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 the some of the source code text.

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

    a. The Document Editor first determines the text represents source code and then determines what file contains the source code.

    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.

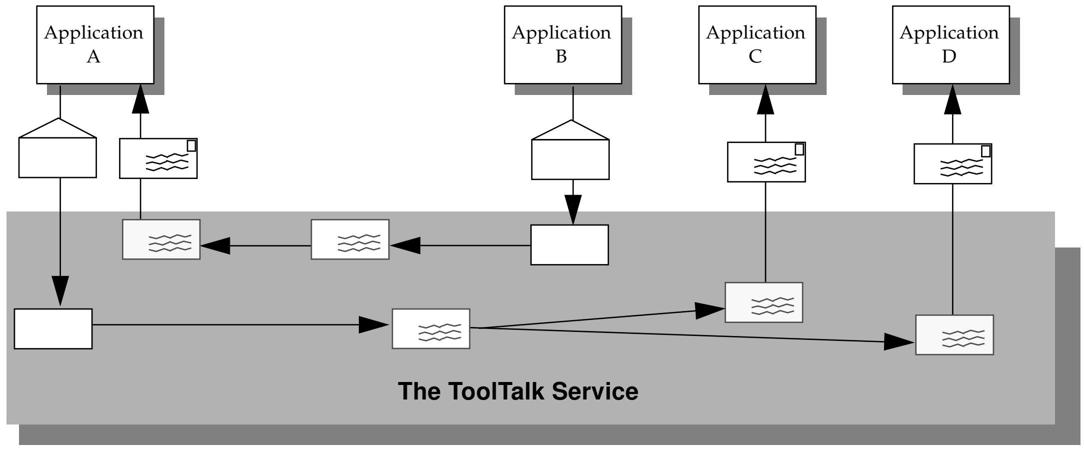

Figure 1 Applications Using the ToolTalk Service

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 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 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 3.

Table 1 ToolTalk Desktop Services 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.

Generic messages can be sent and received by any tool.

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.

Table 2 ToolTalk Desktop Services Message Set

Requests	**Notices**
Do_Command	Created
Get_Environment	Closed
Get_Geometry	Deleted
Get_Iconified	DeIconified
Get_Mapped	Iconified
Get_Status	Lowered
Get_XInfo	Mapped
Quit	Modified
Set_Environment	Moved
Set_Geometry	Opened
Set_Iconified	Raised
Set_Mapped	Reverted
Set_XInfo	Saved
Signal	Started
	Status
	Stopped
	UnMapped

These generic messages can be sent and received by any tool.

Created(Notice)

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

Synopsis

```
[file both] Created(  [in  string  filename ...] );

[file both] Deleted(  [in  string  filename ...] );

[file both session] Created(  in  string  type,
                             in  string  ID  [...] );

[file both session] Deleted(  in  string  type,
                             in  string  ID [...] );
```

Description

The Created message notifies interested tools whenever a tool creates or deletes one or more entities. There are two forms of this message.

1. When the entity that was created or deleted is a *file*, use the form:

```
[file both] Created(  [in  string  filename ...] );
```

```
[file both] Deleted(  [in  string  filename ...] );
```

- If the message has any arguments, each argument is the name of an entity which has been created or deleted in the directory to which the message is scoped; that is, each argument is a single, final component of a pathname.
- If the message has no arguments, then the entity created or deleted is the file to which the message is scoped.

2. When the entity that was created or deleted is a type other than *file*, use the form:

```
[file both session] Created(  in  string  type,
                              in  string  ID  [...] );
```

```
[file both session] Deleted(  in  string  type,
                              in  string  ID [...] );
```

Required Arguments

string type
The type of the created or deleted entity.

Note – The namespace for the *type* argument is currently empty. To define a type, either provide a vendor-specific name (for example, Sun_ToolTalk) or register the type through the SunSoft Developer Integration Format Registration program.

---

string ID
The identifier of the created or deleted entity. If more than one entity ia created or deleted in the same logical event, extra ID arguments may be present.

Note – The namespace for the *ID* argument depends on the *type* argument.

Optional Arguments

string filename
The last component of the pathname of a file which was created or deleted in the directory to which the message is scoped. If more than one file is created or deleted in the directory in the same logical event, extra filename arguments may be present.

Do_Command(Request)

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

Synopsis

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

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 *msgID* 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 msgID
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.

Get_Status(Request)

Requests that a tool's current status be returned.

Synopsis

```
session Get_Status(  out      string       status,
                     out      string       vendor,
                     out      string       toolName,
                     out      string       toolVersion
                     [in      messageID    msgID] );
```

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 msgID
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.

