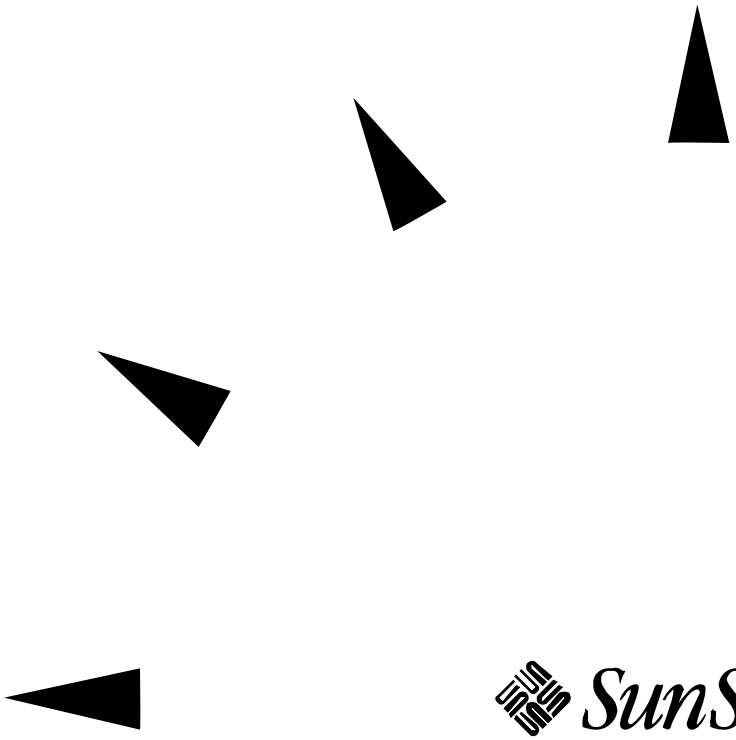
ToolTalk Reference Guide



Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 802-5872-10
Revision A, August 1997

SunSoft
A Sun Microsystems, Inc. Business

NAME

Intro – introduction to the ToolTalk commands and application programs.

OVERVIEW

This section describes commands which are used by ToolTalk or the Tooltalk developer to monitor ToolTalks' passing of messages, or to interact with ToolTalks' storage of message data on disk.

NAME

tt_enumerated_types – Introduction to ToolTalk enumerated types

Tt_address

Possible Values for Tt_address

Value	Description
TT_HANDLER	Addressed to a specific handler that can perform this operation with these arguments. Fill in *handler*, *op*, and *arg* attributes of the message or pattern.
TT_OBJECT	Addressed to a specific object that performs this operation with these arguments. Fill in *object*, *op*, and *arg* attributes of the message or pattern.
TT_OTYPE	Addressed to the type of object that can perform this operation with these arguments. Fill in *otype*, *op*, and *arg* attributes of the message or pattern.
TT_PROCEDURE	Addressed to any process that can perform this operation with these arguments. Fill in the *op* and *arg* attributes of the message or pattern.

Tt_callback

Possible Values for Tt_callback

Value	*Description*
TT_CALLBACK_CONTINUE	If the callback returns TT_CALLBACK_CONTINUE, other callbacks will be run.
TT_CALLBACK_PROCESSED	If the callback returns TT_CALLBACK_PROCESSED, no further callbacks will be invoked for this event, and the message will not be returned by tt_message_receive().

Tt_category

Possible Values for Tt_category

Value	Description
TT_OBSERVE	Just looking at the message. No feedback will be given to the sender.
TT_HANDLE_PUSH	Like TT_HANDLE, but will pick the most recently registered pattern given several equally qualified choices
TT_HANDLE_ROTATE	Like TT_HANDLE, but if there are no TT_HANDLE_PUSH patterns, pick the least recently used TT_HANDLE_ROTATE pattern before trying TT_HANDLE patterns.
TT_HANDLE	Will process the message, including filling in return values if any.

Tt_class

Possible Values for Tt_class

Value	Description
TT_NOTICE	Notice of an event. Sender does not want feedback on this message.

TT_OFFER	The term "offer" was chosen because the effect is like passing a plate of goodies around -- everybody takes one if they want; when the plate comes back you know everybody's been offered one.
TT_REQUEST	Request for some action to be taken. Sender must be notified of progress, success or failure, and must receive any return values.

Tt_disposition

Possible Values for Tt_disposition

Value	Description
TT_DISCARD	No receiver for this message. Message is returned to sender with the Tt_status field containing TT_FAILED.
TT_QUEUE	Queue the message until a process of the proper ptype receives the message.
TT_START	Attempt to start a process of the proper ptype if none is running.

Tt_feature

Possible Values for Tt_feature

Value	Description
TT_FEATURE_MULTITHREADED	Indicates that this version of the ToolTalk API can support multi threaded ToolTalk API calls.
TT_FEATURE_LAST	This code should be unused.

Tt_filter

Possible Values for Tt_filter

Value	Description
TT_FILTER_CONTINUE	Continue the query, feed more values to the callback.
TT_FILTER_STOP	Stop the query, don't look for any more values.

Tt_mode

Possible Values for Tt_mode

Value	Description
TT_IN	The argument is written by the sender and read by the handler and any observers.
TT_OUT	The argument is written by the handler and read by the sender and any reply observers.
TT_INOUT	The argument is written by the sender and the handler and read by all.

Tt_scope

Possible Values for Tt_scope

Value	Description
TT_SESSION	All processes joined to the indicated session are eligible.
TT_FILE	All processes joined to the indicated file are eligible.
TT_BOTH	All processes joined to either the indicated file or the indicated session are eligible.
TT_FILE_IN_SESSION	All processes joined to both the indicated session and the indicated file are eligible.

Tt_state

Possible Values for Tt_state

Value	Description
TT_ABSTAINED	Offers (only) enter this state when a receiving procid does the next tt_message_receive without accepting or rejecting the offer. One can think of TT_ABSTAINED also being entered when a procid destroys an offer without accepting or rejecting it, but since the message is destroyed at that time the procid will never see the state. This state is seen only by the receiver.
TT_ACCEPTED	Offers (only) enter this state when tt_message_accept is done on them by a receiver. The state is seen only by the receiver.
TT_CREATED	Message has been created but not yet sent.
TT_SENT	Message has been sent but not yet handled.
TT_HANDLED	Message has been handled, return values are valid.
TT_FAILED	Message could not be delivered to a handler.
TT_QUEUED	Message has been queued for later delivery.
TT_RETURNED	All observers (and the handler, if there is one) have accepted, rejected, or destroyed the TT_OFFER. The original sender sees this state, and it can be observed. This comes back to the original sender like the reply for a request. In particular, any message callbacks for the offer are run, and user data attached to the message before sending are available.
TT_STARTED	Attempting to start a process to handle the message.
TT_REJECTED	Message has been rejected by a possible handler. This state is seen only by the rejecting process. The ToolTalk service changes the state back to TT_SENT before delivering the message to another possible handler. If all possible handlers have rejected the message, the ToolTalk service changes the state to TT_FAILED before returning the message to the sender. A receiver that gets an offer will see this message in the TT_REJECTED state.

Tt_status

A Tt_status code is returned by all functions, sometimes directly and sometimes encoded in an error return value. See the ToolTalk 1.3 User's Guide for instructions on to determine whether the Tt_status code is a warning or an error and for retrieving the error message string for a Tt_status code.

The following section lists the Tt_status codes.

Error Messages

TT_ERR_ACCESS

Error Message String:
TT_ERR_ACCESS
An attempt was made to access a ToolTalk object in a way forbidden by the protection system.

Description:
You do not have the necessary access to the object and the application; for example, you do not have permission to destroy an object spec. Therefore, the operation cannot be performed.

Solution:
a. Obtain proper access to the object.
b. Retry the operation.

TT_ERR_ADDRESS

Error Message String:
TT_ERR_ADDRESS
The Tt_address value passed is not valid.

Description:
The ToolTalk service does not recognize the address value specified.

Solution:
The Tt_address values are TT_PROCEDURE, TT_OBJECT, TT_HANDLER, and TT_OTYPE. Retry the call with one of these values.

TT_ERR_APPFIRST

Error Message String:
TT_ERR_APPFIRST
This code should be unused.

Description:
This code marks the beginning of the messages allocated for ToolTalk application errors.

Solution:
NA

TT_ERR_CATEGORY

Error Message String:
TT_ERR_CATEGORY
Pattern object has no category set.

Description:
The category was not set.

Solution:
NA

TT_ERR_CLASS

Error Message String:
TT_ERR_CLASS
The Tt_class value passed is invalid.

Description:
The ToolTalk service does not recognize the class value specified.
Solution:
The Tt_class values are TT_NOTICE and TT_REQUEST. Retry the call with one of these values.

TT_ERR_DBAVAIL

Error Message String:
TT_ERR_DBAVAIL
A required database is not available. The condition may be temporary, trying again later may work.
Description:
The ToolTalk service could not access the ToolTalk database needed for this operation.
Solution:
a. Check if the file server or workstation that contains the database is available.
b. Try the operation again later.

TT_ERR_DBCONSIST

Error Message String:
Database is access information is incomplete or database is corrupt (run ttdbck).
Description:
The ToolTalk service could not write to the database because it is either corrupt, or the access inform
Solution:
Run the ttdbck utility to repair the database.

TT_ERR_DBEXIST

Error Message String:
TT_ERR_DBEXIST
A required database does not exist. The database must be created before this action will work.
Description:
The ToolTalk service did not find the specified ToolTalk database in the expected place.
Solution:
Install the rpc.ttdbserved program on the machine that stores the file or object involved in this operation.

TT_ERR_DBFULL

Error Message String:
ToolTalk database is full.
Description:
The ToolTalk service could not write to the database because it is full.
Solution:
Create more space on the file system in which the database is stored.

TT_ERR_DBUPDATE

Error Message String:
TT_ERR_DBUPDATE
The database is inconsistent: another tt_spec_write updated object first.
Description:
The ToolTalk service could not update the database because the specified object was already updated by a previous tt_spec_write call.
Solution:
NA

TT_ERR_DISPOSITION

Error Message String:
TT_ERR_DISPOSITION
The Tt_disposition value passed is not valid.
Description:
The disposition passed is not recognized by the ToolTalk service.
Solution:
The Tt_disposition values are TT_DISCARD, TT_QUEUE, and TT_START. Retry the call with one of these values.

TT_ERR_FILE

Error Message String:
TT_ERR_FILE
File object could not be found.
Description:
The file specified does not exist or is not accessible.
Solution:
a. Check the file path name and retry the operation.
b. Check if the machine where the file is stored is accessible.

TT_ERR_INTERNAL

Error Message String:
TT_ERR_INTERNAL
Internal error (bug)
Description:
The ToolTalk service has suffered an internal error.
Solution:
a. Restart all applications that are using the ToolTalk service.
b. Report the error to the your system vendor support center.

TT_ERR_LAST

Error Message String:
TT_ERR_LAST
This code should be unused.
Description:
This code marks the last of the messages allocated for ToolTalk errors.
Solution:

NA

TT_ERR_MODE

Error Message String:
TT_ERR_MODE
The Tt_mode value is not valid.

Description:
The ToolTalk service does not recognize the specified mode value.

Solution:
The Tt_mode values are TT_IN, TT_OUT, and TT_INOUT. Retry the call with one of these values.

TT_ERR_NO_MATCH

Error Message String:
TT_ERR_NO_MATCH
No handler could be found for this message, and the disposition was not queue or start.

Description:
The message the application sent could not be delivered.
No applications that are running have registered interest in this type of message.

Solution:
Use tt_disposition_set() to change the disposition to TT_QUEUE or TT_START and resend the message.
If no recipients are found, no application has registered interest in this type of message.

TT_ERR_NO_VALUE

Error Message String:
TT_ERR_NO_VALUE
No property value with the given name and number exists.

Description:
The ToolTalk service could not locate a value for the property specified in the ToolTalk database.

Solution:
Retrieve the current list of properties to find the property.

TT_ERR_NOMEM

Error Message String:
No more memory.

Description:
There is not enough available memory to perform the operation.

Solution:
Check the swap space, then retry the operation.

TT_ERR_NOMP

Error Message String:
TT_ERR_NOMP
No ttsession process is running, probably because tt_open() has not been called yet. If this code is returned from tt_open() it means ttsession could not be started, which generally means ToolTalk is not installed on this system.
Description:
The ttsession process is not available. The ToolTalk service tries to restart ttsession if it is not running. This error indicates that the ToolTalk service is either not installed or not installed correctly.
Solution:
a. Verify that the ToolTalk service is installed.
b. Verify that ttsession is installed on the machine in use.

TT_ERR_NOTHANDLER

Error Message String:
TT_ERR_NOTHANDLE
Only the handler of the message can do this.
Description:
Only the handler of a message can perform this operation. This application is not the handler for this message.
Solution:
NA

TT_ERR_NUM

Error Message String:
TT_ERR_NUM
The integer value passed is not valid.
Description:
An invalid integer value that was out-of-range was passed to the ToolTalk service.
Note: Simple out-of-range conditions, such as requesting the third value of a property that has only two values, return a null value.
Solution:
Check the integer specified.

TT_ERR_OBJID

Error Message String:
TT_ERR_OBJID
The object id passed does not refer to any existing object spec.
Description:
The objid does not reference an existing object.
Solution:
Update the spec property that contains the objid specified.

TT_ERR_OP

Error Message String:
TT_ERR_OP
The operation name passed is not syntactically valid.
Description:
The specified operation name is null or contains non-alphanumeric characters.
Solution:
a. Remove any non-alphanumeric characters.
b. Retry the operation.

TT_ERR_OTYPE

Error Message String:
TT_ERR_OTYPE
The object type passed is not the name of an installed object type.
Description:
The ToolTalk service could not locate the specified otype.
Solution:
Check the type of the object with tt_spec_type(). If the application was recently installed and the ToolTalk service has not reread the ToolTalk Types Database:
a. Locate the process id for the ttsession.
b. Force the reread with the USR-2 signal:
% ps -elf | grep ttsession
% kill -USR2 <*ttsession pid*>

TT_ERR_OVERFLOW

Error Message String:
TT_ERR_OVERFLOW
Too many active messages (try again later).
Description:
The ToolTalk service has received the maximum amount of active messages (2000) it can properly handle.
Solution:
Either:
a. Retrieve any messages that the ToolTalk service may be queueing for the application, and send the message again later.
b. Start ttsession with the -A option. Specify the maximum number of messages in progress before a TT_ERR_OVERFLOW
condition is returned. The default is 2000 messages.

TT_ERR_PATH

Error Message String:
TT_ERR_PATH
One of the directories in the file path passed does not exist or cannot be read.
Description:
The ToolTalk service was not able to read a directory in the specified file path name.
Solution:

a. Check the pathname to ensure access to the specified directories.
b. Check the machine where the file resides to make sure it is accessible.

TT_ERR_POINTER

Error Message String:
TT_ERR_POINTER
The opaque pointer (handle) passed does not indicate an object of the proper type.

Description:
The pointer passed does not point at an object of the correct type for this operation. For example, the pointer may point to an integer when a character string is needed.

Solution:
a. Check the arguments for the ToolTalk function to find what arguments the function expects.
b. Retry the operation with a pointer for a valid object.

TT_ERR_PROCID

Error Message String:
TT_ERR_PROCID
The process id passed is not valid.

Description:
The process identifier specified is out of date or invalid.

Solution:
Retrieve the default procid with tt_default_procid().

TT_ERR_PROPLEN

Error Message String:
TT_ERR_PROPLEN
The property value passed is too long.

Description:
The ToolTalk service accepts property values of up to 64 characters.

Solution:
Shorten the property value to less than 64 characters.

TT_ERR_PROPNAME

Error Message String:
TT_ERR_PROPNAME
The property name passed is syntactically invalid.

Description:
The property name is too long, contains non-alphanumeric

Solution:
Check the property name, modify if necessary, and retry the operation.

TT_ERR_PTYPE

Error Message String:
TT_ERR_PTYPE
The process type passed is not the name of an installed process type.
Description:
The ToolTalk service could not locate the specified ptype.
Solution:
If the application was recently installed and the ToolTalk service has not reread the ToolTalk Types Database:
a. Locate the process id for the ttsession.
b. Force the reread with the USR-2 signal:
% ps -elf | grep
ttsession
% kill -USR2
<*ttsession pid*>

TT_ERR_PTYPE_START

Error Message String:
TT_ERR_PTYPE_START
Attempt to launch a client specified in the start attribute of a ptype failed.
Description:
The ToolTalk service could not start the type of process specified.
Solution:
Verify that the application that the ptype represents is properly installed and has execute permission.

TT_ERR_READONLY

Error Message String:
TT_ERR_READONLY
The attribute cannot be changed.
Description:
The application does not have ownership or write permissions for the attribute. Therefore, this operation cannot be performed.
Solution:
NA

TT_ERR_SCOPE

Error Message String:
TT_ERR_SCOPE
The Tt_scope value passed is not valid.
Description:
The scope passed is not recognized by the ToolTalk service.
Solution:
The Tt_scope values are TT_SESSION and TT_FILE. Retry the call with one of these values.

TT_ERR_SESSION

Error Message String:
TT_ERR_SESSION
The session id passed is not the name of an active session.
Description:
An out-of-date or invalid ToolTalk session was specified.
Solution:
Either:
a. obtain the sessid of the current default session using tt_default_session()
b. obtain the sessid of the initial session in which the application was started using tt_initial_session()

TT_ERR_SLOTNAME

Error Message String:
The slot name is syntactically invalid.
Description:
The syntax for the slot name is not valid.
Solution:
Correct the syntax for the slot name.

TT_ERR_STATE

Error Message String:
The Tt_message is in a state that is not valid for the attempted operation.
Description:
The state of the message is invalid for the type of operation being requested.
Solution:
NA

TT_ERR_TOOLATE

Error Message String:
This must be the first call made into the ToolTalk API and can therefore no longer be performed.
Description:
This error will be returned from ToolTalk API calls which require that a specific call be made into the API before any other call is made (such as is the case with use of the ToolTalk multi-thread API calls).
Solution:
NA

TT_ERR_UNIMP

Error Message String:
TT_ERR_UNIMP
Function not implemented.
Description:
The ToolTalk function called is not implemented.
Solution:
NA

TT_ERR_VTYPE

Error Message String:
TT_ERR_VTYPE
The value type name passed is not valid.

Description:
The specified property exists in the ToolTalk database but the type of value does not match the specified type; or the value type is not one that the ToolTalk service recognizes. The ToolTalk service supports types of int and string.

Solution:
a. Change the type of the value to either int or string.
b. Retry the operation.

TT_ERR_XDR

Error Message String:
The XDR procedure failed on the given data, or evaluated to a 0 length structure.

Description:
The XDR procedure failed on the given data, or evaluated to a 0 length structure.

Solution:
NA

TT_OK

Error Message String:
TT_OK
Request successful.

Description:
The call was completed successfully.

Solution:
NA

TT_STATUS_LAST

Error Message String:
TT_STATUS_LAST
This code should be unused.

Description:
This code marks the last of the messages allocated for ToolTalk status.

Solution:
NA

TT_WRN_APPFIRST

Error Message String:
TT_WRN_APPFIRST
This code should be unused.

Description:
This code marks the beginning of the messages allocated for ToolTalk application warnings.

Solution:
NA

TT_WRN_LAST

Error Message String:
TT_WRN_LAST
This code should be unused.
Description:
This code marks the last of the messages allocated for ToolTalk warnings.
Solution:
NA

TT_WRN_NOT_ENABLED

Error Message String:
TT_WRN_NOT_ENABLED
The ToolTalk feature has not been enabled yet in this process.
Description:
This warning can be returned from certain ToolTalk API calls if a particular optional feature of the ToolTalk API (such as multi-threading) has not yet been enabled.
Solution:
NA

TT_WRN_NOTFOUND

Error Message String:
TT_WRN_NOTFOUND
The object was not removed because it was not found.
Description:
The ToolTalk service could not find the specified object in the ToolTalk database. The destroy operation did not succeed.
Solution:
NA

TT_WRN_SAME_OBJID

Error Message String:
TT_WRN_SAME_OBJID
The moved object retains the same objid.
Description:
The object moved stayed within the same file system. The ToolTalk service will retain the same objid and update the location.
Solution:
NA

TT_WRN_STALE_OBJID

Error Message String:
TT_WRN_STALE_OBJID
The object attribute in the message has been replaced with a newer one. Update

the place from which the object id was obtained.

Description:

When the ToolTalk service looked up the specified object in the ToolTalk database, it found a forwarding pointer to the object.

Solution:

The ToolTalk service automatically puts the new objid in the message.
a. Use tt_message_object() to retrieve the new objid.
b. Update any internal application references to the new objid.

TT_WRN_START_MESSAGE

Error Message String:

TT_WRN_START_MESSAGE
This message caused this process to be started. This message should be replied to even if it is a notice.

Description:

When the ToolTalk service starts an application to deliver a message to it, a reply to that message must be sent even if the message which ToolTalk is attempting to deliver is a notice.

Solution:

Use tt_message_accept() or tt_message_reply() to reply to, fail, or reject the message after the process is started by the ToolTalk service.

TT_WRN_STOPPED

Error Message String:

TT_WRN_STOPPED
The query was halted by the filter procedure.

Description:

The query operation being performed was halted by the Tt_filter_function.

Solution:

NA

NAME

tt_type_comp – compile ToolTalk otypes and ptypes

SYNOPSIS

tt_type_comp [–mMs] [–d *db***]** *source_file*

tt_type_comp –r [–s] [–d *db***]** *type* **...**

tt_type_comp –p | –O | –P [–sE] [–d *db***]**

tt_type_comp –p | –O | –P [–s] *source_file*

tt_type_comp –x [–s] [–o *compiled_file***]** *source_file*

tt_type_comp [–hv]

DESCRIPTION

The **tt_type_comp** utility processes otypes and ptypes. The default action of **tt_type_comp** is to compile types from source form into compiled form and then merge the compiled types into the standard ToolTalk types databases. The **tt_type_comp** utility preprocesses the source types with **cpp**(1), and can optionally write out the compiled types instead of merging them into the standard databases. The **tt_type_comp** utility can also remove types from the standard databases or write out the contents of these databases.

The **tt_type_comp** utility operates in two fundamental modes: XDR and Classing Engine. XDR mode is the default. In XDR mode, the standard databases are simply serialized ToolTalk data structures, and the format of **tt_type_comp** output files is the same as that of the databases. In Classing Engine mode, the standard databases are in fact the Classing Engine's own databases, and the format of **tt_type_comp** output files is that expected for input to **ce_db_build**(1) and **ce_db_merge**(1).

OPTIONS

The following options are available:

–d *db* Specify the database to work on, which must be one of **user**, **system** or **network**. For Classing Engine mode these are defined as:

user	**$HOME/.cetables/cetables**
system	**/etc/cetables/cetables**
network	**$OPENWINHOME/lib/cetables/cetables**

For the XDR format these are defined respectively as the first, second, and last elements of **$TTPATH**.

These databases form a hierarchy in which the definition of a type in the user database overrides the definition in the system database, and so on. For the merge and remove options, the default database is **user**. For the **–p**, **–O** and **–P** options, the default is all three databases.

–E Use Classing Engine mode, instead of the default XDR mode.

–G Perform garbage collection on the ToolTalk database. See **ttdbserverd**(1M).

–h Write a help message for invoking **tt_type_comp** and then exit.

–m Merge types into the specified database, updating any existing type with the new definition given. This is the default action. This action is not supported

for Classing Engine mode.

–M Merge types into the specified database (see **–m**), but only if they do not already exist in that database. This action is not supported for Classing Engine mode.

–O Write the names of all otypes read.

–p Write the ToolTalk types read in a source format suitable for recompilation with **tt_type_comp**.

–P Write the names of all ptypes read.

–o *compiled_file*
Write the compiled types into the specified file, or to standard output if *compiled_file* is –.

–r Remove the given ptypes or otypes from the specified database, as indicated by the *type* operands.

–s Silent mode. Write nothing to standard output.

–v Write the version number of **tt_type_comp** and then exit.

–x Compile source types into a compiled types file, instead of merging them into the standard types databases.

These options will be passed through **tt_type_comp** to **cpp**: **–undf –Dname –Idirectory –Uname –Ydirectory**.

OPERANDS

The following operands are supported:

source_file
A pathname of a text file containing ToolTalk source code. If *source_file* is –, standard input is used.

type A name of a type to be removed by the **–r** option.

STDIN

The standard input is used only if a *source_file* operand is –.

INPUT FILES

The input file named by *source_file* is a text file containing ToolTalk source code.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of **tt_type_comp**:

CEPATH In Classing Engine mode, a colon-separated list of directories that tells the Classing Engine where to find the databases that contain (among other things) ToolTalk types. See **ce_db_build**(1).

LANG Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

TTPATH In XDR mode, a colon-separated list of directories that tells the ToolTalk service where to find the ToolTalk types databases. If *TTPATH* has no value or is not set, it is considered to be:

$HOME/.tt:
/etc/tt:
/usr/dt/appconfig/tttypes:
$OPENWINHOME/etc/tt

RESOURCES None.

ASYNCHRONOUS EVENTS The **tt_type_comp** utility takes the standard action for all signals.

STDOUT When the –**h** option is used, **tt_type_comp** writes to standard output a help message in an unspecified format.

When the –**o** option is used, **tt_type_comp** writes to standard output a listing of all otypes read.

When the –**p** option is used, **tt_type_comp** writes to standard output a listing of all the ToolTalk types read, in a source format suitable for recompilation with **tt_type_comp**.

When the –**P** option is used, **tt_type_comp** writes to standard output a listing of all ptypes read.

When the –**v** option is used, **tt_type_comp** writes to standard output a version number in an unspecified format.

STDERR Used only for diagnostic messages.

OUTPUT FILES When the –**x** or –**d user** option is used, **tt_type_comp** writes the compiled types in an unspecified format into a user-specified file. Otherwise, it writes the compiled types into the databases described under –**d**.

EXTENDED DESCRIPTION None.

EXIT STATUS The following exit values are returned:

0 Successful completion.

1 Usage; **tt_type_comp** was given invalid command line options.

2 A syntax error was found in the source types given to **tt_type_comp**.

3 System error; **tt_type_comp** was interrupted by **SIGINT**, or encountered some system or internal error.

CONSEQUENCES OF ERRORS

Default.

FILES

$HOME/.tt/types.xdr
User's ToolTalk types database for XDR mode

$HOME/.tt/.tt_lock
Lock file for serializing updates to user's ToolTalk types database for XDR mode. If this file exists, will refuse to rewrite the database. If a previous execution of exited abnormally, a copy of this file may be left around; future executions of will exit after printing the message ".tt_lock: File exists" several times. To clear this condition, make sure there are no other processes running, and remove the file.

/etc/tt/types.xdr
System ToolTalk types database for XDR mode

/usr/dt/appconfig/tttypes/types.xdr

$OPENWINHOME/etc/tt/types.xdr
Network ToolTalk types databases for XDR mode

$HOME/.cetables/cetables
/etc/cetables/cetables
$OPENWINHOME/lib/cetables/cetables
Classing Engine databases containing ToolTalk types for CE mode. See **ce_db_build**(1).

APPLICATION USAGE

None.

EXAMPLES

None.

SEE ALSO

ttsession(1), **ce_db_build**(1), **ce_db_merge**(1), **cpp**(1).

NAME

ttcp – copy files and inform the ToolTalk service

SYNOPSIS

ttcp [–**pL]** *filename1 filename2*

ttcp –**r [**–**pL]** *directory1 directory2*

ttcp [–**prL]** *filename* **...** *directory*

ttcp –**h** | –**v**

DESCRIPTION

The **ttcp** utility invokes the **cp**(1) utility to copy files and directories, and informs ToolTalk about its actions so that the ToolTalk objects associated with those files and directories can also be copied.

OPTIONS

The following options are available:

–**h** Write a help message for invoking **ttcp** and then exit.

–**L** Copy the ToolTalk objects of the files, but do not invoke **cp**(1) to copy the actual files.

–**p** Preserve. Invoke **cp**(1) with the –**p** option, which duplicates not only the contents of the original files or directories, but also the modification time and permission modes. The modification times of ToolTalk objects are preserved only if the invoking process has appropriate privileges. (Super-user permissions are required.)

–**r** Recursively copy the ToolTalk objects of any directories named, along with their files (including any subdirectories and their files), and pass the –**r** option to **cp**(1).

–**v** Write the version number of **ttcp** and then exit.

The –**f**, –**i** or –**R** options to **cp**(1) are not supported.

OPERANDS

The following operands are supported:

filename
filename1
A pathname of a file to be copied.

filename2
A pathname of an existing or nonexisting file, used for the output when a single file is copied.

directory
directory2
A pathname of a directory to contain the copied files.

directory1
A pathname of a file hierarchy to be copied with –**r**.

STDIN

Not used.

INPUT FILES

The input files specified as operands can be of any file type.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of **ttcp**:

LANG Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

RESOURCES

None.

ASYNCHRONOUS EVENTS

The **ttcp** utility takes the standard action for all signals.

STDOUT

When the –**h** option is used, **ttcp** writes to standard output a help message in an unspecified format.

When the –**v** option is used, **ttcp** writes to standard output a version number in an unspecified format.

STDERR

Used only for diagnostic messages.

OUTPUT FILES

The output files can be of any type.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 All files and ToolTalk objects were copied successfully.

>0 An error occurred or the invoked **cp**(1) command exited with a non-zero value.

CONSEQUENCES OF ERRORS

Default.

FILES

/*mountpoint*/TT_DB The directory used as a database for the ToolTalk objects of files in the file system mounted at */mountpoint*.

APPLICATION USAGE

None.

EXAMPLES

None.

SEE ALSO

cp(1), **ttmv**(1), **tttar**(1), **ttsession**(1).

NAME

ttmv – move or rename files and inform the ToolTalk service

SYNOPSIS

ttmv [–fL] *pathname1 pathname2*

ttmv [–fL] *pathname* **...** *directory*

ttmv –h | –v

DESCRIPTION

The **ttmv** utility invokes **mv**(1) to move files and directories around in the file system and informs ToolTalk about its actions so that the ToolTalk objects associated with those files and directories can also be moved.

The **ttmv** utility moves the ToolTalk objects before it moves the files and does not check whether the file-moving operation will succeed before performing the object-moving operation.

OPTIONS

The following options are available:

–f Force. Do not report any errors, and pass the **–f** option to **mv**(1).

–h Write a help message for invoking **ttmv** and then exit.

–L Move the ToolTalk objects of the files, but do not invoke **mv**(1) to move the actual files.

–v Write the version number of **ttmv** and then exit.

The **–i** option to **cp**(1) is not supported.

OPERANDS

The following operands are supported:

pathname1
A pathname of a file to be moved.

pathname2
A pathname of an existing or nonexisting file, used for the output when a single file is moved.

directory
A pathname of a directory to contain the moved files.

STDIN

Not used.

INPUT FILES

The input files specified as operands can be of any file type.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of **ttmv**:

LANG Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other

internationalization variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

RESOURCES

None.

ASYNCHRONOUS EVENTS

The **ttmv** utility takes the standard action for all signals.

STDOUT

When the **–h** option is used, **ttmv** writes to standard output a help message in an unspecified format.

When the **–v** option is used, **ttmv** writes to standard output a version number in an unspecified format.

STDERR

Used only for diagnostic messages.

OUTPUT FILES

The output files can be of any type.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 All files and ToolTalk objects were moved successfully.

>0 An error occurred or the invoked **mv**(1) command exited with a non-zero value.

CONSEQUENCES OF ERRORS

Default.

FILES

/mountpoint/TT_DB The directory used as a database for the ToolTalk objects of files in the file system mounted at */mountpoint*.

APPLICATION USAGE

None.

EXAMPLES

None.

SEE ALSO

mv(1), **ttsession**(1).

NAME

ttrm – remove files or directories and inform the ToolTalk service

SYNOPSIS

ttrm [–frL] *pathname* **...**

ttrm –h | –v

DESCRIPTION

The **ttrm** utility invokes **rm**(1) to remove files and directories and informs ToolTalk about its actions so that the ToolTalk objects associated with the deleted files and directories can also be deleted.

The **ttrm** utility removes the ToolTalk objects before it removes the files and does not check whether the file-removing operation will succeed before performing the object-removing operation.

OPTIONS

The following options are available:

–f Force. Do not report any errors, and pass the **–f** option to **rm**(1).

–h Write a help message for invoking **ttrm** and then exit.

–L Remove the ToolTalk objects of the files or directories, but do not invoke **rm**(1) to remove the actual files or directories.

–r Recursively remove the ToolTalk objects of any directories named, along with their files (including any subdirectories and their files), and pass the **–r** option to **rm**(1).

–v Write the version number of **ttrm** and then exit.

The **–i** or **–R** options to **rm**(1) are not supported.

OPERANDS

The following operand is supported:

pathname
A pathname of a file to be removed.

STDIN

Not used.

INPUT FILES

The input files specified as operands can be of any file type.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of **ttrm**:

LANG Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative

messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

RESOURCES None.

ASYNCHRONOUS EVENTS The **ttrm** utility takes the standard action for all signals.

STDOUT When the **–h** option is used, **ttrm** writes to standard output a help message in an unspecified format.

When the **–v** option is used, **ttrm** writes to standard output a version number in an unspecified format.

STDERR Used only for diagnostic messages.

OUTPUT FILES None.

EXTENDED DESCRIPTION None.

EXIT STATUS The following exit values are returned:

0 All files and ToolTalk objects were removed successfully.

>0 An error occurred or the invoked **rm**(1) command exited with a non-zero value.

CONSEQUENCES OF ERRORS Default.

FILES

/mountpoint/TT_DB The directory used as a database for the ToolTalk objects of files in the file system mounted at */mountpoint*.

APPLICATION USAGE None.

EXAMPLES None.

SEE ALSO **rm**(1), **ttrmdir**(1), **ttsession**(1).

NAME

ttrmdir – remove empty directories and inform the ToolTalk service

SYNOPSIS

ttrmdir [–L] *directory* **...**

ttrmdir –h | **–v**

DESCRIPTION

The **ttrmdir** utility invokes **rmdir**(1) to remove empty directories and informs ToolTalk about its actions so that the ToolTalk objects associated with the deleted directories can also be deleted.

The **ttrmdir** utility removes the ToolTalk objects before it removes the directories and does not check whether a directory is empty or whether the directory-removing operation will succeed before performing the object-removing operation.

OPTIONS

The following options are available:

–h Write a help message for invoking **ttrmdir** and then exit.

–L Remove the ToolTalk objects of the directories, but do not invoke **rmdir**(1) to remove the actual directories.

–v Write the version number of **ttrmdir** and then exit.

The **–p** option to **cp**(1) is not supported.

OPERANDS

The following operand is supported:

directory
A pathname of an empty directory to be removed.

STDIN

Not used.

INPUT FILES

The input files specified as operands can be of any file type.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of **ttrmdir**:

LANG Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

RESOURCES

None.

ASYNCHRONOUS EVENTS

The **ttrmdir** utility takes the standard action for all signals.

STDOUT

When the –**h** option is used, **ttrmdir** writes to standard output a help message in an unspecified format.

When the –**v** option is used, **ttrmdir** writes to standard output a version number in an unspecified format.

STDERR

Used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values are returned:

0 All directories and ToolTalk objects were removed successfully.

>0 An error occurred or the invoked **rmdir**(1) command exited with a non-zero value.

CONSEQUENCES OF ERRORS

Default.

FILES

/*mountpoint*/TT_DB The directory used as a database for the ToolTalk objects of files in the file system mounted at */mountpoint*.

APPLICATION USAGE

The definition of an empty directory is one that contains, at most, directory entries for dot and dot-dot.

EXAMPLES

None.

SEE ALSO

rmdir(1), **ttrm**(1), **ttsession**(1).

NAME

ttsession – the ToolTalk message server

SYNOPSIS

ttsession [–hNpsStv] [–E | –X] [–a *level***] [–d** *display***] [–c [***command***]]**

DESCRIPTION

The **ttsession** utility is the ToolTalk message server. This background process must be running before any messages can be sent or received. Each message server defines a *session*.

The message server has no user interface and typically runs in the background, started either by the user's **.xinitrc** file or automatically by any program that needs to send or receive a message.

OPTIONS

The following options are available:

–a *level*

Set the server authentication level. The following *level* string values are supported:

unix The sender and receiver must have the same user ID.

des The underlying RPC calls use **AUTH_DES**.

gss The underlying RPC calls use **RPCSEC_GSS**.

For **gss**, additional options may be specified after an immediately following comma in a comma separated (without spaces) list of suboptions and keyword-attribute pairs:

protect={*access*,*integrity*,*privacy*}
Specify the GSS service type. If no service type is specified, **protect**=*access* is assumed.

protect=*access*
means the GSS mechanism is used to verify that clients have the credentials of the user that started **ttsession**.

protect=*integrity*
means the GSS mechanism is used to verify integrity of the data transmitted between **ttsession** and its clients. **protect**=*integrity* implies **protect**=*access*

protect=*privacy*
means the Gss mechanism is used to provide privacy against the data transmitted between **ttsession** and its clients. **protect**=*integrity* implies **protect**=*access.*

mechanism=<*mechanism name*>
Use the *named security mechanism*. If no mechanism is specifierd, **ttsession** wil arbitrarily choose one of the installed mechanisms

qop=<*quality of protection*>
Use the named *quality of protection*. If the quality of protection is not specified, the default for the mechanism is used.

EXAMPLE

To specify *GSS* authentication using the *kerberos_v5* mechanism and the *GSS_KRB5_CONF_C_QOP_DES* quality of protection, start **ttsession** as:

```
ttsession -a gss,mechanism=kerberos_v5,qop=GSS_KRB5_CONF_C_QOP_DES
```

This assumes that the *kerberos_v5* mechanism is installed.

Security options can be overridden on a systemwide basis via the file /etc/default/ttsession (ttsession_file(4)).

–c [*command***]**
Start a process tree session and run the given command. The **ttsession** utility sets the environment variable *TT_SESSION* to the name of this session. Any process started with this variable in the environment defaults to being in this session. If *command* is omitted, **ttsession** invokes the shell named by the *SHELL* environment variable. Everything after **–c** on the command line is used as the command to be executed.

–d *display*
Specify an X Windows display. The ToolTalk session will consist of those applications displaying on the named display. The default display is identified by the *DISPLAY* environment variable.

–E Read in the types from the Classing Engine database. If neither **–E** nor **–X** is given, **–X** is assumed.

–h Write a help message to standard error that describes the command syntax of **ttsession**, and exit.

–N Maximize the number of clients allowed to connect to (in other words, open procids in) this session by attempting to raise the limit of open file descriptors. The precise number of clients is system-dependent; on some systems this option may have no effect. On Solaris 2.6 and later, **ttsession** always maximizes the number of clients, so there is no need to specify this option.

–o *allow_unauth_types_load=<yes | no>*
By default calls to tt_session_types_load(3) in the ToolTalk API will fail with TT_ERR_ACCESS. The system wide default in this regard may be changed via ttsession_file(4). The behavior for a particular ttsession may be changed via this option, if and only if the ttsession_file(4) has not "locked" per-ttsession changes to this option.

–p Write the name of a new process tree session to standard output, and then fork a background instance of **ttsession** to manage this new session.

–s Silent. Do not write any warning messages to standard error.

–S Do not fork a background instance to manage the **ttsession** session.

–t Turn on trace mode. See **ASYNCHRONOUS EVENTS** for how to turn

tracing on and off during execution. Tracing displays the state of a message when it is first seen by **ttsession**. The lifetime of the message is then shown by showing the result of matching the message against type signatures (dispatch stage) and then showing the result of matching the message against any registered message patterns (delivery stage). Any attempt to send the message to a given process is also shown together with the success of that attempt.

–v Write the version number to standard output and exit.

–X Read in the types from the XDR format databases. (Default)

OPERANDS

None.

STDIN

Not used.

INPUT FILES

The XDR format databases listed by the **–X** option are serialized ToolTalk data structures of an unspecified format, except that it is the same as the format of **tt_type_comp**(1) output files.

The file /etc/default/ttsession (ttsession_file(4)) can be used to change the system-wide behavior of the ttsession process depending on the contents of the file.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of **ttsession**:

CEPATH In Classing Engine mode, this variable tells the Classing Engine where to find the databases that contain ToolTalk types. See **ce_db_build**(1).

DISPLAY If *TT_SESSION* is not set and *DISPLAY* is set, then the value of *DISPLAY* will be used by all ToolTalk clients to identify the **ttsession** process serving their X display. If no such process is running, the ToolTalk service will auto-start one.

If **ttsession** is run with the **–d** option and *DISPLAY* is not set, **ttsession** sets *DISPLAY* to be the value of the **–d** option for itself and all processes it forks. This helps ToolTalk clients to find the right X display when they are auto-started by **ttsession**.

LANG Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of

LC_MESSAGES.

TT_ARG_TRACE_WIDTH
Specify the number of bytes of argument and context values to write when in trace mode. The default is to print the first 40 bytes.

TTPATH In XDR mode, a colon-separated list of directories that tells ToolTalk where to find the ToolTalk types databases. See **tt_type_comp**(1).

TTSESSION_CMD
Specify the shell command to be used by all ToolTalk clients for auto-starting **ttsession**.

The **ttsession** utility creates the following variable when it invokes another process:

TT_FILE When **ttsession** invokes a tool to receive a message, it copies the file attribute (if any) of the message into this variable, formatted in the same manner as returned by the **tt_message_file**(3) function.

TT_SESSION The **ttsession** utility uses this variable to communicate its session ID to the tools that it starts. The format of the variable is implementation specific. If this variable is set, the ToolTalk client library uses its value as the default session ID.

TT_TOKEN Inform the ToolTalk client library that it has been invoked by **ttsession**, so that the client can confirm to **ttsession** that it started successfully. The format of the variable is implementation specific.

A tool started by **ttsession** must ensure that the *TT_SESSION* and *TT_TOKEN* are present in the environment of any processes it invokes.

RESOURCES

None.

ASYNCHRONOUS EVENTS

The **ttsession** utility reacts to two signals. If it receives the **SIGUSR1** signal, it toggles trace mode on or off (see the **–t** option). If it receives the **SIGUSR2** signal, it rereads the types file. The **ttsession** utility takes the standard action for all other signals.

STDOUT

When the **–v** option is used, **ttsession** writes the version number in an unspecified format. When **–p** is used, **ttsession** writes the name of a new process tree session.

STDERR

Used only for diagnostic messages and the help message written by the **–h** option.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

When the **–c** child process exits, **ttsession** exits with the status of the exited child. Otherwise, the following exit values are returned:

0 Normal termination. Without the **–c** or **–S** options, a zero exit status means **ttsession** has successfully forked an instance of itself that has begun serving the session.

1 Abnormal termination. The **ttsession** utility was given invalid command line options, was interrupted by **SIGINT**, or encountered some internal error.

2 Collision. Another **ttsession** was found to be serving the session already.

CONSEQUENCES OF ERRORS

The **ttsession** utility takes the standard action for all signals.

APPLICATION USAGE

Since everything after **–c** on the command line is used as the command to be executed, **–c** should be the last option.

Tracing is helpful for seeing how messages are dispatched and delivered, but the output may be voluminous.

EXAMPLES

None.

SEE ALSO

tt_type_comp(1), **tttrace**(1), **tt_message_file**(3), **ttsession_file**(4).

NAME

ttsnoop – ToolTalk graphical user interface

SYNOPSIS

ttsnoop [*options***] [–F** *scopefile***] [–<** *procid***] [–v** *media***] [–m** *op***]**

ttsnoop [*options***] [–e** *script***] command [***args***]**

ttsnoop [*options***] –n | –N**

ttsnoop [–TuX] [–S *sessid***] [–w** *n***] [–l** *n***] [–o** *snoopfile***] [–O** *tracefile***]**

DESCRIPTION

The **ttsnoop** utility interactively monitors ToolTalk message traffic, **ttsession**(1) pattern matching, and ToolTalk client function calls. **ttsnoop** allows the interactive execution of almost any valid sequence of ToolTalk function calls, while optionally tracing those calls. **ttsnoop** can interactively create and send any ToolTalk message, and can automatically create many of the standard ToolTalk messages. **ttsnoop** can interactively create and register any ToolTalk pattern. Messages received by virtue of these patterns can be processed (e.g. replied to) interactively or automatically. For any message encountered, **ttsnoop** can clone a copy of it or generate a pattern that will match similar messages. For any message encountered, **ttsnoop** can generate C source code for recreating it, **dtactionfile**(4) source for a message action that will send it, or **tt_type_comp**(1) source for a static pattern that will match it. **ttsnoop** can dump the system's installed static patterns and message actions.

Like **tttrace**(1), **ttsnoop** operates in two fundamental modes. **ttsnoop** either runs **command** with ToolTalk client tracing turned on, or (if **command** is omitted) snoops message traffic in the default ToolTalk scope. For client tracing, **ttsnoop** simply invokes **tttrace** and logs the trace output to the **ttsnoop** terminal pane. For message snooping, **ttsnoop** registers a ToolTalk pattern in the default scope and prints each matching message in the terminal pane. When **ttsnoop** receives a message, it prints the current time, the address of the internal **Tt_message_callback** that received the message, and a description of the message as generated by **tt_message_print**(3).

OPTIONS

The following options are available:

–F *scopefile*
: Scope initial pattern also to *scopefile*.

–< *procid*
: Limit initial pattern to messages from *procid*.

–v *mediaType*
: Limit initial pattern to messages for *mediaType*.

–m *op*
: Limit initial pattern to messages with *op*.

–n
: Skip initial pattern.

