The ToolTalk Desktop Services Message Set

Ronald Lee, Brian Holtz, and Astrid Julienne

What is 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.

Using the ToolTalk Desktop Services Message Set

The ToolTalk Desktop Services Message Set allows an application to integrate and control other applications without user intervention. This section illustrates two scenarios that show how the Desktop Services Message Set might be implemented.

The Smart Desktop

A common user requirement for a graphic user interface (GUI) front-end is the ability to have data files be aware (or "know") of their applications. To do this, an application-level program is needed to interpret the user's requests. Examples of this application-level program (known as *smart desktops*) are the Apple Macintosh finder, Microsoft Windows File Manager, and the Solaris File Manager. The key common requirements for smart desktops are:

1. Takes a file
2. Determines its application
3. Invokes the application

The ToolTalk Service encompasses additional flexibility by allowing classes of tools to edit a specific data type. The following scenario illustrates how the Desktop Services Message Set might be implemented as a smart desktop transparent to the end-user.

1. **Diane double clicks on the File Manager icon.**
   - The File Manager opens and displays the files in Diane's current directory.
2. **Diane double clicks on an icon for a data file.**
   a. The File Manager requests that the file represented by the icon be displayed. The File Manager encodes the file type in the *display* message.
   b. The ToolTalk session manager matches the pattern in the *display* message to a registered application (in this case, the Icon Editor), and finds an instance of the application running on Diane's desktop.

Note – If the ToolTalk session manager does not find a running instance of the application, it checks the statically-defined ptypes and starts an application that best matches the pattern in the message. If none of the ptypes match, it returns failure to the File Manager application.

---

   c. The Icon Editor accepts the *display* message, de-iconifies itself, and raises itself to the top of the display.
3. **Diane manually edits the file.**

Integrated Toolsets

Another significant application for which the Desktop Services Message Set can be implemented is *integrated toolsets*. These environments can be applied in vertical applications (such as a CASE software developer toolset) or in horizontal environments (such as compound documents). Common to both of these applications is the premise that the overall solution is built out of specialized applications designed to perform one particular task well. Examples of integrated toolset applications are text editors, drawing packages, video or audio display tools, compiler front-ends, and debuggers. The integrated toolset environment requires applications to interact by calling on each other to handle user requests. For example, to display video, an editor calls a video display program; or to check a block of completed code, an editor calls a compiler. The following scenario illustrates how Desktop Services Message Set might be implemented as an integrated toolset:

1. **George is working on a compound document using his favorite editor.** He decides to change some of the source code text.

2. **George double clicks on the source code text.**

   a. The Document Editor first determines the location of the source code. The location is expressed as a file name and a index into the file.

   b. The Document Editor sends an *edit* message request, using the file name as a parameter for the message.

   c. The ToolTalk session manager matches the pattern in the *edit* message to a registered application (in this case, the Source Code Editor), and finds an instance of the application running on George's desktop.

Note – If the ToolTalk session manager does not find a running instance of the application, it checks the statically-defined ptypes and starts an application that best matches the pattern in the message. If none of the ptypes match, it returns failure to the Document Editor application.

---

   d. The Source Code Editor accepts the *edit* message request.

   e. The Source Code editor determines that the source code file is under configuration control, and sends a message to check out the file.

   f. The Source Code Control application accepts the message and creates a read/write copy of the requested file. It then passes the name of the file back to the Source Code Editor.

g. The Source Code Editor opens a window that contains the source file.

3. **George edits the source code text.**

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.

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.