Iconified(Notice)

Notification that a tool has iconified or de-iconified.

Synopsis

```
session   Iconified(   in    string       vendor,
                       in    string       toolName,
                       in    string       toolVersion
                      [in    messageID    msgID
                       in    bufferID     buffID ...] );

session DeIconified(  in     string       vendor,
                      in     string       toolName,
                      in     string       toolVersion
                     [in     messageID    msgID
                      in     bufferID     buffID ...] );
```

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 msgID
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.

bufferID buffID

The identifier of any buffer 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

```
session   Mapped(   in        string       vendor,
                  in        string       toolName,
                  in        string       toolVersion
                 [in        messageID    msgID
                  in        bufferID     buffID ...] );

session   UnMapped(   in      string       vendor,
                    in      string       toolName,
                    in      string       toolVersion
                   [in      messageID    msgID
                    in      bufferID     buffID ...] );
```

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 msgID
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.

bufferID buffID
The identifier of any buffer 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 both]  Modified();

[file both] Reverted();

[file both session] Modified(  in       string       type,
                              in       string       ID );

[file both session] Reverted(  in       string       type,
                              in       string       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). There are two forms to this message.

1. When the entity that was modified or reverted is a *file*, use the form:

```
[file both]  Modified();
```

```
[file both] Reverted();
```

2. When the entity that was modified or reverted is a type other than *file*, use the form:

```
[file both session] Modified(  in      string      type,
                               in      string      ID );
```

```
[file both session] Reverted(  in      string      type,
                               in      string      ID );
```

Required Arguments

string type
The type of the modified or reverted entity.

Note – The namespace for the *type* argument is currently empty. To define a type, either provide a vendor-specific name (for example, Sun_ToolTalk) or register the type through the SunSoft Developer Integration Format Registration program.

---

string ID
The identifier of the modified or reverted entity.

Note – The namespace for the *ID* argument depends on the *type* argument.

Moved(Notice)

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

Synopsis

```
[file both] Moved(   in        string        oldPath );

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

Description

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

1. When the entity that was moved is a *file*, use the form:

```
[file both] Moved(   in       string       oldPath );
```

The message is scoped to the path to which the file has been renamed; the *oldPath* argument indicates the path from which the file has been renamed.

2. When the entity that was moved is a type other than *file*, use the form:

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

Required Arguments

string oldPath
The path from which the file has been renamed. If *oldPath* is a relative path, then it is relative to the directory that contains the path to which the file has been renamed.

Note – When *oldPath* is not relative to the path to which the message is scoped, there is no guarantee that the given path is correct (or even valid) in every interested tool's view of the filesystem. You can use the ToolTalk services's file-scoping mechanism to solve this problem; however, you must install a ToolTalk daemon on each file server.

---

string type
The type of the moved entity.

Note – The namespace for the *type* argument is currently empty. To define a type, either provide a vendor-specific name (for example, Sun_ToolTalk) or register the type through the SunSoft Developer Integration Format Registration program.

---

string oldID
string newID
The old and new identifiers of the moved entity.

Note – The namespace for the *ID* argument depends on the *type* argument.

Opened(Notice)

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

Synopsis

```
[file both]           Opened();

[file both]           Closed();

[file both session] Opened(  in       string       type,
                            in       string       ID );

[file both session] Closed(  in       string       type,
                            in       string       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). There are two forms of this message.

1. When the entity that was opened or closed is a *file*, use the form:

```
[file both]           Opened();
```

```
[file both]           Closed();
```

2. When the entity that was opened or closed is a type other than *file*, use the form:

```
[file both session] Opened(  in      string      type,
                             in      string      ID );
```

```
[file both session] Closed(  in      string      type,
                             in      string      ID );
```

Required Arguments

string type
The type of the opened or closed entity.

Note – The namespace for the *type* argument is currently empty. To define a type, either provide a vendor-specific name (for example, Sun_ToolTalk) or register the type through the SunSoft Developer Integration Format Registration program.

---

string ID
The identifier of the opened or closed entity.

Note – The namespace for the *ID* argument depends on the *type* argument.

Quit(Request)

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

Synopsis

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

Description

The Quit message requests that the receiving tool terminate either an operation it is performing, or itself.

- If the optional *theRequest* argument is omitted, this message requests that the receiving tool's procID terminate. If the request succeeds, one or more ToolTalk procID's call the tt_close() function, and zero or more processes will exit.

Note – Zero or more processes will exit because a single process can instantiate multiple independent procID's; and a single procID can conceivably be implemented by a set of cooperating processes.

- If the optional *theRequest* argument is included, this message requests that the receiving tool terminate the indicated operation.