–N
: Skip initial **ttdt_open**(3), also.

–e *script*
: Take *script* as a **tttrace**(1) setting. See **tttracefile**(4).

–T Trace (even initial) ToolTalk API calls made by **ttsnoop.**

–u Map (de-iconify) on snoop output.

–S *sessid*
Set default session to *sessid*.

–X Set default session to the X session of $DISPLAY.

–w *n* Set global timeout to *n* seconds.

–l *n* Set **tttrace dtterm** saveLines to *n* lines.

–o *snoopfile*
Log snoop output to *snoopfile*.

–o *tracefile*
Log API tracing to *tracefile*.

OPERANDS

The following operands are supported:

command [args]
Invoke *command* [with *args*] and snoop its ToolTalk API calls.

RESOURCES

The main widgets that make up the **ttsnoop** hierarchy are shown under this heading to aid in specifying resources. The widget instance name is shown first, followed by the widget class name in parentheses. Indentation indicates hierarchical structure.

Ttsnoop
ttsnoopWin (topLevelShellWidgetClass)
dtb_ttsnoop_ttsnoop_pane_pane (DtTerm)
dtb_message_props_message_props (DialogShell)
dtb_pattern_props_pattern_props (DialogShell)
dtb_api_tracer_tracer (DialogShell)
dtb_api_tracer_trace_pane_obj_pane (DtTerm)

STDIN

Not used.

INPUT FILES

None.

ASYNCHRONOUS EVENTS

ToolTalk Messages

If *command* is omitted, **ttsnoop** by default registers a pattern in the default scope to observe all messages, and prints all observed messages to the terminal pane. **ttsnoop** can be made to register the handler patterns described in **ttdt_session_join**(3), **ttdt_file_join**(3), **ttdt_message_accept**(3), **ttdt_subcontract_manage**(3), and **ttmedia_ptype_declare**(3). **ttsnoop** can interactively create and register any ToolTalk handler pattern. The user can install a pattern callback to open the matched messages in a dialog window, or automatically accept, reject, reply, or fail it.

STDOUT

Not used.

STDERR

Errors encountered during initialization are written to stderr. After initialization, stderr is not used.

OUTPUT FILES

None.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

1 Could not open X display.

2 Incorrect command line options.

3 Could not fork *command*.

4 ToolTalk initialization error.

5 Caught a fatal signal.

NOTES

Like any ToolTalk client, **ttsnoop** can observe multicast messages, but not **TT_HANDLER**-addressed messages. Only **tttrace**(1) can monitor **TT_HANDLER**-addressed messages. Run **tttrace**(1) by choosing "ttsession" from the "Snoop" menu.

ttsnoop allows a message to be opened as long as it has not been destroyed. However, **ttsnoop** exposes certain ToolTalk convenience routines -- **ttdt_Get_Modified**(3), **ttdt_Save**(3), and **ttdt_Revert**(3) -- that send and destroy requests without ever returning them above the API. When **ttsnoop** uses these routines to send a request to itself, the message should not be manipulated after the convenience routines have destroyed it.

FILES

/usr/dt/app-defaults/$LANG/Ttsnoop
ttsnoop Application Defaults.

SEE ALSO

tttrace(1), **ttsession**(1), **tt_type_comp**(1), **dttypes**(1), **truss**(1), **DtTerm**(3), **dtactionfile**(4)

NAME

tttar – process files and ToolTalk objects in an archive

SYNOPSIS

tttar c| t| x [EfhpSv] [*tarfile* **]** *pathname* **...**

tttar c| t| xfL [EhpRSv] *tttarfile* **[[–rename** *oldname newname***] ...]** *pathname* **...**

tttar –h| –help

tttar –v

DESCRIPTION

The **tttar** utility has two fundamentally different modes.

- Without the **L** function modifier, **tttar** acts as a ToolTalk-aware wrapper for **tar**(1), archiving (or extracting) multiple files and their ToolTalk objects onto (or from) a single archive, called a *tarfile*.
- With the **L** function modifier, **tttar** does not invoke **tar** to archive actual files, but instead archives (or extracts) only ToolTalk objects onto (or from) a single archive, called a *tttarfile*. Since without the **L** function modifier **tttar** acts like an ToolTalk-aware **tar**(1), the description below is phrased as if the **L** function modifier is in effect. That is, the text refers to *tttarfiles* instead of *tarfiles*, and it describes archiving and de-archiving only ''the ToolTalk objects of the named files'' rather than archiving and de-archiving both ''the named files and their ToolTalk objects.''

The actions of **tttar** are controlled by the first argument, the *key*, a string of characters containing exactly one function letter from the set **ctx**, and one or more of the optional function modifiers listed under **OPERANDS**. Other arguments to **tttar** are file or directory names that specify which files to archive or extract ToolTalk objects for. By default, the appearance of a directory name refers recursively to the files and subdirectories of that directory.

A file does not have to exist for a ToolTalk object to be associated with its pathname. When **tttar** descends into a directory, it does not attempt to archive the objects associated with any files that do not exist in the directory.

When extracting from a **tar** archive that is given to **tttar** either on magnetic tape or on the standard input, the current working directory must be writable, so that the *tttarfile* can be placed there temporarily.

OPTIONS

The following options are available:

–h
–help Write a help message for invoking **tttar** and then exit.

–rename *oldname newname*
Interpret the next two arguments as an *oldname* and a *newname*, respectively, and rename any entry archived as *oldname* to *newname*. If *oldname* is a directory, then **tttar** recursively renames the entries as well. If more than one **–rename** option applies to an entry (because of one or more parent directories being renamed), the most specific **–rename** option applies.

–v Write the version number of **tttar** and then exit.

OPERANDS

The following operands are supported:

key The *key* operand consists of a function letter followed immediately by zero or more modifying letters.

The function letter is one of the following:

c Create a new archive and write the ToolTalk objects of the named files onto it.

t Write to standard output the names of all the files in the archive.

x Extract the ToolTalk objects of the named files from the archive. If a named file matches a directory with contents in the archive, this directory is (recursively) extracted. The owner and modification time of the ToolTalk objects are restored (if possible). If no *filename* arguments are given, the ToolTalk objects of all files named in the archive are extracted.

The following characters can be appended to the function letter. Appending the same character more than once produces undefined results.

f Use the next argument as the name of the *tttarfile*. If *tttarfile* is given as '–', **tttar** writes to the standard output or reads from the standard input, whichever is appropriate.

h Follow symbolic links as if they were normal files or directories. Normally, **tttar** does not follow symbolic links.

p Preserve. Restore the named files to their original modes, ignoring the present *umask* value (see **umask**(2)). The **tttar** utility also extracts setUID and sticky information for the super-user. This option is only useful with the **x** function letter, and has no meaning if the **L** function letter is given.

L Do not invoke **tar**(1). This modifier must be used with the **f** function modifier, since reading and writing an **tttar** archive directly to or from magnetic tape is unimplemented.

R Do not recurse into directories. This modifier is valid only with the **L** function modifier.

v Verbose. Write to standard error the name of each file processed, preceded by a string indicating the operation being performed, as follows:

Key Letter	String
c	"a "
x	"x "

The file name may be followed by additional information, such as the size of the file in the archive or file system, in an unspecified format. When used with the **t** function letter, **v** writes to standard

output more information about the archive entries than just the name.

The following functions and modifiers are not supported:

- The **r** and **u** function letters of **tar**(1), for incrementally updating an archive.
- The **X** and **F** function modifiers and the –**I** option of **tar**(1), for including or excluding files from being archived based on SCCS status or being listed in a special file.
- The **w** function modifier and the –**C** option of **tar**(1), for pausing or changing directories between the files listed on the command line.
- Writing and reading *tttarfiles* (that is, archives produced with the **L** function modifier) directly to and from magnetic tape.

pathname
A pathname of a regular file or directory to be archived (when the **c** function letter is used), extracted (**x**) or listed (**t**). When *pathname* is the pathname of a directory, the action applies to all of the files and (recursively) subdirectories of that directory. When the **f** letter is used in the *key* operand, the initial *pathname* operand is interpreted as an archive name, as described previously.

tarfile
A pathname of a regular file to be read or written as an archive of files.

ttarfile
A pathname of a regular file to be read or written as an archive of ToolTalk objects.

STDIN

When the **f** modifier is used with the **t** or **x** function letter and the pathname is –, the standard input is an archive file formatted as described in **EXTENDED DESCRIPTION**. Otherwise, the standard input is not used.

INPUT FILES

The files identified by the *pathname* operands are regular files or directories. The file identified by the *tarfile* operand is a regular file formatted as described in **tar**(1). The file identified by the *tttarfile* operand is a regular file formatted as described in **EXTENDED DESCRIPTION**.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of **tttar**:

LANG Provide a default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-specific default locale will be used. If any of the internationalization variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_MESSAGES Determine the locale that is used to affect the format and contents of

diagnostic messages written to standard error and informative messages written to standard output.

NLSPATH Determine the location of message catalogues for the processing of *LC_MESSAGES*.

TZ Determine the timezone used with date and time strings.

RESOURCES

None.

ASYNCHRONOUS EVENTS

The **tttar** utility takes the standard action for all signals.

STDOUT

When the **–h** option is used, **tttar** writes to standard output a help message in an unspecified format.

When the **–v** option is used, **tttar** writes to standard output a version number in an unspecified format.

When the **f** modifier is used with the **c** function letter and the pathname is –, the standard output is an archive file formatted as described in **EXTENDED DESCRIPTION**.

Otherwise, the standard output is not used.

STDERR

The standard error is used for diagnostic messages and the file name output described under the **v** modifier (when the **t** function letter is not used).

OUTPUT FILES

Output files are created, as specified by the archive, when the **x** function letter is used.

EXTENDED DESCRIPTION

The archive file produced and read by **tttar** is formatted as described in **tar**(1), with the addition of one extra file named **tttarfile**. (If one of the user files being archived is also named **tttarfile**, the results are unspecified.) The **tttarfile** contains all the ToolTalk *spec* information for the ToolTalk objects in the other files in the archive. The contents of **tttarfile** are written according to the referenced XDR specification (RFC 1014). The only XDR data types used are:

int A four-octet signed integer, most significant octet first

string A four-octet unsigned integer length, most significant octet first, followed by the characters of the string, followed by sufficient (0 to 3) residual zero octets to make the total number of octets a multiple of four.

The **tttarfile** starts with two integers. The first is always 1, to mark this as the header record. The second is always 1, indicating this is version 1 of the *tttarfile* format. Any future revisions of the *tttarfile* format should increment the version number so older programs processing the *tttarfile* can diagnose the incompatiblity.

The end of the **tttarfile** is a integer 3, marking the end-of-file record.

In between, there is one logical record for each spec. Each logical record starts with an integer 2, marking it as a spec record. Other integer values are reserved for assignment to future data types.

After the record identifier, the spec record contains, in sequence:

1. A string giving the Tooltalk object identifier (*objid*) of the object represented by the spec
2. A string giving the name of the file (as found in the archive table of contents) that contains the contents of the ToolTalk object represented by the spec
3. A string giving the ToolTalk object type identifier (*otid*) of the ToolTalk object represented by the spec
4. An integer giving the number of properties for this object

The properties of the object immediately follow the number of properties. Each property consists of:

1. A string giving the name of the property
2. An integer, which is always zero (for historical compatibility)
3. An integer giving the number of values for this property
4. A string for each value

After the values, the next property is found, until all properties for the object have been accounted for; then the next spec is found, until all specs for objects associated with files in the archive are accounted for.

EXIT STATUS

The following exit values are returned:

0 All files and ToolTalk objects were moved successfully.

>0 An error occurred or the invoked **tar**(1) command exited with a non-zero value.

CONSEQUENCES OF ERRORS

Default.

FILES

/*mountpoint*/TT_DB The directory used as a database for the ToolTalk objects of files in the file system mounted at */mountpoint*.

APPLICATION USAGE

None.

EXAMPLES

None.

SEE ALSO

tar(1), **ttcp**(1), **ttsession**(1).

NAME

tttrace - trace ToolTalk calls and messages

SYNOPSIS

tttrace [**-0FCa**] [**-o** *outfile*] [**-S** *session* | *command*]
tttrace [**-e** *script* | -f *scriptfile*] [**-S** *session* | *command*]

DESCRIPTION

tttrace traces message traffic through the server for the indicated ToolTalk *session*, or runs *command* with ToolTalk client tracing turned on. If neither *session* nor *command* is given, the default session is traced. By default, tracing terminates when **tttrace** exits.

Tracing of ToolTalk functions looks like this:
[*pid*] *function_name*(*params*) = *return_value* (*Tt_status*)

With the **-a** option, message attributes are printed after a one-line summary of the message:
Tt_state Tt_paradigm Tt_class (*Tt_disposition* in *Tt_scope*): *status* == *Tt_status*

State changes are indicated by:
old_state => *new_state*.

Deliveries are indicated by:
Tt_message => procid <*recipient_procid*>

When dispatching is being traced, the reason for each dispatch is one of:

tt_message_send()
tt_message_reject()
tt_message_fail()
tt_message_reply()
tt_session_join()
tt_file_join()
tt_message_reply()
: A client called the indicated function.

tt_message_send_on_exit()
: **ttsession** is dispatching on_exit messages for a client that disconnected before calling **tt_close**().

tt_message_accept()
: **ttsession** is dispatching messages that had been blocked while a ptype was being started. The started client has now called either **tt_message_accept**() or **tt_message_reply**() to indicate that the ptype should be unblocked.

TT_ERR_PTYPE_START
: A ptype instance was started to receive the message, but the start command exited before it connected to **ttsession**.

TT_ERR_PROCID
: **ttsession** lost its connection to the client that was working on this request.

ttsession -> ttsession
: Another session wants this session to find recipients for the message.

ttsession <- ttsession

Another session wants to update (e.g. fail) a message originating in this session.

When dispatching is being traced, matching is indicated by one of
Tt_message & Tt_pattern {
Tt_message & ptype *ptid* {
Tt_message & otype *otid* {

The pattern or signature is printed, followed by
} == *match_score*; [/* *mismatch_reason* */]

OPTIONS

-0 Turn off message tracing in *session*, or run *command* without message tracing (i.e., with only call tracing).

-F Follow all children forked by *command* or subsequently started in *session* by **ttsession**. Normally, only the indicated *command* or **ttsession** instance is traced. When **-F** is specified, the process id is included with each line of trace output to indicate which process generated it.

-C Do not trace client calls into the ToolTalk API. Default is to trace them.

-a Print all attributes, arguments, and context slots of traced messages. The default is to use only a single line when printing a message on the trace output.

-e *script* Take *script* as a **tttrace** setting. See **tttracefile**(4).

-f *scriptfile*
File to read **tttrace** settings from. See **tttracefile**(4). **-f -** causes **tttrace** to read standard input until EOF, which may prevent *command* from using standard input.

-o *outfile*
File to be used for the trace output.

For session tracing, output goes to standard output of **tttrace**.

For client tracing, output goes by default to standard error of **tttrace**. For client tracing, **-o -** causes trace output to go to standard output of **tttrace**.

If the server for *session* is running on a remote host and either
- *outfile* is not mounted on that host, or
- the **-o** option is omitted,

then **tttrace** will fail.

-S *session*
Session to trace. Defaults to the *default session* -- the session that tt_open() would contact.

command
The ToolTalk client command to invoke and trace.

EXAMPLES

Here we trace a client that registers a pattern and sends a notice that matches it:

% **tttrace** -a *myclientprogram*

```
tt_open() = 0x51708=="7.jOHHM X 129.144.153.55 0" (TT_OK)
tt_fd() = 11 (TT_OK)
tt_pattern_create() = 0x50318 (TT_OK)
tt_pattern_category_set(0x50318, TT_OBSERVE) = 0 (TT_OK)
tt_pattern_scope_add(0x50318, TT_SESSION) = 0 (TT_OK)
tt_pattern_op_add(0x50318, 0x2f308=="Hello World") = 0 (TT_OK)
tt_default_session() = 0x519e0=="X 129.144.153.55 0" (TT_OK)
tt_pattern_session_add(0x50318, 0x519e0=="X 129.144.153.55 0") = 0 (TT_OK)
tt_pattern_register(0x50318) = 0 (TT_OK)
tt_message_create() = 0x51af0 (TT_OK)
tt_message_class_set(0x51af0, TT_NOTICE) = 0 (TT_OK)
tt_message_address_set(0x51af0, TT_PROCEDURE) = 0 (TT_OK)
tt_message_scope_set(0x51af0, TT_SESSION) = 0 (TT_OK)
tt_message_op_set(0x51af0, 0x2f308=="Hello World") = 0 (TT_OK)
tt_message_send(0x51af0)   ...
   TT_CREATED => TT_SENT:
      TT_SENT TT_PROCEDURE TT_NOTICE (TT_DISCARD in TT_SESSION): 0 == TT_OK
      id:      0 7.jOHHM X 129.144.153.55 0
      op:      Hello World
      session: X 129.144.153.55 0
      sender:  7.jOHHM X 129.144.153.55 0
= 0 (TT_OK)
tt_message_receive()       ...
   Tt_message => procid <7.jOHHM X 129.144.153.55 0>
   TT_SENT TT_PROCEDURE TT_NOTICE (TT_DISCARD in TT_SESSION): 0 == TT_OK
   id:         0 7.jOHHM X 129.144.153.55 0
   op:         Hello World
   session:    X 129.144.153.55 0
   sender:     7.jOHHM X 129.144.153.55 0
   pattern:    0:7.jOHHM X 129.144.153.55 0
= 0x51af0 (TT_OK)
```

ttsession's view of this traffic can be seen as follows. Note that the first message traced will almost always be **ttsession**'s reply to the request sent it by **tttrace**.

% **tttrace** -a

```
tt_message_reply:
 TT_SENT => TT_HANDLED:
  TT_HANDLED TT_PROCEDURE TT_REQUEST (TT_DISCARD in TT_SESSION): 0 == TT_OK
  id:         0 2.jOHHM X 129.144.153.55 0
  op:         Session_Trace
  args:       TT_IN string: "> /tmp/traceAAAa002oL; version 1; states"[...]
  session:    X 129.144.153.55 0
  sender:     2.jOHHM X 129.144.153.55 0
  pattern:    0:X 129.144.153.55 0
  handler:    0.jOHHM X 129.144.153.55 0
```

```
  Tt_message => procid <2.jOHHM X 129.144.153.55 0>
tt_message_send:
   TT_CREATED TT_PROCEDURE TT_NOTICE (TT_DISCARD in TT_SESSION): 0 == TT_OK
   id:         0 7.jOHHM X 129.144.153.55 0
   op:         Hello World
   session:    X 129.144.153.55 0
   sender:     7.jOHHM X 129.144.153.55 0
   TT_CREATED => TT_SENT:
      TT_SENT TT_PROCEDURE TT_NOTICE (TT_DISCARD in TT_SESSION): 0 == TT_OK
      id:         0 7.jOHHM X 129.144.153.55 0
      op:         Hello World
      session:    X 129.144.153.55 0
      sender:     7.jOHHM X 129.144.153.55 0
   Tt_message & Tt_pattern {
      id:         0:7.jOHHM X 129.144.153.55 0
      category:   TT_OBSERVE
      scopes:     TT_SESSION
      sessions:   X 129.144.153.55 0
      ops:          Hello World
   } == 3;
   Tt_message => procid <7.jOHHM X 129.144.153.55 0>
```

To trace message flow in a specific, non-default session,

% **tttrace -S** "01 15303 1342177284 1 0 13691 129.144.153.55 2"

ENVIRONMENT

tttrace is implemented purely as a ToolTalk client, using the message interface to **ttsession** and the following environmental hook into libtt.

TT_TRACE_SCRIPT

If set, tells libtt to turn on client-side tracing as specified in the trace script. If the first character of of the value is '.' or '/', the value is taken to be the pathname of file containing the trace script to use. Otherwise, the value is taken to be an inline trace script.

FILES

$TMPDIR/tttrace.nnn

A *named pipe* (see **mkfifo**(3C)) in **$TMPDIR** (see **tempnam**(3S)) from which trace output for *session* is read when the **-o** option is ommitted.

WARNINGS

Since (with the **-F** option) tracing can follow clients to remote hosts if the environment is properly propagated, it is possible for different processes in the same trace output to be labeled with the same process id.

SEE ALSO

ttsession(1), **tttracefile**(4), the **Session_Trace**() ToolTalk request

DIAGNOSTICS

If *command* is run, then **tttrace** will exit with the exit status of *command*. Otherwise, exit codes are as follows:

0 Normal termination. Any *session* tracing turned on by this invocation of **tttrace** has now been turned off.

1 Usage. **tttrace** was given invalid command line options.

2 Failure. **tttrace** encountered an error while trying to do its job. An error message has been emitted on standard error.

3 Runaway *session* tracing. **tttrace** could not terminate tracing in *session* before exiting.

4 Remote *session*. **ttsession** is remote, and *outfile* (if given) is not visible there. Choose a visible file, or run **tttrace** on that remote host.

5 Old *session*. The **ttsession** for *session* does not support the Session_Trace() request. Run **kill -USR1** on it to turn on old-style tracing.

NOTES

For security purposes, client-side tracing is disabled inside a client when its effective uid or gid is different from its real uid or gid and the real uid is not the super-user.

NAME

ttdbck – display, check, or repair ToolTalk databases

SYNOPSIS

ttdbck [*selection opts*] [*diagnosis opts*] [*display opts*] [*repair opts*]
[*data-base-directory*]...

DESCRIPTION

ttdbck is the ToolTalk database maintenance tool. It allows direct inspection of ToolTalk spec data, detection of inconsistencies, and repair of problems.

OPTIONS

data-base-directory

Names the directory or directories containing the ToolTalk database to be inspected or repaired. If no directories are named, the current directory is assumed. If a directory path does not end in ‘‘TT_DB’’, ‘‘TT_DB’’ is appended.

The user running the command must have read access to the files in the directory to inspect the data and write access to repair the data. Since ToolTalk databases are typically accessible only to root, this command is normally run as root.

Selection options

The selection options determine which specs in the database are displayed or modified. If no selection options are given, all specs in the database are displayed. To prevent massive accidental changes to ToolTalk databases, no repair options except **-I** are allowed unless a selection or diagnosis option is given.

–**f** *filename*

Restricts the set of specs to be inspected or modified to those which describe objects in the named file. The file name can contain shell-style wildcards which must be escaped to prevent the shell from expanding them.

–**k** *objidkey*

An object id key, specifying a particular spec to be displayed or modified. The object id key can be obtained from a previous invocation of **ttdbck**; one might display a set of specs, determine the one that needs repair, and specify its key here.

–**t** *type* Restricts the set of specs to be inspected or modified to those with otype *type*. The type name can contain shell-style wildcards which must be escaped to prevent the shell from expanding them.

Diagnosis options

These options check for and report on inconsistencies in the selected specs. Only specs selected by the selection options are checked. If a diagnosis option is given, any display or repair option is applied only to specs which fail the diagnostic check.

–**b** Check for badly formed specs: those which have no file or type or those which have types not defined in the type database.

–**x** Check for specs which refer to files that no longer exist.

Display options

These options determine which data is printed for each selected spec.

–**i** Display the object id (including the object id key.)

–**m** Display the mandatory data that must appear in every spec: the otype of the

object described by the spec and the file in which the spec is stored.

–p Display all the properties and values for each selected spec.

–a Display all data (equivalent to specifying **–imp**)

Repair options

–I Invoke the NetISAM isrepair() function for all files accessed. This action is applied before any other inspection or repair action. This option should be used when normal operations return EBADFILE (error code 105).

–F *filename*
Change the file name for the selected specs to the supplied file name.

–T *otypeid*
Change the type of the selected specs to the given otype.

–Z Remove the selected specs entirely.

EXAMPLES

ttdbck -bxi /home

In the /home/TT_DB directory, finds all badly formed specs and specs that refer to non-existent files and prints their ids.

ttdbck -f /home/sample/data -F /home/sample/data1 /home

In the /home/TT_DB directory, finds all specs that refer to objects in file /home/sample/data and changes them to refer to /home/sample/data1.

ttdbck -t Sample_Otype_Name -Z /export/TT_DB

In the /export/TT_DB directory, finds all specs that refer to objects of type *Sample_Otype_Name* and deletes the specs.

FILES

/path/TT_DB ToolTalk database

NOTES

The **ttdbck** command should be run on the same machine where the TT_DB files being inspected and repaired physically exist. That is, don't try to access the TT_DB files via NFS.

NAME

ttdbserver, rpc.ttdbserverd – RPC-based ToolTalk database server

SYNOPSIS

rpc.ttdbserverd [**-G**] [**-m** DTMOUNTPOINT_value**]** [**-n**] [**-v**] [**-?**]

DESCRIPTION

rpc.ttdbserverd manages ToolTalk objects created by **tt_spec_create**(3), and handles certain queries related to the netfiles returned by **tt_file_netfile**(3). One instance of **rpc.ttdbserverd** (normally started by **inetd**) runs on each host that has a local filesystem. **rpc.ttdbserverd** serves four purposes:

1. Mapping a spec to its associated file and a file to its associated specs.
2. Mapping a spec to its properties.
3. Mapping a file to a list of sessions with clients having patterns registered in the scope of that file.
4. Answering netfile queries; see **tt_file_netfile**(3) and **tt_host_file_netfile**(3).

For each filesystem that **rpc.ttdbserverd** needs to store information about, it creates a directory called TT_DB at the mountpoint of that file system. In that directory it creates the databases it needs to store its tables and indices. If the partition is not writable, then **rpc.ttdbserverd** can be told, via **partition_map**(4), to create the databases in another local partition. If **rpc.ttdbserverd** is not installed on a particular file server, ToolTalk can be told, via **hostname_map**(4), to manage that file server's partitions using the **rpc.ttdbserverd** on a different host.

OPTIONS

–G Perform garbage collection. This cleans up the TT_DB directories and the associated internal database files.

–m DTMOUNTPOINT_value
Sets the DTMOUNTPOINT environment variable for **rpc.ttdbserverd**. If there is already an environment variable called DTMOUNTPOINT, -m will override it.

–S Runs rpc.ttdbserverd in the foreground.

–n Turn off permission checking. Normally the protection of the file passed to **tt_spec_create**(3) determines who may read and write that spec. This option disables this checking and allows anyone to read and write any spec. This option should be used with caution.

–v Print out the version number.

–? Prints out the command usage information.

ENVIRONMENT

DTMOUNTPOINT
If set, the value of this environment variable will be used in place of "/net" in pathnames constructed to answer **tt_host_netfile_file**(3) queries. This environment variable can also be set by using the -m flag for **rpc.ttdbserverd**.

TT_PARTITION_MAP
If **$TT_PARTITION_MAP** is set, it is used in place of /etc/tt/partition_map. See **partition_map**(4).

FILES

TT_DB/*	spec and session database files are kept in the TT_DB directory under each disk partition mount point.
tt/hostname_map	Host redirection map. See **hostname_map**(4).
/etc/tt/partition_map	Partition redirection map. See **partition_map**(4).

SEE ALSO

ttsession(1) **tt_file_netfile**(3) **tt_host_file_netfile**(3) **tt_spec_create**(3) **hostname_map**(4) **partition_map**(4)

NAME

tt_bcontext_join – add a byte-array value to the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_bcontext_join(const char *slotname,
                           const unsigned char *value,
                           int length);
```

DESCRIPTION

The **tt_bcontext_join**( ) function adds the given byte-array value to the list of values for the named contexts of all patterns. The context is compared to currently registered patterns for the procid. If a pattern has a slot with the specified name, the given byte-array value is added to the list of values for that slot.

The *slotname* argument is the name of the context. The *value* argument is the value to be added. The *length* argument is the length in bytes of the value.

RETURN VALUE

Upon successful completion, the **tt_bcontext_join**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_bcontext_quit – remove a byte-array value from the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_bcontext_quit(const char *slotname,
                           const unsigned char *value,
                           int length);
```

DESCRIPTION

The **tt_bcontext_quit**( ) function removes the given byte-array value from the list of values for the contexts of all patterns. The context is compared to currently registered patterns for the procid. If a pattern has a slot with the specified name, the given byte string value is removed from the list of values for that slot. If there are duplicate values, only one value is removed.

The *slotname* argument is the name of the context. The *value* argument is the value to be removed. The *length* argument is the length in bytes of the value.

RETURN VALUE

Upon successful completion, the **tt_bcontext_quit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_close – close the current default procid

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_close(void);
```

DESCRIPTION

The **tt_close**( ) function closes the current default procid.

RETURN VALUE

Upon successful completion, the **tt_close**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

APPLICATION USAGE

When the **tt_close**( ) function call is successful, the procid will no longer be active. For any subsequent API calls the process must, therefore, first call **tt_default_procid_set**( ) to specify a procid.

SEE ALSO

tt_c(5), **tt_open**(3), **tt_context_join**(3).

NAME

tt_context_join – add a string value to the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_context_join(const char *slotname,
                          const char *value);
```

DESCRIPTION

The **tt_context_join**( ) function adds the given string value to the list of values for the context of all patterns.

The context is compared to currently registered patterns for the procid. If a pattern has a slot with the specified name, the given string value is added to the list of values for that slot.

The *slotname* argument is the name of the context. The *value* argument is the value to be added.

RETURN VALUE

Upon successful completion, the **tt_context_join**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_context_quit – remove a string value from the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_context_quit(const char *slotname,
                          const char *value);
```

DESCRIPTION

The **tt_context_quit**( ) function removes the given string value from the list of values for the contexts of all patterns.

The context is compared to currently registered patterns for the procid. If a pattern has a slot with the specified name, **tt_context_quit**( ) removes the given string value from the list of values for that slot. If there are duplicate values, only one value is removed.

The *slotname* argument is the name of the context. The *value* argument is the value to be added.

RETURN VALUE

Upon successful completion, the **tt_context_quit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_default_file – return the current default file

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_default_file(void);
```

DESCRIPTION

The **tt_default_file**( ) function returns the current default file.

When the application joins a file, the file becomes the default.

RETURN VALUE

Upon successful completion, the **tt_default_file**( ) function returns the pointer to a character string that specifies the current default file. If the pointer is **NULL**, no default file is set. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_file_join**(3), **tt_default_file_set**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_default_file_set – set the default file to a file

SYNOPSIS

#include <Tt/tt_c.h>

Tt_status tt_default_file_set(const char **docid***);**

DESCRIPTION

The **tt_default_file_set**( ) function sets the default file to the specified file.

The *docid* argument is a pointer to a character string that specifies the file that is to be the default file.

RETURN VALUE

Upon successful completion, the **tt_default_file_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

TT_ERR_FILE
The specified file does not exist or it is inaccessible.

SEE ALSO

tt_c(5).

NAME

tt_default_procid – identify the current default process

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_default_procid(void);
```

DESCRIPTION

The **tt_default_procid**( ) function retrieves the current default procid for the process.

RETURN VALUE

Upon successful completion, the **tt_default_procid**( ) function returns the pointer to a character string that uniquely identifies the current default process. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_default_procid_set – set the current default procid

SYNOPSIS

#include <Tt/tt_c.h>

Tt_status tt_default_procid_set(const char ∗*procid***);**

DESCRIPTION

The **tt_default_procid_set**( ) function sets the current default procid.

The *procid* argument is the name of process that is to be the default process.

RETURN VALUE

Upon successful completion, the **tt_default_procid_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

SEE ALSO

tt_c(5), **tt_open**(3).

NAME

tt_default_ptype – retrieve the current default ptype

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_default_ptype(void);
```

DESCRIPTION

The **tt_default_ptype**( ) function retrieves the current default ptype.

When the application declares a ptype, the ptype becomes the default.

RETURN VALUE

Upon successful completion, the **tt_default_ptype**( ) function returns a pointer to a character string that uniquely identifies the current default process type. If the pointer is **NULL**, no default ptype is set. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptype_declare**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_default_ptype_set – set the default ptype

SYNOPSIS

#include <Tt/tt_c.h>

Tt_status tt_default_ptype_set(const char **ptid***);**

DESCRIPTION

The **tt_default_ptype_set**( ) function sets the default ptype.

The *ptid* argument must be the character string that uniquely identifies the process that is to be the default process.

RETURN VALUE

Upon successful completion, the **tt_default_ptype_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

SEE ALSO

tt_c(5).

NAME

tt_default_session – retrieve the current default session identifier

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_default_session(void);
```

DESCRIPTION

The **tt_default_session**( ) function retrieves the current default session identifier.

RETURN VALUE

Upon successful completion, the **tt_default_session**( ) function returns the pointer to the unique identifier for the current default session. If the pointer is **NULL**, no default session is set. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

The operation completed successfully.
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.
The current default process identifier is out of date or invalid.

APPLICATION USAGE

A session can have more than one session identifier. This means that the application cannot compare the result of **tt_default_session**( ) with the result of **tt_message_session**(3) to verify that the message was sent in the default session.

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_default_session_set – set the current default session identifier

SYNOPSIS

#include <Tt/tt_c.h>

Tt_status tt_default_session_set(const char **sessid***);**

DESCRIPTION

The **tt_default_session_set**( ) function sets the current default session identifier.

The ToolTalk service uses the initial user session as the default session and supports one session per procid. The application can make this call before it calls **tt_open**(3) to specify the session to which it wants to connect.

The *sessid* argument is a pointer to the unique identifier for the session in which the procid is interested.

RETURN VALUE

Upon successful completion, the **tt_default_session_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

APPLICATION USAGE

To change to another opened session, the application must use the **tt_default_procid_set**( ) function.

To join other sessions, the procid must first set the new session as the default session, and then initialize and register with the ToolTalk service. The calls required must be in the following order:

tt_default_session_set( )
tt_open(3)

The **tt_open**(3) may create another ToolTalk procid, the connection to which is identified by a procid. Only one ToolTalk session per procid is allowed. (However, multiple procids are allowed in a client.) There are no API calls to determine to which session a particular procid is connected. If it is important for the application to know the session to which it is connected, it must make the following calls in the indicated order:

tt_open(3)
tt_default_session(3)

The application can then store the information by indexing it by the procid returned by the **tt_open**(3) call.

SEE ALSO **tt_c**(5), **tt_open**(3), **tt_default_procid**(3), **tt_default_session**(3).

NAME

tt_error – Interpose a function to detect errors returned from the ToolTalk API.

SYNOPSIS

#include <**Tt/tt_c.h**>
void tt_error (*const char *funcname, Tt_status status*);

DESCRIPTION

The **tt_error()** function is a publicly-known null function. This functions is called by the ToolTalk library just before it returns from any ToolTalk API call that has a status other than *TT_OK*. The name of the function that is about to return and the status code is passed.

You can use this call to set a *dbx* breakpoint in **tt_error** to quickly catch and trace back any ToolTalk errors. You can also interpose this function, for example, to log ToolTalk errors to **stderr**.

APPLICATION USAGE

The following code example shows how an application might interpose this function to log ToolTalk errors to **stderr**

```
void tt_error(const char *funcname, Tt_status status) {
       fprintf(stderr, "ToolTalk function %s returned %s.\n",
              funcname, tt_status_message(status));
}
```

SEE ALSO

tt_c(5)

NAME

tt_error_int – return an integer error object that encodes the code

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_error_int(Tt_status ttrc);
```

DESCRIPTION

The **tt_error_int**( ) function returns an integer error object that encodes a **Tt_status** return value, which is within the range of TT_OK == 0 <= ttrc < TT_STATUS_LAST.

The *ttrc* argument is the **Tt_status** code that is to be encoded.

RETURN VALUE

Upon successful completion, the **tt_error_int**( ) function returns the encoded **Tt_status** code.

APPLICATION USAGE

The integer error objects are negative integers; an application should use this call only when the valid integer values are non-negative.

SEE ALSO

tt_c(5).

NAME

tt_error_pointer – return a pointer to an error object that encodes the code

SYNOPSIS

```
#include <Tt/tt_c.h>

void *tt_error_pointer(Tt_status ttrc);
```

DESCRIPTION

The **tt_error_pointer**( ) function returns a pointer to an error object that encodes a **Tt_status** return value, which is within the range of TT_OK == 0 <= ttrc < TT_STATUS_LAST.

The *ttrc* argument is the **Tt_status** code that is to be encoded.

RETURN VALUE

Upon successful completion, the **tt_error_pointer**( ) function returns a pointer to the encoded **Tt_status** code.

SEE ALSO

tt_c(5).

NAME

tt_fd – return a file descriptor

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_fd(void);
```

DESCRIPTION

The **tt_fd**( ) function returns a file descriptor. The returned file descriptor alerts the process that a message has arrived for the default procid in the default session.

File descriptors are either active or inactive. When the file descriptor becomes active, the process must call **tt_message_receive**(3) to receive the message.

RETURN VALUE

Upon successful completion, the **tt_fd**( ) function returns the file descriptor for the current procid. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

APPLICATION USAGE

The application must have a separate file descriptor for each procid. To get an associated file descriptor, the application should use **tt_fd**( ) each time it calls **tt_open**(3).

SEE ALSO

tt_c(5), **tt_open**(3), **tt_int_error**(3), **tt_message_receive**(3).

NAME

tt_feature_enabled - Query the ToolTalk library to see if a particular feature has been enabled previously

SYNOPSIS

#include <**Tt/tt_c.h**>
Tt_status tt_feature_enabled (*Tt_feature feature*);

DESCRIPTION

The **tt_feature_enabled()** call queries the ToolTalk service to see if the indicated feature has previously been enabled.

The *feature* argument is a value indicating a particular feature in which the calling code is interested.

DESCRIPTION

Upon successful completion, the **tt_feature_enabled()** function returns the status of the operation as one of the following *Tt_status* values:

TT_OK The operation completed successfully.

TT_WRN_NOT_ENABLED
The feature has not yet been enabled.

TT_ERR_UNIMP
The version of the ToolTalk library linked with the calling code does not support the indicated feature.

APPLICATION USAGE

A library using ToolTalk could check to see if the calling application had previously turned on ToolTalk's multithreading feature with the following code:

```
Tt_status ttstat;
ttstat = tt_feature_enabled(TT_FEATURE_MULTITHREADED);
if (ttstat != TT_OK) {
       ttstat = tt_feature_required(TT_FEATURE_MULTITHREADED);
}
```

SEE ALSO

tt_c(5), **tt_feature_required**(3)

NAME

tt_feature_required - Declare a feature to be required by the calling code

SYNOPSIS

#include <**Tt/tt_c.h**>
Tt_status tt_feature_required (*Tt_feature feature*);

DESCRIPTION

The **tt_feature_required()** call declares a feature to be required by the calling code. If the feature requires the ToolTalk service to perform some initialization (e.g. **TT_FEATURE_MULTITHREADED**), the initialization is performed in this call.

The *feature* argument is a **Tt_feature** enum value indicating a particular feature to be used by the calling code.

DESCRIPTION

Upon successful completion, the **tt_feature_required()** function returns the status of the operation as one of the following *Tt_status* values:

TT_OK The operation completed successfully.

TT_WRN_NOT_ENABLED
The feature has not yet been enabled.

TT_ERR_UNIMP
The version of the ToolTalk library linked with the calling code does not support the indicated feature.

TT_ERR_TOOLATE
The indicated feature must be declared to be required before calls to the ToolTalk API already made.

APPLICATION USAGE

To use the ToolTalk library in a multithreaded environment, an application would declare multithreading to be required before a call to **tt_open** or **ttdt_open**:

```
Tt_status ttstat;
ttstat = tt_feature_required(TT_FEATURE_MULTITHREADED);
tt_open();
```

SEE ALSO

tt_c(5), **tt_open**(3), **ttdt_open**(3), **tt_feature_enabled**(3)

NAME

tt_file_copy – copy objects from one file to a new file

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_file_copy(const char *oldfilepath,
                       const char *newfilepath);
```

DESCRIPTION

The **tt_file_copy**( ) function copies all objects that exist on the specified file to a new file. If any objects already exist on *newfilepath*, they are not overwritten by the copy (that is, they are not removed.)

The *oldfilepath* argument is a pointer to the name of the file whose objects are to be copied. The *newfilepath* argument is a pointer to the name of the file on which to create the copied objects.

RETURN VALUE

Upon successful completion, the **tt_file_copy**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place. It does not appear to exist; administrative action is required.

TT_ERR_FILE
The specified file does not exist or it is inaccessible.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PATH
The specified pathname included an unsearchable directory.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_file_move**(3), **tt_file_destroy**(3).

NAME

tt_file_destroy – remove objected rooted on a file

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_file_destroy(const char *filepath);
```

DESCRIPTION

The **tt_file_destroy**( ) function removes all objects that exist on the files and directories rooted at *filepath*. The application must call this function when the application unlinks a file or removes a directory.

The *filepath* argument is a pointer to the pathname of the file or directory to be removed.

RETURN VALUE

Upon successful completion, the **tt_file_destroy**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_ACCESS
The user does not have the necessary access to the object and / or the process.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place. It does not appear to exist; administrative action is required.

TT_ERR_FILE
The specified file does not exist or it is inaccessible.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PATH
The specified pathname included an unsearchable directory.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_file_copy**(3), **tt_file_move**(3), **rmdir**(2), **unlink**(2);

NAME

tt_file_join – register interest in messages involving a file

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_file_join(const char *filepath);
```

DESCRIPTION

The **tt_file_join**( ) function informs the ToolTalk service that the process is interested in messages that involve the specified file.

The ToolTalk service adds this file value to any currently registered patterns. The named file becomes the default file.

When the process joins a file, the ToolTalk service updates the file field of its registered patterns. The **tt_file_join**( ) call causes the pattern's ToolTalk session to be stored in the database.

The *filepath* argument is a pointer to the pathname of the file in which the process is interested.

RETURN VALUE

Upon successful completion, the **tt_file_join**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PATH
The specified pathname included an unsearchable directory.

SEE ALSO

tt_c(5).

NAME

tt_file_move – move objects from one file to another

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_file_move(const char *oldfilepath,
                       const char *newfilepath);
```

DESCRIPTION

The **tt_file_move**( ) function destroys all objects that exist on the files and directories rooted at *newfilepath*, then moves all objects that exist on *oldfilepath* to *newfilepath*.

If *oldfilepath* and *newfilepath* reside in the same file system, **tt_file_move**( ) replaces *oldfilepath* with *newfilepath* in the path associated with every object in that file system; that is, all the objects in the directory tree rooted at *oldfilepath* are overlaid onto *newfilepath*. In this mode, the behavior of **tt_file_move**( ) is similar to **rename**(2).

If *oldfilepath* and *newfilepath* reside in different file systems, neither can be a directory.

The *oldfilepath* argument is the name of the file or directory whose objects are to be moved. The *newfilepath* argument is the name of the file or directory to which the objects are to be moved.

RETURN VALUE

Upon successful completion, the **tt_file_move**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_ACCESS
The user does not have the necessary access to the object and/or the process.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_FILE
The specified file does not exist or it is inaccessible.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PATH
The specified pathname included an unsearchable directory, or *oldfilepath* and *newfilepath* reside in different file systems, and either is a directory.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO **tt_c**(5), **tt_file_copy**(3), **tt_file_destroy**(3), **rename**(2).

NAME

tt_file_netfile – map between local and canonical pathnames on the local host

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_file_netfile(const char *filename);
```

DESCRIPTION

The **tt_file_netfile**( ) function converts a local pathname to a *netfilename*, a form that can be passed to other hosts on the network and converted back to a local pathname for the same file with **tt_netfile_file**(3).

The *filename* argument is a pathname (absolute or relative) that is valid on the local host. Every component of *filename* must exist, except that the last component need not exist.

RETURN VALUE

Upon successful completion, the **tt_file_netfile**( ) function returns a freshly allocated null-terminated string of unspecified format, which can be passed to **tt_netfile_file**(3) or **tt_host_netfile_file**(3); otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_ERR_PATH
: The *filename* argument is a path that is not valid on this host.

APPLICATION USAGE

The **tt_file_netfile**(3), **tt_netfile_file**(3), **tt_host_file_netfile**(3) and **tt_host_netfile_file**(3) functions allow an application to determine a path valid on remote hosts, perhaps for purposes of constructing a command string valid for remote execution on that host. By composing the two calls, paths for files not accessible from the current host can be constructed. For example, if path **/sample/file** is valid on host A, a program running on host B can use

```
tt_host_netfile_file("C", tt_host_file_netfile("A", "/sample/file"))
```

to determine a path to the same file valid on host C, if such a path is possible.

The *netfile* string returned by **tt_file_netfile**( ) should be considered opaque; the content and format of the strings are not a public interface. These strings can be safely copied (with **strcpy**(3C) or similar methods), written to files, or transmitted to other processes, perhaps on other hosts.

Allocated strings should be freed using either **tt_free**(3) or **tt_release**(3).

The **tt_open**(3) function need not be called before **tt_file_netfile**( ).

SEE ALSO

tt_c(5), **tt_netfile_file**(3), **tt_host_file_netfile**(3), **tt_host_netfile_file**(3), **tt_open**(3), **tt_free**(3), **tt_release**(3).