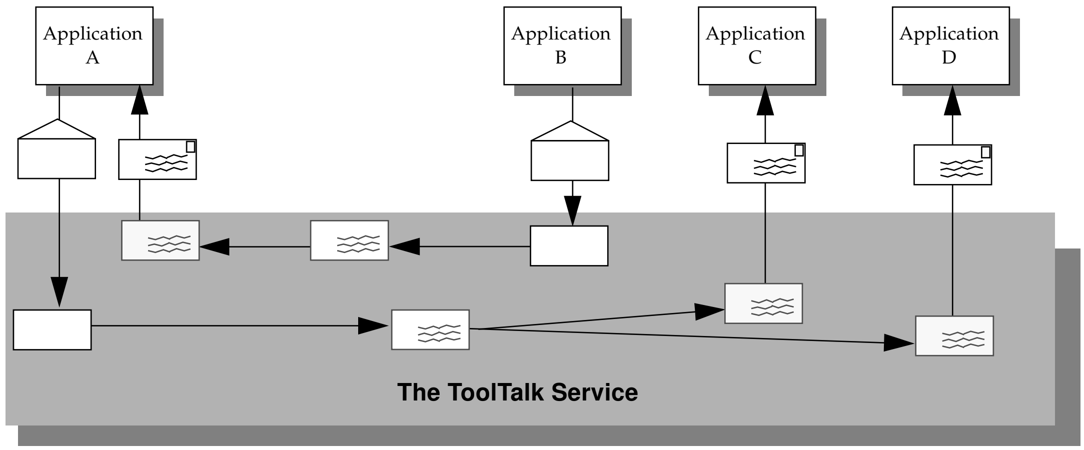


Figure 1 Applications Using the ToolTalk Service

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 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 Desktop Services Messaging Alliance at *ToolTalk_desktop_services@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 Desktop Services Message Set

This section contains a description of each of the generic messages which constitute the ToolTalk Desktop Services Message Set. Each message described in this section contains the information described in Table 1. 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 1 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 1 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 1 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.

General Description of the ToolTalk Desktop Services Message Set

The ToolTalk Desktop Services Message Set is composed of several request and notification messages, listed in Table 1.In addition we define Desktop service unique terms and error messages in this section.

Table 2 ToolTalk Desktop Services Message Set

Requests	**Notices**
Do_Command	Created
Get_Environment	Deleted
Get_Geometry	DeIconified
Get_Iconified	Iconified
Get_Locale	Lowered
Get_Mapped	Mapped
Get_Modified	Modified
Get_Situation	Moved

Table 2 ToolTalk Desktop Services Message Set

Requests	Notices
Get_Status	Opened
Get_Sysinfo	Raised
Get_XInfo	Reverted
Lower	Saved
Quit	Started
Raise	Status
Revert	Stopped
Save	UnMapped
Set_Environment	
Set_Geometry	
Set_Iconified	
Set_Locale	
Set_Mapped	
Set_Situation	
Set_XInfo	
Signal	

Definitions

boolean
A vtype for logical values. The underlying data type of boolean is integer; that is, arguments of this vtype should be manipulated with `tt_*_arg_ival[_set]()` and `tt_*_iarg_add()`. Zero means false; non-zero means true.

buffer
A volatile, non-shared (for example, in-memory) representation of some persistent data.

bufferID
A vtype for uniquely identifying buffers. The underlying data type of bufferID is string. A good way to guarantee bufferID uniqueness is to make them of the form `<internal_counter> <procID>`.

messageID

A vtype for uniquely identifying messages. The underlying data type of messageID is string; that is, arguments of this vtype should be manipulated with `tt_*_arg_val[_set]() and tt_*_arg_add()`. A good way to guarantee messageID uniqueness is to make them of the form <internal_counter> <procID>. ToolTalk 1.1 introduces `tt_message_id()`, which returns an opaque string of similar uniqueness. `tt_message_id()` may be used to generate a message's messageID, but these inter-client conventions will explicitly include the messageID as a message argument, to support interoperation with versions of ToolTalk prior to 1.1.

<type>

Any of the vtypes that are the name of the kind of objects in a particular persistent-object system. For example, the vtype for the kind of objects in filesystems is `File`. The vtype for ToolTalk objects is `ToolTalk_Object`.

vendor
toolName
toolVersion

Names of arguments that appear in several of the messages in the Desktop suite of messages. These strings are not defined rigorously; they merely should be presentable to the user as descriptions of these three attributes of the relevant procID.

view

A chunk of screen real estate, such as a (portion of a) window, that presents to the user part or all of a document.

viewID

A vtype for uniquely identifying views. The underlying data type of viewID is string. A good way to guaranteed viewID uniqueness is to make them of the form <internal_counter> <procID>.

Errors

`1538 TT_DESKTOP_ENOENT`

No such file or directory

1558 TT_DESKTOP_EINVAL

Invalid argument

An argument's value was not valid in these circumstances. -- For example, a locale in Set_Locale() that is not valid on the handler's host. However, TT_DESKTOP_EINVAL should only be used when no more-specific status (for example, TT_DESKTOP_ENOMSG, TT_DESKTOP_EPROTO) applies.

1571 TT_DESKTOP_ENOMSG

No message of desired type

A messageID does not refer to any message currently known by the handler.

