The ToolTalk Standard Message Sets

Standard message sets help developers to develop applications that will automatically integrate with applications developed by others that follow the same message protocols. Extensive work has been done with leading software suppliers and end-users to define standard message sets. The ToolTalk Standard Message Sets are higher-level interfaces of the ToolTalk API that provide common definitions and conventions to easily achieve control and data integration between applications.

See the *ToolTalk Reference Manual* for a complete description of the standard ToolTalk message sets.

C.1 The ToolTalk Desktop Services Message Set

In order to achieve basic desktop integration, applications need to support a basic set of messages to enable inter-application control. The *ToolTalk Desktop Services Message Set* is the common message set that provides this functionality for all applications. A powerful messaging protocol that benefits both developers and users of desktop applications, the ToolTalk Desktop Services Message Set allows applications to easily interact with other desktop applications. Using the ToolTalk Desktop Services Message Set, applications can communicate with each other in a transparent manner, both locally and over networks.

Why the ToolTalk Desktop Services Message Set was Developed

In order to provide integrated control of applications, certain basic features are needed to launch, halt, control display appearance, and pass information regarding input and output data. All applications need to have these facilities so that other applications in the toolset can inter-change basic control information. This kind of functionality enables the development of smart desktops and integrated smart toolsets. Groups of applications can now call upon each other to perform tasks and to interact as one solution environment for the end-user.

Key Benefits of the ToolTalk Desktop Services Message Set

The ToolTalk Desktop Services Message Set offers developers two key benefits:

1. Allows basic control of applications without direct intervention from the user. Routine or common procedures may be automated for the convenience of the user.

2. Allows tool specialization through a common set of interactions. All ToolTalk aware applications can perform these functions.

C.2 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.

C

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.

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.

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.

C.3 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 C-1 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 C-1 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 C-1 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.

C.4 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.

C.5 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.

C.6 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.)

C.7 *Messaging Alliances*

Send questions, comments, and requests for information to the Desktop Services Messaging Alliance at *ToolTalk_desktop_services@sun.com*.

Send questions, comments, and requests for information to the Document and Media Exchange Messaging Alliance at *media_exchange@Sun.Com*.

C