NAME

tt_file_objects_query – find all objects in the named file

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_file_objects_query(const char *filepath,
                                Tt_filter_function filter,
                                void *context,
                                void *accumulator);
```

DESCRIPTION

The **tt_file_objects_query**( ) function instructs the ToolTalk service to find all objects in the named file and pass the objids to the filter function. The context pointer and accumulator pointer initially specified are also passed to the filter function.

As the ToolTalk service finds each object, it calls the filter function, passing the objid of the object and the two application-supplied pointers. The filter function performs its computation and returns a **Tt_filter_action** value that tells the query function whether to continue or to stop. **Tt_filter_action** values are:

TT_FILTER_CONTINUE
The query function should continue.

TT_FILTER_STOP
The query function should stop.

The *filepath* argument is the name of the file to be searched for objects. The *filter* argument is the filter function to which the objids are to be passed. The *context* argument is a pointer to any information the filter needs to execute. The ToolTalk service does not interpret this argument, but passes it directly to the filter function. The *accumulator* argument is a pointer to where the filter is to store the results of the query and filter operations. The ToolTalk service does not interpret this argument, but passes it directly to the filter function.

RETURN VALUE

Upon successful completion, the **tt_file_objects_query**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PATH
The specified pathname included an unsearchable directory.

TT_WRN_STOPPED
The query operation being performed was halted by **Tt_filter_function**.

SEE ALSO **tt_c**(5).

NAME

tt_file_quit – register lack of interest in messages that involve a file

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_file_quit(const char *filepath);
```

DESCRIPTION

The **tt_file_quit**( ) function informs the ToolTalk service that the process is no longer interested in messages that involve the specified file.

The ToolTalk service removes this file value from any currently registered patterns and sets the default file to **NULL**.

The *filepath* argument is the name of the file in which the process is no longer interested.

RETURN VALUE

Upon successful completion, the **tt_file_quit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_PATH
The specified pathname included an unsearchable directory.

SEE ALSO

tt_c(5), **tt_default_file**(3).

NAME

tt_free – free storage from the ToolTalk API allocation stack

SYNOPSIS

```
#include <Tt/tt_c.h>

void tt_free(caddr_t p);
```

DESCRIPTION

The **tt_free**( ) function frees storage from the ToolTalk API allocation stack.

The *p* argument is the address of the storage in the ToolTalk API allocation stack to be freed.

RETURN VALUE

The **tt_free**( ) function returns no value.

APPLICATION USAGE

The application should use the **tt_free**( ) function instead of **tt_mark**(3) and **tt_release**(3) if, for example, the process is in a loop (that is, it obtains strings from the ToolTalk service and processes each in turn).

SEE ALSO

tt_c(5), **tt_malloc**(3), **tt_mark**(3), **tt_release**(3), **tt_free**(3).

NAME

tt_host_file_netfile – map between local and canonical pathnames on a remote host

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_host_file_netfile(const char *host,
                           const char *filename);
```

DESCRIPTION

The **tt_host_file_netfile**( ) function performs a conversion equivalent to that of the **tt_file_netfile**(3) function, but performs it on a remote host.

The *filename* argument is a pathname (absolute or relative) that is valid on the remote host. Every component of *filename* must exist, except for the last component. The *host* argument is a name of a remote host.

RETURN VALUE

Upon successful completion, the **tt_host_file_netfile**( ) function returns a freshly allocated null-terminated string of unspecified format, which can be passed to **tt_netfile_file**(3) or **tt_host_netfile_file**(3); otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_ERR_PATH
: The *filename* argument is a path that is not valid on the remote host.

TT_ERR_DBAVAIL
: The ToolTalk database server could not be reached on *host*, perhaps because the host is unavailable or cannot be reached through the network.

TT_ERR_DBEXIST
: The ToolTalk database server is not properly installed on *host*.

TT_ERR_UNIMP
: The ToolTalk database server contacted is of a version that does not support **tt_host_file_netfile**( ).

APPLICATION USAGE

The **tt_file_netfile**(3), **tt_netfile_file**(3), **tt_host_file_netfile**(3) and **tt_host_netfile_file**(3) functions allow an application to determine a path valid on remote hosts, perhaps for purposes of constructing a command string valid for remote execution on that host. By composing the two calls, paths for files not accessible from the current host can be constructed. For example, if path **/sample/file** is valid on host A, a program running on host B can use

```
tt_host_netfile_file("C", tt_host_file_netfile("A", "/sample/file"))
```

to determine a path to the same file valid on host C, if such a path is possible.

Allocated strings should be freed using either **tt_free**(3) or **tt_release**(3).

The **tt_open**(3) function need not be called before **tt_host_file_netfile**( ).

SEE ALSO

tt_c(5), **tt_file_netfile**(3), **tt_netfile_file**(3), **tt_host_netfile_file**(3), **tt_open**(3), **tt_free**(3), **tt_release**(3).

NAME

tt_host_netfile_file – map between canonical and local pathnames on a remote host

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_host_netfile_file(const char *host,
                           const char *netfilename);
```

DESCRIPTION

The **tt_host_netfile_file**( ) function performs a conversion equivalent to that of the **tt_netfile_file**(3) function, but performs it on a remote host.

The *host* argument is the host on which the file resides. The *netfilename* argument is a copy of a null-terminated string returned by **tt_netfile_file**(3) or **tt_host_netfile_file**(3).

RETURN VALUE

Upon successful completion, the **tt_host_netfile_file**( ) function returns a freshly allocated null-terminated string of unspecified format, which can be passed to **tt_host_netfile_file**(3); otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_ERR_DBAVAIL
: The ToolTalk database server could not be reached on *host*, perhaps because the host is unavailable or cannot be reached through the network.

TT_ERR_DBEXIST
: The ToolTalk database server is not properly installed on *host*.

TT_ERR_NETFILE
: The *netfilename* is not a valid netfilename.

TT_ERR_UNIMP
: The ToolTalk database server contacted is of a version that does not support **tt_host_netfile_file**( ).

APPLICATION USAGE

The **tt_file_netfile**(3), **tt_netfile_file**(3), **tt_host_file_netfile**(3) and **tt_host_netfile_file**(3) functions allow an application to determine a path valid on remote hosts, perhaps for purposes of constructing a command string valid for remote execution on that host. By composing the two calls, paths for files not accessible from the current host can be constructed. For example, if path **/sample/file** is valid on host A, a program running on host B can use

```
tt_host_netfile_file("C", tt_host_file_netfile("A", "/sample/file"))
```

to determine a path to the same file valid on host C, if such a path is possible.

Allocated strings should be freed using either **tt_free**(3) or **tt_release**(3).

The **tt_open**(3) function need not be called before **tt_host_netfile_file**( ).

SEE ALSO

tt_c(5), **tt_file_netfile**(3), **tt_netfile_file**(3), **tt_host_file_netfile**(3), **tt_open**(3), **tt_free**(3), **tt_release**(3).

NAME

tt_icontext_join – add an integer value to the list of values

SYNOPSIS

#include <Tt/tt_c.h>

Tt_status tt_icontext_join(const char **slotname***, int** *value***);**

DESCRIPTION

The **tt_icontext_join**( ) function adds the given integer value to the list of values for the contexts of all patterns.

The context is compared to currently registered patterns for the procid. If a pattern has a slot with the specified name, the given integer value is added to the list of values for that slot.

The *slotname* argument is the name of the context. The *value* argument is the value to be added.

RETURN VALUE

Upon successful completion, the **tt_icontext_join**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_icontext_quit – remove an integer value from the list of values

SYNOPSIS

#include <Tt/tt_c.h>

Tt_status tt_icontext_quit(const char **slotname*, **int** *value*);

DESCRIPTION

The **tt_icontext_quit**( ) function removes the given integer value from the list of values for the contexts of all patterns.

The context is compared to currently registered patterns for the procid. If a pattern has a slot with the specified name, the given integer value is removed from the list of values for that slot.

If there are duplicate values, only one value is removed.

The *slotname* argument is the name of the context. The *value* argument is the value to be added.

RETURN VALUE

Upon successful completion, the **tt_icontext_quit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_initial_session – return the initial session identifier

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_initial_session(void);
```

DESCRIPTION

The **tt_initial_session**( ) function returns the initial session identifier of the **ttsession**(1) with which the current process identifier is associated.

The current process identifier is obtained by calling **tt_open**(3).

RETURN VALUE

Upon successful completion, the **tt_initial_session**( ) function returns the identifier for the current ToolTalk session. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_open**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_int_error – return the status of an error object

SYNOPSIS

#include <Tt/tt_c.h>

Tt_status tt_int_error(int *return_val***);**

DESCRIPTION

The **tt_int_error**( ) function returns the status of an error object.

The *return_val* argument is the integer returned by a ToolTalk function.

RETURN VALUE

Upon successful completion, the **tt_int_error**( ) function returns either **TT_OK**, if the integer is not an error object, or the encoded **Tt_status** value if the integer is an error object.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_is_err – check status value

SYNOPSIS

```
#include <Tt/tt_c.h>
int tt_is_err(Tt_status s);
```

DESCRIPTION

The **tt_is_err**( ) function checks whether a status value is a warning or an error. The *s* argument is the **Tt_status** code to check.

RETURN VALUE

Upon successful completion, the **tt_is_err**( ) function returns one of the following integers:

0 The **Tt_status** is either a warning or **TT_OK**.

1 The **Tt_status** is an error.

SEE ALSO

tt_c(5).

NAME

tt_malloc – allocate storage on the ToolTalk API allocation stack

SYNOPSIS

```
#include <Tt/tt_c.h>

caddr_t tt_malloc(size_t s);
```

DESCRIPTION

The **tt_malloc**( ) function allocates storage on the ToolTalk API allocation stack. The *s* argument is the amount of storage to be allocated in bytes.

RETURN VALUE

Upon successful completion, the **tt_malloc**( ) function returns the address of the storage in the ToolTalk API allocation stack that is to be allocated. If **NULL** is returned, no storage is available.

APPLICATION USAGE

This function allows the application-provided callback routines to take advantage of the allocation stack; for example, a query filter function can allocate storage to accumulate a result.

SEE ALSO

tt_c(5), **tt_free**(3).

NAME

tt_mark – mark a storage position in the ToolTalk API allocation stack

SYNOPSIS

```
#include <Tt/tt_c.h>
int tt_mark(void);
```

DESCRIPTION

The **tt_mark**( ) function marks a storage position in the ToolTalk API allocation stack.

RETURN VALUE

Upon successful completion, the **tt_mark**( ) function returns an integer that marks the storage position in the ToolTalk API allocation stack.

SEE ALSO

tt_c(5), **tt_release**(3).

NAME

tt_message_abstainer – return offer's nth abstaining procid

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_abstainer(Tt_message m,
                           int n);
```

DESCRIPTION

The **tt_message_abstainer**( ) function returns the procid of the *n*th abstainer of the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the abstainer to be returned. The first abstainer is numbered zero.

RETURN VALUE

Upon successful completion, the **tt_message_abstainer**( ) function returns the procid of the *n*th abstainer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_PROCID
There is no valid default procid, perhaps because **tt_open**(3) has not yet been called.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_STATE
The specified message is not in state **TT_RETURNED**. Since only **TT_OFFER**s can be in state **TT_RETURNED**, this status will be returned if the specified message is a **TT_NOTICE** or a **TT_REQUEST**.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application can use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_message_abstainers_count**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_abstainers_count – return a count of the offer's abstaining procids

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_message_abstainers_count(Tt_message m);
```

DESCRIPTION

The **tt_message_abstainers_count**( ) function returns a count of the procids that are recorded in the offer *m* as having abstained from it.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_abstainers_count**( ) function returns a count of the procids that are recorded in the offer *m* as having abstained from it. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_PROCID
There is no valid default procid, perhaps because **tt_open**(3) has not yet been called.

TT_ERR_STATE
The specified message is not in state **TT_RETURNED**. Since only **TT_OFFER**s can be in state **TT_RETURNED**, this status will be returned if the specified message is a **TT_NOTICE** or a **TT_REQUEST**.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_message_abstainer**(3), **tt_int_error**(3).

NAME

tt_message_accept – declare that the process has been initialized and can accept messages

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_accept(Tt_message m);
```

DESCRIPTION

The **tt_message_accept**( ) function declares that the process has been initialized and can accept messages.

The ToolTalk service invokes this function for start messages.

The *m* argument is the opaque handle for the message involved in this operation.

Note: For TT_OFFER messages, the API call will count this *procid* as accepting the offer.

RETURN VALUE

Upon successful completion, the **tt_message_accept**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_UNIMP
The ToolTalk function called is not implemented.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_STATE
The Tt_message is in a Tt_state that is invalid for the attempted operation.

SEE ALSO

tt_c(5).

NAME

tt_message_accepter – return offer's nth accepting procid

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_accepter(Tt_message m,
                          int n);
```

DESCRIPTION

The **tt_message_accepter**( ) function returns the procid of the *n*th accepter of the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the accepter to be returned. The first accepter is numbered zero.

RETURN VALUE

Upon successful completion, the **tt_message_accepter**( ) function returns the procid of the *n*th accepter. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_PROCID
There is no valid default procid, perhaps because **tt_open**(3) has not yet been called.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_STATE
The specified message is not in state **TT_RETURNED**. Since only **TT_OFFER**s can be in state **TT_RETURNED**, this status will be returned if the specified message is a **TT_NOTICE** or a **TT_REQUEST**.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application can use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_message_accepters_count**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_accepters_count – return a count of the offer's accepting procids

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_message_accepters_count(Tt_message m);
```

DESCRIPTION

The **tt_message_accepters_count**( ) function returns a count of the procids that are recorded in the offer *m* as having accepted it.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_accepters_count**( ) function returns a count of the procids that are recorded in the offer *m* as having accepted it. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_PROCID
There is no valid default procid, perhaps because **tt_open**(3) has not yet been called.

TT_ERR_STATE
The specified message is not in state **TT_RETURNED**. Since only **TT_OFFER**s can be in state **TT_RETURNED**, this status will be returned if the specified message is a **TT_NOTICE** or a **TT_REQUEST**.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_message_accepter**(3), **tt_int_error**(3).

NAME

tt_message_address – retrieve the address attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_address tt_message_address(Tt_message m);
```

DESCRIPTION

The **tt_message_address**( ) function retrieves the address attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_address**( ) function returns a value that specifies which message attributes form the address of this message. The **tt_message_address**( ) function returns one of the following **Tt_address** values:

TT_HANDLER
The message is addressed to a specific handler that can perform this operation with these arguments.

TT_OBJECT
The message is addressed to a specific object that can perform this operation with these arguments.

TT_OTYPE
The message is addressed to the type of object that can perform this operation with these arguments.

TT_PROCEDURE
The message is addressed to any process that can perform this operation with these arguments.

Note that messages of class TT_OFFER may only be sent with an address of TT_PROCEDURE. Attempting to send an Offer with any other address will result in an error return of TT_ERR_ADDRESS.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_ADDRESS
The specified **Tt_address** is invalid.

SEE ALSO **tt_c**(5), **tt_int_error**(3).

NAME

tt_message_address_set – set the address attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_address_set(Tt_message m, Tt_address a);
```

DESCRIPTION

The **tt_message_address_set**( ) function sets the address attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *a* argument specifies which message attributes form the address to which the message will be delivered. The following values are defined:

TT_HANDLER
: The message is addressed to a specific handler that can perform this operation with these arguments.

TT_OBJECT
: The message is addressed to a specific object that can perform this operation with these arguments.

TT_OTYPE
: The message is addressed to the type of object that can perform this operation with these arguments.

TT_PROCEDURE
: The message is addressed to any process that can perform this operation with these arguments.

RETURN VALUE

Upon successful completion, the **tt_message_address_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
: The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
: The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_arg_add – add a new argument to a message object

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_arg_add(Tt_message m,
                             Tt_mode n,
                             const char *vtype,
                             const char *value);
```

DESCRIPTION

The **tt_message_arg_add**( ) function adds a new argument to a message object.

The application must add all arguments before the message is sent. To change existing argument values, the application must use only modes **TT_OUT** or **TT_INOUT**.

Adding arguments when replying to a message produces undefined results.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument specifies who (sender, handler, observers) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The vtype argument describes the type of argument data being added. The *value* argument is the contents for the message argument attribute. The application can use **NULL** either for values of mode **TT_OUT**, or if the value is to be filled in later with one of the following:

tt_message_arg_val_set(3)
tt_message_barg_val_set(3)
tt_message_iarg_val_set(3)

RETURN VALUE

Upon successful completion, the **tt_message_arg_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_MODE
The specified **Tt_mode** is invalid.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this

operation.

SEE ALSO **tt_c**(5), **tt_message_arg_val_set**(3), **tt_message_barg_add**(3), **tt_message_iarg_add**(3).

NAME

tt_message_arg_bval – retrieve the byte-array value of a message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_arg_bval(Tt_message m,
                              int n,
                              unsigned char **value,
                              int *len);
```

DESCRIPTION

The **tt_message_arg_bval**( ) function retrieves the byte-array value of the *n*th message argument.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to be retrieved. The first argument is numbered zero. The *value* argument is the address of a character pointer to which the ToolTalk service is to point a string that contains the contents of the argument. The *len* argument is the address of an integer to which the ToolTalk service is to set the length of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_message_arg_bval**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_arg_bval_set – set the byte-array value and type of a message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_arg_bval_set(Tt_message m,
                                  int n,
                                  const unsigned char *value,
                                  int len);
```

DESCRIPTION

The **tt_message_arg_bval_set**( ) function sets the byte-array value and the type of the *n*th message argument.

This function also changes the value of an existing *n*th message argument to a byte string.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to set. The first argument is numbered zero. The *value* argument is the byte string with the contents for the message argument. The *len* argument is the length of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_message_arg_bval_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The sending process can use **tt_message_arg_bval_set**(3) to fill in opaque data.

SEE ALSO

tt_c(5), **tt_message_barg_add**(3), **tt_message_arg_val_set**(3), **tt_message_arg_ival_set**(3).

NAME

tt_message_arg_ival – retrieve the integer value of a message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_arg_ival(Tt_message m,
                              int n,
                              int *value);
```

DESCRIPTION

The **tt_message_arg_ival**( ) function retrieves the integer value of the *n*th message argument.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to be retrieved. The first argument is numbered zero. The *value* argument is a pointer to an integer where the ToolTalk service is to store the contents of the argument.

RETURN VALUE

Upon successful completion, the **tt_message_arg_ival**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range), or, if the value in the message is not yet set (as it would be in observing a request in state TT_SENT and trying to look at the TT_OUT arguments).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_arg_ival_set – add an integer value in a message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_arg_ival_set(Tt_message m,
                                  int n,
                                  int value);
```

DESCRIPTION

The **tt_message_arg_ival_set**( ) function adds an integer value in the *n*th message argument.

This function also changes the value of an existing *n*th message argument to an integer.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to be set. The first argument is numbered zero. The *value* argument is the contents for the message argument.

RETURN VALUE

Upon successful completion, the **tt_message_arg_ival_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_message_iarg_add**(3), **tt_message_arg_val_set**(3), **tt_message_arg_bval_set**(3).

NAME

tt_message_arg_mode – return the mode of a message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_mode tt_message_arg_mode(Tt_message m,
                            int n);
```

DESCRIPTION

The **tt_message_arg_mode**( ) function returns the mode of the *n*th message argument.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to be returned. The first argument is numbered zero.

RETURN VALUE

Upon successful completion, the **tt_message_arg_mode**( ) function returns a value that specifies who (sender, handler, observers) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the **Tt_mode** integer return value:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_message_arg_type – retrieve the type of a message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_arg_type(Tt_message m,
                          int n);
```

DESCRIPTION

The **tt_message_arg_type**( ) function retrieves the type of the *n*th message argument.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to be retrieved. The first argument is numbered zero.

RETURN VALUE

Upon successful completion, the **tt_message_arg_type**( ) function returns the type of the *n*th message argument. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application can use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_arg_val – return a pointer to the value of a message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_arg_val(Tt_message m,
                         int n);
```

DESCRIPTION

The **tt_message_arg_val**( ) function returns a pointer to the value of the *n*th message argument.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to be returned. The first argument is numbered zero.

RETURN VALUE

Upon successful completion, the **tt_message_arg_val**( ) function returns the contents for the message argument. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application can use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_arg_val_set – change the value of a message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_arg_val_set(Tt_message m,
                                 int n,
                                 const char *value);
```

DESCRIPTION

The **tt_message_arg_val_set**( ) function changes the value of the *n*th message argument.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to be changed. The first argument is numbered zero. The *value* argument is the contents for the message argument.

RETURN VALUE

Upon successful completion, the **tt_message_arg_val_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_arg_xval – retrieve and deserialize the data from a message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_arg_xval(Tt_message m,
                              int n,
                              xdrproc_t xdr_proc,
                              void **value);
```

DESCRIPTION

The **tt_message_arg_xval**( ) function retrieves and deserializes the data from a message argument. This function uses an XDR routine that is supplied by the client.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to be returned. The first argument is numbered zero. The *xdr_proc* argument points to the XDR procedure to be used to deserialize the data in the *n*th argument into newly allocated storage, the address of which will be stored in the pointer whose address is *value*.

The *value* argument is the data to be deserialized.

RETURN VALUE

Upon successful completion, the **tt_message_arg_xval**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_MODE
The specified **Tt_mode** is invalid.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_XDR
The XDR procedure failed on the given data, or evaluated to a zero-length structure.

APPLICATION USAGE

The allocation calls are made by the XDR procedure; therefore, any storage allocated is not allocated from the ToolTalk allocation stack. The application should use the **xdr_free**(3) call to free this storage.

SEE ALSO

tt_c(5).

NAME

tt_message_arg_xval_set – serialize and set data into an existing message argument

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_arg_xval_set(Tt_message m,
                                  int n,
                                  xdrproc_t xdr_proc,
                                  void *value);
```

DESCRIPTION

The **tt_message_arg_xval_set**( ) function serializes and sets data into an existing message argument.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the argument to be changed. The first argument is numbered zero. The *xdr_proc* argument causes **tt_message_arg_xval_set**( ) to serialize the data pointed to by *value* and store it as a byte string value of the *n*th argument of the message. The *value* argument is the data to be serialized.

RETURN VALUE

Upon successful completion, the **tt_message_arg_xval_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_MODE
The specified **Tt_mode** is invalid.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_XDR
The XDR procedure failed on the given data, or evaluated to a zero-length expression.

SEE ALSO

tt_c(5).

NAME

tt_message_args_count – return the number of arguments in the message

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_message_args_count(Tt_message m);
```

DESCRIPTION

The **tt_message_args_count**( ) function returns the number of arguments in the message. The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_args_count**( ) function returns the total number of arguments in the message. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_message_barg_add – add an argument to a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_barg_add(Tt_message m,
                              Tt_mode n,
                              const char *vtype,
                              const unsigned char *value,
                              int len);
```

DESCRIPTION

The **tt_message_barg_add**( ) function adds an argument to a pattern that may have a byte-array value that contains embedded nulls.

To change existing argument values, the application must use only modes **TT_OUT** or **TT_INOUT**.

Adding arguments when replying to a message produces undefined results.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument specifies who (sender, handler, observers) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The vtype argument describes the type of argument data being added.

The ToolTalk service treats the value as an opaque byte string. To pass structured data, the application and the receiving application must encode and decode these opaque byte strings. The most common method to do this is XDR.

The *value* argument is the value to be added. The *len* argument is the length of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_message_barg_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO **tt_c**(5), **tt_message_arg_bval_set**(3), **tt_message_arg_add**(3), **tt_message_iarg_add**(3).

NAME

tt_message_bcontext_set – set the byte-array value of a message's context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_bcontext_set(Tt_message m,
                                  const char *slotname,
                                  const unsigned char *value,
                                  int length);
```

DESCRIPTION

The **tt_message_bcontext_set**( ) function sets the byte-array value of a message's context.

This function overwrites any previous value associated with *slotname*.

The *m* argument is the opaque handle for the message involved in this operation. The *slotname* argument describes the slotname in this message. The *value* argument is the byte string with the contents for the message argument. The *length* argument is the length of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_message_bcontext_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_message_callback_add – register a callback function

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_callback_add(Tt_message m,
                                  Tt_message_callback f);
```

DESCRIPTION

The **tt_message_callback_add**( ) function registers a callback function to be automatically invoked by **tt_message_receive**(3) whenever a reply or other state-change to this message is returned.

The callback is defined in **<Tt/tt_c.h>**. If the callback returns **TT_CALLBACK_CONTINUE**, other callbacks will be run; if no callback returns **TT_CALLBACK_PROCESSED**, **tt_message_receive**( ) returns the message. If the callback returns **TT_CALLBACK_PROCESSED**, no further callbacks are invoked for this event; **tt_message_receive**( ) does not return the message.

The *m* argument is the opaque handle for the message involved in this operation. The *f* argument is the message callback to be run.

The pattern handle will be **NULL** if the message did not match a dynamic pattern. This is usually the case for message callbacks.

RETURN VALUE

Upon successful completion, the **tt_message_callback_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

These callbacks are invoked from **tt_message_receive**( ); the program must, therefore, call **tt_message_receive**( ) when the file descriptor returned by **tt_fd**( ) becomes active.

The application can use **tt_message_callback_add**( ) to create wrappers for ToolTalk messages. For example, a library routine can construct a request, attach a callback to the message, send the message, and process the reply in the callback. When the callback returns **TT_CALLBACK_PROCESSED**, the message reply is not returned to the main program; the message and reply are, therefore, completely hidden.

SEE ALSO

tt_c(5), **tt_message_receive**(3).

NAME

tt_message_class – retrieve the class attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_class tt_message_class(Tt_message m);
```

DESCRIPTION

The **tt_message_class**( ) function retrieves the class attribute from the specified message. The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_class**( ) function returns a value that indicates whether the sender wants an action to take place after the message is received. The **tt_message_class**( ) function returns one of the following **Tt_status** values:

TT_NOTICE
A notice of an event. The sender does not want feedback on this message.

TT_REQUEST
A request for some action to be taken. The sender must be notified of progress, success or failure, and must receive any return values.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the **Tt_class** integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_message_class_set – set the class attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_class_set(Tt_message m,
                               Tt_class c);
```

DESCRIPTION

The **tt_message_class_set**( ) function sets the class attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *c* argument indicates whether an action is to take place after the message is received. The following values are defined:

TT_NOTICE
A notice of an event. The sender does not want feedback on this message.

TT_REQUEST
A request for some action to be taken. The sender must be notified of progress, success or failure, and must receive any return values.

RETURN VALUE

Upon successful completion, the **tt_message_class_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_context_bval – retrieve the byte-array value and length of a message's context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_context_bval(Tt_message m,
                                  const char *slotname,
                                  unsigned char **value,
                                  int *len);
```

DESCRIPTION

The **tt_message_context_bval**( ) function retrieves the byte-array value and length of a message's context.

If there is no context slot associated with *slotname*, **tt_message_context_bval**( ) returns a **NULL** pointer in *slotname* and zero in *len*.

The *m* argument is the opaque handle for the message involved in this operation. The *slotname* argument describes the context of this message. The *value* argument points to the location to return the value. The *len* argument is the length of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_message_context_bval**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_message_context_ival – retrieve the integer value of a message's context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_context_ival(Tt_message m,
                                  const char *slotname,
                                  int *value);
```

DESCRIPTION

The **tt_message_context_ival**( ) function retrieves the integer value of a message's context.

The *m* argument is the opaque handle for the message involved in this operation. The *slotname* argument describes the context of this message. The *value* argument points to the location to return the value.

If there is no context slot associated with *slotname*, **tt_message_context_ival**( ) returns a **NULL** pointer in *slotname*.

RETURN VALUE

Upon successful completion, the **tt_message_context_ival**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

TT_WRN_NOTFOUND
The named context does not exist on the specified message.

APPLICATION USAGE

The application can use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_free**(3).

NAME

tt_message_context_set – set the character string value of a message's context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_context_set(Tt_message m,
                                 const char *slotname,
                                 const char *value);
```

DESCRIPTION

The **tt_message_context_set**( ) function sets the character string value of a message's context.

The *m* argument is the opaque handle for the message involved in this operation. The *slotname* argument describes the context of this message. This function overwrites any previous value associated with *slotname*. The *value* argument is the character string to be set.

RETURN VALUE

Upon successful completion, the **tt_message_context_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_message_context_slotname – return the name of a message's nth context

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_context_slotname(Tt_message m,
                                  int n);
```

DESCRIPTION

The **tt_message_context_slotname**( ) function returns the name of a message's *n*th context.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the context to be retrieved. The first context is numbered zero.

RETURN VALUE

Upon successful completion, the **tt_message_context_slotname**( ) function returns the contents for the message argument. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application can use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_context_val – retrieve the character string of a message's context

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_context_val(Tt_message m,
                             const char *slotname);
```

DESCRIPTION

The **tt_message_context_val**( ) function retrieves the character string of a message's context.

The *m* argument is the opaque handle for the message involved in this operation. The *slotname* argument describes the context of this message.

If there is no context slot associated with *slotname*, **tt_message_context_val**( ) returns a **NULL** pointer in *slotname*.

RETURN VALUE

Upon successful completion, the **tt_message_context_val**( ) function returns the contents for the message argument. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_context_xval – retrieve and deserialize the data from a message's context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_context_xval(Tt_message m,
                                  const char *slotname,
                                  xdrproc_t xdr_proc,
                                  void **value);
```

DESCRIPTION

The **tt_message_context_xval**( ) function retrieves and deserializes the data from a message's context.

The *m* argument is the opaque handle for the message involved in this operation. The *slotname* argument describes the context of this message. The *xdr_proc* argument points to the XDR procedure to be used to deserialize the data in the *n*th argument into newly allocated storage, the address of which will be stored in the pointer whose address is *value*.

The *value* argument is the data to be deserialized.

RETURN VALUE

Upon successful completion, the **tt_message_context_xval**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_MODE
The specified **Tt_mode** is invalid.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_XDR
The XDR procedure failed on the given data, or evaluated to a zero-length expression.

APPLICATION USAGE

The allocation calls are made by the XDR procedure; therefore, any storage allocated is not allocated from the ToolTalk allocation stack. The application should use the **xdr_free**(3) call to free this storage.

SEE ALSO

tt_c(5).

NAME

tt_message_contexts_count – return the number of contexts in a message

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_message_contexts_count(Tt_message m);
```

DESCRIPTION

The **tt_message_contexts_count**( ) function returns the number of contexts in a message. The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_contexts_count**( ) function returns the total number of contexts in the message. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_message_create – create a new message object

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_message tt_message_create(void);
```

DESCRIPTION

The **tt_message_create**( ) function creates a new message object.

The ToolTalk service returns a message handle that is an opaque pointer to a ToolTalk structure.

RETURN VALUE

Upon successful completion, the **tt_message_create**( ) function returns the unique opaque handle that identifies the message object. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The process identification is not valid.

APPLICATION USAGE

A return value of **TT_ERR_PROCID** implies that **tt_open**(3) was not issued before **tt_message_create**( ).

If the ToolTalk service is unable to create a message when requested, **tt_message_create**( ) returns an invalid handle. When the application attempts to use this handle with another ToolTalk function, the ToolTalk service will return **TT_ERR_POINTER**.

SEE ALSO

tt_c(5), **tt_open**(3), **tt_message_send**(3), **tt_message_destroy**(3).

NAME

tt_message_create_super – create and re-address a copy of a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_message tt_message_create_super(Tt_message m);
```

DESCRIPTION

The **tt_message_create_super**( ) function creates a copy of the specified message and re-addresses the copy of the message to the parent of the otype contained within the message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_create_super**( ) function returns the opaque unique handle for the re-addressed message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_OK The operation completed successfully.

TT_ERR_ADDRESS
The specified **Tt_address** is invalid.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The objid passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The otype of the message *m* can be determined using the **tt_message_otype**(3) function.

SEE ALSO

tt_c(5), **tt_message_otype**(3), **tt_message_send**(3), **tt_message_destroy**(3).

NAME

tt_message_destroy – destroy a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_destroy(Tt_message m);
```

DESCRIPTION

The **tt_message_destroy**( ) function destroys the message.

Destroying a message has no effect on the delivery of a message already sent.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_destroy**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

If the application sent a request and is expecting a reply with return values, the application should destroy the message after it have received the reply. If the application sends a notice, the application can destroy the message immediately after it sends the notice.

SEE ALSO

tt_c(5), **tt_message_create**(3), **tt_message_create_super**(3).

NAME

tt_message_disposition – retrieve the disposition attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_disposition tt_message_disposition(Tt_message m);
```

DESCRIPTION

The **tt_message_disposition**( ) function retrieves the disposition attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_disposition**( ) function returns a value that indicates whether an instance of the receiving process should be started to receive the message immediately, or whether the message is to be queued until the receiving process is started at a later time. The **tt_message_disposition**( ) function returns one of the following **Tt_disposition** values:

TT_DISCARD
: There is no receiver for this message. The message will be returned to the sender with the **Tt_status** field containing **TT_FAILED**.

TT_QUEUE
: Queue the message until a process of the proper ptype receives the message.

TT_START
: Attempt to start a process of the proper ptype if none is running.

TT_QUEUE+TT_START
: Queue the message and attempt to start a process of the proper ptype if none is running.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the **Tt_disposition** integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
: The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
: The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_message_disposition_set – set the disposition attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_disposition_set(Tt_message m,
                                     Tt_disposition r);
```

DESCRIPTION

The **tt_message_disposition_set**( ) function sets the disposition attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *r* argument indicates whether an instance of the receiving process is to be started to receive the message immediately, or whether the message is to be queued until the receiving process is started at a later time. The following values are defined:

TT_DISCARD
: There is no receiver for this message. The message will be returned to the sender with the **Tt_status** field containing **TT_FAILED**.

TT_QUEUE
: Queue the message until a process of the proper ptype receives the message.

TT_START
: Attempt to start a process of the proper ptype if none is running.

TT_QUEUE+TT_START
: Queue the message and attempt to start a process of the proper ptype if none is running.

RETURN VALUE

Upon successful completion, the **tt_message_disposition_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
: The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
: The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_fail – indicate a message cannot be handled

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_fail(Tt_message m);
```

DESCRIPTION

The **tt_message_fail**( ) function informs the ToolTalk service that the process cannot handle the request just received.

This function also informs the ToolTalk service that the message is not be offered to other processes of the same ptype. The ToolTalk service will send the message back to the sender with state **TT_FAILED**.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_fail**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NOTHANDLER
This application is not the handler for this message.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

The status value must be greater than **TT_ERR_LAST** to avoid confusion with the ToolTalk service status values.

APPLICATION USAGE

To distinguish this case from the case where a message failed because no matching handler could be found, the application should place an explanatory message code in the status attribute of the message with **tt_message_status_set**(3) and **tt_message_status_string_set**(3) before calling **tt_message_fail**( ).

SEE ALSO

tt_c(5), **tt_message_status_set**(3), **tt_message_status_string_set**(3).

NAME

tt_message_file – retrieves the file attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_file(Tt_message m);
```

DESCRIPTION

The **tt_message_file**( ) function retrieves the file attribute from the specified message. The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_file**( ) function returns a string containing the file attribute of the specified message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_file_set – set the file attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_file_set(Tt_message m,
                              const char *file);
```

DESCRIPTION

The **tt_message_file_set**( ) function sets the file attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *file* argument is the name of the file involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_file_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_FILE
The specified file does not exist or it is inaccessible.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_gid – retrieve the group identifier attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

gid_t tt_message_gid(Tt_message m);
```

DESCRIPTION

The **tt_message_gid**( ) function retrieves the group identifier attribute from the specified message.

The ToolTalk service automatically sets the group identifier of a message with the group identifier of the process that created the message.

The *m* argument is the opaque handle for the message involved in this operation.

The application should check the tt_message_uid(3) and **tt_message_gid( )** against the User ID and Group ID of the application receiving the messsage. If the UID and / or GID of the application do not match that of the message then the receiving application should consider failing the message with TT_DESKTOP_EACCES.

RETURN VALUE

Upon successful completion, the **tt_message_gid**( ) function returns the group identifier of the message. If the group **nobody** is returned, the message handle is not valid.

SEE ALSO

tt_c(5), **tt_message_uid**(3).

NAME

tt_message_handler – retrieve the handler attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_handler(Tt_message m);
```

DESCRIPTION

The **tt_message_handler**( ) function retrieves the handler attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_handler**( ) function returns the character value that uniquely identifies the process that is to handle the message (**Tt_state**= **TT_CREATED** or **TT_SENT**) or the process that did handle the message (**Tt_state** = **TT_SENT** or **TT_HANDLED**). The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_handler_ptype – retrieve the handler ptype attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_handler_ptype(Tt_message m);
```

DESCRIPTION

The **tt_message_handler_ptype**( ) function retrieves the handler ptype attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_handler_ptype**( ) function returns the type of process that should handle this message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_handler_ptype_set – set the handler ptype attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_handler_ptype_set(Tt_message m,
                                       const char *ptid);
```

DESCRIPTION

The **tt_message_handler_ptype_set**( ) function sets the handler ptype attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *ptid* argument is the type of process that is to handle this message.

RETURN VALUE

Upon successful completion, the **tt_message_handler_ptype_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_handler_set – set the handler attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_handler_set(Tt_message m,
                                 const char *procid);
```

DESCRIPTION

The **tt_message_handler_set**( ) function sets the handler attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The procid argument is the character value that uniquely identifies the process that is to handle the message.

RETURN VALUE

Upon successful completion, the **tt_message_handler_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_iarg_add – add a new argument to a message object

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_iarg_add(Tt_message m,
                              Tt_mode n,
                              const char *vtype,
                              int value);
```

DESCRIPTION

The **tt_message_iarg_add**( ) function adds a new argument to a message object and sets the value to a given integer.

Add all arguments before the message is sent. To change existing argument values, the application must use only modes **TT_OUT** or **TT_INOUT**.

Adding arguments when replying to a message produces undefined results.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument specifies who (sender, handler, observers) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The vtype argument describes the type of argument data being added. The *value* argument is the value to be added.

RETURN VALUE

Upon successful completion, the **tt_message_iarg_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_MODE
The specified **Tt_mode** is invalid.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_message_arg_ival_set**(3), **tt_message_arg_add**(3), **tt_message_barg_add**(3).

NAME

tt_message_icontext_set – set the integer value of a message's context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_icontext_set(Tt_message m,
                                  const char *slotname,
                                  int value);
```

DESCRIPTION

The **tt_message_icontext_set**( ) function sets the integer value of a message's context.

This function overwrites any previous value associated with *slotname*.

The *m* argument is the opaque handle for the message involved in this operation. The *slotname* argument describes the context of this message. The *value* argument is the integer value to be set.

RETURN VALUE

Upon successful completion, the **tt_message_icontext_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_message_id – retrieve the identifier of a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_id(Tt_message m);
```

DESCRIPTION

The **tt_message_id**( ) function retrieves the identifier of the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_id**( ) function returns the character string value that uniquely identifies the message across all running ToolTalk sessions. The identifier of the message is set at its creation and never changes. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_object – retrieve the object attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_object(Tt_message m);
```

DESCRIPTION

The **tt_message_object**( ) function retrieves the object attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_object**( ) function returns the object involved in this message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The objid passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_object_set – set the object attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_object_set(Tt_message m,
                                const char *objid);
```

DESCRIPTION

The **tt_message_object_set**( ) function sets the object attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *objid* argument is the identifier of the specified object.

RETURN VALUE

Upon successful completion, the **tt_message_object_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_op – retrieve the operation attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_op(Tt_message m);
```

DESCRIPTION

The **tt_message_op**( ) function retrieves the operation attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_op**( ) function returns the operation which the receiving process is to perform. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_op_set – set the operation attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_op_set(Tt_message m,
                            const char *opname);
```

DESCRIPTION

The **tt_message_op_set**( ) function sets the operation attribute for the specified message. The *m* argument is the opaque handle for the message involved in this operation. The *opname* argument is the operation that the receiving process is to perform.

RETURN VALUE

Upon successful completion, the **tt_message_op_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_opnum – retrieve the operation number attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_message_opnum(Tt_message m);
```

DESCRIPTION

The **tt_message_opnum**( ) function retrieves the operation number attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_opnum**( ) function returns the number of the operation involved in this message. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_message_otype – retrieve the object type attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_otype(Tt_message m);
```

DESCRIPTION

The **tt_message_otype**( ) function retrieves the object type attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_otype**( ) function returns the type of the object involved in this message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_otype_set – set the otype attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_otype_set(Tt_message m,
                               const char *otype);
```

DESCRIPTION

The **tt_message_otype_set**( ) function sets the object type (*otype*) attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The otype argument is the type of the object involved in this message.

RETURN VALUE

Upon successful completion, the **tt_message_otype_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_pattern – return the pattern matched by a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_pattern tt_message_pattern(Tt_message m);
```

DESCRIPTION

The **tt_message_pattern**( ) function returns the pattern that the specified message matched.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_pattern**( ) function returns the opaque handle for a message pattern. The application can use **tt_ptr_error**(3) to determine if the handle is valid. The **tt_message_pattern**( ) function returns one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_print – format a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_print(Tt_message m);
```

DESCRIPTION

The **tt_message_print**( ) function formats a message in the same way a message is formatted for the **ttsession**(1) trace and returns a string containing it.

The *m* argument is the message to be formatted.

RETURN VALUE

Upon successful completion, the **tt_message_print**( ) function returns the formatted string. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The **tt_message_print**( ) function allows an application to dump out messages that are received but not understood.

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_free**(3), **tt_ptr_error**(3).

NAME

tt_message_receive – receive a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_message tt_message_receive(void);
```

DESCRIPTION

The **tt_message_receive**( ) function returns a handle for the next message queued to be delivered to the process and also runs any message or pattern callbacks applicable to the queued message.

If the return value of **tt_message_status**(3) for this message is **TT_WRN_START_MESSAGE**, the ToolTalk service started the process to deliver the queued message; the process must reply to this message. If the return value of **tt_message_receive**( ) is zero, no message is available.

RETURN VALUE

Upon successful completion, the **tt_message_receive**( ) function returns the handle for the message object. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_STATE
The Tt_message is in a Tt_state that is invalid for the attempted operation.

APPLICATION USAGE

A zero value can occur if a message or pattern callback processes the message. It can also occur if the interval is too long between the time the file descriptor became active and the **tt_message_receive**( ) call was made. In the latter case, the ToolTalk service will time out and offer the message to another process.

The application should check the tt_message_uid(3) and tt_message_gid(3) against the User ID and Group ID of the application receiving the messsage. If the UID and / or GID of the application do not match that of the message then the receiving application should consider failing the message with TT_DESKTOP_EACCES.

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_reject – reject a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_reject(Tt_message m);
```

DESCRIPTION

The **tt_message_reject**( ) function informs the ToolTalk service that the process cannot handle this message. The ToolTalk service will attempt to deliver the message to other handlers.

The *m* argument is the opaque handle for the message involved in this operation.

Note: If *m* is a TT_OFFER and the one that started the process, this API call will unblock the ptype.

RETURN VALUE

Upon successful completion, the **tt_message_reject**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NOTHANDLER
This application is not the handler for this message.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_STATE
The Tt_message is in a Tt_state that is invalid for the attempted operation.

SEE ALSO

tt_c(5).

NAME

tt_message_rejecter – return offer's nth rejecting procid

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_rejecter(Tt_message m,
                          int n);
```

DESCRIPTION

The **tt_message_rejecter**( ) function returns the procid of the *n*th rejecter of the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument is the number of the rejecter to be returned. The first rejecter is numbered zero.

RETURN VALUE

Upon successful completion, the **tt_message_rejecter**( ) function returns the procid of the *n*th rejecter. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_PROCID
There is no valid default procid, perhaps because **tt_open**(3) has not yet been called.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_STATE
The specified message is not in state **TT_RETURNED**. Since only **TT_OFFER**s can be in state **TT_RETURNED**, this status will be returned if the specified message is a **TT_NOTICE** or a **TT_REQUEST**.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application can use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_message_rejecters_count**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_rejecters_count – return a count of the offer's rejecting procids

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_message_rejecters_count(Tt_message m);
```

DESCRIPTION

The **tt_message_rejecters_count**( ) function returns a count of the procids that are recorded in the offer *m* as having rejected it.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_rejecters_count**( ) function returns a count of the procids that are recorded in the offer *m* as having rejected it. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_PROCID
There is no valid default procid, perhaps because **tt_open**(3) has not yet been called.

TT_ERR_STATE
The specified message is not in state **TT_RETURNED**. Since only **TT_OFFER**s can be in state **TT_RETURNED**, this status will be returned if the specified message is a **TT_NOTICE** or a **TT_REQUEST**.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_message_rejecter**(3), **tt_int_error**(3).

NAME

tt_message_reply – reply to a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_reply(Tt_message m);
```

DESCRIPTION

The **tt_message_reply**( ) function informs the ToolTalk service that the process has handled the message and filled in all return values.

The ToolTalk service sends the message back to the sending process and fills in the state attribute with **TT_HANDLED**.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_reply**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NOTHANDLER
This application is not the handler for this message.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

SEE ALSO

tt_c(5).

NAME

tt_message_scope – retrieve the scope attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_scope tt_message_scope(Tt_message m);
```

DESCRIPTION

The **tt_message_scope**( ) function retrieves the scope attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_scope**( ) function returns a value that identifies the set of processes eligible to receive the message. The following values are defined:

TT_SESSION
All processes joined to the indicated session are eligible.

TT_FILE
All processes joined to the indicated file are eligible.

TT_BOTH
All processes joined to either indicated file or the indicated session are eligible.

TT_FILE_IN_SESSION
All processes joined to both the indicated file and the indicated session are eligible.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the **Tt_scope** integer return value:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_message_scope_set – set the scope attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_scope_set(Tt_message m,
                               Tt_scope s);
```

DESCRIPTION

The **tt_message_scope_set**( ) function sets the scope attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *s* argument identifies the set of processes eligible to receive the message. The following values are defined:

TT_SESSION
All processes joined to the indicated session are eligible.

TT_FILE
All processes joined to the indicated file are eligible.

TT_BOTH
All processes joined to either indicated file or the indicated session are eligible.

TT_FILE_IN_SESSION
All processes joined to both the indicated file and the indicated session are eligible.