1610 TT_DESKTOP_EPROTO

Protocol error

A message could not be understood because

- a required argument was omitted.
- an argument had the wrong vtype, or a vtype not allowed in this message -- for example, boolean in Get_Geometry().
- an argument's value was not legal for its vtype -- for example, negative values for width in Set_Geometry().
- an argument's value was not legal for this message -- for example, PATH=/foo as a variable in Get_Environment(). In general, TT_DESKTOP_EPROTO means that one could see that the message is malformed simply by comparing it with the reference page for the message.

1698 TT_DESKTOP_CANCELED

Operation was canceled

The operation was canceled because of direct or indirect user intervention. An example of indirect intervention is termination of the handling process caused by the user, or receipt of a Quit() request. (All messages should be taken as authentically representing the wishes of the user whose uid is indicated by tt_message_uid().)

1699 TT_DESKTOP_UNMODIFIED

Operation does not apply to unmodified entities

Warnings

The vtype namespace for persistent objects currently only contains `File` and `ToolTalk_Object`. Vendors who want to define a type should either give it a vendor-specific name or register it through SunSoft's Developer Integration Format Registration program. SunSoft can be reached at 1-800-227-9227.

Created(Notice)

Notification that entities (for example, files) have been created or deleted.

Synopsis

```
[file] Created([in <type> ID[...]);

[file] Deleted([in <type> ID[...]);
```

Description

The Created notice is sent whenever a tool creates or deletes one or more entities that may be of interest to other tools.

Required Arguments

<type> ID

The identity of the created entity. If more than one entity are created in the same logical event, extra ID args may be present.

When <type> is File, each non-empty ID argument is the name of an entry which has been created in the directory named in the message's file attribute. (Thus, each argument is a single, final component of a pathname.)

When type is File and this argument is empty (i.e., has a value of (char *)0), then it refers to the file or directory named in the message's file attribute.

Optional Arguments

<type> ID

Extra instances of this argument may be included.

Do_Command(Request)

Requests in a tool's native command language that a command be performed.

Synopsis

```
Do_Command(in    string     command,
                   out  string      results
                  [in    messageID    counterfoil]);
```

Description

The Do_Command message requests that the receiving tool perform a command. The request is stated in the receiving tool's native command language.

When the request includes the optional *counterfoil* argument, the handler can send an immediate point-to-point status notice back to the requesting tool if the requested operation is expected to require an extended amount of time.

Required Arguments

string command
: The command being requested to be performed.

string results
: The results of the completed command. The results are returned as if the command had been executed locally to the requesting tool.

Optional Arguments

messageID counterfoil

Unique string created by the message sender (typically by concatenating a counter and a procID) to give both sender and receiver a way to refer to this request in other correspondence. This argument should be included if the sender anticipates a need to communicate with the handler about this request before it is completed, e.g., to cancel it.

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.

Warnings

This request lets tools provide a message interface to functionality that is not supported through any standard (or even tool-specific) message interface. Therefore this message constitutes a deprecated interface when the intended function is available through an existing message interface.

Get_Modified(Request)

Asks whether an entity (for example, a file) has been modified.

Synopsis

```
[file] Get_Modified(in <type> ID,
                     out boolean modified);
```

Description

The Get_Modified message asks whether any tool has modified a volatile, non-shared for example, in-memory) representation of the persistent state of an entity (such as a file) with the intention of eventually making that representation persistent. Therefore, a tool should register a dynamic pattern for this request when it has modified an entity of possible shared interest.

Required Arguments

<type> ID

The identity of the entity that may have been modified.

When type is `File`, this argument is empty (that is, it has a value of (char *) 0) and it references the file or directory named in the message's file attribute.

boolean modified

The boolean value that indicates whether a volatile, non-shared (for example, in-memory) representation of the entity has been modified with the intention of eventually making that representation persistent.

Errors

`TT_ERR_NO_MATCH`

The Get_Modified request failed because no handler was found and the named entity is assumed not to be modified.

Get_Status(Request)

Requests that a tool's current status be returned.

Synopsis

```
Get_Status(out    string      status,
                  out       string      vendor,
                  out       string      toolName,
                  out       string      toolVersion
                 [in        messageID    operation2Query]

                  [out      domain       statusIsKeyIn]);
```

Description

The Get_Status message retrieves either the current status of a tool or the current status of a specific operation that is being performed by a tool.

