required arguments	The arguments that must always be included in the message. A particular argument is described in the following format: <vtype> <argument name> where *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.
optional arguments	Additional arguments which may be included in a message.

Editor messages are sent and received by tools that display or edit some kind of media. The parts of an editor 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:

- ISO_Latin_1
- PostScript
- RTF
- MIF
- WKS
- EPS
- GIF
- TIFF
- X_Pixmap
- X_Bitmap
- Sun_Raster
- Sun_XView_icon
- Sun_Audio
- JPEG_Movie

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

<component>
The label given to a unit of the document. A component can be one of the following:

- Chapter
- Section
- Page
- Paragraph

- Sentence
- Line
- Word
- Character
- InsertionPoint
- Link
- Time
- X
- Y
- Z
- Row
- Column

Note – Not all components apply to all mediaTypes. The component list will be extended as required.

<locator>

A description of a location in a document. The locator consists of pairs of components and component-identifiers in the following format:

```
<locator>::= <locator>,<locator>
<locator>::= <component>:<componentID>
<componentID>::= [0-9]+[.[0-9]*]
<componentID>::= "string"[in string, \\ ::= \, \" ::= "]
```

The following example describes the location of a character:

```
Character:862
Page:4,Line:17,Character:48
Section:"The \"Magic Bullet\" Theory"
Time:0.03333
```

<region>
The portion of a document bounded by a series of locators in the following format:

```
<region>::= <locator>[;<locator>]+
```

Abstract(Request)

Requests a summary representation of a document.

Synopsis

```
[file both session] Abstract ( in    <mediaType>   contents,
                                out    mediaType>   output
                               [in     messageID    msgID,
                                inout  region        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 the message is scoped to file, 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*.

<mediaType> output
  The abstracted document.

Optional Arguments

messageID msgID

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.

region size

The *size* of the abstraction. The error `EDITOR_ERR_SIZE` is returned if the requested size is too small.

- On input, is the maximum size of the abstraction. The returned abstraction must not exceed this size.
- On output, is the actual size of the abstraction that is being returned; if the error `EDITOR_ERR_SIZE` is returned, the indicated region size is 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 reresentation; 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.

Deposit(Request)

Saves the document to its backing store.

Synopsis

```
[file both session] Deposit ( in <mediaType> contents
                              in   string     bufferID );
```

Description

The Deposit message requests that the receiving tool save the document to its backing store. For example, you can use this message to save checkpointed modifications to a document that is the subject of an Edit request in progress.

Required Arguments

string bufferID

An editor-assigned identifier for a volatile, non-shared representation (for example, in-memory) of a document. The bufferID is either the identifier returned when this buffer was created by an Open request or the messageID of the Edit request that created this buffer.

Optional Arguments

<mediaType> contents

The contents of the document. Include this argument when the buffer was created implicitly.

Example

To save checkpoint changes made to a document that is the subject of an Edit request in progress, send a Deposit request such as the following:

```
Deposit (in WKS contents);
```

Display(Request)

Displays a document.

Synopsis

```
[file both session] Display( in <mediaType>   contents
                            [in   messageID   msgID,
                             in   string       title ] );
```

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, an audio tool, or a text-to-speech synthesizer). 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 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*.

Optional Arguments

messageID msgID

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

string title

The name of the document.

Edit(Request)

Edits or composes a document.

Synopsis

```
[file both session] Edit ( [in]out <mediaType>   contents
                           [in      messageID    msgID ]
                           [in      string        title ]   );
```

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 theedit.”).
- 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 msgID

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 be able to later cancel the request.

string title

The name of the document.

Translate(Request)

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

Synopsis

```
[file both session] Translate ( in    <mediaType>   contents,
                                out   <mediaType>    output
                               [in     messageID     msgID ] );
```

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 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*.

<mediaType> output
: The translated document.

Optional Arguments

messageID msgID

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 be able to later cancel the request.

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 optical characters to text, send a Translate request such as the following:

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

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 1.0

A White Paper

Media Exchange

SunSoft

A Sun Microsystems, Inc. Business

Table of Contents

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

For more information, call 1 800 227-9227.

Printed in USA 7/92 92204-001 .5K