RETURN VALUE

Upon successful completion, the **tt_message_scope_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_send – send a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_send(Tt_message m);
```

DESCRIPTION

The **tt_message_send**( ) function sends the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_send**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_ADDRESS
The specified **Tt_address** is invalid.

TT_ERR_CLASS
The specified **Tt_class** is invalid.

TT_ERR_FILE
The specified file does not exist or it is inaccessible.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The objid passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

TT_ERR_OVERFLOW
The ToolTalk service has more active messages than it can handle. (The maximum number of active messages is implementation specific, but is at least 2000.)

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

TT_WRN_STALE_OBJID
The object attribute in the message has been replaced with a newer one.

TT_ERR_UNIMP
The ToolTalk function called is not implemented.

SEE ALSO **tt_c**(5).

NAME

tt_message_send_on_exit – set up a message to send upon unexpected exit

SYNOPSIS

#include <Tt/tt_c.h>

Tt_status tt_message_send_on_exit(Tt_message *m***);**

DESCRIPTION

The **tt_message_send_on_exit**( ) function requests that the ToolTalk service send this message if the process exits unexpectedly. The message is sent to the ToolTalk service, which queues the message internally until either of two events occur:

1. The procid that sent the **tt_message_send_on_exit**( ) message to the ToolTalk service calls **tt_close**(3). In this case, the queued message is deleted.
2. The connection between the **ttsession**(1) server and the process that sent the **tt_message_send_on_exit**( ) message to the ToolTalk service is broken; for example, if the application has crashed.

In this case, the ToolTalk service matches the queued message to its patterns and delivers it in the same manner as if the process had sent the message normally before exiting.

If a process sends a normal termination message but exits without calling **tt_close**(3), both the normal termination message and the on_exit message are delivered.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_send_on_exit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_ADDRESS
The specified **Tt_address** is invalid.

TT_ERR_CLASS
The specified **Tt_class** is invalid.

TT_ERR_FILE
The specified file does not exist or it is inaccessible.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The objid passed to the ToolTalk service does not reference an existing object spec.

SEE ALSO

tt_c(5), **tt_close**(3).

NAME

tt_message_sender – retrieve the sender attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_sender(Tt_message m);
```

DESCRIPTION

The **tt_message_sender**( ) function retrieves the sender attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_sender**( ) function returns the character value that uniquely identifies the sending process. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_sender_ptype – retrieve the sender ptype attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_sender_ptype(Tt_message m);
```

DESCRIPTION

The **tt_message_sender_ptype**( ) function retrieves the sender ptype attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_sender_ptype**( ) function returns the sending process. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_sender_ptype_set – set the sender ptype attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_sender_ptype_set(Tt_message m,
                                      const char *ptid);
```

DESCRIPTION

The **tt_message_sender_ptype_set**( ) function sets the sender ptype attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *ptid* argument is the type of process that is sending this message.

RETURN VALUE

Upon successful completion, the **tt_message_sender_ptype_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_session – retrieve the session attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_session(Tt_message m);
```

DESCRIPTION

The **tt_message_session**( ) function retrieves the session attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_session**( ) function returns the identifier of the session to which this message applies. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_session_set – set the session attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_session_set(Tt_message m,
                                 const char *sessid);
```

DESCRIPTION

The **tt_message_session_set**( ) function sets the session attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *sessid* argument is the identifier of the session in which the process is interested.

RETURN VALUE

Upon successful completion, the **tt_message_session_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5).

NAME

tt_message_state – retrieve the state attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_state tt_message_state(Tt_message m);
```

DESCRIPTION

The **tt_message_state**( ) function retrieves the state attribute from the specified message. The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_state**( ) function returns a value that indicates the current delivery state of the message. The **tt_message_state**( ) function returns one of the following **Tt_status** values:

TT_CREATED
The message has been created, but not yet sent.

TT_SENT
The message has been sent, but not yet handled.

TT_HANDLED
The message has been handled; return values are valid.

TT_FAILED
The message could not be delivered to a handler.

TT_QUEUED
The message has been queued for delivery.

TT_STARTED
The ToolTalk service is attempting to start a process to handle the message.

TT_REJECTED
The message has been rejected by a possible handler.

TT_RETURNED
all observers (and the handler, if there is one) have accepted, rejected, or destroyed the TT_OFFER. The original sender sees this state, and it can be observed.
This comes back to the original sender like the reply for a request. In particular, any message callbacks for the offer are run, and user data attached to the message before sending are available.

TT_ACCEPTED
Offers (only) enter this state when tt_message_accept is done on them by a receiver. The state is seen only by the receiver.

TT_REJECTED
This state already exists; a receiver can tt_message_reject a request that matched a handle pattern, which puts the message into state TT_REJECTED for it. This will be extended to offers -- a receiver that gets an offer will see this message in the TT_REJECTED state.

TT_ABSTAINED
Offers (only) enter this state when a receiving procid does the next tt_message_receive without accepting or rejecting the offer. One can think of TT_ABSTAINED also being entered when a procid destroys an offer without accepting or rejecting it, but since the message is destroyed at that time the procid will never see the state. This state is seen only by the receiver.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the **Tt_state** integer return value:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_message_status – retrieve the status attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_message_status(Tt_message m);
```

DESCRIPTION

The **tt_message_status**( ) function retrieves the status attribute from the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_status**( ) function returns an integer that describes the status stored in the status attribute of this message. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_message_status_string**(3), **tt_int_error**(3).

NAME

tt_message_status_set – set the status attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_status_set(Tt_message m,
                                int status);
```

DESCRIPTION

The **tt_message_status_set**( ) function sets the status attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *status* argument is the status to be stored in this message.

RETURN VALUE

Upon successful completion, the **tt_message_status_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

The status value must be greater than **TT_ERR_LAST** to avoid confusion with the ToolTalk service status values.

SEE ALSO

tt_c(5).

NAME

tt_message_status_string – retrieve the character string stored with the status attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_message_status_string(Tt_message m);
```

DESCRIPTION

The **tt_message_status_string**( ) function retrieves the character string stored with the status attribute for the specified message.

The *m* argument is the opaque handle for the message involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_message_status_string**( ) function returns the status string stored in this message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_message_status**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_status_string_set – set a character string with the status attribute for a message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_status_string_set(Tt_message m,
                                       const char *status_str);
```

DESCRIPTION

The **tt_message_status_string_set**( ) function sets status string of the specified message.

The *m* argument is the opaque handle for the message involved in this operation. The *status_str* argument is the status string to be stored in this message.

RETURN VALUE

Upon successful completion, the **tt_message_status_string_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The status string should be used by the application developer to amplify on, for example, why the application is failing a message.

SEE ALSO

tt_c(5), **tt_message_status_set**(3).

NAME

tt_message_uid – retrieve the user identifier attribute from a message

SYNOPSIS

```
#include <Tt/tt_c.h>

uid_t tt_message_uid(Tt_message m);
```

DESCRIPTION

The **tt_message_uid**( ) function retrieves the user identifier attribute from the specified message.

The ToolTalk service automatically sets the user identifier of a message with the user identifier of the process that created the message.

The *m* argument is the opaque handle for the message involved in this operation.

The application should check the **tt_message_uid( )** and tt_message_gid(3) against the User ID and Group ID of the application receiving the messsage. If the UID and / or GID of the application do not match that of the message then the receiving application should consider failing the message with TT_DESKTOP_EACCES.

RETURN VALUE

Upon successful completion, the **tt_message_uid**( ) function returns the user identifier of the message. If the group **nobody** is returned, the message handle is not valid.

SEE ALSO

tt_c(5), **tt_message_gid**(3).

NAME

tt_message_user – retrieve the user information associated with a message object

SYNOPSIS

```
#include <Tt/tt_c.h>

void *tt_message_user(Tt_message m,
                      int key);
```

DESCRIPTION

The **tt_message_user**( ) function retrieves the user information stored in data cells associated with the specified message object.

The user data is part of the message object (that is, the storage buffer in the application); it is not a part of the actual message. The application can, therefore, only retrieve user information that the application placed in the message.

The *m* argument is the opaque handle for the message involved in this operation. The *key* argument is the user data cell to be retrieved. The user data cell must be unique for this message.

RETURN VALUE

Upon successful completion, the **tt_message_user**( ) function returns the data cell, a piece of arbitrary user data that can hold a **void** *. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned data:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

The user data cell is intended to hold an address. If the address selected is equal to one of the **Tt_status** enumerated values, the result of the **tt_ptr_error**(3) function will not be reliable.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_message_user_set – stores user information associated with a message object

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_user_set(Tt_message m,
                              int key,
                              void *v);
```

DESCRIPTION

The **tt_message_user_set**( ) function stores user information in data cells associated with the specified message object.

The user data is part of the message object (that is, the storage buffer in the application); it is not part of the actual message. Data stored by the sending process in user data cells is not seen by handlers and observers. The application can use arguments for data that needs to be seen by handlers or observers.

The *m* argument is the opaque handle for the message involved in this operation. The *key* argument is the user data cell in which user information is to be stored. The *v* argument is the data cell, a piece of arbitrary user data that can hold a **void** *.

RETURN VALUE

Upon successful completion, the **tt_message_user_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

SEE ALSO

tt_c(5), **tt_message_arg_add**(3).

NAME

tt_message_xarg_add – add an argument with an XDR-interpreted value to a message object

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_xarg_add(Tt_message m,
                              Tt_mode n,
                              const char *vtype,
                              xdrproc_t xdr_proc,
                              void *value);
```

DESCRIPTION

The **tt_message_xarg_add**( ) function adds an argument with an XDR-interpreted value to a message object.

To change existing argument values, the application must use only modes **TT_OUT** or **TT_INOUT**.

Adding arguments when replying to a message produces undefined results.

The *m* argument is the opaque handle for the message involved in this operation. The *n* argument specifies who (sender, handler, observers) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The vtype argument describes the type of argument data being added. The *xdr_proc* argument points to the XDR procedure to be used to serialize the data pointed to by *value*. The *value* argument is the data to be serialized.

RETURN VALUE

Upon successful completion, the **tt_message_xarg_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_MODE
The specified **Tt_mode** is invalid.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_XDR
The XDR procedure failed on the given data, or evaluated to a zero-length expression.

SEE ALSO

tt_c(5).

NAME

tt_message_xcontext_set – set the XDR-interpreted byte-array value of a message's context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_message_xcontext_set(Tt_message m,
                                  const char *slotname,
                                  xdrproc_t xdr_proc,
                                  void *value);
```

DESCRIPTION

The **tt_message_xcontext_set**( ) function sets the XDR-interpreted byte-array value of a message's context.

The *m* argument is the opaque handle for the message involved in this operation. The *slotname* argument describes the slotname in this message. The *value* argument is the byte string with the contents for the message argument. The *xdr_proc* argument points to the XDR procedure to be used to serialize the data pointed to by *value*. The *value* argument is the data to be serialized.

RETURN VALUE

Upon successful completion, the **tt_message_xcontext_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer does not point at an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

TT_ERR_XDR
The XDR procedure failed on the given data, or evaluated to a zero-length expression.

SEE ALSO

tt_c(5).

NAME

tt_netfile_file – map between canonical and local pathnames on the local host

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_netfile_file(const char *netfilename);
```

DESCRIPTION

The **tt_netfile_file**( ) function converts a *netfilename* of the format returned by **tt_file_netfile**(3) to a pathname that is valid on the local host. If the file is not currently mounted on the local host, **tt_netfile_file**( ) constructs a pathname of the form:

/ *mountpoint* / *host* / *filepath*

where *mountpoint* is the mount point pathname in the environment variable *DTMOUNTPOINT*, or **/net** if the variable is null or unset.

The *netfilename* argument is a copy of a null-terminated string returned by **tt_netfile_file**(3) or **tt_host_netfile_file**(3).

RETURN VALUE

Upon successful completion, the **tt_netfile_file**( ) function returns a null-terminated local filename; otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_ERR_NETFILE
The *netfilename* argument is not a valid netfilename.

APPLICATION USAGE

The **tt_file_netfile**(3), **tt_netfile_file**(3), **tt_host_file_netfile**(3) and **tt_host_netfile_file**(3) functions allow an application to determine a path valid on remote hosts, perhaps for purposes of constructing a command string valid for remote execution on that host. By composing the two calls, paths for files not accessible from the current host can be constructed. For example, if path **/sample/file** is valid on host A, a program running on host B can use

```
tt_host_netfile_file("C", tt_host_file_netfile("A", "/sample/file"))
```

to determine a path to the same file valid on host C, if such a path is possible.

The *netfilename* string input to **tt_netfile_file**( ) should be considered opaque; the content and format of the strings are not a public interface. These strings can be safely copied (with **strcpy**(3C) or similar methods), written to files, or transmitted to other processes, perhaps on other hosts.

The *mountpoint* value is intended to be the mount point for the automounter's host map on those systems supporting automounting services.

Allocated strings should be freed using either **tt_free**(3) or **tt_release**(3).

The **tt_open**(3) function need not be called before **tt_netfile_file**( ).

SEE ALSO

tt_c(5), **tt_file_netfile**(3), **tt_host_file_netfile**(3), **tt_host_netfile_file**(3), **tt_open**(3), **tt_free**(3), **tt_release**(3).

NAME

tt_objid_equal – test whether two objids are equal

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_objid_equal(const char *objid1,
                   const char *objid2);
```

DESCRIPTION

The **tt_objid_equal**( ) function tests whether two objids are equal.

The **tt_objid_equal**(3) function is recommended rather than **strcmp**(3) for this purpose because the **tt_objid_equal**(3) function returns 1 even in the case where one objid is a forwarding pointer for the other.

The *objid1* argument is the identifier of the first object involved in this operation. The *objid2* argument is the identifier of the second object involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_objid_equal**( ) function returns an integer that indicates whether the objids are equal. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

0 The *objid1* and *objid2* objects are not equal.

1 The *objid1* and *objid2* objects are equal.

The application can use **tt_int_error**(3) to determine if the integer is valid. The **tt_objid_equal**( ) function returns one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The objid passed to the ToolTalk service does not reference an existing object spec.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_objid_objkey – return the unique key of an objid

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_objid_objkey(const char *objid);
```

DESCRIPTION

The **tt_objid_objkey**( ) function returns the unique key of an objid.

The *objid* argument is the identifier of the object involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_objid_objkey**( ) function returns the unique key of the *objid*. No two objids have the same unique key. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_onotice_create – create a notice

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_message tt_onotice_create(const char *objid,
                             const char *op);
```

DESCRIPTION

The **tt_onotice_create**( ) function creates a message. The created message contains the following:

Tt_address = **TT_OBJECT**
Tt_class = **TT_NOTICE**

The application can use the returned handle to add arguments and other attributes, and to send the message.

The *objid* argument is the identifier of the specified object. The *op* argument is the operation to be performed by the receiving process.

RETURN VALUE

Upon successful completion, the **tt_onotice_create**( ) function returns the unique handle that identifies the message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_open – return the process identifier for the calling process

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_open(void);
```

DESCRIPTION

The **tt_open**( ) function returns the process identifier for the calling process.

RETURN VALUE

The **tt_open**( ) function also sets this identifier as the default procid for the process. The **tt_open**(3) function is typically the first ToolTalk function called by a process.

The application must call **tt_open**(3) before other *tt_* calls are made. However, there are three exceptions: **tt_default_session_set**(3), **tt_feature_required**(3), and **tt_X_session**(3) can be called before **tt_open**( ).

A process can call **tt_open**( ) more than once to obtain multiple procids. To open another session, the process must make the following calls in the order specified:

tt_default_session_set(3)
tt_open( )

RETURN VALUE

Upon successful completion, the **tt_open**( ) function returns the character value that uniquely identifies the process. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

Each procid has its own associated file descriptor, and can join another session. To switch to another procid, the application should call **tt_default_procid_set**( ).

SEE ALSO

tt_c(5), **tt_fd**(3), **tt_default_procid**(3), **tt_default_procid_set**(3), **tt_default_session**(3), **tt_default_session_set**(3), **tt_feature_required**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_orequest_create – create a request message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_message tt_orequest_create(const char *objid,
                              const char *op);
```

DESCRIPTION

The **tt_orequest_create**( ) function creates a message. The created message contains the following:

Tt_address = **TT_OBJECT**
Tt_class = **TT_REQUEST**

The application can use the returned handle to add arguments and other attributes, and to send the message.

The *objid* argument is the identifier of the specified object. The *op* argument is the operation to be performed by the receiving process.

RETURN VALUE

Upon successful completion, the **tt_orequest_create**( ) function returns the unique handle that identifies the message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_otype_base – return the base otype of an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_otype_base(const char *otype);
```

DESCRIPTION

The **tt_otype_base**( ) function returns the base otype of the given otype, or **NULL** if the given otype is not derived.

The otype argument is the object type involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_otype_base**( ) function returns the name of the base otype; if the given otype is not derived, this value is **NULL**. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_otype_is_derived**(3), **tt_otype_derived**(3), **tt_otype_deriveds_count**(3), **tt_spec_type**(3), **tt_message_otype**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_otype_derived – return the ith otype derived from the given otype

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_otype_derived(const char *otype,
                       int i);
```

DESCRIPTION

The **tt_otype_derived**( ) function returns the *i*th otype derived from the given otype.

The otype argument is the object type involved in this operation. The *i* argument is the zero-based index into the otypes derived from the given otype.

RETURN VALUE

Upon successful completion, the **tt_otype_derived**( ) function returns the name of the *i*th otype derived from the given otype. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_otype_is_derived**(3), **tt_otype_base**(3), **tt_otype_deriveds_count**(3), **tt_spec_type**(3), **tt_message_otype**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_otype_deriveds_count – return the number of otypes derived from an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_otype_deriveds_count(const char *otype);
```

DESCRIPTION

The **tt_otype_deriveds_count**( ) function returns the number of otypes derived from the given otype.

The otype argument is the object type involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_otype_deriveds_count**( ) function returns the number of otypes derived from the given otype. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

SEE ALSO

tt_c(5), **tt_otype_is_derived**(3), **tt_otype_base**(3), **tt_otype_derived**(3), **tt_spec_type**(3), **tt_message_otype**(3), **tt_int_error**(3).

NAME

tt_otype_hsig_arg_mode – return the mode of an argument of a request signature of an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_mode tt_otype_hsig_arg_mode(const char *otype,
                               int sig,
                               int arg);
```

DESCRIPTION

The **tt_otype_hsig_arg_mode**( ) function returns the mode of the *arg*th argument of the *sig*th request signature of the given otype.

The otype argument is the object type involved in this operation. The *sig* argument is the zero-based index into the request signatures of the specified otype. The *arg* argument is the zero-based index into the arguments of the specified signature.

RETURN VALUE

Upon successful completion, the **tt_otype_hsig_arg_mode**( ) function returns a value that determines who (sender or handler) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the **Tt_mode** integer return value:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

SEE ALSO

tt_c(5), **tt_otype_hsig_arg_type**(3), **tt_otype_hsig_count**(3), **tt_otype_hsig_args_count**(3), **tt_otype_hsig_op**(3), **tt_int_error**(3).

NAME

tt_otype_hsig_arg_type – return the data type of an argument of a request signature of an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_otype_hsig_arg_type(const char *otype,
                             int sig,
                             int arg);
```

DESCRIPTION

The **tt_otype_hsig_arg_type**( ) function returns the data type of the *arg*th argument of the *sig*th request signature of the given otype.

The otype argument is the object type involved in this operation. The *sig* argument is the zero-based index into the request signatures of the specified otype. The *arg* argument is the zero-based index into the arguments of the specified signature.

RETURN VALUE

Upon successful completion, the **tt_otype_hsig_arg_type**( ) function returns the data type of the specified argument. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_otype_hsig_arg_mode**(3), **tt_otype_hsig_count**(3), **tt_otype_hsig_args_count**(3), **tt_otype_hsig_op**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_otype_hsig_args_count – return the number of arguments of a request signature of an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_otype_hsig_args_count(const char *otype,
                             int sig);
```

DESCRIPTION

The **tt_otype_hsig_args_count**( ) function returns the number of arguments of the *sig*th request signature of the given otype.

The otype argument is the object type involved in this operation. The *sig* argument is the zero-based index into the request signatures of the specified otype.

RETURN VALUE

Upon successful completion, the **tt_otype_hsig_args_count**( ) function returns the number of arguments of the *sig*th request signature of the given otype. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

SEE ALSO

tt_c(5), **tt_otype_hsig_arg_type**(3), **tt_otype_hsig_arg_mode**(3), **tt_otype_hsig_count**(3), **tt_otype_hsig_op**(3), **tt_int_error**(3).

NAME

tt_otype_hsig_count – return the number of request signatures for an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_otype_hsig_count(const char *otype);
```

DESCRIPTION

The **tt_otype_hsig_count**( ) function returns the number of request signatures for the given otype.

The otype argument is the object type involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_otype_hsig_count**( ) function returns the number of request signatures for the given otype. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

SEE ALSO

tt_c(5), **tt_otype_hsig_arg_type**(3), **tt_otype_hsig_arg_mode**(3), **tt_otype_hsig_args_count**(3), **tt_otype_hsig_op**(3), **tt_int_error**(3).

NAME

tt_otype_hsig_op – return the operation name of a request signature of an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_otype_hsig_op(const char *otype,
                       int sig);
```

DESCRIPTION

The **tt_otype_hsig_op**( ) function returns the operation name of the *sig*th request signature of the given otype.

The otype argument is the object type involved in this operation. The *sig* argument is the zero-based index into the request signatures of the given otype.

RETURN VALUE

Upon successful completion, the **tt_otype_hsig_op**( ) function returns the operation attribute of the specified request signature. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_otype_hsig_arg_type**(3), **tt_otype_hsig_arg_mode**(3), **tt_otype_hsig_args_count**(3), **tt_otype_hsig_count**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_otype_is_derived – indicate the otype derivations

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_otype_is_derived(const char *derivedotype,
                        const char *baseotype);
```

DESCRIPTION

The **tt_otype_is_derived**( ) function specifies whether the derived otype is derived directly or indirectly from the base otype.

The *derivedotype* argument is the specified derived otype. The *baseotype* argument is the specified base otype.

RETURN VALUE

Upon successful completion, the **tt_otype_is_derived**( ) function returns 1 if the *derivedotype* is derived directly or indirectly from *baseotype*; otherwise, it returns zero.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

SEE ALSO

tt_c(5), **tt_otype_deriveds_count**(3), **tt_otype_base**(3), **tt_otype_derived**(3), **tt_spec_type**(3), **tt_message_otype**(3), **tt_int_error**(3).

NAME

tt_otype_opnum_callback_add – return a callback if two opnums are equal

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_otype_opnum_callback_add(const char *otid,
                                      int opnum,
                                      Tt_message_callback f);
```

DESCRIPTION

The **tt_otype_opnum_callback_add**( ) function adds a callback that is automatically invoked when a message is delivered because it matched a pattern derived from a signature in the named otype with an opnum equal to the specified one. The callback is defined in **<Tt/tt_c.h>**.

The *otid* argument is the identifier of the object type involved in this operation. The *opnum* argument is the opnum of the specified otype. The *f* argument is the message callback to be run.

RETURN VALUE

Upon successful completion, the **tt_otype_opnum_callback_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

APPLICATION USAGE

The **tt_otype_opnum_callback_add**( ) function will only be called for messages delivered by virtue of matching handler signatures. The callback cannot be called for observer signatures because the observer ptype is not recorded in the incoming message.

SEE ALSO

tt_c(5), **tt_message_callback_add**(3).

NAME

tt_otype_osig_arg_mode – return the mode of an argument of a notice signature of an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_mode tt_otype_osig_arg_mode(const char *otype,
                               int sig,
                               int arg);
```

DESCRIPTION

The **tt_otype_osig_arg_mode**( ) function returns the mode of the *arg*th argument of the *sig*th notice signature of the given otype.

The otype argument is the object type involved in this operation. The *sig* argument is the zero-based index into the notice signatures of the specified otype. The *arg* argument is the zero-based index into the arguments of the specified signature.

RETURN VALUE

Upon successful completion, the **tt_otype_osig_arg_mode**( ) function returns a value that determines who (sender or handler) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the **Tt_mode** integer return value:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

SEE ALSO

tt_c(5), **tt_otype_osig_arg_type**(3), **tt_otype_osig_count**(3), **tt_otype_osig_args_count**(3), **tt_otype_osig_op**(3), **tt_int_error**(3).

NAME

tt_otype_osig_arg_type – return the data type of an argument of a notice signature of an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_otype_osig_arg_type(const char *otype,
                             int sig,
                             int arg);
```

DESCRIPTION

The **tt_otype_osig_arg_type**( ) function returns the data type of the *arg*th argument of the *sig*th notice signature of the given otype.

The otype argument is the object type involved in this operation. The *sig* argument is the zero-based index into the notice signatures of the specified otype. The *arg* argument is the zero-based index into the arguments of the specified signature.

RETURN VALUE

Upon successful completion, the **tt_otype_osig_arg_type**( ) function returns the data type of the specified argument. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_otype_osig_arg_mode**(3), **tt_otype_osig_count**(3), **tt_otype_osig_args_count**(3), **tt_otype_osig_op**(3), **tt_free**(3).

NAME

tt_otype_osig_args_count – returns the number of arguments of a notice signature of an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_otype_osig_args_count(const char *otype,
                             int sig);
```

DESCRIPTION

The **tt_otype_osig_args_count**( ) function returns the number of arguments of the *sig*th notice signature of the given otype.

The otype argument is the object type involved in this operation. The *sig* argument is the zero-based index into the notice signatures of the specified otype.

RETURN VALUE

Upon successful completion, the **tt_otype_osig_args_count**( ) function returns the number of arguments of the *sig*th notice signature of the given otype. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

SEE ALSO

tt_c(5), **tt_otype_osig_arg_type**(3), **tt_otype_osig_arg_mode**(3), **tt_otype_osig_count**(3), **tt_otype_osig_op**(3), **tt_int_error**(3).

NAME

tt_otype_osig_count – return the number of notice signatures for an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_otype_osig_count(const char*otype);
```

DESCRIPTION

The **tt_otype_osig_count**( ) function returns the number of notice signatures for the given otype.

The otype argument is the object type involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_otype_osig_count**( ) function returns the number of notice signatures for the given otype. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

SEE ALSO

tt_c(5), **tt_otype_osig_arg_type**(3), **tt_otype_osig_arg_mode**(3), **tt_otype_osig_args_count**(3), **tt_otype_osig_op**(3), **tt_int_error**(3).

NAME

tt_otype_osig_op – return the op name of a notice signature of an otype

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_otype_osig_op(const char *otype,
                       int sig);
```

DESCRIPTION

The **tt_otype_osig_op**( ) function returns the op name of the *sig*th notice signature of the given otype.

The otype argument is the object type involved in this operation. The *sig* argument is the zero-based index into the notice signatures of the given otype.

RETURN VALUE

Upon successful completion, the **tt_otype_osig_op**( ) function returns the operation attribute of the specified notice signature. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_otype_osig_arg_type**(3), **tt_otype_osig_arg_mode**(3), **tt_otype_osig_args_count**(3), **tt_otype_osig_count**(3), **tt_free**(3).

NAME

tt_pattern_address_add – add a value to the address field for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_address_add(Tt_pattern p,
                                 Tt_address d);
```

DESCRIPTION

The **tt_pattern_address_add**( ) function adds a value to the address field for the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after a **tt_pattern_create**(3) call has been made.

The *d* argument specifies which pattern attributes form the address that messages will be matched against. The following values are defined:

TT_HANDLER
The message is addressed to a specific handler that can perform this operation with these arguments.

TT_OBJECT
The message is addressed to a specific object that can perform this operation with these arguments.

TT_OTYPE
The message is addressed to the type of object that can perform this operation with these arguments.

TT_PROCEDURE
The message is addressed to any process that can perform this operation with these arguments.

RETURN VALUE

Upon successful completion, the **tt_pattern_address_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_arg_add – add an argument to a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_arg_add(Tt_pattern p,
                             Tt_mode n,
                             const char *vtype,
                             const char *value);
```

DESCRIPTION

The **tt_pattern_arg_add**( ) function adds an argument to a pattern. The application must add pattern arguments before it registers the pattern with the ToolTalk service.

The *p* argument is the opaque handle for the pattern involved in this operation The *n* argument specifies who (sender, handler, observers) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The vtype argument describes the type of argument data being added. The type **ALL** matches any argument value type. The *value* argument is the value to fill in. This value must be an unsigned character string. A **NULL** matches any value.

Pattern arguments are positional paramaters, and thus will only match an incoming message if the arguments have the same type and position within the argument list of the incoming message. In order to match an argument which is not the first in a list of arguments, the programmer must use tt_pattern_arg_add(3) to register wildcard arguments for the intervening arguments between the first argument and the argument which it is desired to match on. Wildcard arguments should have the vtype of "ALL" and a value of NULL.

RETURN VALUE

Upon successful completion, the **tt_pattern_arg_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO **tt_c**(5), **tt_pattern_register**(3), **tt_pattern_barg_add**(3), **tt_pattern_iarg_add**(3).

NAME

tt_pattern_barg_add – add an argument with a value that contains embedded nulls to a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_barg_add(Tt_pattern m,
                              Tt_mode n,
                              const char *vtype,
                              const unsigned char *value,
                              int len);
```

DESCRIPTION

The **tt_pattern_barg_add**( ) function adds an argument with a value that contains embedded nulls to a pattern.

The *m* argument is the opaque handle for the pattern involved in this operation. The *n* argument specifies who (sender, handler, observers) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT The argument is written by the sender and the handler and read by all.

The vtype argument describes the type of argument data being added. Type **ALL** matches any argument value type.

The ToolTalk service treats the value as an opaque byte string. To pass structured data, the application and the receiving application must encode and decode these unique values. The most common method to use is XDR.

The *value* argument is the value to be added. **NULL** matches any value.

The *len* argument is the length of the value in bytes.

Pattern arguments are positional paramaters, and thus will only match an incoming message if the arguments have the same type and position within the argument list of the incoming message. In order to match an argument which is not the first in a list of arguments, the programmer must use tt_pattern_arg_add(3) to register wildcard arguments for the intervening arguments between the first argument and the argument which it is desired to match on. Wildcard arguments should have the vtype of "ALL" and a value of NULL.

RETURN VALUE

Upon successful completion, the **tt_pattern_barg_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO **tt_c**(5), **tt_pattern_register**(3), **tt_pattern_arg_add**(3), **tt_pattern_iarg_add**(3).

NAME

tt_pattern_bcontext_add – add a byte-array value to the values in this pattern's named context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_bcontext_add(Tt_pattern p,
                                  const char *slotname,
                                  const unsigned char *value,
                                  int length);
```

DESCRIPTION

The **tt_pattern_bcontext_add**( ) function adds a byte-array value to the values in this pattern's named context.

The *p* argument is the opaque handle for the pattern involved in this operation. The *slotname* argument describes the context for this pattern. The *value* argument is the byte string with the contents for the message context. The *length* argument is the length of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_pattern_bcontext_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_UNIMP
The ToolTalk function called is not implemented.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_pattern_callback_add – register a message-matching callback function

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_callback_add(Tt_pattern m,
                                  Tt_message_callback f);
```

DESCRIPTION

The **tt_pattern_callback_add**( ) function registers a callback function that will be automatically invoked by **tt_message_receive**(3) whenever a message matches the pattern.

The callback is defined in **<Tt/tt_c.h>**. If the callback returns **TT_CALLBACK_CONTINUE**, other callbacks will be run; if no callback returns **TT_CALLBACK_PROCESSED**, **tt_message_receive**(3) returns the message. If the callback returns **TT_CALLBACK_PROCESSED**, no further callbacks will be invoked for this event; **tt_message_receive**(3) does not return the message.

The *m* argument is the opaque handle for the pattern involved in this operation.

The *f* argument is the message callback to be run.

The application should check the tt_message_uid(3) and tt_message_gid(3) against the User ID and Group ID of the application receiving the messsage. If the UID and / or GID of the application do not match that of the message then the receiving application should consider failing the message with TT_DESKTOP_EACCES.

RETURN VALUE

Upon successful completion, the **tt_pattern_callback_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_register**(3), **tt_message_receive**(3).

NAME

tt_pattern_category – return the category value of a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_category tt_pattern_category(Tt_pattern p);
```

DESCRIPTION

The **tt_pattern_category**( ) function returns the category value of the specified pattern. The *p* argument is the opaque handle for a message pattern.

RETURN VALUE

Upon successful completion, the **tt_pattern_category**( ) function returns a value that indicates whether the receiving process will observe or handle messages. The **tt_pattern_category**( ) function returns one of the following **Tt_status** values:

TT_OBSERVE
The receiving process will observe messages.

TT_HANDLE
The receiving process will handle messages.

The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the **Tt_category** integer return value:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_category_set**(3), **tt_int_error**(3).

NAME

tt_pattern_category_set – fill in the category field for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_category_set(Tt_pattern p,
                                  Tt_category c);
```

DESCRIPTION

The **tt_pattern_category_set**( ) function fills in the category field for the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called.

The *c* argument indicates whether the receiving process will observe or handle messages. The following values are defined:

TT_OBSERVE
The receiving process will observe messages.

TT_HANDLE
The receiving process will handle messages.

RETURN VALUE

Upon successful completion, the **tt_pattern_category_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_CATEGORY
The pattern object has no category set.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_category**(3), **tt_pattern_create**(3).

NAME

tt_pattern_class_add – add a value to the class information for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_class_add(Tt_pattern p,
                               Tt_class c);
```

DESCRIPTION

The **tt_pattern_class_add**( ) function adds a value to the class information for the specified pattern.

If the class is **TT_REQUEST**, the sending process expects a reply to the message.

If the class is **TT_NOTICE**, the sending process does not expect a reply to the message.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *c* argument indicates whether the receiving process is to take action after the message is received. The following values are defined:

TT_NOTICE
A notice of an event. The sender does not want feedback on this message.

TT_REQUEST
A request for some action to be taken. The sender must be notified of progress, success or failure, and must receive any return values.

RETURN VALUE

Upon successful completion, the **tt_pattern_class_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_context_add – add a string value to the values of this pattern's context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_context_add(Tt_pattern p,
                                 const char *slotname,
                                 const char *value);
```

DESCRIPTION

The **tt_pattern_context_add**( ) function adds a string value to the values of this pattern's context.

If the value pointer is **NULL**, a slot is created with the specified name but no value is added.

The *p* argument is the opaque handle for the pattern involved in this operation. The *slotname* argument describes the context of this pattern. The *value* argument is the value to be added.

RETURN VALUE

Upon successful completion, the **tt_pattern_context_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_UNIMP
The ToolTalk function called is not implemented.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_pattern_create – request a new pattern object

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_pattern tt_pattern_create(void);
```

DESCRIPTION

The **tt_pattern_create**( ) function requests a new pattern object.

After receiving the pattern object, the application fills in the message pattern fields to indicate what type of messages the process wants to receive and then registers the pattern with the ToolTalk service.

The application can supply multiple values for each attribute added to a pattern (although some attributes are set and can only have one value). The pattern attribute matches a message attribute if any of the values in the pattern match the value in the message. If no value is specified for an attribute, the ToolTalk service assumes that any value will match.

RETURN VALUE

Upon successful completion, the **tt_pattern_create**( ) function returns the opaque handle for a message pattern. The application can use this handle in future calls to identify the pattern object. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_pattern_register**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_pattern_destroy – destroy a pattern object

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_destroy(Tt_pattern p);
```

DESCRIPTION

The **tt_pattern_destroy**( ) function destroys a pattern object.

Destroying a pattern object automatically unregisters the pattern with the ToolTalk service.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called.

RETURN VALUE

Upon successful completion, the **tt_pattern_destroy**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_register**(3), **tt_pattern_create**(3).

NAME

tt_pattern_disposition_add – add a value to the disposition field for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_disposition_add(Tt_pattern p,
                                     Tt_disposition r);
```

DESCRIPTION

The **tt_pattern_disposition_add**( ) function adds a value to the disposition field for the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called.

The *r* argument indicates whether an instance of the receiver is to be started to receive the message immediately, or whether the message is to be queued until the receiving process is started at a later time or discarded if the receiver is not started. The following values are defined:

TT_DISCARD
: There is no receiver for this message. The message will be returned to the sender with the **Tt_status** field containing **TT_FAILED**.

TT_QUEUE
: Queue the message until a process of the proper ptype receives the message.

TT_START
: Attempt to start a process of the proper ptype if none is running.

TT_QUEUE+TT_START
: Queue the message and attempt to start a process of the proper ptype if none is running.

RETURN VALUE

Upon successful completion, the **tt_pattern_disposition_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
: The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
: The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_file_add – add a value to the file field of a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_file_add(Tt_pattern p,
                              const char *file);
```

DESCRIPTION

The **tt_pattern_file_add**( ) function adds a value to the file field of the specified pattern.

The application can use this call to set individual files on individual patterns.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *file* argument is the name of the file of the specified pattern.

RETURN VALUE

Upon successful completion, the **tt_pattern_file_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

However, this call does not cause the pattern's ToolTalk session to be stored in the database.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_iarg_add – add a new integer argument to a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_iarg_add(Tt_pattern m,
                              Tt_mode n,
                              const char *vtype,
                              int value);
```

DESCRIPTION

The **tt_pattern_iarg_add**( ) function adds a new argument to a pattern and sets the value to a given integer.

Add all arguments before the pattern is registered with the ToolTalk service.

The *m* argument is the opaque handle for the pattern involved in this operation. The *n* argument specifies who (sender, handler, observers) writes and reads a message argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The vtype argument describes the type of argument data being added. **NULL** matches any value. The *value* argument is the value to be added.

Pattern arguments are positional paramaters, and thus will only match an incoming message if the arguments have the same type and position within the argument list of the incoming message. In order to match an argument which is not the first in a list of arguments, the programmer must use tt_pattern_arg_add(3) to register wildcard arguments for the intervening arguments between the first argument and the argument which it is desired to match on. Wildcard arguments should have the vtype of "ALL" and a value of NULL. Note that the **tt_pattern_iarg_add ( )** should not be used to add wildcard arguments because NULL, or 0, is a valid number.

RETURN VALUE

Upon successful completion, the **tt_pattern_iarg_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_MODE
The specified **Tt_mode** is invalid.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER

The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_register**(3).

NAME

tt_pattern_icontext_add – add an integer value to the values of this pattern's context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_icontext_add(Tt_pattern p,
                                  const char *slotname,
                                  int value);
```

DESCRIPTION

The **tt_pattern_icontext_add**( ) function adds an integer value to the values of this pattern's context.

The *p* argument is the opaque handle for the pattern involved in this operation. The *slotname* argument describes the slotname in this pattern. The *value* argument is the value to be added.

RETURN VALUE

Tt_status Upon successful completion, the **tt_pattern_icontext_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_UNIMP
The ToolTalk function called is not implemented.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_pattern_object_add – add a value to the object field of a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_object_add(Tt_pattern p,
                                const char *objid);
```

DESCRIPTION

The **tt_pattern_object_add**( ) function adds a value to the object field of the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *objid* argument is the identifier for the specified object. Both **tt_spec_create**(3) and **tt_spec_move**(3) return objids.

RETURN VALUE

Upon successful completion, the **tt_pattern_object_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_op_add – add a value to the operation field of a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_op_add(Tt_pattern p,
                            const char *opname);
```

DESCRIPTION

The **tt_pattern_op_add**( ) function adds a value to the operation field of the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *opname* argument is the name of the operation the process can perform.

RETURN VALUE

Upon successful completion, the **tt_pattern_op_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_opnum_add – add an operation number to a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_opnum_add(Tt_pattern p,
                               int opnum);
```

DESCRIPTION

The **tt_pattern_opnum_add**( ) function adds an operation number to the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *opnum* argument is the operation number to be added.

RETURN VALUE

Upon successful completion, the **tt_pattern_opnum_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_otype_add – add a value to the object type field for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_otype_add(Tt_pattern p,
                               const char *otype);
```

DESCRIPTION

The **tt_pattern_otype_add**( ) function adds a value to the object type field for the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The otype argument is the name of the object type the application manages.

RETURN VALUE

Upon successful completion, the **tt_pattern_otype_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_print – format a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_pattern_print(Tt_pattern p);
```

DESCRIPTION

The **tt_pattern_print**( ) function formats a pattern in the same way a message is formatted for the **ttsession**(1) trace and returns a string containing it.

The *p* argument is the pattern to be formatted.

RETURN VALUE

Upon successful completion, the **tt_pattern_print**( ) function returns the formatted string. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The **tt_pattern_print**( ) function allows an application writer to dump out patterns for debugging.

SEE ALSO

tt_c(5), **tt_ptr_error**(3).

NAME

tt_pattern_register – register a pattern with the ToolTalk service

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_register(Tt_pattern p);
```

DESCRIPTION

The **tt_pattern_register**( ) function registers a pattern with the ToolTalk service.

When the process is registered, it will start receiving messages that match the specified pattern. Once a pattern is registered, no further changes can be made in the pattern.

When the process joins a session or file, the ToolTalk service updates the file and session field of its registered patterns.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called.

RETURN VALUE

Upon successful completion, the **tt_pattern_register**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

SEE ALSO

tt_c(5), **tt_pattern_unregister**(3), **tt_pattern_create**(3).

NAME

tt_pattern_scope_add – add a value to the scope field for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_scope_add(Tt_pattern p,
                               Tt_scope s);
```

DESCRIPTION

The **tt_pattern_scope_add**( ) function adds a value to the scope field for the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *s* argument specifies what processes are eligible to receive the message. The following values are defined:

TT_SESSION
All processes joined to the indicated session are eligible.

TT_FILE
All processes joined to the indicated file are eligible.

TT_BOTH
All processes joined to either indicated file or the indicated session are eligible.

TT_FILE_IN_SESSION
All processes joined to both the indicated file and the indicated session are eligible.

RETURN VALUE

Upon successful completion, the **tt_pattern_scope_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_sender_add – add a value to the sender field for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_sender_add(Tt_pattern p,
                                const char *procid);
```

DESCRIPTION

The **tt_pattern_sender_add**( ) function adds a value to the sender field for the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *procid* argument is the character value that uniquely identifies the process of interest.

RETURN VALUE

Upon successful completion, the **tt_pattern_sender_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_sender_ptype_add – add a value to the sending process's ptype field for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_sender_ptype_add(Tt_pattern p,
                                      const char *ptid);
```

DESCRIPTION

The **tt_pattern_sender_ptype_add**( ) function adds a value to the sending process's ptype field for the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *ptid* argument is the character string that uniquely identifies the type of process in which the application is interested.

RETURN VALUE

Upon successful completion, the **tt_pattern_sender_ptype_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_session_add – adds a value to the session field for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_session_add(Tt_pattern p,
                                 const char *sessid);
```

DESCRIPTION

The **tt_pattern_session_add**( ) function adds a value to the session field for the specified pattern.

When the process joins a session, the ToolTalk service updates the session field of its registered patterns.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *sessid* argument is the session of interest.

RETURN VALUE

Upon successful completion, the **tt_pattern_session_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_state_add – add a value to the state field for a pattern

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_state_add(Tt_pattern p,
                               Tt_state s);
```

DESCRIPTION

The **tt_pattern_state_add**( ) function adds a value to the state field for the specified pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *s* argument indicates the current delivery state of a message. The following values are defined:

TT_CREATED
The message has been created, but not yet sent.

TT_SENT
The message has been sent, but not yet handled.

TT_HANDLED
The message has been handled; return values are valid.

TT_FAILED
The message could not be delivered to a handler.

TT_QUEUED
The message has been queued for delivery.

TT_STARTED
The ToolTalk service is attempting to start a process to handle the message.

TT_REJECTED
The message has been rejected by a possible handler.

RETURN VALUE

Upon successful completion, the **tt_pattern_state_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_create**(3).

NAME

tt_pattern_unregister – unregister a pattern from the ToolTalk service

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_unregister(Tt_pattern p);
```

DESCRIPTION

The **tt_pattern_unregister**( ) function unregisters the specified pattern from the ToolTalk service. The process will stop receiving messages that match this pattern.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called.

RETURN VALUE

Upon successful completion, the **tt_pattern_unregister**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_register**(3), **tt_pattern_create**(3).

NAME

tt_pattern_user – return the value in a user data cell for a pattern object

SYNOPSIS

```
#include <Tt/tt_c.h>

void *tt_pattern_user(Tt_pattern p,
                      int key);
```

DESCRIPTION

The **tt_pattern_user**( ) function returns the value in the indicated user data cell for the specified pattern object.

Every pattern object allows an arbitrary number of user data cells that are each one word in size. The user data cells are identified by integer keys. The tool can use these keys in any manner to associate arbitrary data with a pattern object.

The user data is part of the pattern object (that is, the storage buffer in the application); it is not part of the actual pattern. The content of user cells has no effect on pattern matching.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**( ) is called. The *key* argument is the specified user data cell. The application can use **tt_pattern_user_set**(3) to assign the keys to the user data cells that are part of the pattern object. The value of each data cell must be unique for this pattern.

RETURN VALUE

Upon successful completion, the **tt_pattern_user**( ) function returns the data cell, a piece of arbitrary user data that can hold a **void** *. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned data:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

APPLICATION USAGE

The user data cell is intended to hold an address. If the address selected is equal to one of the **Tt_status** enumerated values, the result of the **tt_ptr_error**(3) function will not be reliable.

SEE ALSO

tt_c(5), **tt_pattern_user_set**(3), **tt_pattern_create**(3), **tt_ptr_error**(3).