Required Arguments

string status
: The status to be retrieved.

string vendor
: The name of the vendor of the receiving tool.

string toolName
: The name of the receiving tool.

string toolVersion
: The version of the receiving tool.

Optional Arguments

messageID operation2Query

The ID of the request that initiated the operation the status of which is being requested.

domain statusIsKeyIn

The internationalization “domain” in which the status argument is a valid “message ID” (i.e., key). If the status argument is not a valid key in any domain, then this argument should be given a null value.

Get_Sysinfo(Request)

Retrieves information about a tool's host.

Synopsis

```
Get_Sysinfo(out     string     sysname,
            out     string     nodename,
            out     string     release,
            out     string     version,
            out     string     machine,
            out     string     architecture,
            out     string     provider,
            out     string     serial);
```

Description

The Get_SysInfo message retrieves information about the receiver's host.

Required Arguments

string sysname
The name of the host's operating system.

string nodename
The name of the host.

string release
string version
Vendor-determined information about the host's operating system.

string machine
A vendor-determined name that identifies the hardware on which the operating system is running (such as sun4, sun4c, or sun4m).

string architecture
A vendor-determined name that identifies the instruction set architecture of the host (such as sparc, mc68030, m32100, or i80486).

string provider
The name of the hardware manufacturer.

string serial
The ASCII representation of the hardware-specific serial number of the host.

See Also

sysinfo(2), umane(2)

Iconified(Notice)

Notification that a tool has iconified or de-iconified.

Synopsis

```
Iconified(in    string        vendor,
                     in    string        toolName,
                     in    string        toolVersion
                    [in     messageID    commission...
                     in     viewID     viewIconified...]);

DeIconified(in      string        vendor,
                     in      string        toolName,
                     in      string        toolVersion
                    [in      messageID     commission...
                     in      viewID      ViewDeIconified...]);
```

Description

The Iconified message notifies interested tools whenever a tool iconifies or de-iconifies the window(s) associated with the sending tool's procID.

Required Arguments

string vendor
The name of the vendor of the iconified or de-iconified tool.

string toolName
The name of the iconified or de-iconified tool.

string toolVersion
The version of the iconified or de-iconified tool.

Optional Arguments

messageID commission
The identifier of the message (if any) that resulted in the creation of the iconified or de-iconified window.

viewID view[De]Iconified

The identifier of any view associated with the window(s) that were iconified or de-iconified.

Mapped(Notice)

Notification that a tool has mapped or unmapped its window(s) to or from the screen.

Synopsis

```
Mapped(in       string       vendor,
                 in       string       toolName,
                 in       string       toolVersion
                [in       messageID     commission...
                 in       ViewID       ViewMapped...]);

UnMapped(in     string       vendor,
                 in       string       toolName,
                 in       string       toolVersion
                [in       messageID     commission...
                 in       viewID       ViewUnMapped...]);
```

Description

The Mapped message notifies interested tools whenever a tool maps or unmaps the window(s) associated with the sending tool's procID.

Required Arguments

string vendor
The name of the vendor of the mapped or unmapped tool.

string toolName
The name of the mapped or unmapped tool.

string toolVersion
The version of the mapped or unmapped tool.

Optional Arguments

messageID commission
The identifier of the message (if any) that resulted in the creation of the mapped or unmapped window.

viewID view[Un]Mapped

The identifier of any view associated with the window(s) that were mapped or unmapped.

Modified(Notice)

Notification that an entity (for example, a file) has been either modified or reverted to its prior state.

Synopsis

```
[file] Modified(in <type> ID);

[file] Reverted(in <type> ID);
```

Description

The Modified message notifies interested tools whenever a tool first makes changes to a volatile, non-shared (for example, in-memory) representation of the persistent state of an entity (such as a file). The Reverted message notifies interested tools whenever a tool discards the modifications made to a volatile, non-shared (for example, in-memory) representation of the persistent state of an entity (such as a file).

Required Arguments

<type> ID
: The identity of the modified or reverted entity.

When type is File, then this argument is empty (that is, has a value of (char *)0), and it refers to the file or directory named in the message's file attribute.

`Moved(Notice)`

Notification that an entity (for example, a file) has been moved.

Synopsis

```
[file] Moved(in        string        oldID,
                  in        string        newID);
```

Description

The Moved message notifies interested tools whenever a tool changes the location of a persistent entity.

Required Arguments

<type> newID
The new identity of the moved entity.

When type is File, then this argument is empty (i.e., has a value of (char *)0), and it refers to the file or directory named in the message's file attribute.