Note – This request fails when the termination is not performed; for example, because the *silent* argument is false, allowing the user to cancel the quit command.

Required Arguments

boolean silent

The boolean value that indicates whether the receiving tool is allowed to request user input before it terminates.

boolean force

The boolean value that indicates whether the receiving tool can terminate although it would normally not terminate under the current circumstances.

For example, a tool that does not quit with unsaved changes unless it asks the user whether the changes are to be saved will terminate if this argument is true although the data will be lost because user has not been asked whether changes are to be saved.

Optional Arguments

messageID theRequest

The operation that is to be terminated.

Note – To be able to terminate a request, the requesting tool that initiated the operation to be terminated must have included an optional messageID in the requesting message; and the handler must have identified itself to the requesting tool by returning a Status notice to it.

Raised(Notice)

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

Synopsis

```
session Raised(   in        string       vendor,
                  in        string       toolName,
                  in        string       toolVersion
                 [in        messageID    msgID
                  in        bufferID     buffID ...] );

session Lowered(  in        string       vendor,
                  in        string       toolName,
                  in        string       toolVersion
                 [in        messageID    msgID
                  in        bufferID     buffID ...] );
```

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 msgID
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.

bufferID buffID

The identifier of any buffer associated with the window(s) that were raised or lowered.

Saved(Notice)

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

Synopsis

```
[file both] Saved();

[file both session] Saved(  in       string       type,
                            in       string       ID );
```

Description

The Saved message notifies interested tools whenever a tool saves an entity (such as a file) to persistent storage. There are two forms of this message.

1. When the entity that was saved is a *file*, use the form:

```
[file both] Saved();
```

2. When the entity that was saved is a type other than *file*, use the form:

```
[file both session] Saved(  in      string      type,
                            in      string      ID );
```

Required Arguments

string type
    The type of the saved entity.

Note – The namespace for the *type* argument is currently empty. To define a type, either provide a vendor-specific name (for example, Sun_ToolTalk) or register the type through the SunSoft Developer Integration Format Registration program.

string ID
The identifier of the saved entity.

Note – The namespace for the *ID* argument depends on the *type* argument.

Set_Environment(Request)

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

Synopsis

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

session  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 is omitted 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.

Set_Geometry(Request)

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

Synopsis

```
session   Set_Geometry(   [ inout    width        w]
                          [inout    height        h]
                          [inout    xOffset       x]
                          [inout    yOffset       y]
                          [in        messageID    msgID]
                          [in        bufferID     bufID] );

session     Get_Geometry(    [out      width        w]
                            [out      height        h]
                            [out      xOffset       x]
                            [out      yOffset       y]
                            [in        messageID    msgID]
                            [in        bufferID     bufID] );
```

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 msgID

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.

bufferID buffID

The identifier of any buffer associated with the window for which the value is to be set or retrieved.

`Set_Iconified(Request)`

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

Synopsis

```
session   Set_Iconified(   inout    boolean      iconic
                          [in       messageID    msgID]
                          [in       bufferID     bufID] );

session   Get_Iconified(   out      boolean      iconic
                          [in       messageID    msgID]
                          [in       bufferID     bufID] );
```

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 msgID
: 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.

bufferID buffID
: The identifier of any buffer associated with the window for which the value is to be set or retrieved.

Set_Mapped(Request)

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

Synopsis

```
session   Set_Mapped(   inout    boolean      mapped
                       [in       messageID    msgID]
                       [in       bufferID     bufID] );

session    Get_Mapped(  out      boolean      mapped
                       [in       messageID    msgID]
                       [in       bufferID     bufID] );
```

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 msgID
: 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.

bufferID buffID
: The identifier of any buffer associated with the window for which the value is to be set or retrieved.

`Set_XInfo(Request)`

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

Synopsis

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

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

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 msgID
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

```
session   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.

Started(Notice)

Notification that a tool has started or terminated.

Synopsis

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

session    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

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

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 msgID
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.

string domain

The internationalization *domain* in which the *status* argument is a valid message ID (that is, key).

Future Direction

The ToolTalk Desktop Services Message Set will evolve as new requirements arise. It leverages all of the existing desktop services formats because it allows an application to specify the format of the data within the message. The ToolTalk Desktop Services Message Set, therefore, will allow improved, or new, file formats to be used in the future as easily as a currently existing file format.

The ToolTalk Desktop Services Message Set will create new business opportunities for developers and users of desktop services technologies. Developers of non-desktop service technologies will benefit from the ability to add these desktop services to their products quickly without having to become desktop services technology experts.

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 1.0

A White Paper

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 8/92 92206--001 .5K