NAME

tt_pattern_user_set – store information in the user data cells of a pattern object

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_user_set(Tt_pattern p,
                              int key,
                              void *v);
```

DESCRIPTION

The **tt_pattern_user_set**( ) function stores information in the user data cells associated with the specified pattern object.

The *p* argument is a unique handle for a message pattern. This handle is returned after **tt_pattern_create**(3) is called. The *key* argument is the specified user data cell. The value for each data cell must be unique for this pattern. The *v* argument is the data cell, a piece of arbitrary user data that can hold a **void** *.

RETURN VALUE

Upon successful completion, the **tt_pattern_user_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_pattern_user**(3), **tt_pattern_create**(3).

NAME

tt_pattern_xarg_add – add a new argument with an interpreted XDR value to a pattern object

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_xarg_add(Tt_pattern m,
                              Tt_mode n,
                              const char *vtype,
                              xdrproc_t xdr_proc,
                              void *value);
```

DESCRIPTION

The **tt_pattern_xarg_add**( ) function adds a new argument with an interpreted XDR value to a pattern object.

The *m* argument is the opaque handle for the pattern involved in this operation. The *n* argument specifies who (sender, handler, observers) writes and reads a pattern argument. The following modes are defined:

TT_IN The argument is written by the sender and read by the handler and any observers.

TT_OUT
The argument is written by the handler and read by the sender and any reply observers.

TT_INOUT
The argument is written by the sender and the handler and read by all.

The vtype argument describes the type of argument data being added. The *xdr_proc* argument points to the XDR procedure to be used to serialize the data pointed to by *value*. The *value* argument is the data to be serialized.

RETURN VALUE

Upon successful completion, the **tt_pattern_xarg_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_MODE
The specified **Tt_mode** is invalid.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_XDR
The XDR procedure failed on the given data, or evaluated to a zero-length expression.

SEE ALSO **tt_c**(5).

NAME

tt_pattern_xcontext_add – add an XDR-interpreted byte-array value to this pattern's named context

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pattern_xcontext_add(Tt_pattern p,
                                  const char *slotname,
                                  xdrproc_t xdr_proc,
                                  void *value);
```

DESCRIPTION

The **tt_pattern_xcontext_add**( ) function adds an XDR-interpreted byte-array value to the values in this pattern's named context.

The *p* argument is the opaque handle for the pattern involved in this operation. The *slotname* argument describes the context for this pattern. The *xdr_proc* argument points to the XDR procedure to be used to serialize the data pointed to by value. The *value* argument is the data to be serialized.

RETURN VALUE

Upon successful completion, the **tt_pattern_xcontext_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_UNIMP
The ToolTalk function called is not implemented.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

TT_ERR_XDR
The XDR procedure failed on the given data, or evaluated to a zero-length expression.

SEE ALSO

tt_c(5).

NAME

tt_pnotice_create – create a procedure notice

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_message tt_pnotice_create(Tt_scope scope,
                             const char *op);
```

DESCRIPTION

The **tt_pnotice_create**( ) function creates a message. The created message contains the following:

Tt_address = **TT_PROCEDURE**
Tt_class = **TT_NOTICE**

The application can use the returned handle to add arguments and other attributes, and to send the message.

The *scope* argument determines which processes are eligible to receive the message. The following values are defined:

TT_SESSION
All processes joined to the indicated session are eligible.

TT_FILE
All processes joined to the indicated file are eligible.

TT_BOTH
All processes joined to either indicated file or the indicated session are eligible.

TT_FILE_IN_SESSION
All processes joined to both the indicated file and the indicated session are eligible.

The *op* argument is the operation to be performed by the receiving process.

RETURN VALUE

Upon successful completion, the **tt_pnotice_create**( ) function returns the unique handle that identifies this message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

If the ToolTalk service is unable to create a message when requested, **tt_pnotice_create**() returns an invalid handle. When the application attempts to use this handle with another ToolTalk function, the ToolTalk service will return **TT_ERR_POINTER**.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_pointer_error – return the status of a pointer

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_pointer_error(void *pointer);
```

DESCRIPTION

The **tt_pointer_error**( ) function returns the status of the specified pointer.

If an opaque pointer (**Tt_message** or **Tt_pattern**) or character pointer **(char *)** is specified, this function returns **TT_OK** if the pointer is valid or the encoded **Tt_status** value if the pointer is an error object.

The *pointer* argument is the opaque pointer or character pointer to be checked.

RETURN VALUE

Upon successful completion, the **tt_pointer_error**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_ptr_error**(3).

NAME

tt_prequest_create – create a procedure request message

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_message tt_prequest_create(Tt_scope scope,
                              const char *op);
```

DESCRIPTION

The **tt_prequest_create**( ) function creates a message. The created message created contains the following:

Tt_address = **TT_PROCEDURE**
Tt_class = **TT_REQUEST**

The application can use the returned handle to add arguments and other attributes, and to send the message.

The *scope* argument determines which processes are eligible to receive the message. The following values are defined:

TT_SESSION
All processes joined to the indicated session are eligible.

TT_FILE
All processes joined to the indicated file are eligible.

TT_BOTH
All processes joined to either indicated file or the indicated session are eligible.

TT_FILE_IN_SESSION
All processes joined to both the indicated file and the indicated session are eligible.

The *op* argument is the operation to be performed by the receiving process.

RETURN VALUE

Upon successful completion, the **tt_prequest_create**( ) function returns the unique handle that identifies this message. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

If the ToolTalk service is unable to create a message when requested, **tt_prequest_create**( ) returns an invalid handle. When the application attempts to use this handle with another ToolTalk function, the ToolTalk service will return **TT_ERR_POINTER**.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_procid_session - identify the session in which the indicated *procid* was opened

SYNOPSIS

#include <**Tt/tt_c.h**>
char ∗**tt_procid_session** (*char* ∗*procid*);

DESCRIPTION

The **tt_procid_session()** function retrieves the session in which the given *procid* was opened.

The *procid* argument is a pointer to the unique identifier for the ToolTalk process in which the caller is interested.

DESCRIPTION

Upon successful completion, the **tt_procid_session()** function returns the pointer to a character string that uniquely identifies the session which was the default session at the time the *procid* argument was returned by a **tt_open**(3) or **ttdt_open**(3) call. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession(1)** process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free(3)** to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3)

NAME

tt_ptr_error – pointer error macro

SYNOPSIS

#include <Tt/tt_c.h>

Tt_status tt_ptr_error(*pointer***);**

DESCRIPTION

The **tt_ptr_error**( ) macro expands to **tt_pointer_error((void *)(p))**.

The *pointer* argument is the opaque pointer or character pointer to be checked.

RETURN VALUE

Upon successful completion, the **tt_ptr_error**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tt_c(5), **tt_ptr_error**(3).

NAME

tt_ptype_declare – register the process type with the ToolTalk service

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_ptype_declare(const char *ptid);
```

DESCRIPTION

The **tt_ptype_declare**( ) function registers the process type with the ToolTalk service.

The *ptid* argument is the character string specified in the ptype that uniquely identifies this process.

RETURN VALUE

Upon successful completion, the **tt_ptype_declare**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PTYPE
The specified process type is not the name of an installed process type.

SEE ALSO

tt_c(5).

NAME

tt_ptype_exists – indicate whether a ptype is already installed

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_ptype_exists(const char *ptid);
```

DESCRIPTION

The **tt_ptype_exists**( ) function returns an indication of whether a ptype is already installed.

The *ptid* argument is the character string specifying the ptype.

RETURN VALUE

Upon successful completion, the **tt_ptype_exists**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully and the ptype is already installed.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PTYPE
The specified process type is not the name of an installed process type.

SEE ALSO

tt_c(5).

NAME

tt_ptype_opnum_callback_add – return a callback if two opnums are equal

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_ptype_opnum_callback_add(const char *ptid,
                                      int opnum,
                                      Tt_message_callback f);
```

DESCRIPTION

The **tt_ptype_opnum_callback_add**( ) function returns a callback if the specified opnums are equal. The callback is defined in **<Tt/tt_c.h>**.

When a message is delivered because it matched a pattern derived from a signature in the named ptype with an opnum equal to the specified one, the given callback is run in the usual ToolTalk way.

The *ptid* argument is the identifier of the ptype involved in this operation. The *opnum* argument is the opnum of the specified ptype. The *f* argument is the message callback to be run.

RETURN VALUE

Upon successful completion, the **tt_ptype_opnum_callback_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_PTYPE
The specified process type is not the name of an installed process type.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

APPLICATION USAGE

The **tt_ptype_opnum_callback_add**( ) function will only be called for messages delivered by virtue of matching handler signatures. The callback cannot be called for observer signatures because the observer ptype is not recorded in the incoming message.

SEE ALSO

tt_c(5).

NAME

tt_ptype_undeclare – undeclare a ptype

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_ptype_undeclare(const char *ptid);
```

DESCRIPTION

The **tt_ptype_undeclare**( ) function undeclares the indicated ptype and unregisters the patterns associated with the indicated ptype from the ToolTalk service.

The *ptid* argument is the character string specifying the ptype.

RETURN VALUE

Upon successful completion, the **tt_ptype_undeclare**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PTYPE
The specified process type is not the name of an installed process type.

SEE ALSO

tt_c(5).

NAME

tt_release – free storage allocated on the ToolTalk API allocation stack

SYNOPSIS

```
#include <Tt/tt_c.h>

void tt_release(int mark);
```

DESCRIPTION

The **tt_release**( ) function frees all storage allocated on the ToolTalk API allocation stack since *mark* was returned by **tt_mark**(3).

The *mark* argument is an integer that marks the application's storage position in the ToolTalk API allocation stack.

APPLICATION USAGE

This function frees all storage allocated since the **tt_mark**(3) call that returned *mark* and is typically called at the end of a procedure to release all storage allocated within the procedure.

SEE ALSO

tt_c(5), **tt_mark**(3).

NAME

tt_session_bprop – retrieve the ith value of the named property of a session

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_session_bprop(const char *sessid,
                           const char *propname,
                           int i,
                           unsigned char **value,
                           int *length);
```

DESCRIPTION

The **tt_session_bprop**( ) function retrieves the *i*th value of the named property of the specified session.

If there are *i* values or fewer, both the returned value and the returned length are set to zero.

The *sessid* argument is the session joined. The application can use the *sessid* value returned when **tt_default_session**( ) is called. The *propname* argument is the name of the property from which values are to be obtained. The *i* argument is the number of the item in the property list from which the value is to be obtained. The list numbering begins with zero. The *value* argument is the address of a character pointer to which the ToolTalk service is to point a string that contains the contents of the property. The *len* argument is the address of an integer to which the ToolTalk service is to set the length of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_session_bprop**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

SEE ALSO

tt_c(5).

NAME

tt_session_bprop_add – add a new byte-string value to the end of the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_session_bprop_add(const char *sessid,
                               const char *propname,
                               const unsigned char *value,
                               int length);
```

DESCRIPTION

The **tt_session_bprop_add**( ) function adds a new byte-string value to the end of the list of values for the named property of the specified session.

The *sessid* argument is the name of the session joined. The application can use the *sessid* value returned when **tt_default_session**( ) is called. The *propname* argument is the name of the property to which to add values. The *value* argument is the value to add to the session property. The *length* argument is the size of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_session_bprop_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROPLEN
The specified property value is too long. (The maximum size is implementation specific, but is at least 2048.)

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_session_bprop_set – replace current values stored under the named property of a session

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_session_bprop_set(const char *sessid,
                               const char *propname,
                               const unsigned char *value,
                               int length);
```

DESCRIPTION

The **tt_session_bprop_set**( ) function replaces any current values stored under the named property of the specified session with the given byte-string value.

The *sessid* argument is the name of the session joined. The application can use the *sessid* value returned when **tt_default_session**( ) is called. The *propname* argument is the name of the property whose value is to be replaced. The *value* argument is the value to which the session property is set. If value is **NULL**, the property is removed entirely. The *length* argument is the size of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_session_bprop_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROPLEN
The specified property value is too long. (The maximum size is implementation specific, but is at least 2048.)

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

SEE ALSO

tt_c(5).

NAME

tt_session_join – join a session and make it the default

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_session_join(const char *sessid);
```

DESCRIPTION

The **tt_session_join**( ) function joins the named session and makes it the default session. The *sessid* argument is the name of the session to join.

RETURN VALUE

Upon successful completion, the **tt_session_join**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

APPLICATION USAGE

The application can use the *sessid* value returned by **tt_default_session**(3), **tt_X_session**(3), or **tt_initial_session**(3).

SEE ALSO

tt_c(5), **tt_X_session**(3), **tt_default_session**(3), **tt_initial_session**(3).

NAME

tt_session_prop – return the ith value of a session property

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_session_prop(const char *sessid,
                      const char *propname,
                      int i);
```

DESCRIPTION

The **tt_session_prop**( ) function returns the *i*th value of the specified session property.

The *sessid* argument is the name of the session joined. The application can use the *sessid* value returned when **tt_default_session**( ) is called. The *propname* argument is the name of the property from which a value is to be retrieved. The name must be less than 64 bytes. The *i* argument is the number of the item in the property name list for which the value is to be obtained. The list numbering begins with zero.

RETURN VALUE

Upon successful completion, the **tt_session_prop**( ) function returns the value of the requested property. If there are *i* values or fewer, it returns **NULL**. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

If the returned value has embedded nulls, it is impossible to determine how long it is. The application can use **tt_session_bprop**(3) for values with embedded nulls.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_session_prop_add – add a new character-string value to the end of the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_session_prop_add(const char *sessid,
                              const char *propname,
                              const char *value);
```

DESCRIPTION

The **tt_session_prop_add**( ) function adds a new character-string value to the end of the list of values for the property of the specified session.

The *sessid* argument is the name of the session joined. The application can use the *sessid* value returned when **tt_default_session**( ) is called. The *propname* argument is the name of the property to which a value is to be added. The name must be less than 64 bytes. The *value* argument is the character string to add to the property name list.

RETURN VALUE

Upon successful completion, the **tt_session_prop_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROPLEN
The specified property value is too long. (The maximum size is implementation specific, but is at least 64.)

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

SEE ALSO

tt_c(5).

NAME

tt_session_prop_count – return the number of values stored under a property of a session

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_session_prop_count(const char *sessid,
                          const char *propname);
```

DESCRIPTION

The **tt_session_prop_count**( ) function returns the number of values stored under the named property of the specified session.

The *sessid* argument is the name of the session joined. The application can use the *sessid* value returned when **tt_default_session**( ) is called. The *propname* argument is the name of the property to be examined.

RETURN VALUE

Upon successful completion, the **tt_session_prop_count**( ) function returns the number of values in the specified property list. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_session_prop_set – replace current values for a property of a session with a character-string value

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_session_prop_set(const char *sessid,
                              const char *propname,
                              const char *value);
```

DESCRIPTION

The **tt_session_prop_set**( ) function replaces all current values stored under the named property of the specified session with the given character-string value.

The *sessid* argument is the name of the session joined. The application can use the *sessid* value returned when **tt_default_session**( ) is called. The *propname* argument is the name of the property to be examined. The *value* argument is the new value to be inserted. **NULL** removes a value from the property list.

RETURN VALUE

Upon successful completion, the **tt_session_prop_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROPLEN
The specified property value is too long. (The maximum size is implementation specific, but is at least 64.)

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

SEE ALSO

tt_c(5).

NAME

tt_session_propname – returns an element of the list of property names for a session

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_session_propname(const char *sessid,
                          int n);
```

DESCRIPTION

The **tt_session_propname**( ) function returns the *n*th element of the list of currently defined property names for the specified session.

The *sessid* argument is the name of the session joined. The application can use the *sessid* value returned when **tt_default_session**( ) is called. The *n* argument is the number of the item in the property name list for which a name is to be obtained. The list numbering begins with zero.

RETURN VALUE

Upon successful completion, the **tt_session_propname**( ) function returns the name of the specified property from the session property list. If there are *n* properties or fewer, **tt_session_propname**( ) returns **NULL**. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_session_propnames_count – return the number of property names for the session

SYNOPSIS

#include <Tt/tt_c.h>

int tt_session_propnames_count(const char **sessid***);**

DESCRIPTION

The **tt_session_propnames_count**( ) function returns the number of currently defined property names for the session.

The *sessid* argument is the name of the session joined. The application can use the *sessid* value returned when **tt_default_session**( ) is called.

RETURN VALUE

Upon successful completion, the **tt_session_propnames_count**( ) function returns the number of property names for the session. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_session_quit – quit the session

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_session_quit(const char *sessid);
```

DESCRIPTION

The **tt_session_quit**( ) function informs the ToolTalk service that the process is no longer interested in this ToolTalk session. The ToolTalk service stops delivering messages scoped to this session.

The *sessid* argument is the name of the session to quit.

RETURN VALUE

Upon successful completion, the **tt_session_quit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

SEE ALSO

tt_c(5).

NAME

tt_session_types_load – merge a compiled ToolTalk types file into the running ttsession

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_session_types_load(const char *session,
                                const char *filename);
```

DESCRIPTION

The **tt_session_types_load**( ) function merges a compiled ToolTalk types file into the running **ttsession**(1).

The *session* argument is the name of the running session. The *filename* argument is the name of the compiled ToolTalk types file.

RETURN VALUE

Upon successful completion, the **tt_session_types_load**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

TT_ERR_FILE
The specified file does not exist or it is inaccessible.

TT_ERR_UNIMP
The ToolTalk function called is not implemented.

SEE ALSO

tt_c(5), **ttsession**(1), **ttsession_file**(4).

NAME

tt_spec_bprop – retrieve the ith value of a property

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_spec_bprop(const char *objid,
                        const char *propname,
                        int i,
                        unsigned char **value,
                        int *length);
```

DESCRIPTION

The **tt_spec_bprop**( ) function retrieves the *i*th value of the specified property.

The *objid* argument is the identifier of the object involved in this operation. The *propname* argument is the name of the property whose value is to be retrieved. The name must be less than 64 characters. The *i* argument is the item of the list for which a value is to be obtained. The list numbering begins with zero. The *value* argument is the address of a character pointer to which the ToolTalk service is to point a string that contains the contents of the spec's property. If there are *i* values or fewer, the pointer is set to zero. The *length* argument is the address of an integer to which the ToolTalk service is to set the length of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_spec_bprop**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_spec_bprop_add – add a new byte-string to the end of the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_spec_bprop_add(const char *objid,
                            const char *propname,
                            const unsigned char *value,
                            int length);
```

DESCRIPTION

The **tt_spec_bprop_add**( ) function adds a new byte-string to the end of the list of values associated with the specified spec property.

The *objid* argument is the identifier of the object involved in this operation. The *propname* argument is the name of the property to which the byte-string is to be added. The *value* argument is the byte-string to be added to the property value list. The *length* argument is the length in bytes of the byte-string.

RETURN VALUE

Upon successful completion, the **tt_spec_bprop_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_PROPLEN
The specified property value is too long. (The maximum size is implementation specific, but is at least 64.)

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_spec_bprop_set – replace any current values stored under this spec property with a new byte-string

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_spec_bprop_set(const char *objid,
                            const char *propname,
                            const unsigned char *value,
                            int length);
```

DESCRIPTION

The **tt_spec_bprop_set**( ) function replaces any current values stored under this spec property with a new byte-string.

The *objid* argument is the identifier of the object involved in this operation. The *propname* argument is the name of the property which stores the values. The *value* argument is the byte-string to be added to the property value list. If the value is **NULL**, the property is removed entirely. The *length* argument is the length of the value in bytes.

RETURN VALUE

Upon successful completion, the **tt_spec_bprop_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_PROPLEN
The specified property value is too long. (The maximum size is implementation specific, but is at least 64.)

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

SEE ALSO

tt_c(5).

NAME

tt_spec_create – create an in-memory spec for an object

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_spec_create(const char *filepath);
```

DESCRIPTION

The **tt_spec_create**( ) function creates a spec (in memory) for an object.

The application can use the objid returned in future calls to manipulate the object.

The *filepath* argument is the name of the file.

RETURN VALUE

Upon successful completion, the **tt_spec_create**( ) function returns the identifier for this object. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

TT_ERR_PATH
The specified pathname included an unsearchable directory.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

To make the object a permanent ToolTalk item or one visible to other processes, the creating process must call **tt_spec_write**(3).

SEE ALSO

tt_c(5), **tt_spec_type_set**(3), **tt_spec_write**(3), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_spec_destroy – destroy an object's spec

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_spec_destroy(const char *objid);
```

DESCRIPTION

The **tt_spec_destroy**( ) function destroys an object's spec immediately.

The *objid* argument is the identifier of the object involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_spec_destroy**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

SEE ALSO

tt_c(5).

NAME

tt_spec_file – retrieve the name of the file that contains the object described by the spec

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_spec_file(const char *objid);
```

DESCRIPTION

The **tt_spec_file**( ) function retrieves the name of the file that contains the object described by the spec.

The *objid* argument is the identifier of the object involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_spec_file**( ) function returns the absolute pathname of the file. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_spec_move – notify the ToolTalk service that an object has moved to a different file

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_spec_move(const char *objid,
                   const char *newfilepath);
```

DESCRIPTION

The **tt_spec_move**( ) function notifies the ToolTalk service that this object has moved to a different file.

The ToolTalk service returns a new objid for the object and leaves a forwarding pointer from the old objid to the new one.

If a new objid is not required (for example, because the new and old files are in the same file system), **tt_spec_move**( ) returns **TT_WRN_SAME_OBJID**.

The *objid* argument is the identifier of the object involved in this operation.

The *newfilepath* argument is the new file name.

RETURN VALUE

Upon successful completion, the **tt_spec_move**( ) function returns the new unique identifier of the object involved in this operation. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_PATH
The specified pathname included an unsearchable directory.

TT_WRN_SAME_OBJID
A new *objid* is not required.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

For efficiency and reliability, the application should replace any references in the application to the old *objid* with references to the new one.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_spec_prop – retrieve the ith value of the property associated with an object spec

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_spec_prop(const char *objid,
                   const char *propname,
                   int i);
```

DESCRIPTION

The **tt_spec_prop**( ) function retrieves the *i*th value of the property associated with this object spec.

The *objid* argument is the identifier of the object involved in this operation. The *propname* argument is the name of the property associated with the object spec. The *i* argument is the item of the list whose value is to be retrieved. The list numbering begins with zero.

RETURN VALUE

Upon successful completion, the **tt_spec_prop**( ) function returns the contents of the property value. If there are *i* values or less, **tt_spec_prop**( ) returns **NULL**. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

If the returned value has embedded nulls, its length cannot be determined.

SEE ALSO **tt_c**(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_spec_prop_add – add a new item to the end of the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_spec_prop_add(const char *objid,
                           const char *propname,
                           const char *value);
```

DESCRIPTION

The **tt_spec_prop_add**( ) function adds a new item to the end of the list of values associated with this spec property.

The *objid* argument is the identifier of the object involved in this operation. The *propname* argument is the property to which the item is to be added. The *value* argument is the new character-string to be added to the property value list.

RETURN VALUE

Upon successful completion, the **tt_spec_prop_add**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_PROPLEN
The specified property value is too long. (The maximum size is implementation specific, but is at least 2048.)

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

SEE ALSO

tt_c(5), **tt_spec_prop_set**(3).

NAME

tt_spec_prop_count – return the number of values listed in this spec property

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_spec_prop_count(const char *objid,
                       const char *propname);
```

DESCRIPTION

The **tt_spec_prop_count**( ) function returns the number of values listed in this spec property.

The *objid* argument is the identifier of the object involved in this operation. The *propname* argument is the name of the property that contains the value to be returned.

RETURN VALUE

Upon successful completion, the **tt_spec_prop_count**( ) function returns the number of values listed in the spec property. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_spec_prop_set – replace property values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_spec_prop_set(const char *objid,
                           const char *propname,
                           const char *value);
```

DESCRIPTION

The **tt_spec_prop_set**( ) function replaces any values currently stored under this property of the object spec with a new value.

The *objid* argument is the identifier of the object involved in this operation. The *propname* argument is the name of the property which stores the values. The *value* argument is the value to be placed in the property value list. If value is **NULL**, the property is removed entirely.

RETURN VALUE

Upon successful completion, the **tt_spec_prop_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_PROPLEN
The specified property value is too long. (The maximum size is implementation specific, but is at least 2048.)

TT_ERR_PROPNAME
The specified property name is syntactically invalid.

SEE ALSO

tt_c(5), **tt_spec_prop_add**(3).

NAME

tt_spec_propname – return an element of the property name list for an object spec

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_spec_propname(const char *objid,
                       int n);
```

DESCRIPTION

The **tt_spec_propname**( ) function returns the *n*th element of the property name list for this object spec.

The *objid* argument is the identifier of the object involved in this operation. The *n* argument is the item of the list whose element is to be returned. The list numbering begins with zero.

RETURN VALUE

Upon successful completion, the **tt_spec_propname**( ) function returns the property name. If there are *n* properties or less, **tt_spec_propname**( ) returns **NULL**. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_spec_propnames_count – return the number of property names for an object

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_spec_propnames_count(const char *objid);
```

DESCRIPTION

The **tt_spec_propnames_count**( ) function returns the number of property names for this object.

The *objid* argument is the identifier of the object involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_spec_propnames_count**( ) function returns the number of values listed in the spec property. The application can use **tt_int_error**(3) to extract one of the following **Tt_status** values from the returned integer:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

SEE ALSO

tt_c(5), **tt_int_error**(3).

NAME

tt_spec_type – return the name of the object type

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_spec_type(const char *objid);
```

DESCRIPTION

The **tt_spec_type**( ) function returns the name of the object type.

The *objid* argument is the identifier of the object involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_spec_type**( ) function returns the type of this object. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_spec_type_set – assign an object type value to an object spec

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_spec_type_set(const char *objid,
                           const char *otid);
```

DESCRIPTION

The **tt_spec_type_set**( ) function assigns an object type value to the object spec.

The type must be set before the spec is written for the first time and cannot be set thereafter.

The *objid* argument is the identifier of the object involved in this operation. The *otid* argument is the otype to be assigned to the spec.

RETURN VALUE

Upon successful completion, the **tt_spec_type_set**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_READONLY
The attribute the application is trying to change is not owned or writable by the current user.

SEE ALSO

tt_c(5), **tt_spec_create**(3), **tt_spec_write**(3).

NAME

tt_spec_write – write the spec and any associated properties to the ToolTalk database

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_spec_write(const char *objid);
```

DESCRIPTION

The **tt_spec_write**( ) function writes the spec and any associated properties to the ToolTalk database. The type must be set before the spec is written for the first time.

The *objid* argument is the identifier of the object involved in this operation.

RETURN VALUE

Upon successful completion, the **tt_spec_write**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OBJID
The *objid* passed to the ToolTalk service does not reference an existing object spec.

TT_ERR_OTYPE
The specified object type is not the name of an installed object type.

APPLICATION USAGE

It is not necessary to perform a write operation after a destroy operation.

Several changes can be batched between write calls; for example, the application can create an object spec, set some properties, and then write all the changes at once with one write call.

SEE ALSO

tt_c(5), **tt_spec_create**(3), **tt_spec_type_set**(3).

NAME

tt_status_message – provide a message for a problem status code

SYNOPSIS

#include <Tt/tt_c.h>

char *__tt_status_message(Tt_status** *ttrc*__);__

DESCRIPTION

The **tt_status_message**( ) function returns a pointer to a message that describes the problem indicated by this status code.

The *ttrc* argument is the status code received during an operation.

RETURN VALUE

Upon successful completion, the **tt_status_message**( ) function returns a pointer to a character string that describes the status code, which is one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_xxx Any other **TT_** status code is explained in the returned string.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_free**(3).

NAME

tt_thread_procid - identify the default process for the currently active thread

SYNOPSIS

#include <**Tt/tt_c.h**>
char ∗**tt_thread_procid** (*void*);

DESCRIPTION

The **tt_thread_procid()** function retrieves the current default procid for the thread.

If there is no default *procid* set for the currently-active thread, then the process default *procid* is returned.

DESCRIPTION

Upon successful completion, the **tt_thread_procid()** function returns the pointer to a character string that uniquely identifies the default process for the currently active thread. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_prt_error**(3), **tt_free**(3)

NAME

tt_thread_procid_set - set the default *procid* for the currently active thread

SYNOPSIS

#include <**Tt/tt_c.h**>
Tt_status tt_thread_procid_set (*const char* ∗*procid*);

DESCRIPTION

The **tt_thread_procid_set()** function sets the default *procid* for the currently active thread.

The *procid* argument is the name of process that is to be the default process.

DESCRIPTION

Upon successful completion, the **tt_thread_procid_set()** function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

SEE ALSO

tt_c(5), **tt_open**(3)

NAME

tt_thread_session - retrieve the default session identifier for the currently active thread

SYNOPSIS

#include <**Tt/tt_c.h**>
char ∗**tt_thread_session** (*void*);

DESCRIPTION

The **tt_thread_session()** function retrieves the default session identifier for the currently active thread.

If there is no default session set for the currently-active thread, then the process default session identifier is returned.

DESCRIPTION

Upon successful completion, the **tt_thread_session()** function returns the pointer to the unique identifier for the active thread's current default session. If the pointer is NULL, no default session is set. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

APPLICATION USAGE

A session can have more than one session identifier. This means that the application cannot compare the result of **tt_thread_session()** with the result of **tt_message_session**(3) to verify that the message was sent in the default session.

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_prt_error**(3), **tt_free**(3)

NAME

tt_thread_session_set - set the default session identifier for the currently active thread

SYNOPSIS

#include <**Tt/tt_c.h**>
Tt_status tt_thread_session_set (*const char *sessid*);

DESCRIPTION

The **tt_thread_session_set()** function sets the default session identifier for the currently active thread.

The ToolTalk service uses the initial user session as the default session and supports one session per *procid*. The application can make this call before it calls **tt_open**(3) to specify the session to which it wants to connect in the active thread.

The *sessid* argument is a pointer to the unique identifier for the session in which the *procid* is interested.

DESCRIPTION

Upon successful completion, the **tt_thread_session_set()** function returns the status of the operation as one of the following *Tt_status* values:

TT_OK The operation completed successfully.

TT_WRN_NOT_ENABLED
The feature has not yet been enabled.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
The current default process identifier is out of date or invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

APPLICATION USAGE

To change to another opened session, the application must use either **tt_thread_procid_set**(3) or .!! "x X 0 (tt_default_procid_set) link-text" **tt_default_procid_set**(3).

To join other sessions, the procid must first set the new session as the default session, and then initialize and register with the ToolTalk service. The calls required must be in the following order:

tt_thread_session_set() or **tt_default_session_set**(3) **tt_open**(3)
The **tt_open**(3) may create another ToolTalk *procid*, the connection to which is identified by a *procid*. Only one ToolTalk session per *procid* is allowed. (However, multiple *procids* are allowed in a client.) There are no API calls to determine to which session a particular *procid* is connected. If it is important for the application to know the session to which it is connected. It must make the following calls in the indicated order:
tt_open(3) **tt_thread_session**(3)
The application can then store the information by indexing it by the *procid* returned by the **tt_open**(3) call.

SEE ALSO **tt_c**(5), **tt_open**(3), **tt_default_procid**(3), **tt_thread_procid**(3), **tt_default_session**(3), **tt_thread_session**(3).

NAME

tt_trace_control – control client-side tracing

SYNOPSIS

```
#include <Tt/tt_c.h>

int tt_trace_control(int onoff);
```

DESCRIPTION

The **tt_trace_control**( ) function sets or clears an internal flag controlling all client-side tracing. The intent of this is to be called from debugger breakpoints, allowing a programmer to narrow the trace to the suspect area.

The value of the *onoff* argument affects tracing as follows:

- 0 Tracing is turned off.
- 1 Tracing is turned on.
- –1 Tracing is turned on if it was off and vice-versa.

RETURN VALUE

The **tt_trace_control**( ) function returns the previous setting of the trace flag.

APPLICATION USAGE

This call does not return one of the **TT_xxx** type of errors or warnings, but only the numbers 1 or zero.

SEE ALSO

tt_c(5), **ttsession**(1).

NAME

tt_xcontext_join – add an XDR-interpreted byte-array to the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_xcontext_join(const char *slotname,
                           xdrproc_t xdr_proc,
                           void *value);
```

DESCRIPTION

The **tt_xcontext_join**( ) function adds the given XDR-interpreted byte-array value to the list of values for the named contexts of all patterns.

The *slotname* argument describes the slotname in this message. The *xdr_proc* argument points to the XDR procedure to be used to serialize the data pointed to by *value*. The *value* argument is the data to be serialized.

RETURN VALUE

Upon successful completion, the **tt_xcontext_join**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

TT_ERR_XDR
The XDR procedure failed on the given data, or evaluated to a zero-length expression.

SEE ALSO

tt_c(5).

NAME

tt_xcontext_quit – remove an XDR-interpreted byte-array value from the list of values

SYNOPSIS

```
#include <Tt/tt_c.h>

Tt_status tt_xcontext_quit(const char *slotname,
                           xdrproc_t xdr_proc,
                           void *value);
```

DESCRIPTION

The **tt_xcontext_quit**( ) function removes the given XDR-interpreted byte-array value from the list of values for the contexts of all patterns.

The *slotname* argument describes the slotname in this message. The *xdr_proc* argument points to the XDR procedure to be used to serialize the data pointed to by *value*.

The *value* argument is the data to be serialized.

RETURN VALUE

Upon successful completion, the **tt_xcontext_quit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_SLOTNAME
The specified slotname is syntactically invalid.

TT_ERR_XDR
The XDR procedure failed on the given data, or evaluated to a zero-length expression.

SEE ALSO

tt_c(5).

NAME

tt_X_session – return the session associated with an X window system display

SYNOPSIS

```
#include <Tt/tt_c.h>

char *tt_X_session(const char *xdisplaystring);
```

DESCRIPTION

The **tt_X_session**( ) function returns the session associated with the named X window system display.

The application can call **tt_X_session**( ) before it calls **tt_open**(3).

The *xdisplaystring* argument is the name of an X display server; for example, **somehost:0** or **:0**.

RETURN VALUE

Upon successful completion, the **tt_X_session**( ) function returns the identifier for the ToolTalk session associated with the named X window system display. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned pointer:

TT_OK The operation completed successfully.

TT_ERR_SESSION
The *xdisplaystring* does not name an X display.

TT_ERR_POINTER
The *xdisplaystring* is **NULL**.

APPLICATION USAGE

The application should use **tt_free**(3) to free any data stored in the address returned by the ToolTalk API.

SEE ALSO

tt_c(5), **tt_ptr_error**(3), **tt_open**(3), **tt_free**(3),

NAME

ttdt_Get_Modified – ask if any ToolTalk client has changes pending on a file

SYNOPSIS

```
#include <Tt/tttk.h>

int ttdt_Get_Modified(Tt_message context,
                      const char *pathname,
                      Tt_scope the_scope,
                      XtAppContext app2run,
                      int ms_timeout);
```

DESCRIPTION

The **ttdt_Get_Modified**( ) function sends a *Get_Modified* request in the scope *the_scope* and waits for the reply. A *Get_Modified* request asks if any ToolTalk client has changes pending on *pathname* that it intends to make persistent.

The *context* argument describes the environment to use. If *context* is not zero, messages created by **ttdt_Get_Modified**( ) inherit from *context* all contexts whose slotname begins with the characters **ENV_**. That is, the environment described in *context* is propagated to messages created by **ttdt_Get_Modified**( ).

The *pathname* argument is a pointer to a pathname on which the client is operating.

The *the_scope* argument identifies the scope of the request. If *the_scope* is **TT_SCOPE_NONE**, **ttdt_Get_Modified**( ) tries **TT_BOTH**, and falls back to **TT_FILE_IN_SESSION** if, for example, the ToolTalk database server is not installed on the file server that owns *pathname*.

The **ttdt_Get_Modified**( ) function passes *app2run* and *ms_timeout* to **tttk_block_while**(3), blocking on the reply to the *Get_Modified* request it sends.

RETURN VALUE

Upon successful completion, the **ttdt_Get_Modified**( ) function returns non-zero if the *Get_Modified* request receives an affirmative reply within *ms_timeout* milliseconds; otherwise, it returns zero.

SEE ALSO

tttk(5), **ttdt_file_join**(3), **ttdt_file_event**(3), **tttk_block_while**(3).

NAME

ttdt_Revert – request a ToolTalk client to revert a file

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status ttdt_Revert(Tt_message context,
                      const char *pathname,
                      Tt_scope the_scope,
                      XtAppContext app2run,
                      int ms_timeout);
```

DESCRIPTION

The **ttdt_Revert**( ) function sends a *Revert* request in the *the_scope* argument and waits for the reply. A *Revert* request asks the handling ToolTalk client to discard any changes pending on *pathname*.

The *context* argument describes the environment to use. If *context* is not zero, messages created by **ttdt_Revert**( ) inherit from *context* all contexts whose slotname begins with the characters **ENV_**.

The *the_scope* argument identifies the scope of the request. If *the_scope* is **TT_SCOPE_NONE**, **ttdt_Revert**( ) tries **TT_BOTH**, and falls back to **TT_FILE_IN_SESSION** if, for example, the ToolTalk database server is not installed on the file server that owns *pathname*.

The **ttdt_Revert**( ) function passes *app2run* and *ms_timeout* to **tttk_block_while**(3), blocking on the reply to the *Save* request it sends.

RETURN VALUE

Upon successful completion, the **ttdt_Revert**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The sent request received an affirmative reply within *ms_timeout* milliseconds.

TT_DESKTOP_ETIMEDOUT
No reply was received within *ms_timeout* milliseconds.

TT_DESKTOP_EPROTO
The request was failed, but the handler set the **tt_message_status**( ) of the failure reply to **TT_OK**, instead of a specific error status.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot

restart it.

TT_ERR_OVERFLOW
The ToolTalk service has more active messages than it can handle. (The maximum number of active messages is implementation specific, but is at least 2000.)

TT_ERR_POINTER
The *pathname* argument was **NULL** or was a ToolTalk error pointer.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

SEE ALSO

tttk(5), **ttdt_Save**(3), **ttdt_file_join**(3), **ttdt_file_event**(3), **tttk_block_while**(3).

NAME

ttdt_Save – request a ToolTalk client to save a file

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status ttdt_Save(Tt_message context,
                    const char *pathname,
                    Tt_scope the_scope,
                    XtAppContext app2run,
                    int ms_timeout);
```

DESCRIPTION

The **ttdt_Save**( ) function sends a *Save* request in the *the_scope* argument and waits for the reply. A *Save* request asks the handling ToolTalk client to save any changes pending on *pathname*.

The *context* argument describes the environment to use. If *context* is not zero, messages created by **ttdt_Save**( ) inherit from *context* all contexts whose slotname begins with the characters **ENV_**.

The *the_scope* argument identifies the scope of the request. If *the_scope* is **TT_SCOPE_NONE**, **ttdt_Save**( ) tries **TT_BOTH**, and falls back to **TT_FILE_IN_SESSION** if, for example, the ToolTalk database server is not installed on the file server that owns *pathname*.

The **ttdt_Save**( ) function passes *app2run* and *ms_timeout* to **tttk_block_while**(3), blocking on the reply to the *Save* request it sends.

RETURN VALUE

Upon successful completion, the **ttdt_Save**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The sent request received an affirmative reply within *ms_timeout* milliseconds.

TT_DESKTOP_ETIMEDOUT
No reply was received within *ms_timeout* milliseconds.

TT_DESKTOP_EPROTO
The request was failed, but the handler set the **tt_message_status**( ) of the failure reply to **TT_OK**, instead of a specific error status.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot

restart it.

TT_ERR_OVERFLOW
The ToolTalk service has more active messages than it can handle. (The maximum number of active messages is implementation specific, but is at least 2000.)

TT_ERR_POINTER
The *pathname* argument was **NULL** or was a ToolTalk error pointer.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

SEE ALSO

tttk(5), **ttdt_Revert**(3), **ttdt_file_join**(3), **ttdt_file_event**(3), **tttk_block_while**(3).

NAME

ttdt_close – destroy a ToolTalk communication endpoint

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status ttdt_close(const char *procid,
                     const char *new_procid,
                     int sendStopped);
```

DESCRIPTION

The **ttdt_close**( ) function destroys a ToolTalk communication endpoint.

If *sendStopped* is True, the **ttdt_close**( ) function sends a *Stopped* notice; otherwise, it sends no notice. If *procid* is not **NULL**, **ttdt_close**( ) calls **tt_default_procid_set**( ) with a *procid* argument and then calls **tt_close**( ); otherwise, it closes the current default procid. If *new_procid* is not **NULL**, **ttdt_close**( ) calls **tt_default_procid_set**( ) with a *new_procid* argument.

RETURN VALUE

The **ttdt_close**( ) function may return any of the errors returned by **tt_default_procid_set**(3) and **tt_close**(3).

No errors are propagated if sending the *Stopped* notice fails.

SEE ALSO

tttk(5), **ttdt_open**(3), **tt_default_procid_set**(3), **tt_close**(3).

NAME

ttdt_file_event – use ToolTalk to announce an event about a file

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status ttdt_file_event(Tt_message context,
                          Tttk_op event,
                          Tt_pattern *patterns,
                          int send);
```

DESCRIPTION

The **ttdt_file_event**( ) function is used to create and send a ToolTalk notice announcing an event pertaining to a file. The file is indicated by the *pathname* argument that was passed to **ttdt_file_join**(3) when *patterns* was created.

The *event* argument identifies the event. If *event* is **TTDT_MODIFIED**, **ttdt_file_event**( ) registers in the *the_scope* argument passed to **ttdt_file_join**(3) to handle *Get_Modified*, *Save*, and *Revert* requests. *Get_Modified* is handled transparently by associating the modified state of the file with *patterns*. *Save* and *Revert* requests are passed to the **Ttdt_file_cb** that was given to **ttdt_file_join**(3). If *send* is True, **ttdt_file_event**( ) sends *Modified* in *the_scope*. If *event* is **TTDT_SAVED** or **TTDT_REVERTED**, **ttdt_file_event**( ) unregisters handler patterns for *Get_Modified*, *Save*, and *Revert* requests. If *send* is True, **ttdt_file_event**( ) sends *Saved* or *Reverted*, respectively, in *the_scope*.

The *context* argument describes the environment to use. If *context* is not zero, messages created by **ttdt_file_event**( ) inherit from *context* all contexts whose slotname begins with the characters **ENV_**.

RETURN VALUE

Upon successful completion, the **ttdt_file_event**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OVERFLOW
The ToolTalk service has more active messages than it can handle. (The maximum number of active messages is implementation specific, but is at least 2000.)

TT_ERR_POINTER
The *patterns* argument was **NULL**.

SEE ALSO

tttk(5), **ttdt_file_join**(3), **ttdt_Get_Modified**(3), **ttdt_file_quit**(3).

NAME

ttdt_file_join – register to observe ToolTalk events on a file

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_pattern *ttdt_file_join(const char *pathname,
                           Tt_scope the_scope,
                           int join,
                           Ttdt_file_cb cb,
                           void *clientdata);
```

DESCRIPTION

The **ttdt_file_join**( ) function registers to observe *Deleted*, *Modified*, *Reverted*, *Moved*, and *Saved* notices.

If *join* is True, **ttdt_file_join**( ) calls **tt_file_join**( ) with a *pathname* argument.

The *the_scope* argument identifies the scope of the request. If *the_scope* is **TT_SCOPE_NONE**, it tries **TT_BOTH**, and falls back to **TT_FILE_IN_SESSION** if, for example, the ToolTalk database server is not installed on the file server that owns *pathname*.

The **ttdt_file_join**( ) function associates *the_scope* and a copy of *pathname* with the **Tt_pattern**s returned, so that **ttdt_file_quit**( ) can access them. Thus, the caller is free to modify or free *pathname* after **ttdt_file_join**( ) returns.

The *clientdata* argument points to arbitrary data that will be passed into the callback unmodified.

The **Ttdt_file_cb** argument is a callback defined as:

```
Tt_message (*Ttdt_file_cb)(Tt_message msg,
                           Tttk_op op,
                           char *pathname,
                           void *clientdata,
                           int same_euid_egid,
                           int same_procid);
```

The *message* argument is the message. The *op* argument is the operation. The *pathname* argument is the pathname of the file the message is about. The *clientdata* argument is the client data passed into **ttdt_file_join**( ). The *same_euid_egid* argument is True if the sender can be trusted; otherwise it is False. The *same_procid* argument is True if the sender is the same procid as the receiver; otherwise it is False. A **Ttdt_file_cb** must return the message if it does not consume the message. (Consuming means replying, rejecting or failing a request, and then destroying the message.) Otherwise, it must consume the message and return either zero or a **tt_error_pointer**( ) cast to **Tt_message**.

RETURN VALUE

Upon successful completion, the **ttdt_file_join**( ) function returns a null-terminated array of **Tt_pattern**, which can be passed to **ttdt_file_event**(3) to register for requests that the application should handle once it begins to modify the file; otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PATH
The specified pathname included an unsearchable directory.

APPLICATION USAGE

The null-terminated array of **Tt_pattern** returned by **ttdt_file_join**( ) should be destroyed by passing the array to **ttdt_file_quit**(3).

The *pathname* argument to **Ttdt_file_cb** is a copy that can be freed using **tt_free**(3).

EXAMPLES

This is the typical algorithm of a **Ttdt_file_cb**:

```
Tt_message myFileCB(Tt_message     msg,
        Tttk_op       op,
        char         *pathname,
        int          trust,
        int          isMe)
{
        tt_free(pathname);
        Tt_status status = TT_OK;
        switch(op) {
          case TTDT_MODIFIED:
                if ((_modifiedByMe)&&(! isMe)) {
                        /* Hmm, the other editor either does not know or
                         * does not care that we are already modifying the
                         * file, so the last saver will win.
                         */
                } else {
                        /* Interrogate user if she ever modifies the buffer */
                        _modifiedByOther = 1;
                        XtAddCallback(myTextWidget, XmNmodifyVerifyCallback,
                                myTextModifyCB, 0);
                }
                break;
            case TTDT_GET_MODIFIED:
                tt_message_arg_ival_set(msg, 1, _modifiedByMe);
```

```
                tt_message_reply(msg);
                break;
        case TTDT_SAVE:
                status = mySave(trust);
                if (status == TT_OK) {
                        tt_message_reply(msg);
                } else {
                        tttk_message_fail(msg, status, 0, 0);
                }
                break;
        case TTDT_REVERT:
                status = myRevert(trust);
                if (status == TT_OK) {
                        tt_message_reply(msg);
                } else {
                        tttk_message_fail(msg, status, 0, 0);
                }
                break;
        case TTDT_REVERTED:
                if (! isMe) {
                        _modifiedByOther = 0;
                }
                break;
        case TTDT_SAVED:
                if (! isMe) {
                        _modifiedByOther = 0;
                        int choice = myUserChoice(myContext, myBaseFrame,
                                                  "Another tool has saved "
                                                  "this file.", 2, "Ignore",
                                                  "Revert");
                        switch(choice) {
                          case 1:
                                myRevert(1);
                                break;
                        }
                }
                break;
        case TTDT_MOVED:
        case TTDT_DELETED:
                /* Do something appropriate */
                break;
    }
    tttk_message_destroy(msg);
    return 0;
}
```

SEE ALSO **tttk**(5), **ttdt_file_quit**(3), **ttdt_file_event**(3), **ttdt_Get_Modified**(3), **ttdt_Save**(3), **ttdt_Revert**(3), **tt_file_join**(3), **tt_free**(3).

NAME

ttdt_file_notice – create and send a standard ToolTalk notice about a file

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_message ttdt_file_notice(Tt_message context,
                            Tttk_op op,
                            Tt_scope scope,
                            const char *pathname,
                            int send_and_destroy);
```

DESCRIPTION

The **ttdt_file_notice**( ) function is used to create (and optionally send) any of the standard file notices: *Created*, *Deleted*, *Moved*, *Reverted*, *Saved*, and *Modified*.