<type> oldID
The old identity of the moved entity.

When type is File, then this argument is either an absolute pathname, or a pathname relative to the directory named in (or containing) the path in the message's file attribute.

Opened(Notice)

Notification that an entity (for example, a file) has been opened or closed.

Synopsis

```
[file]          Opened(in <type> ID);

[file]          Closed(in <type> ID);
```

Description

The Opened message notifies interested tools whenever a tool opens or closes a volatile, non-shared (for example, in-memory) representation of a persistent entity (such as a file).

Required Arguments

<type> ID
The identity of the opened or closed entity.

When type is File, then this argument is empty (i.e., has a value of (char *)0), and it refers to the file or directory named in the message's file attribute.The namespace for the *ID* argument depends on the *type* argument.

Quit(Request)

Requests that an operation, or an entire tool, terminate.

Synopsis

```
Quit(in        boolean       silent,
               in        boolean       force
               [in        messageID     operation2Quit]);
```

Description

Without the optional 'operation2Quit' argument, this request asks the recipient procID to quit. If the request succeeds, one or more ToolTalk procID's should call tt_close(), and zero or more processes should exit. "Zero or more", because a single process can instantiate multiple independent procID's, and a single procID can conceivably be implemented by a set of cooperating processes.

With the optional 'operation2Quit' argument, this request asks the recipient to terminate the indicated request. (Whether the terminated request must therefore be failed depends on its semantics. Often, termination can be considered to mean that the requested operation has been carried out to the requestor's satisfaction.)

This request should be failed (and the status code set appropriately) when the termination is not performed -- for example, because the 'silent' argument was false and the user canceled the quit.

Required Arguments

boolean silent
: Boolean value indicating whether the recipient tool is allowed to request user input before terminating.

boolean force

Boolean value indicating whether the recipient tool should terminate even if circumstances are such that the tool ordinarily would not terminate under them.

For example, a tool might have a policy of not quitting with unsaved changes unless the user has been asked whether the changes should be saved. When this argument is true, such a tool should terminate even when doing so would lose changes that the user has not been asked about saving.

Optional Arguments

messageID operation2Quit

The request that should be terminated. For a request to be terminable, an (optional) counterfoil messageID shall have been included in the request, and the handler shall have sent a Status() notice back to the requestor (thus identifying itself to the requestor).

Errors

TT_DESKTOP_CANCELED

Warnings

Quit may also be sent as a multicast notice, as an edict to all tools within the sound of one's voice. The consequences of doing so can be severe and unexpected.

Raise(Request)

Raises or lowers a tool's window(s) to the front or back, respectively.

Synopsis

```
Raise(     [in       messageID     commission...]
              [in       viewID        view2Raise...]);

Lower(     [in       messageID     commission...]
              [in       viewID        view2Lower...]);
```

Description

The Raise message raises or lowers the window(s) associated with the recipient's procid. If any optional arguments are present, only the indicated window(s) are raise or lowered.

Note – The Raise and Lower requests may also be sent as a multicast notice; however, the consequences can be severe and unexpected.

Optional Arguments

messageID commission
The identifier of the message (if any) that resulted in the creation of the raised or lowered window(s).

viewID view2Raise
viewID view2Lower
The identifier of the view whose associated window(s) is (are) be raised or lowered.

Warnings

Raise and Lower may also be sent as a multicast notice, as an edict to all tools within the sound of one’s voice. The consequences of doing so can be severe and unexpected.

Raised(Notice)

Notification that the stacking order of a tool's window(s) have changed.

Synopsis

```
Raised(in       string       vendor,
                 in       string       toolName,
                 in       string       toolVersion
                [in       messageID    commission...]
                 in       viewID       viewRaised...]);

Lowered(in       string       vendor,
                  in       string       toolName,
                  in       string       toolVersion
                 [in       messageID    commission...]
                  in       viewID       viewLowered...]);
```

Description

The Raised message notifies interested tools whenever a tool raises or lowers the window(s) associated with the sending tool's procID.

Required Arguments

string vendor
: The name of the vendor of the raised or lowered tool.

string toolName
: The name of the raised or lowered tool.

string toolVersion
: The version of the raised or lowered tool.

Optional Arguments

messageID commission
: The identifier of the message (if any) that resulted in the creation of the raised or lowered window(s).

viewID viewRaised
viewID viewLowered
The identifier of any view associated with the window(s) that were raised or lowered.