The **ttdt_file_notice**( ) function creates a notice with the specified *op* and *scope*, and sets its file attribute to *pathname*. The function adds an unset argument of **Tt_mode TT_IN** and vtype *File* to the notice, per the Desktop messaging conventions. If *send_and_destroy* is True, **ttdt_file_notice**( ) sends the message and then destroys it; otherwise, it only creates the message.

The *context* argument describes the environment to use. If *context* is not zero, messages created by **ttdt_file_notice**( ) inherit from *context* all contexts whose slotname begins with the characters **ENV_**.

RETURN VALUE

If *send_and_destroy* is False, the **ttdt_file_notice**( ) function returns the created **Tt_message**. If *send_and_destroy* is True, it returns zero; otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_DESKTOP_EINVAL
The *op* argument was **TTDT_MOVED** and *send_and_destroy* was True.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OVERFLOW
The ToolTalk service has more active messages than it can handle. (The maximum number of active messages is implementation specific, but is at least 2000.)

TT_ERR_POINTER
The *pathname* argument was **NULL** or was a ToolTalk error pointer.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

APPLICATION USAGE

The **ttdt_file_event**( ) function is a higher-level interface than **ttdt_file_notice**( ), and is the preferred way to send all but the *Moved* notice.

SEE ALSO

tttk(5), **ttdt_file_event**(3).

NAME

ttdt_file_quit – unregister interest in ToolTalk events about a file

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status ttdt_file_quit(Tt_pattern *patterns,
                         int quit);
```

DESCRIPTION

The **ttdt_file_quit**( ) function is used to unregister interest in the *pathname* that was passed to **ttdt_file_join**(3) when *patterns* was created. The **ttdt_file_quit**( ) function destroys *patterns* and sets the default file to **NULL**.

If *quit* is True, **ttdt_file_quit**( ) calls **tt_file_quit**(3) with a *pathname* argument; otherwise, it returns without quitting.

RETURN VALUE

Upon successful completion, the **ttdt_file_quit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The *patterns* argument was **NULL** or otherwise invalid.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

SEE ALSO

tttk(5), **ttdt_file_join**(3), **tt_default_file**(3), **tt_file_quit**(3).

NAME

ttdt_file_request – create and send a standard ToolTalk request about a file

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_message ttdt_file_request(Tt_message context,
                             Tttk_op op,
                             Tt_scope scope,
                             const char *pathname,
                             Ttdt_file_cb cb,
                             void *client_data,
                             int send_and_destroy);
```

DESCRIPTION

The **ttdt_file_request**( ) function is used to create (and optionally send) any of the standard Desktop file requests such as *Get_Modified*, *Save*, and *Revert*.

The **ttdt_file_request**( ) function creates a request with the specified *op* and *scope*, and sets its file attribute to *pathname*. The function adds an unset argument of **Tt_mode TT_IN** and vtype *File* to the request, per the Desktop messaging conventions. If *op* is **TTDT_GET_MODIFIED**, **ttdt_file_request**( ) also adds an unset **TT_OUT** argument of vtype *Boolean* to the request. The **ttdt_file_request**( ) function installs *cb* as a message callback for the created request, and ensures that *client_data* will be passed into the callback. (The **Ttdt_file_cb** callback is described under **ttdt_file_join**(3)). If *send* is True, **ttdt_file_request**( ) sends the request before returning the handle to it; otherwise, it only creates the request.

The *context* argument describes the environment to use. If *context* is not zero, messages created by **ttdt_file_request**( ) inherit from *context* all contexts whose slotname begins with the characters **ENV_**.

RETURN VALUE

Upon successful completion, the **ttdt_file_request**( ) function returns the created **Tt_message**; otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_ERR_DBAVAIL
The ToolTalk service could not access the ToolTalk database needed for this operation.

TT_ERR_DBEXIST
The ToolTalk service could not access the specified ToolTalk database in the expected place.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OVERFLOW
The ToolTalk service has more active messages than it can handle. (The

maximum number of active messages is implementation specific, but is at least 2000.)

TT_ERR_POINTER
The *pathname* argument was **NULL** or was a ToolTalk error pointer.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

APPLICATION USAGE

The **ttdt_file_request**( ) function is a lower-level interface than **ttdt_Get_Modified**( ), **ttdt_Save**( ), and **ttdt_Revert**( ), since the latter functions create and send the request and then block on its reply.

SEE ALSO

tttk(5), **ttdt_Get_Modified**(3), **ttdt_Save**(3), **ttdt_Revert**(3), **ttdt_file_join**(3).

NAME

ttdt_message_accept – accept a contract to handle a ToolTalk request

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_pattern *ttdt_message_accept(Tt_message contract,
                                Ttdt_contract_cb cb,
                                void *clientdata,
                                Widget shell,
                                int accept,
                                int sendStatus);
```

DESCRIPTION

The **ttdt_message_accept**( ) function registers in the default session for **TT_HANDLER**-addressed requests:

(1) *Get_Geometry*, *Set_Geometry*, *Get_Iconified*, *Set_Iconified*, *Get_Mapped*, *Set_Mapped*, *Raise*, *Lower*, *Get_XInfo*, *Set_XInfo*

(2) *Pause*, *Resume*

(3) *Quit*, *Get_Status*

If the *shell* argument is not **NULL**, the ToolTalk service handles messages in (1) transparently; otherwise, it treats them like messages in (3).

If *shell* is non-**NULL** and *cb* is **NULL**, then the ToolTalk service handles messages in (2) transparently by passing *shell* and the appropriate boolean value to **XtSetSensitive**(3X). If *cb* is **NULL**, then the ToolTalk service treats messages in (2) like (3).

If *cb* is not **NULL**, **ttdt_message_accept**( ) passes messages in (3) to the *cb* callback; otherwise it fails with **TT_DESKTOP_ENOTSUP**.

If *accept* is True, **ttdt_message_accept**( ) calls **tt_message_accept**(3) with a *contract* argument. If *contract* has a returned value from **tt_message_status**( ) of **TT_WRN_START_MESSAGE**, it is the message that caused the tool to be started. The tool should join any scopes it wants to serve before accepting *contract*, so that it will receive any other messages already dispatched to its ptype. Otherwise, those messages will cause other instances of the ptype to be started. If that is in fact desired (for example, because the tool can only service one message at a time), then the tool should undeclare its ptype while it is busy.

If *sendStatus* is True, **ttdt_message_accept**( ) sends a *Status* notice to the requester, using the arguments (if any) passed to **ttdt_open**( ).

RETURN VALUE

Upon successful completion, the **ttdt_message_accept**( ) function returns a null-terminated array of **Tt_pattern**, and associates this array with *contract*; otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_UNIMP
The **ttsession**(1) for the default session is of a version that does not support **tt_message_accept**( ). If *contract* is a **TT_WRN_START_MESSAGE**, messages to the tool's ptype will remain blocked until *contract* is rejected, replied to, or failed.

APPLICATION USAGE

The **ttdt_message_accept**( ) function is what a tool calls when it wants to accept responsibility for handling (that is, failing or rejecting) a request.

If *contract* is destroyed by **tttk_message_destroy**(3), then the patterns will also be destroyed. Otherwise, the caller is responsible for iterating over the array and destroying each pattern.

EXAMPLES

See **ttdt_session_join**(3) for an example of a **Ttdt_contract_cb** callback that can be used with **ttdt_message_accept**( ).

SEE ALSO

tttk(5), **ttdt_open**(3), **ttmedia_ptype_declare**(3), **tt_ptype_declare**(3), **ttdt_session_join**(3), **ttdt_file_join**(3), **tt_ptype_undeclare**(3), **tt_ptype_undeclare**(3), **XtSetSensitive**(3X).

NAME

ttdt_open – create a ToolTalk communication endpoint

SYNOPSIS

```
#include <Tt/tttk.h>

char *ttdt_open(int *ttfd,
                const char *toolname,
                const char *vendor,
                const char *version,
                int sendStarted);
```

DESCRIPTION

The **ttdt_open**( ) function calls **tt_open**(3) and **tt_fd**(3). It associates *toolname*, *vendor* and *version* with the created procid, and initializes the new procid's default contexts from the process environment. If *sendStarted* is True, **ttdt_open**( ) sends a *Started* notice.

RETURN VALUE

Upon successful completion, the **ttdt_open**( ) function returns the created procid in a string that can be freed with **tt_free**( ); otherwise, the **ttdt_open**( ) function may return any of the errors returned by **tt_open**(3) and **tt_fd**(3).

No errors are propagated if sending the *Started* notice fails.

SEE ALSO

tttk(5), **ttdt_close**(3), **tt_open**(3), **tt_fd**(3), **tt_free**(3).

NAME

ttdt_sender_imprint_on – act like a child of the specified tool

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status ttdt_sender_imprint_on(const char *handler,
                                 Tt_message contract,
                                 char **display,
                                 int *width,
                                 int *height,
                                 int *xoffset,
                                 int *yoffset,
                                 XtAppContext app2run,
                                 int ms_timeout);
```

DESCRIPTION

The **ttdt_sender_imprint_on**( ) function is used to make the calling tool act equivalently to a child of another specified tool. The calling tool adopts the other tool's X11 display, locale, and current working directory. It also learns the other tool's X11 geometry, so that it may position itself appropriately.

If the *handler* argument is non-**NULL**, the requests are addressed to that procid using **TT_HANDLER**. If *handler* is **NULL** and the *contract* argument is non-**NULL**, the requests are addressed to the **tt_message_sender**(3) of the contract, using **TT_HANDLER**.

The *contract* argument is passed to **tttk_message_create**(3) as the *context* argument.

If the *display* argument is not **NULL**, **ttdt_sender_imprint_on**( ) returns the other tool's display in **display*. If *display* is **NULL**, **ttdt_sender_imprint_on**( ) sets the *DISPLAY* environment variable to the other tool's display.

If each of the *width*, *height*, *xoffset*, and *yoffset* arguments are **NULL**, then **ttdt_sender_imprint_on**( ) does not send the other tool a *Get_Geometry* request.

The **ttdt_sender_imprint_on**( ) function passes the *app2run* and *ms_timeout* arguments to **tttk_block_while**(3), blocking on the replies to the requests it sends.

If the *display* argument is not **NULL**, **ttdt_sender_imprint_on**( ) sets **display* to a string that can be freed with **tt_free**( ).

If for some reason no width or height is returned by the other tool, **ttdt_sender_imprint_on**( ) sets **width* or **height* to –1. If no positional information is returned, **ttdt_sender_imprint_on**( ) sets **xoffset* and **yoffset* to {INT_MAX}.

RETURN VALUE

Upon successful completion, the **ttdt_sender_imprint_on**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_DESKTOP_ETIMEDOUT
One or more of the sent requests did not complete within *ms_timeout* milliseconds.

TT_ERR_NOMEM

There is insufficient memory available to perform the function.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OVERFLOW
The ToolTalk service has more active messages than it can handle. (The maximum number of active messages is implementation specific, but is at least 2000.)

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

APPLICATION USAGE

If both the *handler* and *contract* are zero, the requests are addressed to no tool in particular, using **TT_PROCEDURE**; this is not recommended.

SEE ALSO

tttk(5), **tt_free**(3), **tt_message_sender**(3), **tttk_block_while**(3), **tttk_message_create**(3).

NAME

ttdt_session_join – join a ToolTalk session

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_pattern *ttdt_session_join(const char *sessid,
                              Ttdt_contract_cb cb,
                              Widget shell,
                              void *clientdata,
                              int join);
```

DESCRIPTION

The **ttdt_session_join**( ) function joins the session *sessid*, registering patterns and default callbacks for many standard Desktop message interfaces. If *sessid* is **NULL**, the default session is joined.

The **ttdt_session_join**( ) function registers for the following **TT_HANDLER**-addressed requests:

(1) *Get_Environment*, *Set_Environment*, *Get_Locale*, *Set_Locale*, *Get_Situation*, *Set_Situation*, *Signal*, *Get_Sysinfo*

(2) *Get_Geometry*, *Set_Geometry*, *Get_Iconified*, *Set_Iconified*, *Get_Mapped*, *Set_Mapped*, *Raise*, *Lower*, *Get_XInfo*, *Set_XInfo*

(3) *Pause*, *Resume*, *Quit*

(4) *Get_Status*, *Do_Command*

If *join* is True, **ttdt_session_join**( ) actually joins the indicated session.

The ToolTalk service handles messages in (1) transparently.

If *shell* is non-**NULL**, then it is expected to be a realized *mappedWhenManaged applicationShellWidget*, and the ToolTalk service handles messages in (2) transparently. (If *shell* is merely a realized widget, then the ToolTalk service handles only the *Get_XInfo* request, and **ttdt_session_join**( ) fails the rest of (2) with **TT_DESKTOP_ENOTSUP**.) If *shell* is **NULL**, then the ToolTalk service treats messages in (2) equivalently to those in (4).

If *shell* is non-**NULL** and *cb* is **NULL**, then the ToolTalk service handles messages in (3) transparently as follows; otherwise, it treats them as equivalent to those in (4). The *Quit* request results in a **WM_DELETE_WINDOW** event on *shell* if the *silent* and *force* arguments of the *Quit* request are both False. In other words, if *shell* is supplied without a *cb*, then a *Quit* request may imply that the user quit the application's top-level window using the window manager. *Pause* and *Resume* requests result in the ToolTalk service passing *shell* and the appropriate boolean value to **XtSetSensitive**(3X).

If *cb* is not **NULL**, the ToolTalk service passes messages in (4) to *cb*; otherwise, **ttdt_session_join**( ) fails with **TT_DESKTOP_ENOTSUP**.

The **Ttdt_contract_cb** argument is a callback defined as:

```
Tt_message (*Ttdt_contract_cb)(Tt_message msg,
                               void *clientdata,
                               Tt_message contract);
```

The *msg* argument is a message in **Tt_state TT_SENT**. If *msg* is a **TT_REQUEST**, the client program becomes responsible for either failing, rejecting or replying to *msg*. After doing so, the client program may dispose of *msg* with **tttk_message_destroy**( ). The *clientdata* argument is the *clientdata* passed to **ttdt_session_join**( ) or **ttdt_message_accept**(3). The *contract* argument is the *contract* passed to **ttdt_message_accept**( ). For callbacks installed by **ttdt_session_join**( ), *contract* is always zero.

RETURN VALUE

Upon successful completion, the **ttdt_session_join**( ) function returns a null-terminated array of **Tt_pattern**; otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_ERR_NOMEM
: There is insufficient memory available to perform the function.

TT_ERR_NOMP
: The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
: The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_PROCID
: The specified process identifier is out of date or invalid.

TT_ERR_SESSION
: The specified ToolTalk session is out of date or invalid.

APPLICATION USAGE

The null-terminated array of **Tt_pattern** returned by **ttdt_session_join**( ) should be destroyed by passing the array to **ttdt_file_quit**(3).

The ToolTalk service will reply to the *Quit* request before generating the **WM_DELETE_WINDOW** event. If the application catches and cancels this event, then the sender of the *Quit* request will be misled into thinking the application actually quit. Applications that can cancel **WM_DELETE_WINDOW** should install a real **Ttdt_contract_cb**.

The ToolTalk service handles the *Pause* and *Resume* requests by setting the sensitivity of *widget*. If *widget* is the parent of any top-level pop-up shells, **XtSetSensitive**(3X) will not affect them. Applications that can have such pop-ups should install a real **Ttdt_contract_cb**.

A **Ttdt_contract_cb** should return zero if it processes *msg* successfully, or a **tt_error_pointer**( ) cast to **Tt_message** if processing results in an error. It should return the *msg* if it does not consume it. If *msg* is returned, then the ToolTalk service passes **TT_CALLBACK_CONTINUE** down the call stack, so that *msg* will be offered to other callbacks or (more likely) be returned from **tt_message_receive**(3). Applications will rarely want *msg* to get processed by other callbacks or in the main event loop.

EXAMPLES

This is the typical algorithm of a **Ttdt_contract_cb** for an application that handles *Pause*, *Resume* or *Quit* requests for itself, but lets the ToolTalk service handle the X11-related requests listed in (2). Since this example callback deals with the case when *contract* is not zero, it can also be used as the **Ttdt_contract_cb** passed to **ttdt_message_accept**( ).

```
Tt_message myContractCB(Tt_message     msg,
        void        *clientdata,
        Tt_message     contract)
{
        char *opString = tt_message_op(msg);
        Tttk_op op = tttk_string_op(opString);
        tt_free(opString);
        int silent = 0;
        int force  = 0;
        Boolean cancel = False;
        Boolean sensitive = True;
        char *status, command;
        switch(op) {
          case TTDT_QUIT:
                tt_message_arg_ival(msg, 0, &silent);
                tt_message_arg_ival(msg, 1, &force);
                if (contract == 0) {
                        /* Quit entire application */
                        cancel = ! myQuitWholeApp(silent, force);
                } else {
                        /* Quit just the specified request being worked on */
                        cancel = ! myCancelThisRequest(contract, silent, force);
                }
                if (cancel) {
                        /* User canceled Quit; fail the Quit request */
                        tttk_message_fail(msg, TT_DESKTOP_ECANCELED, 0, 1);
                } else {
                        tt_message_reply(msg);
                        tttk_message_destroy(msg);
                }
                return 0;
          case TTDT_PAUSE:
                sensitive = False;
          case TTDT_RESUME:
                if (contract == 0) {
                        int already = 1;
                        if (XtIsSensitive(myTopShell) != sensitive) {
                                already = 0;
                                XtSetSensitive(myTopShell, sensitive);
                        }
                        if (already) {
```

```
                                tt_message_status_set(msg, TT_DESKTOP_EALREADY);
                        }
                } else {
                        if (XtIsSensitive(thisShell) == sensitive) {
                                tt_message_status_set(msg, TT_DESKTOP_EALREADY);
                        } else {
                                XtSetSensitive(thisShell, sensitive);
                        }
                }
                tt_message_reply(msg);
                tttk_message_destroy(msg);
                return 0;
        case TTDT_GET_STATUS:
                if (contract == 0) {
                        status = "Message about status of entire app";
                } else {
                        status = "Message about status of this request";
                }
                tt_message_arg_val_set(msg, 0, status);
                tt_message_reply(msg);
                tttk_message_destroy(msg);
                return 0;
        case TTDT_DO_COMMAND:
                if (! haveExtensionLanguage) {
                        tttk_message_fail(msg, TT_DESKTOP_ENOTSUP, 0, 1);
                        return 0;
                }
                command = tt_message_arg_val(msg, 0);
                result = myEval(command);
                tt_free(command);
                tt_message_status_set(msg, result);
                if (tt_is_err(result)) {
                        tttk_message_fail(msg, result, 0, 1);
                } else {
                        tt_message_reply(msg);
                        tttk_message_destroy(msg);
                }
                return 0;
        }
        /* Unrecognized message; do not consume it */
        return msg;
}
```

SEE ALSO **tttk**(5), **ttdt_session_quit**(3), **tt_session_join**(3), **XtSetSensitive**(3X).

NAME

ttdt_session_quit – quit a ToolTalk session

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status ttdt_session_quit(const char *sessid,
                            Tt_pattern *sess_pats,
                            int quit);
```

DESCRIPTION

The **ttdt_session_quit**( ) function destroys the patterns in *sess_pats*. If *quit* is True, it quits the session *sessid*, or the default session if *sessid* is **NULL**.

RETURN VALUE

Upon successful completion, the **ttdt_session_quit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

TT_ERR_SESSION
The specified ToolTalk session is out of date or invalid.

SEE ALSO

tttk(5), **ttdt_session_join**(3), **tt_session_quit**(3).

NAME

ttdt_subcontract_manage – manage an outstanding request

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_pattern *ttdt_subcontract_manage(Tt_message subcontract,
                                    Ttdt_contract_cb cb,
                                    Widget shell,
                                    void *clientdata);
```

DESCRIPTION

The **ttdt_subcontract_manage**( ) function allows a requester to manage the standard Desktop interactions with the tool that is handling the request. The **ttdt_subcontract_manage**( ) function registers in the default session for **TT_HANDLER**-addressed requests *Get_Geometry* and *Get_XInfo*, and *Status* notices.

If *shell* is not **NULL**, the ToolTalk service handles the *Get_Geometry* and *Get_XInfo*, and *Status* notices transparently; otherwise, it passes them to *cb*.

See **ttdt_session_join**(3) for a description of a **Ttdt_contract_cb** callback.

If *subcontract* is destroyed by **tttk_message_destroy**( ), then the patterns will also be destroyed; otherwise, the caller is responsible for iterating over the array and destroying each pattern.

RETURN VALUE

Upon successful completion, the **ttdt_subcontract_manage**( ) function returns a null-terminated array of **Tt_pattern**, and associates this array with *subcontract*; otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_DESKTOP_EINVAL
Both the *shell* and *cb* arguments were **NULL**.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The *subcontract* argument was not a valid **Tt_message**.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

SEE ALSO

tttk(5), **ttdt_session_join**(3), **tttk_message_destroy**(3).

NAME

ttmedia_Deposit – send a Deposit request to checkpoint a document

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status ttmedia_Deposit(Tt_message load_contract,
                          const char *buffer_id,
                          const char *media_type,
                          const unsigned char *new_contents,
                          int new_len,
                          const char *pathname,
                          XtAppContext app2run,
                          int ms_timeout);
```

DESCRIPTION

The **ttmedia_Deposit**( ) function is used to perform a checkpoint save on a document that was the subject of a Media Exchange *load_contract* request such as *Edit*, *Compose*, or *Open*. To carry out a checkpoint save, the editor must send the new document contents back to the sender of *load_contract*.

The **ttmedia_Deposit**( ) function creates and sends a *Deposit* request and returns the success or failure of that request. The *load_contract* argument is the request that caused this editor to load the document. The *buffer_id* argument is the identifier of the buffer this editor created if the document was loaded via an *Open* request. If *buffer_id* is **NULL**, the the ToolTalk service gives the *Deposit* request a **Tt_address** of **TT_HANDLER** and sends it directly to the **tt_message_sender**( ) of *load_contract*; otherwise, the the ToolTalk service will address it as a **TT_PROCEDURE** and insert *buffer_id* into the request to match the pattern registered by the sender of the *load_contract*.

The **ttmedia_Deposit**( ) function uses the *media_type* argument as the vtype of the contents argument of the sent request, and *new_contents* and *new_len* as its value. The latter two must be zero if *pathname* is used to name a temporary file into which the editor will place the checkpointed document. The editor is free to remove the temporary file after the reply to the *Deposit* request is received; that is, after **ttmedia_Deposit**( ) has returned.

After the request is sent, **ttmedia_Deposit**( ) passes *app2run* and *ms_timeout* to **tttk_block_while**(3) to wait for the reply.

RETURN VALUE

Upon successful completion, the **ttmedia_Deposit**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_DESKTOP_ETIMEDOUT
No reply was received within *ms_timeout* milliseconds.

TT_ERR_NOMEM
There is insufficient memory available to perform the function.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OVERFLOW
The ToolTalk service has more active messages than it can handle. (The maximum number of active messages is implementation specific, but is at least 2000.)

TT_ERR_POINTER
The *pathname* argument was **NULL** or was a ToolTalk error pointer.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

SEE ALSO

tttk(5), **ttmedia_load**(3), **ttmedia_load_reply**(3), **ttmedia_ptype_declare**(3), **ttdt_Save**(3), **tttk_block_while**(3).

NAME

ttmedia_load – send a Display, Edit or Compose request

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_message ttmedia_load(Tt_message context,
                        Ttmedia_load_msg_cb cb,
                        void *clientdata,
                        Tttk_op op,
                        const char *media_type,
                        const unsigned char *contents,
                        int len,
                        const char *file,
                        const char *docname,
                        int send);
```

DESCRIPTION

The **ttmedia_load**( ) function is used to create and optionally send a Media Exchange request to display, edit or compose a document.

The *cb* argument will be passed *clientdata* when the reply is received, or when intermediate versions of the document are checkpointed through *Deposit* requests. The *op* argument must be one of **TTME_DISPLAY**, **TTME_EDIT** or **TTME_COMPOSE**. The *media_type* argument names the data format of the document, and is usually the primary determinant of which application will be chosen to handle the request. The *contents* and *len* arguments specify the document; if they are **NULL** and zero, respectively, and *file* is not **NULL**, then the document is assumed to be contained in *file*. If *docname* is not **NULL**, then **ttmedia_load**( ) uses it as the title of the document. If *send* is True, the message is sent before being returned.

The *context* argument describes the environment to use. If *context* is not zero, messages created by **ttmedia_load**( ) inherit from *context* all contexts whose slotname begins with the characters **ENV_**.

The **Ttmedia_load_msg_cb** argument is a callback defined as:

```
Tt_message (*Ttmedia_load_msg_cb)(Tt_message msg,
                                  void *clientdata),
                                  Tttk_op op,
                                  unsigned char *contents,
                                  int len,
                                  char *file);
```

The *msg* argument is the reply to the load request, or a *Deposit* request with a *messageID* argument naming the identifier (see **tt_message_id**(3)) of the load request. In the latter case, the client program becomes responsible for either failing or replying to the request. In either case, *msg* should be destroyed after being processed.

The *op* argument is the op of *msg*. It must be either **TTME_DEPOSIT** or the *op* passed to **ttmedia_load**(3).

The *contents*, *len* and *file* arguments represent the contents of the arriving document. If *len* is zero, then the document is contained in *file*. If *contents* or *file* are non-**NULL**, they can be freed using **tt_free**( ).

The *clientdata* argument is the *clientdata* passed to **ttmedia_load**( ).

RETURN VALUE

Upon successful completion, the **ttmedia_load**( ) function returns the request it was asked to build; otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_ERR_NOMEM
: There is insufficient memory available to perform the function.

TT_ERR_NOMP
: The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_OVERFLOW
: The ToolTalk service has more active messages than it can handle. (The maximum number of active messages is implementation specific, but is at least 2000.)

TT_ERR_PROCID
: The specified process identifier is out of date or invalid.

APPLICATION USAGE

After the request created by **ttmedia_load**( ) is sent, the application will probably want to use **ttdt_subcontract_manage**( ) immediately afterwards to manage the standard interactions with the handler of the request.

A **Ttmedia_load_msg_cb** callback should return **NULL** if it processes *msg* successfully, or a **tt_error_pointer**( ) cast to **Tt_message** if processing results in an error. It should return the *msg* if it does not consume it, in which case the ToolTalk service will pass **TT_CALLBACK_CONTINUE** down the call stack, so that *msg* will be offered to other callbacks or (more likely) be returned from **tt_message_receive**(3). Applications will rarely want *msg* to get processed by other callbacks or in the main event loop.

EXAMPLES

This is the typical algorithm of a **Ttmedia_load_msg_cb**:

```
Tt_message
myLoadMsgCB(Tt_message  msg,
    void        *clientData,
    Tttk_op       op,
    unsigned char  *contents,
    int          len,
    char         *file)
{
    if (len > 0) {
        /* Replace data with len bytes in contents */
    } else if (file != 0) {
        /* Replace data with data read from file */
```

```
        }
        if (op == TTME_DEPOSIT) {
            tt_message_reply(msg);
        }
        tttk_message_destroy(msg);
        return 0;
    }
```

SEE ALSO

tttk(5), **ttmedia_load_reply**(3), **ttmedia_ptype_declare**(3), **ttmedia_Deposit**(3), **tt_free**(3), **tt_message_receive**(3).

NAME

ttmedia_load_reply – reply to a Display, Edit or Compose request

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_message ttmedia_load_reply(Tt_message contract,
                              const unsigned char *new_contents,
                              int new_len,
                              int reply_and_destroy);
```

DESCRIPTION

The **ttmedia_load_reply**( ) function is used to reply to a Media Exchange request to display, edit or compose a document. The editor working on the request usually calls **ttmedia_load_reply**( ) when the user has indicated in some way that he or she is finished viewing or modifying the document.

If *new_contents* and *new_len* are non-**NULL** and non-zero, respectively, **ttmedia_load_reply**( ) uses their values to set the new contents of the document back in the appropriate output argument of *contract*. If *reply_and_destroy* is True, **ttmedia_load_reply**( ) replies to *contract* and then destroys it.

RETURN VALUE

Upon successful completion, the **ttmedia_load_reply**( ) function returns the created **Tt_message**; otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NOTHANDLER
This application is not the handler for this message.

TT_ERR_NUM
The integer value passed was invalid (out of range).

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

APPLICATION USAGE

If *contract* is a *Display* request, then *new_contents* and *new_len* should be zero.

SEE ALSO

tttk(5), **ttmedia_load**(3), **ttmedia_ptype_declare**(3), **ttmedia_Deposit**(3).

NAME

ttmedia_ptype_declare – declare the ptype of a Media Exchange media editor

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status ttmedia_ptype_declare(const char *ptype,
                                int base_opnum,
                                Ttmedia_load_pat_cb cb,
                                void *clientdata,
                                int declare);
```

DESCRIPTION

The **ttmedia_ptype_declare**( ) function is used to initialize an editor that implements the Media Exchange message interface for a particular media type. The **ttmedia_ptype_declare**( ) function notifies the ToolTalk service that the *cb* callback is to be called when the editor is asked to edit a document of the kind supported by ptype. The **ttmedia_ptype_declare**( ) function installs an implementation-specific *opnum* callback on a series of signatures that ptype is assumed to contain. These signatures are listed below, with their corresponding opnum offsets. Opnums in ptype for these signatures start at *base_opnum*, which must be zero or a multiple of 1000. The implementation-specific opnum callback will pass *clientdata* to *cb* when a request is received that matches one of these signatures.

If *declare* is True, **ttmedia_ptype_declare**( ) calls **tt_ptype_declare**( ) with the ptype argument. If ptype implements Media Exchange for several different media types, then **ttmedia_ptype_declare**( ) can be called multiple times, with a different *base_opnum* each time, and with *declare* being True only once.

The **Ttmedia_load_pat_cb** argument is a callback defined as:

```
Tt_message (*Ttmedia_load_pat_cb)(Tt_message msg,
                                  void *clientdata,
                                  Tttk_op op,
                                  Tt_status diagnosis,
                                  unsigned char *contents,
                                  int len,
                                  char *file,
                                  char *docname);
```

The *msg* argument is a **TT_REQUEST** in **Tt_state TT_SENT**. The client program becomes responsible for either failing, rejecting or replying to it. This can either be done inside the callback, or the message can be saved and dismissed later (that is, after the callback returns). Usually, the callback will either immediately reject/fail the request, or it will start processing the request, perhaps by associating it with a new window. When the request is finally dismissed, it should be destroyed, for example, using **tt_message_destroy**( ).

If the callback knows it will handle the request (either fail or reply to it, but not reject it), then it should call **ttdt_message_accept**(3). But if the return value of **tt_message_status**(3) of *msg* is **TT_WRN_START_MESSAGE**, then the callback should

probably do **ttdt_session_join**(), and perhaps a **ttdt_file_join**(), before accepting the message. The *op* argument is the op of the incoming request, one of **TTME_COMPOSE**, **TTME_EDIT** or **TTME_DISPLAY**. The *diagnosis* argument is the recommended error code; if the ToolTalk service detects a problem with the request (for example, **TT_DESKTOP_ENODATA**), then it passes in the error code that it recommends the request should be failed with. If *diagnosis* was not **TT_OK** and the **Ttmedia_load_pat_cb** returns *msg*, then the ToolTalk service will fail and destroy *msg*.

The ToolTalk service does not simply fail the request transparently, because the request may be the reason that the client process was started by ToolTalk in the first place. So if *diagnosis* is not **TT_OK** and the **tt_message_status**() of *msg* is **TT_WRN_START_MESSAGE**, then many applications will decide that they have no reason to continue running. If such an application chooses to exit in the callback, then it should first dismiss the request. Otherwise, it can set some global flag, return *msg* (thus allowing the ToolTalk service to dismiss the message), and then have **main**() check the flag and exit before even entering the event loop. (Exiting without dismissing the request would fail it with status **TT_ERR_PROCID**, instead of with *diagnostic*.)

The *contents*, *len*, and *file* arguments represent the contents of the arriving document. If *len* is zero, then the document is contained in *file*. If *contents* or *file* are non-**NULL**, they can be freed using **tt_free**().

The *docname* argument is the name of the document, if any. The *clientdata* argument is the *clientdata* passed to **ttmedia_ptype_declare**().

A **Ttmedia_load_pat_cb** should return zero if it processes *msg* successfully, or a **tt_error_pointer**() cast to **Tt_message** if processing results in an error. It should return the *msg* if it does not consume it. If *diagnosis* is not **TT_OK** and *msg* is returned, then the ToolTalk service will consume (namely, fail and destroy) it. If *diagnosis* is **TT_OK** and *msg* is returned, then the ToolTalk service will pass **TT_CALLBACK_CONTINUE** down the call stack, so that *msg* will be offered to other callbacks or (more likely) be returned from **tt_message_receive**(3). Applications will rarely want *msg* to get processed by other callbacks or in the main event loop.

RETURN VALUE

Upon successful completion, the **ttmedia_ptype_declare**() function returns the status of the operation. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_ERR_PROCID
The specified process identifier is out of date or invalid.

TT_ERR_PTYPE

The specified process type is not the name of an installed process type.

EXAMPLES

This is the typical algorithm of a **Ttmedia_load_pat_cb**:

```
Tt_message
myAcmeSheetLoadCB(
        Tt_message     msg,
        void          *client_data,
        Tttk_op        op,
        Tt_status      diagnosis,
        unsigned char  *contents,
        int            len,
        char          *file,
        char          *docname
)
{
        Tt_status status = TT_OK;
        if (diagnosis != TT_OK) {
                /* toolkit detected an error */
                if (tt_message_status(msg) == TT_WRN_START_MESSAGE) {
                        /*
                         * Error is in start message!  We now have no
                         * reason to live, so tell main() to exit().
                         */
                        myAbortCode = 2;
                }
                /* let toolkit handle the error */
                return msg;
        }

        /* We should only operate on files we own. */
        if (tt_message_uid() != getuid() || tt_message_gid() != getgid()) {
                tttk_message_fail(msg, TT_DESKTOP_EACCES, 0, 1);
                /* tt_free as appropriate... */
                return 0;
        }

        if ((op == TTME_COMPOSE)&&(file == 0)) {
                /* open empty new buffer */
        } else if (len > 0) {
                /* load contents into new buffer */
        } else if (file != 0) {
                if (ttdt_Get_Modified(msg, file, TT_BOTH, myCntxt, 5000)) {
                        switch(myUserChoice("Save, Revert, Ignore?")) {
                          case 0:
                                ttdt_Save(msg, file, TT_BOTH, myCntxt, 5000);
```

```
                                        break;
                                case 1:
                                        ttdt_Revert(msg, file, TT_BOTH, myCntxt, 5000);
                                        break;
                                }
                        }
                        /* load file into new buffer */
                } else {
                        tttk_message_fail(msg, TT_DESKTOP_ENODATA, 0, 1);
                        tt_free(contents); tt_free(file); tt_free(docname);
                        return 0;
                }
                int w, h, x, y = INT_MAX;
                ttdt_sender_imprint_on(0, msg, 0, &w, &h, &x, &y, myCntxt, 5000);
                positionMyWindowRelativeTo(w, h, x, y);
                if (maxBuffersAreNowOpen) {
                        /* Un-volunteer to handle future requests until less busy */
                        tt_ptype_undeclare("Acme_Calc");
                }
                if (tt_message_status(msg) == TT_WRN_START_MESSAGE) {
                        /*
                         * Join session before accepting start message,
                         * to prevent unnecessary starts of our ptype
                         */
                        ttdt_session_join(0, myContractCB, myShell, 0, 1);
                }
                ttdt_message_accept(msg, myContractCB, myShell, 0, 1, 1);
                tt_free(contents); tt_free(file); tt_free(docname);
                return 0;
}
```

This is the signature layout that ptype should conform to:

```
ptype Acme_Calc {
   start "acalc";
   handle:
        /*
         * Display Acme_Sheet
         * Include in tool's ptype if tool can display a document.
         */
        session Display( in   Acme_Sheet  contents    ) => start opnum = 1;
        session Display( in   Acme_Sheet  contents,
                         in   messageID   counterfoil  ) => start opnum = 2;
        session Display( in   Acme_Sheet  contents,
                         in   title       docName     ) => start opnum = 3;
        session Display( in   Acme_Sheet  contents,
                         in   messageID   counterfoil,
```

```
                        in   title     docName     ) => start opnum = 4;
        /*
         * Edit Acme_Sheet
         * Include in tool's ptype if tool can edit a document.
         */
        session Edit(   inout Acme_Sheet  contents    ) => start opnum = 101;
        session Edit(   inout Acme_Sheet  contents,
                        in   messageID   counterfoil  ) => start opnum = 102;
        session Edit(   inout Acme_Sheet  contents,
                        in   title     docName     ) => start opnum = 103;
        session Edit(   inout Acme_Sheet  contents,
                        in   messageID   counterfoil,
                        in   title     docName     ) => start opnum = 104;
        /*
         * Compose Acme_Sheet
         * Include in tool's ptype if tool can compose a document from scratch.
         */
        session Edit(   out   Acme_Sheet  contents    ) => start opnum = 201;
        session Edit(   out   Acme_Sheet  contents,
                        in   messageID   counterfoil  ) => start opnum = 202;
        session Edit(   out   Acme_Sheet  contents,
                        in   title     docName     ) => start opnum = 203;
        session Edit(   out   Acme_Sheet  contents,
                        in   messageID   counterfoil,
                        in   title     docName     ) => start opnum = 204;
        /*
         * Mail Acme_Sheet
         * Include in tool's ptype if tool can mail a document.
         */
        session Mail(   in    Acme_Sheet  contents    ) => start opnum = 301;
        session Mail(   inout Acme_Sheet  contents    ) => start opnum = 311;
        session Mail(   inout Acme_Sheet  contents,
                        in   title     docName     ) => start opnum = 313;
        session Mail(   out   Acme_Sheet  contents    ) => start opnum = 321;
        session Mail(   out   Acme_Sheet  contents,
                        in   messageID   counterfoil  ) => start opnum = 323;
};
```

SEE ALSO **tttk**(5), **tt_ptype_declare**(3), **tt_ptype_undeclare**(3), **ttdt_message_accept**(3), **ttdt_session_join**(3), **ttdt_file_join**(3), **tt_free**(3), **tt_message_receive**(3).

NAME

tttk_Xt_input_handler – Process ToolTalk events for Xt clients

SYNOPSIS

```
#include <Tt/tttk.h>

void tttk_Xt_input_handler(XtPointer procid,
                           int *source,
                           XtInputId *id);
```

DESCRIPTION

If *procid* is not **NULL**, **tttk_Xt_input_handler**( ) passes it to **tt_default_procid_set**(3). The **tttk_Xt_input_handler**( ) function then calls **tt_message_receive**(3), which retrieves the next message available, if any, for the default procid. If **tt_message_receive**(3) returns **TT_ERR_NOMP**, then **tttk_Xt_input_handler**( ) closes the default procid with **ttdt_close**(3), and removes the input source **id* with **XtRemoveInput**(3X) if *id* is not zero. If a message is available and **tt_message_receive**(3) returns it (indicating it was not consumed by any message or pattern callback), then the ToolTalk service passes the message to **tttk_message_abandon**(3).

RETURN VALUE

The **tttk_Xt_input_handler**( ) function returns no value.

APPLICATION USAGE

The application should use **tttk_Xt_input_handler**( ) as its Xt input handler unless some messages are expected not to be consumed by callbacks. (The only messages that absolutely cannot be intercepted and consumed by callbacks are those that match observe signatures in a ptype or otype.)

EXAMPLES

```
int myTtFd;
char *myProcID;
myProcID = ttdt_open(&myTtFd, "WhizzyCalc", "Acme", "1.0", 1);
/* ... */
/* Process the message that started us, if any */
tttk_Xt_input_handler(myProcID, 0, 0);
/* ... */
XtAppAddInput(myContext, myTtFd, (XtPointer)XtInputReadMask,
                        tttk_Xt_input_handler, myProcID);
```

SEE ALSO

tttk(5), **ttdt_close**(3), **tttk_message_abandon**(3), **tt_default_procid_set**(3), **tt_message_receive**(3), **XtAppAddInput**(3X), **XtRemoveInput**(3X).

NAME

tttk_block_while – block while a counter is greater than zero

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status tttk_block_while(XtAppContext app2run,
                           const int *blocked,
                           int ms_timeout);
```

DESCRIPTION

The **tttk_block_while**( ) function is used to process asynchronous events, such as ToolTalk messages or window system events, while waiting for a condition or timeout.

If *app2run* is not zero, then an event loop is run for that application context, by repeatedly calling **XtAppProcessEvent**(3X) with *ms_timeout* being effected using **XtAppAddTimeOut**(3X). If *app2run* is zero, then the file descriptor (as returned by **tt_fd**(3)) of the default procid is polled (using the **poll**(2) function) and **tttk_Xt_input_handler**(3) is called whenever the file descriptor is active.

If *blocked* is zero, then **tttk_block_while**( ) runs until *ms_timeout* occurs. If *blocked* is non-zero, then the loop is run until either *ms_timeout* occurs, or **blocked* is less than 1.

If *ms_timeout* is zero, **tttk_block_while**( ) checks once for events, processes the first one, and then returns. If *ms_timeout* is negative, no timeout is in effect.

RETURN VALUE

Upon successful completion, the **tttk_block_while**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_DESKTOP_ETIMEDOUT
The timeout occurred within *ms_timeout* milliseconds, or *ms_timeout* was zero and no input was available.

TT_DESKTOP_EINTR
The *app2run* argument was zero, and **poll**(2) was interrupted by a signal.

TT_DESKTOP_EAGAIN
The *app2run* argument was zero, and **poll**(2) returned **EAGAIN**.

If *app2run* is not zero and *ms_timeout* is negative, then **tttk_block_while**( ) will only return when **blocked* is less than 1, with **TT_OK** being returned.

If *app2run* is not zero, *ms_timeout* is negative, and *blocked* is zero, then **tttk_block_while**( ) behaves equivalent to **XtAppMainLoop**(3X), and will never return.

APPLICATION USAGE

If *app2run* is zero, then only messaging events for the default procid will be serviced. Events for other procids will be blocked, as will window system events, so that the graphical user interface of the application will not update itself even, for example, after expose events.

On the other hand, if the application passes its Xt context in as *app2run*, then window system events will continue to be handled, as will message activity for all procids for which an **XtAppAddInput**(3X) has been done. Since the window system event loop is fully

operational in this case, the application should take care to disable any user interface controls that the user should not operate while the application is waiting for **tttk_block_while**( ) to return.

SEE ALSO

tttk(5), **tttk_Xt_input_handler**(3); **poll**(2), **XtAppPending**(3X), **XtAppAddTimeOut**(3X), **XtAppNextEvent**(3X), **XtDispatchEvent**(3X).

NAME

tttk_message_abandon – finalize a message properly

SYNOPSIS

#include <Tt/tttk.h>

Tt_status tttk_message_abandon(Tt_message *msg***);**

DESCRIPTION

The **tttk_message_abandon**( ) function is used by an application when it does not understand a message and wants to get rid of it. The **tttk_message_abandon**( ) function fails or rejects *msg* if appropriate, and then destroys it. The **tttk_message_abandon**( ) will reject or fail the message only if *msg* is a **TT_REQUEST** in **Tt_state TT_SENT**. If it has a **Tt_address** of **TT_HANDLER** or a **tt_message_status**( ) of **TT_WRN_START_MESSAGE**, then it fails the message; otherwise, it rejects it. In either case, **tttk_message_abandon**( ) gives *msg* a message status (see **tt_message_status**(3)) of **TT_DESKTOP_ENOTSUP**.

RETURN VALUE

Upon successful completion, the **tttk_message_abandon**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NOTHANDLER
This application is not the handler for this message.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tttk(5), **tt_message_status**(3), **tttk_message_fail**(3), **tttk_message_reject**(3).

NAME

tttk_message_create – create a message conforming to the CDE conventions

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_message tttk_message_create(Tt_message context,
                               Tt_class the_class,
                               Tt_scope the_scope,
                               const char *handler,
                               const char *op,
                               Tt_message_callback callback);
```

DESCRIPTION

The **tttk_message_create**( ) function creates a message that propagates inherited contexts from one message to another. The **tttk_message_create**( ) function creates a message and copies onto it all the context slots from *context* whose slotname begins with the characters **ENV_**. It gives the created message a **Tt_class** of *the_class* and a **Tt_scope** of *the_scope*. If *handler* is not **NULL**, then **tttk_message_create**( ) addresses the message as a **TT_HANDLER** to that procid; otherwise, it gives the message a **Tt_address** of **TT_PROCEDURE**. It sets the message's op to *op* if *op* is not **NULL**. If *callback* is not **NULL**, **tttk_message_create**( ) adds it to the message as a message callback.

RETURN VALUE

Upon successful completion, the **tttk_message_create**( ) function returns the created **Tt_message ,** which can be modified, sent, and destroyed like any other **Tt_message ;** otherwise, it returns an error pointer. The application can use **tt_ptr_error**(3) to extract one of the following **Tt_status** values from the returned handle:

TT_ERR_NOMEM
: There is insufficient memory available to perform the function.

TT_ERR_NOMP
: The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_PROCID
: The specified process identifier is out of date or invalid.

SEE ALSO

tttk(5), **tt_message_create**(3), **tttk_message_create**(3), **ttdt_file_notice**(3), **ttdt_file_request**(3).

NAME

tttk_message_destroy – destroy a message conforming to the CDE conventions

SYNOPSIS

#include <Tt/tttk.h>

Tt_status tttk_message_destroy(Tt_message *msg***);**

DESCRIPTION

The **tttk_message_destroy**( ) function can be used in place of **tt_message_destroy**(3). It destroys any patterns that may have been stored on *msg* by **ttdt_message_accept**(3) or **ttdt_subcontract_manage**(3). Then it passes *msg* to **tt_message_destroy**(3).

RETURN VALUE

Upon successful completion, the **tttk_message_destroy**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

TT_WRN_STOPPED
The message is not actually destroyed. (A message is not destroyed if it is in a non-final state; for example, a request for which the reply has not been received.)

SEE ALSO

tttk(5), **tt_message_create**(3), **tt_message_destroy**(3), **ttdt_file_notice**(3), **ttdt_file_request**(3).

NAME

tttk_message_fail – fail a message

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status tttk_message_fail(Tt_message msg,
                            Tt_status status,
                            const char *status_string,
                            int destroy);
```

DESCRIPTION

The **tttk_message_fail**( ) function sets the status and status string of the **TT_REQUEST** *msg*, fails *msg*, and then destroys *msg* if *destroy* is True.

RETURN VALUE

Upon successful completion, the **tttk_message_fail**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NOTHANDLER
This application is not the handler for this message.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tttk(5), **tt_message_fail**(3), **tttk_message_abandon**(3), **tttk_message_reject**(3).

NAME

tttk_message_reject – reject a message

SYNOPSIS

```
#include <Tt/tttk.h>

Tt_status tttk_message_reject(Tt_message msg,
                              Tt_status status,
                              const char *status_string,
                              int destroy);
```

DESCRIPTION

The **tttk_message_reject**( ) function sets the status and status string of the **TT_REQUEST** *msg*, rejects the *msg*, and then destroys *msg* if *destroy* is True.

RETURN VALUE

Upon successful completion, the **tttk_message_reject**( ) function returns the status of the operation as one of the following **Tt_status** values:

TT_OK The operation completed successfully.

TT_ERR_NOMP
The **ttsession**(1) process is not running and the ToolTalk service cannot restart it.

TT_ERR_NOTHANDLER
This application is not the handler for this message.

TT_ERR_POINTER
The pointer passed does not point to an object of the correct type for this operation.

SEE ALSO

tttk(5), **tt_message_reject**(3), **tttk_message_fail**(3), **tttk_message_abandon**(3), **tttk_message_fail**(3).

NAME

tttk_op_string – map a ToolTalk op code to a string

SYNOPSIS

#include <Tt/tttk.h>

char *__**tttk_op_string(Tttk_op** *opcode***);**

DESCRIPTION

The **tttk_op_string**( ) function returns a string containing the op for *opcode*.

RETURN VALUE

Upon successful completion, the **tttk_op_string**( ) function returns a string that can be freed using **tt_free**(3); otherwise, it returns **NULL**.

APPLICATION USAGE

The distinctions in the **Tttk_op** enumerated type are for programmer convenience, and elements of **Tttk_op** do not necessarily map one-to-one with op strings, since ToolTalk allows ops to be overloaded. For example, **TTME_EDIT** and **TTME_COMPOSE** are overloaded on the same op (*Edit*), and the messages only vary by the **Tt_mode** of the first argument.

SEE ALSO

tttk(5), **tt_message_op**(3), **tt_free**(3).

NAME

tttk_string_op – map a string to a ToolTalk op code

SYNOPSIS

#include <Tt/tttk.h>

Tttk_op tttk_string_op(const char **opstring***);**

DESCRIPTION

The **tttk_string_op**( ) function returns the **Tttk_op** named by *opstring*.

RETURN VALUE

Upon successful completion, the **tttk_string_op**( ) function a **Tttk_op** value; otherwise, it returns **TTDT_OP_NONE**.

APPLICATION USAGE

See **tttk_op_string**(3).

SEE ALSO

tttk(5), **tttk_op_string**(3), **tt_message_op**(3).

NAME

ttDesktop – introduction to desktop messaging policy

DESCRIPTION

The **Desktop** message policies apply to any tool in a POSIX or **X11**(7) environment. In addition to standard messages for these environments, the **Desktop** policies define data types and error codes that apply to all of the ToolTalk message policies.

LIST OF MESSAGES

```
      Do_Command(       in      string      command,
                        out     string      results ] );
      Set_Environment(  in      string      variable,
                        in      string      value
                                            [...] );
      Get_Environment(  in      string      variable,
                        out     string      value
                                            [...] );
      Set_Geometry(     inout   width       w
                        inout   height      h
                        inout   xOffset     x
                        inout   yOffset     y
                        [in     messageID   commission ] );
      Set_Iconified(    in      boolean     iconic
                        [in     messageID   commission ] );
      Set_Locale(       in      string      category,
                        in      string      locale
                                            [...] );
      Get_Locale(       in      string      category,
                        out     string      locale
                                            [...] );
      Set_Mapped(       in      boolean     mapped
                        [in     messageID   commission ] );
[file] Modified(        in      type         ID );
[file] Reverted(        in      type         ID );
[file] Get_Modified(    in      type         ID,
                        out     boolean     modified );
      Pause(            [in     messageID   operation ] );
      Quit(             in      boolean     silent,
                        in      boolean     force
                        [in     messageID   operation2Quit ] );
      Raise(            [in     messageID   commission ] );
[file] Save(            in      type         ID );
[file] Revert(          in      type         ID );
[file] Saved(           in      type         ID );
      Set_Situation(    in      string      path );
      Get_Situation(    out     string      path );
      Signal(           in      string      theSignal );
      Started(          in      string      vendor,
```

```
                    in    string     toolName,
                    in    string     toolVersion );
      Stopped(      in    string     vendor,
                    in    string     toolName,
                    in    string     toolVersion );
      Status(       in    string     status,
                    in    string     vendor,
                    in    string     toolName,
                    in    string     toolVersion
                    [in   messageID  commission]);
      Get_Status(   out   string     status,
                    out   string     vendor,
                    out   string     toolName,
                    out   string     toolVersion
                    [in   messageID  operation2Query]);
      Get_Sysinfo(  out   string     sysname,
                    out   string     nodename,
                    out   string     release,
                    out   string     version,
                    out   string     machine );
      Get_XInfo(    out   string     display,
                    out   string     visual,
                    out   integer    depth
                    [in   messageID  commission] );
```

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.

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**(). The messageID of a **Tt_message** is returned by **tt_message_id**().

type — Any of the vtypes that are the name of the kind of objects in a particular system of persistent objects. 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.

ERRORS

1102 ITT_DESKTOP_ENOENT
No such file or directory

1113 TT_DESKTOP_EACCES
Permission denied

1122 TT_DESKTOP_EINVAL
Invalid argument

An argument's value was not valid in these circumstances -- e.g., 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 (e.g. **TT_DESKTOP_ENOMSG, TT_DESKTOP_EPROTO**) applies.

1135 TT_DESKTOP_ENOMSG
No message of desired type

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

1171 TT_DESKTOP_EPROTO
Protocol error

A message does could not be understood, because
- a required argument was omitted
- an argument had the wrong vtype, or a vtype not allowed in this message -- e.g., boolean in Get_Geometry()
- an argument had the wrong **Tt_mode**
- an argument's value was not legal for its vtype -- e.g., negative values for width in Set_Geometry()
- an argument's value was not legal for this message -- e.g., PATH=/foo as a *variable* in Get_Environment()

In general, **TT_DESKTOP_EPROTO** means that one could see that the request is malformed simply by comparing it with the reference page for the message.

1147 TT_DESKTOP_ECANCELED
Operation 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**().)

1148 TT_DESKTOP_ENOTSUP
Operation not supported

The requested operation is not supported by this handler. Normally, a well-formed request that a handler does not support should be **tt_message_reject**()ed, thus causing it to fail with **TT_ERR_NO_MATCH** if no supporting handler can be found or started. But sometimes a handler can safely assume that, if it rejects a request, no other handler will be able to perform the operation. Examples: a **TT_HANDLER** -addressed request such as Set_Iconified(), or a request referring to state that is managed by this handler and no other. In these cases, it is better to explicitly fail the request with **TT_DESKTOP_ENOTSUP,** in order to distinguish the case of

an incompletely-implemented handler from the case of the absence of a handler.

TT_ERR_UNIMP should not be used in place of **TT_DESKTOP_ENOTSUP,** because **TT_ERR_UNIMP** means that a particular feature of ToolTalk itself is not implemented.

1299 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 wish 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.

SEE ALSO

ttsession(1), **intro**(2), **X11**(7), **Intro**(TT**Policy)**

NAME

ttMedia – introduction to Media Exchange messaging conventions

DESCRIPTION

The **Media** conventions allow a tool to be a container for arbitrary media, or to be a media player/editor that can be driven from such a container. These conventions allow a container application to compose, display, edit, print, or transform a document of an arbitrary media type, without understanding anything about the format of that media type. ToolTalk routes container requests to the user's preferred tool for the given media type and operation. This includes routing the request to an already-running instance of the tool if that instance is best-positioned to handle the request.

LIST OF MESSAGES

```
[file] Deposit(  in      mediaType   contents,
                 [in     bufferID    beingDeposited
                 in      messageID   commission] );
[file] Display(  in      mediaType   contents,
                 [in     messageID   conterfoil]
                 [in     title       docName] );
[file] Edit(     [in]out mediaType   contents,
                 [in     messageID   counterfoil]
                 [in     title       docName ])
[file] Mail(     in      mediaType   contents );
[file] Mail(     out     mediaType   contents
                 [in     title       docName] );
[file] Mail(     inout   mediaType   contents
                 [in     title       docName] );
[file] Print(    in      mediaType   contents,
                 in      boolean     inquisitive,
                 in      boolean     covert
[file] Translate(in      mediaType   contents,
                 out     mediaType   output,
                 in      boolean     inquisitive,
                 in      boolean     covert
                 [in     messageID   counterfoil] );
```

DEFINITIONS

document A vector of bytes with an associated *mediaType*.

mediaType The name of a media format. The media type of a document allows messages about that document to be dispatched to the right tool. Standard media types include:

ISO_Latin_1	ISO 8859-1 (+TAB+NEWLINE)	ISO
EUC	Multi-National Lang. Supplement	AT&T
PostScript	PostScript Lang. Ref. Manual	Adobe
Sun_Raster	rasterfile(5)	Sun
TIFF	"TIFF Rev. 5" Technical Memo	Aldus
GIF	Graphics Interchange Format	CompuServe
XPM	XPM -- The X PixMap Format	Groupe Bull

JPEG		ISO-CCITT
JPEG_Movie		Parallax
Sun_Audio	audio_intro(3), audio_hdr(3)	Sun
RFC_822_Message	RFC 822	NIC
MIME_Message	RFC MIME	NIC
UNIX_Mail_Folder		
RTF	MS Word Technical Reference	Microsoft
EPS		
Sun_CM_Appointment		Sun

ERRORS

1300 TT_MEDIA_ERR_SIZE
The specified size was too big or too small.

1301 TT_MEDIA_ERR_FORMAT
The data do not conform to their alleged format.

NOTES

It is possible to extract from the ToolTalk types database a list of the installed media types.

NAME

ttPolicy – introduction to ToolTalk messaging policy

DESCRIPTION

ToolTalk is purely an inter-application communication mechanism, and does not specify communication policy. This document sets forth messaging conventions that good ToolTalk citizens should adhere to. The purpose of these conventions is threefold:

1. Prevent collisions, so that no two tools use the same ToolTalk syntax for different semantics.
2. Prevent "passing in the night", so that no two tools fail to talk to each other just because they use different ToolTalk syntax for identical semantics.
3. Encourage socialization, as tool authors are exposed to message interfaces that they might not have thought to add to their tools.

Most of these conventions consist of descriptions of standard ToolTalk messages. Conventions not related to any particular standard message are described either below, or in the **Intro** page for the set of messages they apply to.

Reference page layout

Each message is described on a separate reference page, consisting of:

Name

The name of the message and a one-line description of it.

Synopsis

A representation of the message in a syntax similar to that understood by the ToolTalk type compiler **tt_type_comp**(1). The format is essentially

[*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. ToolTalk allows each message to refer to a file, and has a mechanism (called file-scoping) for delivering messages to clients who are "interested" in the named file.

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_Hoark_My_Frammistat.`

requiredArgs, optionalArgs

In the synopsis, arguments are expressed as *mode vtype argumentName*. *vtype* and *argumentName* are described below. *mode* is one of in, out, or inout, and indicates the direction(s) in which the data of that argument flow.

Description

An explanation of the operation that the request entreats, or the event that the notice announces.

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 an ANSI C function prototype.

Optional Arguments

The extra arguments that may be included in a message. Any optional arguments in a message must be in the specified order, and must follow the required arguments.

Errors

A list of the error codes that can be set by the handler of the request (or the sender of the notice).

Examples

Scenarios in which the message can be useful, and sample ToolTalk code for sending and receiving the message.

Versioning

All messages are individually versioned. When no version information is available, messages may be assumed to be version 0. Version information is carried in a *context slot* with the slotname `version`. (*Contexts* are a new feature in ToolTalk 1.1. In previous releases, arguments can only be positional. That is, they are set and retrieved by ordinal numbers. Context arguments may be set and retrieved by keyword. These ToolTalk messaging policies currently only specify positional arguments for passing data.)

DEFINITIONS

Edict

A notice that looks like a request. If a request returns no data (or if the sender doesn't 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 will be returned, no replies will be received, and the sender is not told whether any tool gets the message.

Handler

The distinguished recipient procid of a request. This procid is responsible for carrying out the indicated operation.

Notice

A message announcing an event. Zero or more tools may receive a given notice. The sender is not told whether any tools receive its notice. A notice cannot be replied to.

Procid A principal that can send and receive ToolTalk messages. A procid is an identity, created and handed over by ToolTalk 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 message that asks an operation to be performed. A request has a distinguished recipient, called a handler, who is responsible for performing 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 of them can fail it or reply to it. If no running handler can be found to accept a request, ToolTalk 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 with a **Tt_state** of **TT_FAILED.**

ERRORS An integer status code can be read from a reply via **tt_message_status**(). This status defaults to 0 (**TT_OK**), or can be set by the handler via **tt_message_status_set**(). In extraordinary circumstances such as no matching handler, ToolTalk itself sets the message status, to a **Tt_status** code.

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

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

ToolTalk allows an arbitrary status string to be included in any reply. Since a standard localized message string can be derived for each status code, the **tt_message_status_string**() may be used as a free-form elucidation of the status. For example, if a request is failed with **TT_DESKTOP_EPROTO,** then 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**(Desktop) request.

SEE ALSO **ttsession**(1), **tt_type_comp**(1), **intro**(2), **Get_Locale**(Desktop), *Solaris 2.2 Developer's Guide to Internationalization*

NAME

ttsession_file - tell **ttsession (1)** about systemwide defaults.

SYNOPSIS

/etc/default/ttsession

DESCRIPTION

The file / etc / default / ttsession can be used to control the behavior of ttsession processes on the machine on which this file resides. Each line within the file is of the form VAR = value. Currently there are two values which VAR may be assigned:

AUTH = <security_options> [, lock=yes | no]
COMPAT = allow_unauth_types_load = yes | no [, lock=yes | no]

Format rules

The format rules for a **ttsesssion_file** are:

1. All lines must be command (VAR = value) lines.

2. Words may be delimited by white space.

3. A command line of the form:

AUTH = <security_options>

follows the same format fules which are used in the "-a" option to ttsession(1).

4. A command line of the form:

COMPAT = allow_unauth_types_load = yes | no

means to allow("yes") or disallow("no") users to successfully call the ToolTalk API function tt_session_types_load(3). The default is "no".

5. Adding the comma seperated suffix:

"lock = yes | no"

to either of the format lines allows ("no") or disallows ("yes") the value to be overridden by a given instance of ttsession(1) via arguments passed to the ttsession process. The default is "no".

EXAMPLES

This example allows calls to tt_session_types_load(3) systemwide:

COMPAT = allow_unauth_types_load = yes

This example sets systemwide security to a specified level, and disallows per-session override on this host:

AUTH = gss,mechanism=kerberos_v5,qop=GSS_KRB5_CONF_C_QOP_DES, lock = yes

SEE ALSO

ttsession(1)

NAME

Deposit – save a document to its backing store

SYNOPSIS

[*file***] Deposit(in** *mediaType contents*
[in messageID *commission***]);**

DESCRIPTION

The *Deposit* request saves a document to its backing store. This request is different from the *Save* request in that the requester (not the handler) has the data to be saved.

The *contents* argument is the contents of the document. If this argument is unset (in other words, has a value of **(char *)0),** then the contents of the document are in the file named in the message's *file* attribute. The data type (*mediaType*) of the *contents* argument should be **string**, unless nulls are valid in the given media type, in which case the data type must be **bytes**.

The *commission* argument contains the message ID of the *Edit* request that caused the creation of this buffer.

APPLICATION USAGE

The **ttmedia_load**(3) function can be used to register for, and help process, this message.

This message can be sent with the **ttmedia_Deposit**(3) function.

The *Deposit* request is useful for cases where the user may perform an intermediate save of modifications to a document that is the subject of an *Edit* or *Display* request in progress. In the latter case, the *Deposit* may fail on a **TT_DESKTOP_EACCES** error if the handler does not allow updates to the document being displayed.

Handlers receiving this request should reply before deleting any file named in the message's *file* attribute, but this is optional and applications should not rely on this.

ERRORS

The ToolTalk service may return one of the following errors in processing the *Deposit* request:

TT_DESKTOP_EACCES
The document is read-only.

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

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

TT_MEDIA_ERR_FORMAT
The document is not a valid instance of the media type.

SEE ALSO

ttmedia_load(3), **ttmedia_Deposit**(3); *Intro*, *Display*, *Edit*, *Status* requests.

NAME

Display – display a document

SYNOPSIS

[*file***] Display(in** *mediaType contents*
[in title *docName***]);**

DESCRIPTION

The *Display* request causes the handler to display (present or manifest) a document to the user. For example, an audio manipulation utility would be said to ''display'' audio documents when it plays them.

The handler must decide issues such as:

- When the display operation can be deemed completed
- What user gesture signals the completion of the display
- What the handling tool should do with itself after replying

The *contents* argument is the contents of the document. If this argument is unset (in other words, has a value of **(char** ∗**)0),** then the contents of the document are in the file named in the message's *file* attribute. The data type (*mediaType*) of the *contents* argument should be **string**, unless nulls are valid in the given media type, in which case the data type must be **bytes**.

The *docName* argument contains the name of the document. If the *docName* argument is absent and the *file* attribute is set, the file name is considered to be the title of the document. This string would be suitable for display in a window title bar, for example.

APPLICATION USAGE

The **ttmedia_ptype_declare**(3) function can be used to register for, and help process, this message.

This message can be sent with the **ttmedia_load**(3) function.

When the document to be displayed is read-only or unlikely to be modified the *Display* message is frequently used instead of the *Edit* message.

EXAMPLES

To display a PostScript document, the application can send a *Display* request with a first argument whose vtype is **PostScript**, and whose value is a vector of bytes such as:

%!\n/inch {72 mul} def...

The **\n** in the example represents the newline character.

To display a PostScript document contained in a file, the application can send a *Display* request with the *file* attribute set to that file and with an unset first argument whose vtype is **PostScript**.

ERRORS

The ToolTalk service may return one of the following errors in processing the *Display* request:

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

TT_DESKTOP_ENODATA
The in-mode *contents* argument had no value and the *file* attribute of the

message was not set.

TT_MEDIA_ERR_FORMAT
The document is not a valid instance of the media type.

SEE ALSO **ttmedia_ptype_declare**(3), **ttmedia_load**(3); *Intro*, *Deposit*, *Edit*, *Status* requests.

NAME

Edit – compose or edit a document

SYNOPSIS

[*file***] Edit([out**| **inout]** *mediaType contents*
[in title *docName***]);**

DESCRIPTION

The *Edit* request causes the handler to edit a document and reply with the new contents when the editing is completed.

It is up to the handler to decide issues such as:

- When the editing operation can be deemed completed
- What user gesture signals the completion of the editing
- What the handling tool should do with itself after replying

If the handling tool supports some form of intermediate save operation during editing, it must send a *Deposit* request back to the tool that requested the *Edit*.

The *contents* argument is the contents of the document. If this argument is unset (in other words, has a value of **(char** ∗**)0),** then the contents of the document are in the file named in the message's *file* attribute. The data type (*mediaType*) of the *contents* argument should be **string**, unless nulls are valid in the given media type, in which case the data type must be **bytes**.

If the *contents* argument is of mode **out**, then a new document must be composed and its contents returned in this argument.

The *docName* argument contains the name of the document. If the *docName* argument is absent and the *file* attribute is set, the file name is considered to be the title of the document. This string would be suitable for display in a window title bar, for example.

APPLICATION USAGE

The **ttmedia_ptype_declare**(3) function can be used to register for, and help process, this message.

This message can be sent with the **ttmedia_load**(3) function.

EXAMPLES

To edit an X11 XBM bitmap, the application can send an *Edit* request with a first argument whose vtype is **XBM**, and whose value is a string such as:

#define foo_width 44\n#define foo_height 94\n

The **\n** in the example represents the newline character.

To edit an X11 XBM bitmap contained in a file, the application can send an *Edit* request naming that file in its *file* attribute, with a first argument whose vtype is **XBM**, and whose value is not set.

ERRORS

The ToolTalk service may return one of the following errors in processing the *Edit* request:

TT_DESKTOP_ECANCELED
The user overrode the *Edit* request. When an *Edit* request is failed with **TT_DESKTOP_ECANCELED**, the document should not be updated as a result,

but rather should remain as it was before the failure reply was received.

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

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

TT_MEDIA_ERR_FORMAT
The document is not a valid instance of the media type.

SEE ALSO

ttmedia_ptype_declare(3), **ttmedia_load**(3); *Intro*, *Display* requests.

NAME

Get_Environment – get a tool's environment

SYNOPSIS

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

DESCRIPTION

The *Get_Environment* request reports the value of the indicated environment variable(s).

The *variable* argument is the name of the environment variable to get.

The *value* argument is the value of the environment variable. If no value (in other words, **(char *)0**) is returned for this argument, then the variable was not present in the handler's environment. This condition is not an error. If an empty string (in other words, "") is returned for this argument, then the variable was present in the handler's environment, but had a null value.

APPLICATION USAGE

The **ttdt_session_join**(3), function can be used to register for, and transparently process, the *Get_Environment* request.

SEE ALSO

ttdt_session_join(3); *Set_Environment* request.

NAME

Get_Geometry – get a tool's on-screen geometry

SYNOPSIS

Get_Geometry(out width *w***,**
out height *h***,**
out xOffset *x***,**
out yOffset *y*
[in messageID *commission***]);**

DESCRIPTION

The *Get_Geometry* request reports the on-screen geometry of the optionally specified window, or of the window primarily associated with the recipient procid (if no window is specified).

The *w*, *h*, *x* and *y* arguments are integer geometry values, in pixels, representing width, height, x-coordinate and y-coordinate, respectively. Negative offset values are interpreted according to the **X11**(7) man page.

The *commission* argument is the ID of the ongoing request, if any, that resulted in the creation of the window in question.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used by Xt applications to register for, and transparently process, the *Get_Geometry* request. Also, *Get_Geometry* can be sent by **ttdt_sender_imprint_on**(3).

SEE ALSO

ttdt_message_accept(3), **ttdt_sender_imprint_on**(3), **ttdt_session_join**(3); *Set_Geometry* request.

NAME

Get_Iconified – get a tool's iconic state

SYNOPSIS

Get_Iconified(out boolean *iconic*
[in messageID *commission***]);**

DESCRIPTION

The *Get_Iconified* request reports the iconic state of the optionally specified window, or of the window primarily associated with the handling procid (if no window is specified).

The *iconic* argument is a Boolean value indicating whether the specified window is (to be) iconified.

The *commission* argument is the ID of the ongoing request, if any, that resulted in the creation of the window(s) in question.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used by Xt applications to register for, and transparently process, the *Get_Iconified* request.

SEE ALSO

ttdt_message_accept(3), **ttdt_session_join**(3); *Set_Iconified* request.

NAME

Get_Locale – get a tool's locale

SYNOPSIS

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

DESCRIPTION

The *Get_Locale* request reports the POSIX locale of the indicated locale categories.

The *category* argument is the locale category to get. A locale category is a group of data types whose output formatting varies according to locale in a similar way. ISO C and locale categories are:

LC_ALL
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME

The *locale* argument is the name of the current locale of the indicated category. The value of *locale* is implementation-defined. For example, Solaris defines such locales as **C**, **de**, **fr**, **it**, etc.

ERRORS

The ToolTalk service may return the following error in processing the *Get_Locale* request:

TT_DESKTOP_EINVAL
The *locale* argument is not valid on the handler's host.

APPLICATION USAGE

The **ttdt_session_join**(3), function can be used to register for, and transparently process, the *Get_Locale* request.

Also, *Get_Locale* can be sent by **ttdt_sender_imprint_on**(3), with the reply being handled transparently.

SEE ALSO

setlocale(3C) *Solaris 2.3 Developer's Guide to Internationalization*, **ttdt_sender_imprint_on**(3), **ttdt_session_join**(3); *Set_Locale* request.

NAME

Get_Mapped – get whether a tool is mapped to the screen

SYNOPSIS

Get_Mapped(out boolean *mapped*
[in messageID *commission***]);**

DESCRIPTION

The *Get_Mapped* request reports the mapped state of the optionally specified window, or of the window primarily associated with the handling procid (if no window is specified).

The *mapped* argument is a Boolean value indicating whether the specified window is (to be) mapped to the screen.

The *commission* argument is the ID of the ongoing request, if any, that resulted in the creation of the window in question.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used by Xt applications to register for, and transparently process, the *Get_Mapped* request.

SEE ALSO

ttdt_message_accept(3), **ttdt_session_join**(3); *Set_Mapped* request.

NAME

Get_Modified – ask whether an entity has been modified

SYNOPSIS

[*file***] Get_Modified(in** *type ID***,**
out boolean *modified***);**

DESCRIPTION

The *Get_Modified* request 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.

Thus, a tool should register a dynamic pattern for this request when it has modified an entity of possible shared interest.

The *ID* argument is the identity of the persistent entity being asked about. When its *type* is **File**, then *ID* is unset (in other words, has a value of **(char** ∗**)0**), and it refers to the file or directory named in the message's file attribute.

The *modified* argument argument is a Boolean value indicating 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

The ToolTalk service may return one of the following errors in processing the *Get_Modified* request:

TT_ERR_NO_MATCH
Since no handler could be found, the entity in question can be assumed not to be modified.

APPLICATION USAGE

The **ttdt_file_join**(3), function can be used to register for, and transparently process, the *Get_Modified* request.

The *Get_Modified* request can be sent with **ttdt_file_request**(3); **ttdt_Get_Modified**(3), can send the *Get_Modified* request and block on the reply.

SEE ALSO

ttdt_file_join(3), **ttdt_file_request**(3), **ttdt_file_request**(3), **ttdt_Get_Modified**(3); *Set_Modified* request.

NAME

Get_Situation – get a tool's current working directory

SYNOPSIS

Get_Situation(out string *path***);**

DESCRIPTION

The *Get_Situation* request reports the current working directory.

The *path* argument is the pathname of the working directory that the recipient is using.

APPLICATION USAGE

The **ttdt_session_join**(3), function can be used to register for, and transparently process, the *Get_Situation* request.

SEE ALSO

ttdt_session_join(3); *Set_Situation* request.

NAME

Get_Status – retrieve a tool's current status

SYNOPSIS

Get_Status(out string *status***,**
out string *vendor***,**
out string *toolName***,**
out string *toolVersion*
[in messageID *operation2Query***]);**

DESCRIPTION

The *Get_Status* request retrieves the current status of a tool (or, optionally, of a specific operation being performed by a tool).

The *status* argument is the status retrieved.

The *vendor* argument is the name of the vendor of the handling tool.

The *toolName* argument is the name of the handling tool.

The *toolVersion* argument is the version of the handling tool.

OPTIONAL ARGUMENTS

The *operation2Query* argument is the ID of the request that initiated the operation the status of which is being requested.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used to register for, and help process, the *Get_Status* request.

EXAMPLES

After sending a **TT_REQUEST** and storing its handle in **Tt_message** *request_I_sent*, if the handler identifies itself with a *Status* notice saved in **Tt_message** *status_msg_from_handler*, then the status of *request_I_sent* can be queried as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                     tt_message_sender(status_msg_from_handler),
                     TTDT_GET_STATUS, my_callback);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_arg_add(msg, TT_IN, Tttk_string,
                                tt_message_id(request_I_sent));
tt_message_send(msg);
```

SEE ALSO

tt_message_arg_add(3), **tt_message_id**(3), **tt_message_send**(3), **ttdt_message_accept**(3), **tt_message_sender**(3), **ttdt_session_join**(3).

NAME

Get_Sysinfo – get information about a tool's host

SYNOPSIS

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

DESCRIPTION

The *Get_Sysinfo* request gets information about the handler's host.

The *sysname* argument is the name of the host's operating system.

The *nodename* argument is the name of the host.

The *release* and *version* arguments are implementation-specific information about the host's operating system.

The *machine* argument is an implementation-specific name that identifies the hardware on which the operating system is running.

APPLICATION USAGE

The **ttdt_session_join**(3), function can be used to register for, and transparently process, the *Get_Sysinfo* request.

EXAMPLES

The *Get_Sysinfo* message can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_GET_SYSINFO,
                                     my_callback);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_send(msg);
```

SEE ALSO

uname(2) **tt_message_arg_add**(3), **tt_message_send**(3), **ttdt_session_join**(3).

NAME

Get_XInfo – get a tool's X11 attributes

SYNOPSIS

Get_XInfo(out string *display***,**
out string *visual***,**
out integer *depth*
[in messageID *commission***]);**

DESCRIPTION

The *Get_XInfo* request reports the X11 attributes of the optionally specified window, or of the window primarily associated with the recipient procid (if no window is specified).

The *display* argument is an X11 display.

The *visual* argument is an X11 visual class (which determines how a pixel will be displayed as a color). Valid values are:

StaticGray
GrayScale
StaticColor
PseudoColor
TrueColor
DirectColor

The *depth* argument is the number of bits in a pixel.

The *commission* argument is the ID of the ongoing request with respect to which X11 attributes are being set or reported.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used by Xt applications to register for, and transparently process, the *Get_XInfo* request. Also, *Get_XInfo* can be sent by **ttdt_sender_imprint_on**(3).

Since the handler may be running on a different host, it is almost always better to return a *display* value of *hostname*:*n*[.*n*] instead of :*n*[.*n*].)

The *commission* argument is useful to the extent that the handler employs different attributes for the different operations it may be carrying out.

EXAMPLES

The *Get_XInfo* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_GET_XINFO,
                                     my_callback);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_iarg_add(msg, TT_OUT, Tttk_integer, 0);
tt_message_send(msg);
```

SEE ALSO

tt_message_iarg_add(3), **tt_message_send**(3), **ttdt_message_accept**(3), **ttdt_sender_imprint_on**(3), **ttdt_session_join**(3).

NAME

hostname_map - tell ToolTalk clients to use a per-host alternative **rpc.ttdbserverd**(1M)

SYNOPSIS

hostname_map

DESCRIPTION

A ToolTalk **hostname_map** tells ToolTalk clients to use host A's **rpc.ttdbserverd**(1M) as a proxy for host B. In this way, ToolTalk databases can be created for filesystems exported from hosts (like host B) that for whatever reason do not run **rpc.ttdbserverd**(1M).

The **hostname_map** file is called "hostname_map" and resides in the same directories as the ToolTalk types databases; see **tt_type_comp**(1). If $TT_HOSTNAME_MAP is set, it is used in place of $HOME/.tt/hostname_map.

ToolTalk clients read the hostname maps when the ToolTalk client library is initialized in e.g. **tt_open**(3). **ttsession** reads the hostname maps upon startup and rereads them if it receives signal USR2.

Format rules

The format rules for a **hostname_map** are:

1. Any line beginning with a "#" or white-space and a "#" is a comment.
2. Blank lines are comments.
3. Words are delimited by white space.
4. The first word in a non-comment line is the host name from which to map.
5. The second word in a non-comment line is the host name to which to map.

ENVIRONMENT

TT_HOSTNAME_MAP
: If $TT_HOSTNAME_MAP is set, it is used in place of $HOME/.tt/hostname_map.

TTPATH
: A colon-separated list of directories in which to seek hostname_maps. See **tt_type_comp**(1).

EXAMPLES

This example maps "mainframe1" to "sparcstorage":

```
# rpc.ttdbserverd cannot run on filesystems
# exported from mainframe
mainframe1 sparcstorage
```

SEE ALSO

rpc.ttdbserverd(1M), **ttsession**(1), **tt_type_comp**(1), **tt_open**(3), **partition_map**(4)

NAME

Lower – lower a tool's window(s) to the back

SYNOPSIS

Lower([in messageID *commission***]);**

DESCRIPTION

The *Lower* request lowers the window(s) associated with the handling procid. If any optional arguments are present, then it lowers only the indicated window(s).

The *commission* argument is the ID of the message, if any, that resulted in the creation of the window(s) that should be lowered.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used by Xt applications to register for, and transparently process, the *Lower* request.

WARNINGS

Lower can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

SEE ALSO

ttdt_message_accept(3), **ttdt_session_join**(3); *Raise* request.

NAME

Mail – compose or mail a document

SYNOPSIS

[*file***] Mail(in** *mediaType contents***);**

[*file***] Mail([out**| **inout]** *mediaType contents*
[in title *docName***]);**

DESCRIPTION

The *Mail* request causes the handler to route a document to a destination using the mail message handling system. The handler is responsible for finding routing information in the document.

When the *contents* argument is of mode **in**, the handler must deliver the document as is, without interacting with the user.

When the *contents* argument is of mode **inout** or **out**, the handler must allow the user to compose or edit the document (and any embedded routing information) before it is delivered. If the handling tool supports some form of intermediate ''save'' operation, it must send a *Deposit* request back to the tool that initiated the *Mail* request.

The *contents* argument is the contents of the document. If this argument is unset (in other words, has a value of **(char** ∗**)0),** then the contents of the document are in the file named in the message's *file* attribute. The data type (*mediaType*) of the *contents* argument should be **string**, unless nulls are valid in the given media type, in which case the data type must be **bytes**.

The *docName* argument contains the name of the document. If the *docName* argument is absent and the *file* attribute is set, the file name is considered to be the title of the document. This string would be suitable for display in a window title bar, for example.

APPLICATION USAGE

The **ttmedia_ptype_declare**(3) function can be used to register for, and help process, this message.

This message can be sent with the **ttmedia_load**(3) function.

ERRORS

The ToolTalk service may return one of the following errors in processing the *Mail* request:

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

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

TT_MEDIA_ERR_FORMAT
The document is not a valid instance of the media type.

SEE ALSO

ttmedia_ptype_declare(3), **ttmedia_load**(3); *Intro*, *Edit* requests.

NAME

Modified – an entity has been modified

SYNOPSIS

[*file***] Modified(in** *type ID***);**

DESCRIPTION

The *Modified* notice is sent whenever a tool first modifies 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.

The *ID* argument is the identity of the modified entity. When its *type* is **File**, then the *ID* argument is unset (in other words, has a value of **(char** ∗**)0**), and it refers to the file or directory named in the message's file attribute.

APPLICATION USAGE

The **ttdt_file_join**(3), function can be used to register for, and help process, the *Modified* request.

The *Modified* request can be sent with **ttdt_file_event**(3).

SEE ALSO

ttdt_file_event(3). **ttdt_file_join**(3); *Reverted* notice.

NAME

partition_map - tell **rpc.ttdbserverd**(1M) to locate its databases in an alternate file system

SYNOPSIS

partition_map

DESCRIPTION

For each filesystem that **rpc.ttdbserverd** needs to store information about, it creates a directory called TT_DB at the mountpoint of that file system. In that directory it creates the databases it needs to store its tables and indices. If the partition is not writable, then **rpc.ttdbserverd** can be told, via **partition_map**(4), to create the databases in another file system.

The **partition_map** file is called "partition_map" and resides in /etc/tt. If $TT_PARTITION_MAP is set, it is used in place of /etc/tt/partition_map.

rpc.ttdbserverd(1M) reads the partition map upon startup and rereads the map if it receives signal USR2.

Format rules

The format rules for a **partition_map** are:

1. Any line beginning with a "#" or white-space and a "#" is a comment.
2. Blank lines are comments.
3. Words are delimited by white space.
4. The first word in a non-comment line is the partition from which to map.
5. The second word in a non-comment line is the partition to which to map. Although the TT_DB directories are by default at the root of their file systems, the user may in fact map to any local filename that is writeable by UID root.

ENVIRONMENT

TT_PARTITION_MAP
If $TT_PARTITION_MAP is set, it is used in place of /etc/tt/partition_map.

EXAMPLES

This example maps "/cdrom" to "/usr":

```
# cannot write to /cdrom
/cdrom /usr
```

This example maps "/cdrom" to "/usr/TT_maps/cdrom":

```
# cannot write to /cdrom
/cdrom /usr/TT_maps/cdrom
```

SEE ALSO

rpc.ttdbserverd(1M), **hostname_map**(4)

NAME

Pause – pause a tool, operation or data performance

SYNOPSIS

Pause([in messageID *operation***]);**

DESCRIPTION

The *Pause* request pauses the specified tool, operation or data performance.

If the optional *operation* argument is included, the handler should pause the operation that was invoked by the specified request.

The *operation* argument is the request that should be paused. For a request to be eligible for pausing, the handler must have sent a *Status* notice back to the requester (thus identifying itself to the requester).

ERRORS

The ToolTalk service may return the following error in processing the *Pause* request:

TT_DESKTOP_ENOMSG
The *operation* argument does not refer to any message currently known by the handler.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used to register for, and help process, the *Pause* request.

EXAMPLES

The *Pause* message can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_PAUSE,
                                     my_callback);
tt_message_send(msg);
```

WARNINGS

Pause can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

SEE ALSO

tt_message_send(3), **ttdt_message_accept**(3), **ttdt_session_join**(3); *Resume* request.

NAME

Print – print a document

SYNOPSIS

[*file***] Print(in** *mediaType contents***,**
in boolean *inquisitive***,**
in boolean *covert*
[in title *docName***]);**

DESCRIPTION

The *Print* request causes the handler to print a document. The handler must act as if the user had issued, (via the handler's user interface) either a ''Print One'' or ''Print...'' command, depending on the value of the *inquisitive* argument.

The *contents* argument is the contents of the document. If this argument is unset (in other words, has a value of **(char** ∗**)0),** then the contents of the document are in the file named in the message's *file* attribute. The data type (*mediaType*) of the *contents* argument should be **string**, unless nulls are valid in the given media type, in which case the data type must be **bytes**.

The *inquisitive* argument is a **boolean** value indicating whether the handler is allowed to block on user input while carrying out the request. However, even if *inquisitive* is True, the handler is not required to seek such input.

The *covert* argument is a **boolean** value indicating whether the handler may make itself apparent to the user as it carries out the request. If False, the recipient need not make itself apparent.

If both the *inquisitive* argument and the *covert* argument are True, the recipient should attempt to limit its presence to the minimum needed to receive any user input desired; for example, through iconification.

The *docName* argument contains the name of the document. If the *docName* argument is absent and the *file* attribute is set, the file name is considered to be the title of the document. This string would be suitable for display in a window title bar, for example.

APPLICATION USAGE

The **ttmedia_ptype_declare**(3) function can be used to register for, and help process, this message.

This message can be sent with the **ttmedia_load**(3) function.

EXAMPLES

To print a PostScript document, the application can send a request of the form:

Print(in PostScript *contents***,**
in boolean *inquisitive***,**
in boolean *covert***);**

with a first argument whose value is a vector of bytes such as:

%!\n/inch {72 mul} def...

The **\n** in the example represents the newline character.

To print a PostScript document contained in a file, the application can send the *Print* request as above, with the *file* attribute set to the relevant file and with the value of the first argument not set.

ERRORS

The ToolTalk service may return one of the following errors in processing the *Print* request:

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

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

TT_MEDIA_ERR_FORMAT
The document is not a valid instance of the media type.

SEE ALSO

ttmedia_ptype_declare(3), **ttmedia_load**(3); *Intro*, *Status* requests.

NAME

Quit – terminate an operation or an entire tool

SYNOPSIS

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

DESCRIPTION

The *Quit* request terminates an operation or an entire tool. Without the optional *operation2Quit* argument, this request asks the handling procid to quit. If the request succeeds, one or more ToolTalk procids should call **tt_close**(3), and zero or more processes should exit.

With the optional *operation2Quit* argument, this request asks the handler 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 requester's satisfaction.)

The *Quit* 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.

The *silent* argument affects user notification of termination. If *silent* is True, the handler is not allowed to block on user input before terminating itself (or the indicated operation). If it is False, however, the handler may seek such input.

The *force* argument is a Boolean value indicating whether the handler should terminate itself (or the indicated operation) even if circumstances are such that the tool ordinarily would not perform the termination.

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 *force* is true, such a tool should terminate even when doing so would lose changes that the user has not been asked by the tool about saving.

The *operation2Quit* argument is the request that should be terminated. For a request to be terminable, the handler must have sent a *Status* notice back to the requester (thus identifying itself to the requester).

ERRORS

The ToolTalk service may return one of the following errors in processing the *Quit* request:

TT_DESKTOP_ECANCELED
The user overrode the *Quit* request.

TT_DESKTOP_ENOMSG
The *operation2Quit* argument does not refer to any message currently known by the handler.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used to register for, and help process, the *Quit* request.

In the successful case, ''zero or more'' procids are cited because a single process can instantiate multiple independent procids, and a single procid can conceivably be implemented by a set of cooperating processes.

EXAMPLES

The *Quit* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_QUIT,
                                     my_callback);
tt_message_iarg_add(msg, TT_IN, Tttk_boolean, 0);
tt_message_iarg_add(msg, TT_IN, Tttk_boolean, 0);
tt_message_send(msg);
```

WARNINGS

Quit can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

BUGS

The *silent* argument should have its polarity reversed, to be like the *inquisitive* argument of several of the **Media** messages.

SEE ALSO

tt_close(3), **tt_message_iarg_add**(3), **tt_message_send**(3), **ttdt_message_accept**(3), **ttdt_session_join**(3).

NAME

Raise – raise a tool's window(s) to the front

SYNOPSIS

Raise([in messageID *commission***]);**

DESCRIPTION

The *Raise* request raises the window(s) associated with the handling procid. If any optional arguments are present, then it raises only the indicated window(s).

The *commission* argument is the ID of the message, if any, that resulted in the creation of the window(s) that should be raised.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used by Xt applications to register for, and transparently process, the *Raise* request.

EXAMPLES

The *Raise* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_RAISE,
                                     my_callback);
tt_message_send(msg);
```

WARNINGS

Raise can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

SEE ALSO

tt_message_send(3), **ttdt_message_accept**(3), **ttdt_session_join**(3); *Lower* request.

NAME

Resume – resume a tool, operation or data performance

SYNOPSIS

Resume([in messageID *operation***]);**

DESCRIPTION

The *Resume* request resumes the specified tool, operation or data performance.

If the optional *operation* argument is included, the handler should resume the operation that was invoked by the specified request.

The *operation* argument is the request that should be resumed.

ERRORS

The ToolTalk service may return the following error in processing the *Resume* request:

TT_DESKTOP_ENOMSG
The *operation* argument does not refer to any message currently known by the handler.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used to register for, and help process, the *Resume* request.