`Save(Request)`

Saves or discards any modifications to an entity (for example, a file).

Synopsis

```
[file] Save(in        <type>       ID);
[file] Revert(in       <type>       ID);
```

Description

The Save messages requests that any pending, unsaved modifications to a persistent entity (such as a file) be saved or discarded.

Required Arguments

<type> ID
The identity of the entity to save or revert.

When type is `File`, this argument is empty (that is, it has a value of (char *) 0) and references the file or directory named in the message's file attribute.

Errors

`TT_DESKTOP_UNMODIFIED`
The entity had no pending, unsaved modifications.

`TT_DESKTOP_ENOENT`
The file to save or revert does not exist.

`Saved(Notice)`

Notification that an entity (such as a file) has been saved to persistent storage.

Synopsis

```
[file] Saved(in <type> ID);
```

Description

The Saved message notifies interested tools whenever a tool saves an entity (such as a file) to persistent storage.

Required Arguments

<type> ID
The identity of the saved entity.

When type is File, then this argument is empty (i.e., has a value of (char *)0), and it refers to the file or directory named in the message's file attribute.

Set_Environment (Request)

Requests that a tool's environment either be set or retrieved.

Synopsis

```
Set_Environment(in        string       variable,
                           in       string       value
                           [...]);

Get_Environment(in        string       variable,
                          out      string       value
                          [...]);
```

Description

The Set_Environment message requests that the value of the indicated environment variable(s) either be replaced or reported.

Required Arguments

string variable
  The name of the environment variable to be set or retrieved.

string value
  The value of the environment variable to be set or retrieved.

- If this argument does not contain a value for the Set_Environment request, the variable is removed from the environment. It is not considered an error if the specified variable does not exist.
- If this argument does not contain a value when used in the Get_Environment request, the variable was not present in the receiving tool's environment. This condition is not considered an error.

Optional Arguments

string variable
string value
  Extra pairs of these arguments may included.

Warnings

Set_Environment may also be sent as a multicast notice, as an edict to all tools within the sound of one's voice. The consequences of doing so can be severe and unexpected.

Set_Geometry(Request)

Requests that a tool's on-screen geometry either be set or retrieved.

Synopsis

```
Set_Geometry([inout    width         w]
                [inout    height        h]
                [inout    xOffset       x]
                [inout    yOffset       y]
                [in       messageID     commission]
                [in       viewID        view2Set]);

Get_Geometry([out      width         w]
                  [out      height        h]
                  [out      xOffset       x]
                  [out      yOffset       y]
                  [in       messageID     commission]
                  [in       viewID       view2Get]);
```

Description

The Set_Geometry message requests that the value of the on-screen geometry of the optionally-specified window, or the value of the on-screen geometry of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved.

Optional Arguments

width w
height h
xOffset x
yOffset y

The integer geometry values in pixels.

The return values for the Get_Geometry request are the actual new values, not the requested new values.

Note – Negative offset values are interpreted according to the `X(1)` man page.

---

messageID commission

The identifier of the message (if any) that resulted in the creation of the set or retrieved window(s).

viewID view2Set
viewID view2Get

The identifier of any view associated with the window(s) that is (are) to be set or retrieved.

.

Set_Iconified(Request)

Requests that a tool's iconic state be set or retrieved.

Synopsis

```
Set_Iconified(inout    boolean      iconic
                          [in       messageID    commission...]
                          [in       viewID      view2Iconify...]);

Get_Iconified(out      boolean      iconic
                          [in       messageID    commission...]
                          [in       viewID      view2Query]);
```

Description

The Set_Iconified message requests that the value of the iconic state of the optionally-specified window, or the iconic state of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved.

Required Arguments

boolean iconic
The boolean value that indicates whether the specified window is iconified.

Optional Arguments

messageID commission
The identifier of the message (if any) that resulted in the creation of the iconified or queried window(s).

viewID view2Iconify
viewID view2Query
The identifier of any view associated with the window(s) that is (are) to be iconified or queried.

Warnings

Set_Iconified may also be sent as a multicast notice, as an edict to all tools within the sound of one’s voice. The consequences of doing so can be severe and unexpected

Set_Locale(Request)

Sets or retrieves a tool's locale.

Synopsis

```
Set_Locale( in        string        category,
            in        string        locale
            [...]);

Get_Locale( in        string        category,
            out       string        locale
            [...]);
```

Description

The Set_Locale message replaces or reports the locale of the indicated locale categories.

Required Arguments

string category
: The locale category to set or retrieve.

  A locale category is a group of data types whose formatting varies according to locale; for example, in UNIX SVR4, locale categories include:

  - `LC_CTYPE`
  - `LC_NUMERIC`
  - `LC_TIME`
  - `LC_COLLATE`
  - `LC_MESSAGES`
  - `LC_MONETARY`
  - `LC_ALL`

string locale
: The name of the current locale of the indicated category, or the locale to which to set the indicated category; example of these locales defined in UNIX SVR4 are "C", "de", "fr", and "it".

Optional Arguments

string category
string locale
Extra pairs of these arguments may be included.

Warnings

Set_Locale may also be sent as a multicast notice, as an edict to all tools within the sound of one's voice. The consequences of doing so can be severe and unexpected

Set_Mapped(Request)

Requests that a tool's mapping to the screen be set or retrieved.

Synopsis

```
Set_Mapped( inout    boolean      mapped
                      [in        messageID    commission]
                      [in        viewID      View2Map]);

Get_Mapped(out      boolean      mapped
                      [in        messageID    commission]
                      [in        viewID       view2Query]);
```

Description

The Set_Mapped message requests that value of the mapped state of the optionally-specified window, or the mapped state of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved.

Required Arguments

boolean mapped
  The boolean value that indicates whether the specified window is mapped to the screen.

Optional Arguments

messageID commission
  The identifier of the message (if any) that resulted in the creation of the set or retrieved window(s).

viewID view2Map
viewID view2Query
  The identifier of any view associated with the window(s) that is (are) to be set or retrieved.

Warnings

Set_Mapped may also be sent as a multicast notice, as an edict to all tools within the sound of one's voice. The consequences of doing so can be severe and unexpected

Set_XInfo(Request)

Requests that a tool's X11 attributes be set or retrieved.

Synopsis

```
Set_XInfo( inout    string       display,
                     inout    string       visual,
                     inout    integer      depth
                    [inout    string       resourceName,
                     inout    string       resourceVal,...]
                    [in       messageID    commission]);

Get_XInfo( out      string       display,
                     out      string       visual,
                     out      integer      depth
                    [in       string       resourceName,
                     out      string       resourceVal,...]
                    [in       messageID    commission]);
```

Description

The Set_XInfo message requests that the X11 attributes of the optionally-specified window, or the X11 attributes of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved.

Required Arguments

string display
An X11 display.

string visual
An X11 visual class, which determines how a pixel will be displayed as a color. Values include:

StaticGray
GrayScale
StaticColor

PseudoColor
TrueColor
DirectColor

integer depth
The number of bits in a pixel.

Optional Arguments

string resourceName
string resourceVal
An X11 resource name and resource value.

messageID commission
The ID of the message with respect to which X11 attributes are being set or reported. This is useful to the extent that the handler employs different attributes for the different operations it may be carrying out.

Signal(Request)

Requests that a (POSIX-style) signal be sent to a tool.

Synopsis

```
Signal(in        integer      theSignal);
```

Description

The Signal message requests that the receiving tool's procID send the indicated signal to itself.

Required Arguments

integer theSignal
The signal to be sent.

Warnings

Signal may also be sent as a multicast notice, as an edict to all tools within the sound of one's voice. The consequences of doing so can be severe and unexpected.

Started(Notice)

Notification that a tool has started or terminated.

Synopsis

```
Started( in       string       vendor,
           in       string       toolName,
           in       string       toolVersion);

Stopped( in       string       vendor,
           in       string       toolName,
           in       string       toolVersion);
```

Description

The Started message notifies interested tools whenever a tool starts or terminates.

Required Arguments

string vendor
The name of the vendor of the started or terminated tool.

string toolName
The name of the started or terminated tool.

string toolVersion
The version of the started or terminated tool.

Status(Notice)

Notification that a tool has status information to announce.

Synopsis

```
Status( in      string      status,
         in      string      vendor,
         in      string      toolName,
         in      string      toolVersion
        [in      messageID  commission]
        [in      domain      statusIsKeyIn]);
```

Description

The Status message notifies interested tools of a tool's general status information.

Required Arguments

string status
The status which is being announced.

string vendor
The name of the vendor of the tool whose status is being announced.

string toolName
The name of the tool whose status is being announced.

string toolVersion
The version of the tool whose status is being announced.

Optional Arguments

messageID commission
The ID of the request, if any, that initiated the operation the status of which is being announced.

domain statusIsKeyIn
The internationalization "domain" in which the status argument is a valid "message ID" (i.e., key).).

Future Direction

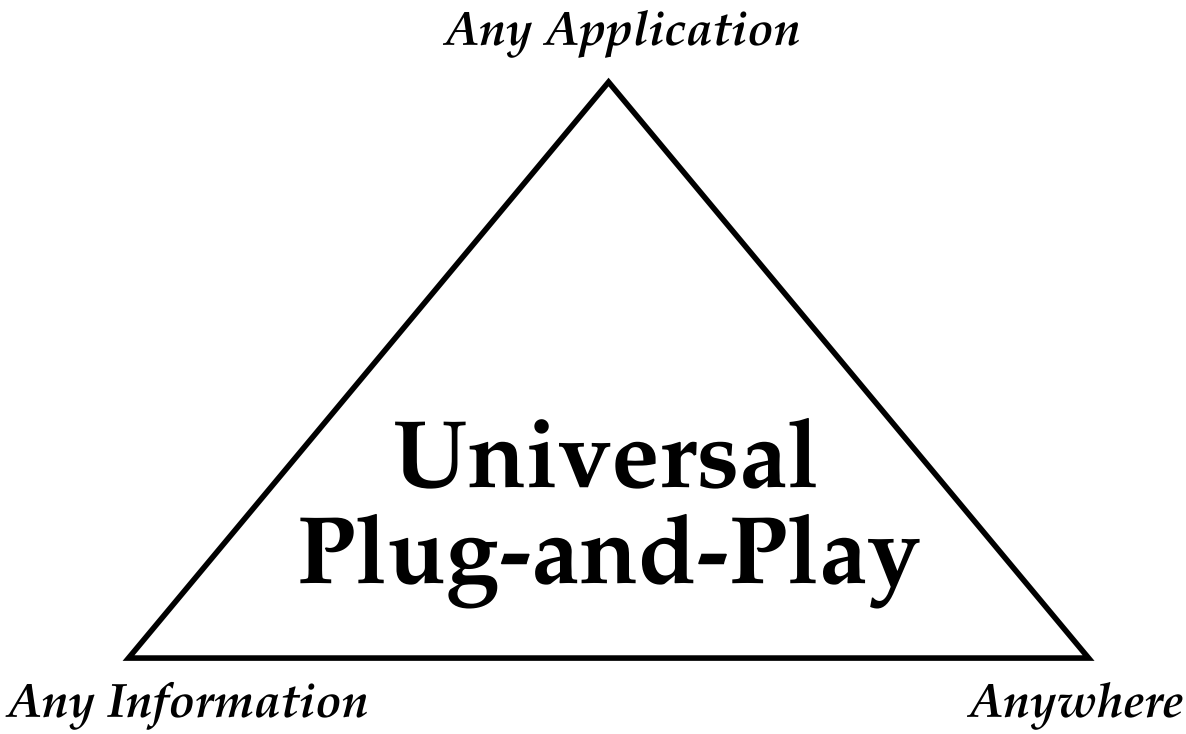
The ToolTalk Desktop Services Message Set will evolve as new requirements arise. We intend work with implementers of these messages to create extensions needed for situation we have not thought of. The idea of the message set is to provide a language in which applications can use to "talk" to each other and achieve integration.

The ToolTalk Desktop Services Message Set will create new business opportunities for developers and users of applications. Developers of applications will benefit from the ability to integrate with the user's application environment without pre-negotiating interfaces with all possible applications that the customer may want to integrate with.

The ToolTalk Desktop Services 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 desktop services developers and users. Send questions, comments, and requests for information to the Desktop Services Messaging Alliance at *ToolTalk_desktop_services@sun.com*.

The ToolTalk Desktop Services Message Set: *Release 1.1*

A White Paper



SunSoft

A Sun Microsystems, Inc. Business

Revision History

Revision	Dash	Date	Comments
92206	–001	August 1992	Initial Release
92206	-002	February 1993	New Messages : • Get_Environment • Get_Local • Get_Modified • Get_Situation • Get_Sysinfo • Raise • Revert • Save • Set_Local • Set_Situation Updated : • Iconified • Mapped • Raised • Set_Geometry • Set_Iconified • Set_Mapped

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 92206--002 .5K