SEE ALSO

ttdt_message_accept(3), **ttdt_session_join**(3); *Pause* request.

NAME

Revert – discard any modifications to an entity

SYNOPSIS

[*file***] Revert(in** *type ID***);**

DESCRIPTION

The *Revert* notice asks that any pending, unsaved modifications to a persistent entity (such as a file) be discarded.

The *ID* argument is the identity of the entity to revert. When its *type* is **File**, then the *ID* argument is unset (in other words, has a value of **(char *)0**), and it refers to the file or directory named in the message's file attribute.

ERRORS

The ToolTalk service may return one of the following errors in processing the *Revert* notice:

TT_DESKTOP_UNMODIFIED
The entity had no pending, unsaved modifications.

TT_DESKTOP_ENOENT
The file to save / revert does not exist.

APPLICATION USAGE

The **ttdt_file_join**(3), function can be used to register for, and help process, the *Revert* request.

The *Revert* request can be sent with **ttdt_file_request**(3). Also, **ttdt_Revert**(3), can send the relevant message and block on the reply.

SEE ALSO

ttdt_Revert(3), **ttdt_file_join**(3), **ttdt_file_request**(3); *Save* notice.

NAME

Reverted – an entity has been reverted

SYNOPSIS

[*file***] Reverted(in** *type ID***);**

DESCRIPTION

The *Reverted* notice is sent when all the modifications (see the *Modified* notice) to an entity have been discarded.

The *ID* argument is the identity of the modified or reverted entity. When its *type* is **File**, then the *ID* argument is unset (in other words, has a value of **(char** ∗**)0**), and it refers to the file or directory named in the message's file attribute.

APPLICATION USAGE

The **ttdt_file_join**(3), function can be used to register for, and help process, the *Reverted* request.

The *Reverted* request can be sent with **ttdt_file_event**(3).

SEE ALSO

ttdt_file_event(3), **ttdt_file_join**(3); *Saved* notice.

NAME

Save – save any modifications to an entity

SYNOPSIS

[*file***] Save(in** *type ID***);**

DESCRIPTION

The *Save* notice asks that any pending, unsaved modifications to a persistent entity (such as a file) be saved.

The *ID* argument is the identity of the entity to save. When its *type* is **File**, then the *ID* argument is unset (in other words, has a value of **(char** ∗**)0**), and it refers to the file or directory named in the message's file attribute.

ERRORS

The ToolTalk service may return one of the following errors in processing the *Save* notice:

TT_DESKTOP_UNMODIFIED
The entity had no pending, unsaved modifications.

TT_DESKTOP_ENOENT
The file to save / revert does not exist.

APPLICATION USAGE

The **ttdt_file_join**(3), function can be used to register for, and help process, the *Save* request.

The *Save* request can be sent with **ttdt_file_request**(3). Also, **ttdt_Save**(3), can send the relevant message and block on the reply.

SEE ALSO

ttdt_Save(3), **ttdt_file_join**(3), **ttdt_file_request**(3); *Revert* notice.

NAME

Saved – an entity has been saved to persistent storage

SYNOPSIS

[*file***] Saved(in** *type ID***);**

DESCRIPTION

The *Saved* notice announces that the persistent storage for an entity (such as a file) has been updated.

The *ID* argument is the identity of the saved entity. When its *type* is **File**, then the *ID* argument is unset (in other words, has a value of **(char** ∗**)0**), and it refers to the file or directory named in the message's file attribute.

APPLICATION USAGE

The **ttdt_file_join**(3), function can be used to register for, and help process, the *Saved* request.

The *Saved* request can be sent with **ttdt_file_event**(3).

SEE ALSO

ttdt_file_event(3), **ttdt_file_join**(3).

NAME

Set_Environment – set a tool's environment

SYNOPSIS

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

DESCRIPTION

The *Set_Environment* request replaces the value of the indicated environment variable(s).

The *variable* argument is the name of the environment variable to set.

The *value* argument is the value of the environment variable. If this argument is unset (in other words, has a value of **(char *)0**), then the variable should be removed from the environment. It is not an error for the variable not to have existed in the first place.

APPLICATION USAGE

The **ttdt_session_join**(3), function can be used to register for, and transparently process, the *Set_Environment* request.

EXAMPLES

The *Set_Environment* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_SET_ENVIRONMENT,
                                     my_callback);
tt_message_arg_add(msg, TT_IN, Tttk_string, "PATH");
tt_message_arg_add(msg, TT_IN, Tttk_string, "/bin:/usr/ucb");
tt_message_send(msg);
```

WARNINGS

Set_Environment can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

SEE ALSO

tt_message_arg_add(3), **tt_message_send**(3), **ttdt_session_join**(3); *Get_Environment* request.

NAME

Set_Geometry – set a tool's on-screen geometry

SYNOPSIS

Set_Geometry(inout width *w*,
inout height *h*,
inout xOffset *x*,
inout yOffset *y*
[in messageID *commission*]);

DESCRIPTION

The *Set_Geometry* request sets the on-screen geometry of the optionally specified window, or of the window primarily associated with the recipient procid (if no window is specified).

The *w*, *h*, *x* and *y* arguments are integer geometry values, in pixels, representing width, height, x-coordinate and y-coordinate, respectively. Negative offset values are interpreted according to the **X11**(7) man page. If any of these arguments are unset, that part of the geometry need not be changed. The return values are the actual new values, in case they differ from the requested new values.

The *commission* argument is the ID of the ongoing request, if any, that resulted in the creation of the window in question.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used by Xt applications to register for, and transparently process, the *Set_Geometry* request.

EXAMPLES

The *Set_Geometry* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_SET_GEOMETRY,
                                     my_callback);
tt_message_iarg_add(msg, TT_INOUT, Tttk_width, 500);
tt_message_iarg_add(msg, TT_INOUT, Tttk_height, 500);
tt_message_arg_add(msg, TT_INOUT, Tttk_xoffset, 0); /* no value */
tt_message_arg_add(msg, TT_INOUT, Tttk_yoffset, 0); /* no value */
tt_message_send(msg);
```

SEE ALSO

tt_message_arg_add(3), **tt_message_iarg_add**(3), **tt_message_send**(3), **ttdt_message_accept**(3), **ttdt_session_join**(3); *Get_Geometry* request.

NAME

Set_Iconified – set a tool's iconic state

SYNOPSIS

Set_Iconified(out boolean *iconic*
[in messageID *commission***]);**

DESCRIPTION

The *Set_Iconified* request sets the iconic state of the optionally specified window, or of the window primarily associated with the handling procid (if no window is specified).

The *iconic* argument is a Boolean value indicating whether the specified window is (to be) iconified.

The *commission* argument is the ID of the ongoing request, if any, that resulted in the creation of the window(s) in question.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used by Xt applications to register for, and transparently process, the *Set_Iconified* request.

If the handler does not map window-system windows one-to-one to commissions or procids, then it may interpret ''iconic state'' liberally. For example, consider a *Display* request on an **ISO_Latin_1** file, handled by a **gnuemacs** instance that then devotes an **emacs** ''window'' to the file. ''Windows'' in **gnuemacs** are not separate X11 windows, and are not separately iconifiable. However, a *Set_Iconified* request issued with respect to the ongoing *Display* request could be liberally interpreted by **gnuemacs** to mean it should minimize the screen real estate devoted to the operation, perhaps by ''burying'' the buffer or dividing its window's real estate among neighboring windows. And, if the *Display* request happens to be the only thing **emacs** is working on at the moment, it could instead take a literal interpretation, and actually iconify itself.

EXAMPLES

The *Set_Iconified* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_SET_ICONIFIED,
                                     my_callback);
tt_message_iarg_add(msg, TT_IN, Tttk_boolean, 1);
tt_message_send(msg);
```

WARNINGS

Set_Iconified can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

SEE ALSO

tt_message_iarg_add(3), **tt_message_send**(3), **ttdt_message_accept**(3), **ttdt_session_join**(3); *Get_Iconified* request.

NAME

Set_Locale – set a tool's locale

SYNOPSIS

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

DESCRIPTION

The *Set_Locale* request reports the POSIX locale of the indicated locale categories.

The *category* argument is the locale category to set. A locale category is a group of data types whose output formatting varies according to locale in a similar way. ISO C and locale categories are:

LC_ALL
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME

The *locale* argument is the name of the locale to set the indicated category to. The value of *locale* is implementation-defined. For example, Solaris defines such locales as **C**, **de**, **fr**, **it**, etc.

ERRORS

The ToolTalk service may return the following error in processing the *Set_Locale* request:

TT_DESKTOP_EINVAL
The *locale* argument is not valid on the handler's host.

APPLICATION USAGE

The **ttdt_session_join**(3), function can be used to register for, and transparently process, the *Set_Locale* request.

EXAMPLES

The *Set_Locale* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_SET_LOCALE,
                                     my_callback);
tt_message_arg_add(msg, TT_IN, Tttk_string, "LC_MONETARY");
tt_message_arg_add(msg, TT_IN, Tttk_string, "de");
tt_message_send(msg);
```

WARNINGS

Set_Locale can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

SEE ALSO

setlocale(3C) *Solaris 2.3 Developer's Guide to Internationalization;* **tt_message_arg_add**(3), **tt_message_send**(3), **ttdt_sender_imprint_on**(3), **ttdt_session_join**(3); *Get_Locale* request.

NAME

Set_Mapped – set whether a tool is mapped to the screen

SYNOPSIS

Set_Mapped(in boolean *mapped*
[in messageID *commission***]);**

DESCRIPTION

The *Set_Mapped* request sets the mapped state of the optionally specified window, or of the window primarily associated with the handling procid (if no window is specified).

The *mapped* argument is a Boolean value indicating whether the specified window is (to be) mapped to the screen.

The *commission* argument is the ID of the ongoing request, if any, that resulted in the creation of the window in question.

APPLICATION USAGE

The **ttdt_session_join**(3), and **ttdt_message_accept**(3), functions can be used by Xt applications to register for, and transparently process, the *Set_Mapped* request.

EXAMPLES

The *Set_Mapped* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_SET_MAPPED,
                                     my_callback);
tt_message_iarg_add(msg, TT_IN, Tttk_boolean, 1);
tt_message_send(msg);
```

WARNINGS

Set_Mapped can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

SEE ALSO

tt_message_iarg_add(3), **tt_message_send**(3), **ttdt_message_accept**(3), **ttdt_session_join**(3); *Get_Mapped* request.

NAME

Set_Situation – set a tool's current working directory

SYNOPSIS

Set_Situation(in string *path***);**

DESCRIPTION

The *Set_Situation* request sets the current working directory.

The *path* argument is the pathname of the working directory that the recipient should use.

APPLICATION USAGE

The **ttdt_session_join**(3), function can be used to register for, and transparently process, the *Set_Situation* request.

EXAMPLES

The *Set_Situation* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_SET_SITUATION,
                                     my_callback);
tt_message_arg_add(msg, TT_OUT, Tttk_string, 0);
tt_message_send(msg);
```

WARNINGS

Set_Situation can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

SEE ALSO

tt_message_arg_add(3), **tt_message_send**(3), **ttdt_session_join**(3); *Get_Situation* request.

NAME

Signal – send a (POSIX-style) signal to a tool

SYNOPSIS

Signal(in string *theSignal***);**

DESCRIPTION

The *Signal* request asks the handling procid to send itself the indicated POSIX signal.

The *theSignal* argument is the signal to send.

APPLICATION USAGE

The **ttdt_session_join**(3), function can be used to register for, and transparently process, the *Signal* request.

EXAMPLES

The *Signal* request can be sent as in the following example:

```
Tt_message msg = tttk_message_create(0, TT_REQUEST, TT_SESSION,
                                     the_recipient_procid, TTDT_SIGNAL,
                                     my_callback);
tt_message_arg_add(msg, TT_IN, Tttk_string, "SIGHUP");
tt_message_send(msg);
```

WARNINGS

Signal can also be sent as a multicast notice, as an *edict* to all tools in the scope of the message. The consequences of doing so can be severe and unexpected.

SEE ALSO

sigaction(2) **tt_message_arg_add**(3), **tt_message_send**(3), **ttdt_session_join**(3),

NAME

Started – a tool has started

SYNOPSIS

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

DESCRIPTION

The *Started* notice announces that a tool has started.

The *vendor* argument is the vendor of the started tool.

The *toolName* argument is the name of the started tool.

The *toolVersion* argument is the version of the started tool.

APPLICATION USAGE

EXAMPLES

A pattern observing the *Started* request can be registered as in the following example:

```
Tt_pattern pat = tt_pattern_create();
tt_pattern_category_set(pat, TT_OBSERVE);
tt_pattern_scope_add(pat, TT_SESSION);
char *ses = tt_default_session();
tt_pattern_session_add(pat, ses);
tt_free(ses);
tt_pattern_op_add(pat, Tttk_Started);
tt_pattern_op_add(pat, Tttk_Stopped);
tt_pattern_callback_add(pat, my_callback);
tt_pattern_register(pat);
```

The *Started* request can be sent with **ttdt_open**(3).

SEE ALSO

tt_free(3), **tt_pattern_callback_add**(3), **tt_pattern_category_set**(3), **tt_pattern_op_add**(3), **tt_pattern_register**(3), **tt_pattern_scope_add**(3), **tt_pattern_session_add**(3), **ttdt_open**(3); *Stopped* notice.

NAME

Status – a tool has some status information to announce

SYNOPSIS

Status(in string *status***,**
in string *vendor***,**
in string *toolName***,**
in string *toolVersion*
[in messageID *commission***]);**

DESCRIPTION

The *Status* notice indicates that a tool has status information to announce.

The *status* argument is the status being announced.

The *vendor* argument is the vendor of the sending tool.

The *toolName* argument is the name of the sending tool.

The *toolVersion* argument is the version of the sending tool.

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

APPLICATION USAGE

The **ttdt_subcontract_manage**(3), function can be used to register for, and help process, the *Status* request.

The *Status* request can be sent with **ttdt_message_accept**(3).

The *Status* notice can be used by handlers of requests invoking protracted operations to provide periodic point-to-point status reports to the requester. Doing so has the nice side effect of identifying the handler to the requester, so that the requester can issue a *Quit* request if it wants to.

SEE ALSO

ttdt_message_accept(3), **ttdt_subcontract_manage**(3); *Quit* request.

NAME

Stopped – a tool has terminated

SYNOPSIS

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

DESCRIPTION

The *Stopped* notice announces that a tool has exited.

The *vendor* argument is the vendor of the terminated tool.

The *toolName* argument is the name of the terminated tool.

The *toolVersion* argument is the version of the terminated tool.

EXAMPLES

A pattern observing the *Stopped* request can be registered as in the following example:

```
Tt_pattern pat = tt_pattern_create();
tt_pattern_category_set(pat, TT_OBSERVE);
tt_pattern_scope_add(pat, TT_SESSION);
char *ses = tt_default_session();
tt_pattern_session_add(pat, ses);
tt_free(ses);
tt_pattern_op_add(pat, Tttk_Started);
tt_pattern_op_add(pat, Tttk_Stopped);
tt_pattern_callback_add(pat, my_callback);
tt_pattern_register(pat);
```

The *Stopped* request can be sent with **ttdt_close**(3).

SEE ALSO

tt_free(3), **tt_pattern_callback_add**(3), **tt_pattern_category_set**(3), **tt_pattern_op_add**(3), **tt_pattern_register**(3), **tt_pattern_scope_add**(3), **tt_pattern_session_add**(3), **ttdt_close**(3); *Started* notice.

NAME

Translate – translate a document from one media type to another

SYNOPSIS

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

DESCRIPTION

The *Translate* request causes the handler to translate a document from one media type to another and return the translation. The translation must be the best possible representation of the document in the target media type, even if the resulting representation cannot be exactly translated back into the original document.

The *contents* argument is the contents of the document. If this argument is unset (in other words, has a value of **(char** ∗**)0),** then the contents of the document are in the file named in the message's *file* attribute. The data type (*mediaType*) of the *contents* argument should be **string**, unless nulls are valid in the given media type, in which case the data type must be **bytes**.

The *output* argument is the translation of the document.

The *inquisitive* argument is a **boolean** value indicating whether the handler is allowed to block on user input while carrying out the request. However, even if *inquisitive* is True, the handler is not required to seek such input.

The *covert* argument is a **boolean** value indicating whether the handler may make itself apparent to the user as it carries out the request. If False, the recipient need not make itself apparent.

If both the *inquisitive* argument and the *covert* argument are True, the recipient should attempt to limit its presence to the minimum needed to receive any user input desired; for example, through iconification.

The *counterfoil* argument is a unique string created by the message sender to give both sender and receiver a way to refer to this request in other correspondence. Typically this string is created by concatenating a process ID and a counter. This argument should be included if the sender anticipates a need to communicate with the handler about this request before it is completed; for example, to cancel it. When this argument is included, and the handler determines that an immediate reply is not possible, the handler must immediately send at least one *Status* notice point-to-point back to the requester, so as to identify itself to the requester.

APPLICATION USAGE

To provide a speech-to-text service, a tool can handle requests of the form:

Translate(in Sun_Audio *contents***,**
out ISO_Latin_1 *output***,**
...);

To provide an OCR (optical character recognition) service, a tool can handle requests of the form:

Translate(in GIF *contents***,**
out ISO_Latin_1 *output***,**
...);

ERRORS

The ToolTalk service may return one of the following errors in processing the *Translate* request:

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

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

TT_MEDIA_ERR_FORMAT
The document is not a valid instance of the media type.

SEE ALSO

Intro, *Abstract*, *Interpret*, *Status* requests.

NAME

tttracefile - script of settings for ToolTalk tracing

DESCRIPTION

A **tttrace** script contains settings that control ToolTalk calls and messages. A **tttrace** script consists of commands separated by semicolons or newlines. The first command must be the **version** command.

Commands

If conflicting values are given for a setting, the last value wins.

version n
: The version of the **tttracefile** command syntax used. The current version is 1.

follow [**off** | **on**]
: Sets whether to follow all children forked by the traced client or subsequently started in the traced session. Default is **off**.

[> | **>>]** *outfile*
: File to be used for the trace output. By default, trace output goes to standard error. Normal **sh**(1) interpretation of > and >> applies.

functions [**all** | **none** | *func*...]
: ToolTalk API functions to trace. *func* may include **sh**(1) wildcard characters. Default is **all**.

attributes [**all** | **none**]
: **none** means use only a single line when printing a message on the trace output. **all** means print all attributes, arguments, and context slots of traced messages. Default is **none**.

states [**none** | **edge** | **deliver** | **dispatch** | *Tt_state*]...
: State(s) through which to trace messages. In addition to the `Tt_state`s defined in `tt_c.h`, valid *state*s are:
  - `none` - disable all message tracing
  - `edge` - messages entering initial (`TT_SENT`) and final (`TT_HANDLED`, `TT_FAILED`) states.
  - `deliver` - all state changes and all client deliveries.
  - `dispatch` - `deliver` + all patterns considered for matching. (default)

ops *toTrace*...
sender_ptypes *toTrace*...
handler_ptypes *toTrace*...
: Trace messages that have *toTrace* as a value for the indicated message attribute. *toTrace* may include **sh**(1) wildcard characters. If no *toTrace* argument is included for a given message attribute, then no value of that attribute excludes a message from tracing.

Comments

A word beginning with **#** causes that word and all the following characters up to a new-line to be ignored.

EXAMPLES

To trace all attribute-getting and -setting messages sent by ptype starting with "Dt",

```
version 1
ops Get* Set*
sender_ptypes Dt*
```

To trace only ToolTalk function calls (but not messages) in a process tree,

```
version 1; follow on; states none
```

SEE ALSO

ttsession(1), **tttrace**(1), the **Session_Trace**() ToolTalk request

NAME

tttk, Tttttk – ToolTalk definitions

SYNOPSIS

#include <Tt/tttk.h>

DESCRIPTION

The **<Tt/tttk.h>** header defines the following enumeration data type, with at least the following members:

Tttk_op

TTDT_CREATED, **TTDT_DELETED**, **TTDT_DO_COMMAND**, **TTDT_GET_ENVIRONMENT**, **TTDT_GET_GEOMETRY**, **TTDT_GET_ICONIFIED**, **TTDT_GET_LOCALE**, **TTDT_GET_MAPPED**, **TTDT_GET_MODIFIED**, **TTDT_GET_SITUATION**, **TTDT_GET_STATUS**, **TTDT_GET_SYSINFO**, **TTDT_GET_XINFO**, **TTDT_LOWER**, **TTDT_MODIFIED**, **TTDT_MOVED**, **TTDT_OP_LAST**, **TTDT_OP_NONE**, **TTDT_PAUSE**, **TTDT_QUIT**, **TTDT_RAISE**, **TTDT_RESUME**, **TTDT_REVERT**, **TTDT_REVERTED**, **TTDT_SAVE**, **TTDT_SAVED**, **TTDT_SET_ENVIRONMENT**, **TTDT_SET_GEOMETRY**, **TTDT_SET_ICONIFIED**, **TTDT_SET_LOCALE**, **TTDT_SET_MAPPED**, **TTDT_SET_SITUATION**, **TTDT_SET_XINFO**, **TTDT_SIGNAL**, **TTDT_STARTED**, **TTDT_STATUS**, **TTDT_STOPPED**, **TTME_ABSTRACT**, **TTME_COMPOSE**, **TTME_DEPOSIT**, **TTME_DISPLAY**, **TTME_EDIT**, **TTME_INTERPRET**, **TTME_MAIL**, **TTME_MAIL_COMPOSE**, **TTME_MAIL_EDIT**, **TTME_PRINT**, **TTME_TRANSLATE**

The header declares the following global string constants for some standard vtypes:

extern const char *Tttk_boolean;
extern const char *Tttk_file;
extern const char *Tttk_height;
extern const char *Tttk_integer;
extern const char *Tttk_message_id;
extern const char *Tttk_string;
extern const char *Tttk_title;
extern const char *Tttk_width;
extern const char *Tttk_xoffset;
extern const char *Tttk_yoffset;

The header declares the following as functions:

```
int ttdt_Get_Modified(Tt_message context,
                      const char *pathname,
                      Tt_scope the_scope,
                      XtAppContext app2run,
                      int ms_timeout);

Tt_status ttdt_Revert(Tt_message context,
                      const char *pathname,
                      Tt_scope the_scope,
                      XtAppContext app2run,
                      int ms_timeout);
```

```
Tt_status ttdt_Save(Tt_message context,
                    const char *pathname,
                    Tt_scope the_scope,
                    XtAppContext app2run,
                    int ms_timeout);
Tt_status ttdt_close(const char *procid,
                     const char *new_procid,
                     int sendStopped);
Tt_status ttdt_file_event(Tt_message context,
                          Tttk_op event,
                          Tt_pattern *patterns,
                          int send);
Tt_pattern *ttdt_file_join(const char *pathname,
                           Tt_scope the_scope,
                           int join,
                           Ttdt_file_cb cb,
                           void *clientdata);
Tt_message ttdt_file_notice(Tt_message context,
                            Tttk_op op,
                            Tt_scope scope,
                            const char *pathname,
                            int send_and_destroy);
Tt_status ttdt_file_quit(Tt_pattern *patterns,
                         int quit);
Tt_message ttdt_file_request(Tt_message context,
                             Tttk_op op,
                             Tt_scope scope,
                             const char *pathname,
                             Ttdt_file_cb cb,
                             void *client_data,
                             int send_and_destroy);
Tt_pattern *ttdt_message_accept(Tt_message contract,
                                Ttdt_contract_cb cb,
                                void *clientdata,
                                Widget shell,
                                int accept,
                                int sendStatus);
char *ttdt_open(int *ttfd,
                const char *toolname,
                const char *vendor,
                const char *version,
                int sendStarted);
```

```
Tt_status ttdt_sender_imprint_on(const char *handler,
                                 Tt_message contract,
                                 char **display,
                                 int *width,
                                 int *height,
                                 int *xoffset,
                                 int *yoffset,
                                 XtAppContext app2run,
                                 int ms_timeout);
Tt_pattern *ttdt_session_join(const char *sessid,
                              Ttdt_contract_cb cb,
                              Widget shell,
                              void *clientdata,
                              int join);
Tt_status ttdt_session_quit(const char *sessid,
                            Tt_pattern *sess_pats,
                            int quit);
Tt_pattern *ttdt_subcontract_manage(Tt_message subcontract,
                                    Ttdt_contract_cb cb,
                                    Widget shell,
                                    void *clientdata);
Tt_status ttmedia_Deposit(Tt_message load_contract,
                          const char *buffer_id,
                          const char *media_type,
                          const unsigned char *new_contents,
                          int new_len,
                          const char *pathname,
                          XtAppContext app2run,
                          int ms_timeout);
Tt_message ttmedia_load(Tt_message context,
                        Ttmedia_load_msg_cb cb,
                        void *clientdata,
                        Tttk_op op,
                        const char *media_type,
                        const unsigned char *contents,
                        int len,
                        const char *file,
                        const char *docname,
                        int send);
Tt_message ttmedia_load_reply(Tt_message contract,
                              const unsigned char *new_contents,
                              int new_len,
                              int reply_and_destroy);
```

```
Tt_status ttmedia_ptype_declare(const char *ptype,
                                int base_opnum,
                                Ttmedia_load_pat_cb cb,
                                void *clientdata,
                                int declare);

void tttk_Xt_input_handler(XtPointer procid,
                           int *source,
                           XtInputId *id);

Tt_status tttk_block_while(XtAppContext app2run,
                           const int *blocked,
                           int ms_timeout);

Tt_status tttk_message_abandon(Tt_message msg);

Tt_message tttk_message_create(Tt_message context,
                               Tt_class the_class,
                               Tt_scope the_scope,
                               const char *handler,
                               const char *op,
                               Tt_message_callback callback);

Tt_status tttk_message_destroy(Tt_message msg);

Tt_status tttk_message_fail(Tt_message msg,
                            Tt_status status,
                            const char *status_string,
                            int destroy);

Tt_status tttk_message_reject(Tt_message msg,
                              Tt_status status,
                              const char *status_string,
                              int destroy);

char *tttk_op_string(Tttk_op opcode);

Tttk_op tttk_string_op(const char *opstring);
```

NAME

tt_c, Tttt_c – ToolTalk definitions

SYNOPSIS

```
#include <Tt/tt_c.h>
```

DESCRIPTION

The **<Tt/tt_c.h>** header includes **typedef**s for the following callback functions:

```
typedef Tt_filter_action (*Tt_filter_function)(const char *nodeid,
                                               void *context,
                                               void *accumulator);

typedef Tt_callback_action (*Tt_message_callback)(Tt_message m,
                                                  Tt_pattern p);
```

The header defines the **TT_VERSION** constant with the value 10200, indicating the version of the ToolTalk API.

The header defines the **Tt_status** enumeration data type, with the following members and specific values:

```
typedef enum tt_status {
    TT_OK             = 0,
    TT_WRN_NOTFOUND   = 1,
    TT_WRN_STALE_OBJID = 2,
    TT_WRN_STOPPED    = 3,
    TT_WRN_SAME_OBJID = 4,
    TT_WRN_START_MESSAGE = 5,
    TT_WRN_NOT_ENABLED = 6,
    TT_WRN_APPFIRST   = 512,
    TT_WRN_LAST       = 1024,
    TT_ERR_CLASS      = 1025,
    TT_ERR_DBAVAIL    = 1026,
    TT_ERR_DBEXIST    = 1027,
    TT_ERR_FILE       = 1028,
    TT_ERR_INVALID    = 1029,
    TT_ERR_MODE       = 1031,
    TT_ERR_ACCESS     = 1032,
    TT_ERR_NOMP       = 1033,
    TT_ERR_NOTHANDLER = 1034,
    TT_ERR_NUM        = 1035,
    TT_ERR_OBJID      = 1036,
    TT_ERR_OP         = 1037,
    TT_ERR_OTYPE      = 1038,
    TT_ERR_ADDRESS    = 1039,
    TT_ERR_PATH       = 1040,
    TT_ERR_POINTER    = 1041,
    TT_ERR_PROCID     = 1042,
    TT_ERR_PROPLEN    = 1043,
    TT_ERR_PROPNAME   = 1044,
```

```
TT_ERR_PTYPE = 1045,
TT_ERR_DISPOSITION = 1046,
TT_ERR_SCOPE = 1047,
TT_ERR_SESSION = 1048,
TT_ERR_VTYPE = 1049,
TT_ERR_NO_VALUE = 1050,
TT_ERR_INTERNAL = 1051,
TT_ERR_READONLY = 1052,
TT_ERR_NO_MATCH = 1053,
TT_ERR_UNIMP = 1054,
TT_ERR_OVERFLOW = 1055,
TT_ERR_PTYPE_START = 1056,
TT_ERR_CATEGORY = 1057,
TT_ERR_DBUPDATE = 1058,
TT_ERR_DBFULL = 1059,
TT_ERR_DBCONSIST = 1060,
TT_ERR_STATE = 1061,
TT_ERR_NOMEM = 1062,
TT_ERR_SLOTNAME = 1063,
TT_ERR_XDR = 1064,
TT_ERR_NETFILE = 1065,
TT_ERR_TOOLATE = 1066,
TT_DESKTOP_ = 1100,
TT_DESKTOP_EPERM = 1101,
TT_DESKTOP_ENOENT = 1102,
TT_DESKTOP_EINTR = 1104,
TT_DESKTOP_EIO = 1105,
TT_DESKTOP_EAGAIN = 1111,
TT_DESKTOP_ENOMEM = 1112,
TT_DESKTOP_EACCES = 1113,
TT_DESKTOP_EFAULT = 1114,
TT_DESKTOP_EEXIST = 1117,
TT_DESKTOP_ENODEV = 1119,
TT_DESKTOP_ENOTDIR = 1120,
TT_DESKTOP_EISDIR = 1121,
TT_DESKTOP_EINVAL = 1122,
TT_DESKTOP_ENFILE = 1123,
TT_DESKTOP_EMFILE = 1124,
TT_DESKTOP_ETXTBSY = 1126,
TT_DESKTOP_EFBIG = 1127,
TT_DESKTOP_ENOSPC = 1128,
TT_DESKTOP_EROFS = 1130,
TT_DESKTOP_EMLINK = 1131,
TT_DESKTOP_EPIPE = 1132,
TT_DESKTOP_ENOMSG = 1135,
```

```
        TT_DESKTOP_EDEADLK     = 1145,
        TT_DESKTOP_ECANCELED   = 1147,
        TT_DESKTOP_ENOTSUP     = 1148,
        TT_DESKTOP_ENODATA     = 1161,
        TT_DESKTOP_EPROTO      = 1171,
        TT_DESKTOP_ENOTEMPTY   = 1193,
        TT_DESKTOP_ETIMEDOUT   = 1245,
        TT_DESKTOP_EALREADY    = 1249,
        TT_DESKTOP_UNMODIFIED  = 1299,
        TT_MEDIA_ERR_SIZE      = 1300,
        TT_MEDIA_ERR_FORMAT    = 1301,
        TT_ERR_APPFIRST        = 1536,
        TT_ERR_LAST            = 2047,
        TT_STATUS_LAST         = 2048
    } Tt_status;
```

Specific values are required because they can be communicated between ToolTalk clients on different platforms, usually via **tt_message_status_set**(3) and **tt_message_status**(3).

The header defines the following enumeration data types, with the following members:

Tt_filter_action
: **TT_FILTER_CONTINUE**, **TT_FILTER_LAST**, **TT_FILTER_STOP**

Tt_callback_action
: **TT_CALLBACK_CONTINUE**, **TT_CALLBACK_LAST**, **TT_CALLBACK_PROCESSED**

Tt_mode
: **TT_IN**, **TT_INOUT**, **TT_MODE_LAST**, **TT_MODE_UNDEFINED**, **TT_OUT**

Tt_scope
: **TT_BOTH**, **TT_FILE**, **TT_FILE_IN_SESSION**, **TT_SCOPE_NONE**, **TT_SESSION**

Tt_class
: **TT_CLASS_LAST**, **TT_CLASS_UNDEFINED**, **TT_NOTICE**, **TT_REQUEST**, **TT_OFFER**

Tt_category
: **TT_CATEGORY_LAST**, **TT_CATEGORY_UNDEFINED**, **TT_HANDLE**, **TT_HANDLE_PUSH**, **TT_HANDLE_ROTATE**, **TT_OBSERVE**

Tt_address
: **TT_ADDRESS_LAST**, **TT_HANDLER**, **TT_OBJECT**, **TT_OTYPE**, **TT_PROCEDURE**

Tt_disposition
: **TT_DISCARD**, **TT_QUEUE**, **TT_START**

Tt_state
: **TT_CREATED**, **TT_FAILED**, **TT_HANDLED**, **TT_QUEUED**, **TT_REJECTED**, **TT_RETURNED**, **TT_ACCEPTED**, **TT_ABSTAINED**, **TT_SENT**, **TT_STARTED**, **TT_STATE_LAST**

Tt_feature

_TT_FEATURE_MULTITHREADED, **_TT_FEATURE_LAST**

The header defines the following as opaque data types: **Tt_message**, **Tt_pattern**.

The header declares the following as functions:

char ∗**tt_X_session(const char** ∗*xdisplaystring***);**

Tt_status tt_bcontext_join(const char ∗*slotname***,**
const unsigned char ∗*value***,**
int *length***);**

Tt_status tt_bcontext_quit(const char ∗*slotname***,**
const unsigned char ∗*value***,**
int *length***);**

Tt_status tt_close(void);

Tt_status tt_context_join(const char ∗*slotname***,**
const char ∗*value***);**

Tt_status tt_context_quit(const char ∗*slotname***,**
const char ∗*value***);**

char ∗**tt_default_file(void);**

Tt_status tt_default_file_set(const char ∗*docid***);**

char ∗**tt_default_procid(void);**

Tt_status tt_default_procid_set(const char ∗*procid***);**

char ∗ **tt_thread_procid(void);**

Tt_status tt_thread_procid_set(const char ∗*procid***);**

char ∗**tt_procid_session(const char** ∗*procid***);**

char ∗**tt_default_ptype(void);**

Tt_status tt_default_ptype_set(const char ∗*ptid***);**

char ∗**tt_default_session(void);**

Tt_status tt_default_session_set(const char ∗*sessid***);**

char ∗ **tt_thread_session(void);**

Tt_status tt_thread_session_set(const char ∗*sessid***);**

int tt_error_int(Tt_status *ttrc***);**

void ∗**tt_error_pointer(Tt_status** *ttrc***);**

int tt_fd(void);

Tt_status tt_file_copy(const char ∗*oldfilepath***,**
const char ∗*newfilepath***);**

Tt_status tt_file_destroy(const char ∗*filepath***);**

Tt_status tt_file_join(const char ∗*filepath***);**

```
Tt_status tt_file_move(const char *oldfilepath,
                       const char *newfilepath);
char *tt_file_netfile(const char *filename);
Tt_status tt_file_objects_query(const char *filepath,
                                Tt_filter_function filter,
                                void *context,
                                void *accumulator);
Tt_status tt_file_quit(const char *filepath);
void tt_free(caddr_t p);
char *tt_host_file_netfile(const char *host,
                           const char *filename);
char *tt_host_netfile_file(const char *host,
                           const char *netfilename);
Tt_status tt_icontext_join(const char *slotname, int value);
Tt_status tt_icontext_quit(const char *slotname, int value);
char *tt_initial_session(void);
Tt_status tt_int_error(int return_val);
int tt_is_err(Tt_status s);
caddr_t tt_malloc(size_t s);
int tt_mark(void);
Tt_status tt_message_accept(Tt_message m);
Tt_address tt_message_address(Tt_message m);
Tt_status tt_message_address_set(Tt_message m, Tt_address a);
Tt_status tt_message_arg_add(Tt_message m,
                             Tt_mode n,
                             const char *vtype,
                             const char *value);
Tt_status tt_message_arg_bval(Tt_message m,
                              int n,
                              unsigned char **value,
                              int *len);
Tt_status tt_message_arg_bval_set(Tt_message m,
                                  int n,
                                  const unsigned char *value,
                                  int len);
Tt_status tt_message_arg_ival(Tt_message m,
                              int n,
                              int *value);
```

```
Tt_status tt_message_arg_ival_set(Tt_message m,
                                  int n,
                                  int value);
Tt_mode tt_message_arg_mode(Tt_message m,
                            int n);
char *tt_message_arg_type(Tt_message m,
                          int n);
char *tt_message_arg_val(Tt_message m,
                         int n);
Tt_status tt_message_arg_val_set(Tt_message m,
                                 int n,
                                 const char *value);
Tt_status tt_message_arg_xval(Tt_message m,
                              int n,
                              xdrproc_t xdr_proc,
                              void **value);
Tt_status tt_message_arg_xval_set(Tt_message m,
                                  int n,
                                  xdrproc_t xdr_proc,
                                  void *value);
int tt_message_args_count(Tt_message m);
Tt_status tt_message_barg_add(Tt_message m,
                              Tt_mode n,
                              const char *vtype,
                              const unsigned char *value,
                              int len);
Tt_status tt_message_bcontext_set(Tt_message m,
                                  const char *slotname,
                                  const unsigned char *value,
                                  int length);
Tt_status tt_message_callback_add(Tt_message m,
                                  Tt_message_callback f);
Tt_class tt_message_class(Tt_message m);
Tt_status tt_message_class_set(Tt_message m,
                               Tt_class c);
Tt_status tt_message_context_bval(Tt_message m,
                                  const char *slotname,
                                  unsigned char **value,
                                  int *len);
```

```
Tt_status tt_message_context_ival(Tt_message m,
                                  const char *slotname,
                                  int *value);
Tt_status tt_message_context_set(Tt_message m,
                                 const char *slotname,
                                 const char *value);
char *tt_message_context_slotname(Tt_message m,
                                  int n);
char *tt_message_context_val(Tt_message m,
                             const char *slotname);
Tt_status tt_message_context_xval(Tt_message m,
                                  const char *slotname,
                                  xdrproc_t xdr_proc,
                                  void **value);
int tt_message_contexts_count(Tt_message m);
Tt_message tt_message_create(void);
Tt_message tt_message_create_super(Tt_message m);
Tt_status tt_message_destroy(Tt_message m);
Tt_disposition tt_message_disposition(Tt_message m);
Tt_status tt_message_disposition_set(Tt_message m,
                                     Tt_disposition r);
Tt_status tt_message_fail(Tt_message m);
char *tt_message_file(Tt_message m);
Tt_status tt_message_file_set(Tt_message m,
                              const char *file);
gid_t tt_message_gid(Tt_message m);
char *tt_message_handler(Tt_message m);
char *tt_message_handler_ptype(Tt_message m);
Tt_status tt_message_handler_ptype_set(Tt_message m,
                                       const char *ptid);
Tt_status tt_message_handler_set(Tt_message m,
                                 const char *procid);
Tt_status tt_message_iarg_add(Tt_message m,
                              Tt_mode n,
                              const char *vtype,
                              int value);
Tt_status tt_message_icontext_set(Tt_message m,
                                  const char *slotname,
                                  int value);
```

```
char *tt_message_id(Tt_message m);
char *tt_message_object(Tt_message m);
Tt_status tt_message_object_set(Tt_message m,
                                const char *objid);
char *tt_message_op(Tt_message m);
Tt_status tt_message_op_set(Tt_message m,
                            const char *opname);
int tt_message_opnum(Tt_message m);
char *tt_message_otype(Tt_message m);
Tt_status tt_message_otype_set(Tt_message m,
                               const char *otype);
Tt_pattern tt_message_pattern(Tt_message m);
char *tt_message_print(Tt_message *m);
Tt_message tt_message_receive(void);
Tt_status tt_message_reject(Tt_message m);
Tt_status tt_message_reply(Tt_message m);
Tt_scope tt_message_scope(Tt_message m);
Tt_status tt_message_scope_set(Tt_message m,
                               Tt_scope s);
Tt_status tt_message_send(Tt_message m);
Tt_status tt_message_send_on_exit(Tt_message m);
char *tt_message_sender(Tt_message m);
char *tt_message_sender_ptype(Tt_message m);
Tt_status tt_message_sender_ptype_set(Tt_message m,
                                      const char *ptid);
char *tt_message_session(Tt_message m);
Tt_status tt_message_session_set(Tt_message m,
                                 const char *sessid);
Tt_state tt_message_state(Tt_message m);
int tt_message_status(Tt_message m);
Tt_status tt_message_status_set(Tt_message m,
                                int status);
char *tt_message_status_string(Tt_message m);
Tt_status tt_message_status_string_set(Tt_message m,
                                       const char *status_str);
```

```
uid_t tt_message_uid(Tt_message m);
void *tt_message_user(Tt_message m,
                      int key);
Tt_status tt_message_user_set(Tt_message m,
                              int key,
                              void *v);
Tt_status tt_message_xarg_add(Tt_message m,
                              Tt_mode n,
                              const char *vtype,
                              xdrproc_t xdr_proc,
                              void *value);
Tt_status tt_message_xcontext_join(const char *slotname,
                                   xdrproc_t xdr_proc,
                                   void *value);
Tt_status tt_message_xcontext_set(Tt_message m,
                                  const char *slotname,
                                  xdrproc_t xdr_proc,
                                  void *value);
char *tt_netfile_file(const char *netfilename);
int tt_objid_equal(const char *objid1,
                   const char *objid2);
char *tt_objid_objkey(const char *objid);
Tt_message tt_onotice_create(const char *objid,
                             const char *op);
char *tt_open(void);
Tt_message tt_orequest_create(const char *objid,
                              const char *op);
char *tt_otype_base(const char *otype);
char *tt_otype_derived(const char *otype,
                       int i);
int tt_otype_deriveds_count(const char *otype);
Tt_mode tt_otype_hsig_arg_mode(const char *otype,
                               int sig,
                               int arg);
char *tt_otype_hsig_arg_type(const char *otype,
                             int sig,
                             int arg);
int tt_otype_hsig_args_count(const char *otype,
                             int sig);
```

```
int tt_otype_hsig_count(const char *otype);
char *tt_otype_hsig_op(const char *otype,
                       int sig);
int tt_otype_is_derived(const char *derivedotype,
                        const char *baseotype);
Tt_status tt_otype_opnum_callback_add(const char *otid,
                                      int opnum,
                                      Tt_message_callback f);
Tt_mode tt_otype_osig_arg_mode(const char *otype,
                               int sig,
                               int arg);
char *tt_otype_osig_arg_type(const char *otype,
                             int sig,
                             int arg);
int tt_otype_osig_args_count(const char *otype,
                             int sig);
int tt_otype_osig_count(const char*otype);
char *tt_otype_osig_op(const char *otype,
                       int sig);
Tt_status tt_pattern_address_add(Tt_pattern p,
                                 Tt_address d);
Tt_status tt_pattern_arg_add(Tt_pattern p,
                             Tt_mode n,
                             const char *vtype,
                             const char *value);
Tt_status tt_pattern_barg_add(Tt_pattern m,
                              Tt_mode n,
                              const char *vtype,
                              const unsigned char *value,
                              int len);
Tt_status tt_pattern_bcontext_add(Tt_pattern p,
                                  const char *slotname,
                                  const unsigned char *value,
                                  int length);
Tt_status tt_pattern_callback_add(Tt_pattern m,
                                  Tt_message_callback f);
Tt_category tt_pattern_category(Tt_pattern p);
Tt_status tt_pattern_category_set(Tt_pattern p,
                                  Tt_category c);
```

```
Tt_status tt_pattern_class_add(Tt_pattern p,
                               Tt_class c);
Tt_status tt_pattern_context_add(Tt_pattern p,
                                 const char *slotname,
                                 const char *value);
Tt_pattern tt_pattern_create(void);
Tt_status tt_pattern_destroy(Tt_pattern p);
Tt_status tt_pattern_disposition_add(Tt_pattern p,
                                     Tt_disposition r);
Tt_status tt_pattern_file_add(Tt_pattern p,
                              const char *file);
Tt_status tt_pattern_iarg_add(Tt_pattern m,
                              Tt_mode n,
                              const char *vtype,
                              int value);
Tt_status tt_pattern_icontext_add(Tt_pattern p,
                                  const char *slotname,
                                  int value);
Tt_status tt_pattern_object_add(Tt_pattern p,
                                const char *objid);
Tt_status tt_pattern_op_add(Tt_pattern p,
                            const char *opname);
Tt_status tt_pattern_opnum_add(Tt_pattern p,
                               int opnum);
Tt_status tt_pattern_otype_add(Tt_pattern p,
                               const char *otype);
char *tt_pattern_print(Tt_pattern *p);
Tt_status tt_pattern_register(Tt_pattern p);
Tt_status tt_pattern_scope_add(Tt_pattern p,
                               Tt_scope s);
Tt_status tt_pattern_sender_add(Tt_pattern p,
                                const char *procid);
Tt_status tt_pattern_sender_ptype_add(Tt_pattern p,
                                      const char *ptid);
Tt_status tt_pattern_session_add(Tt_pattern p,
                                 const char *sessid);
Tt_status tt_pattern_state_add(Tt_pattern p,
                               Tt_state s);
```

```
Tt_status tt_pattern_unregister(Tt_pattern p);
void *tt_pattern_user(Tt_pattern p,
                      int key);
Tt_status tt_pattern_user_set(Tt_pattern p,
                              int key,
                              void *v);
Tt_status tt_pattern_xarg_add(Tt_pattern m,
                              Tt_mode n,
                              const char *vtype,
                              xdrproc_t xdr_proc,
                              void *value);
Tt_status tt_pattern_xcontext_add(Tt_pattern p,
                                  const char *slotname,
                                  xdrproc_t xdr_proc,
                                  void *value);
Tt_message tt_pnotice_create(Tt_scope scope,
                             const char *op);
Tt_status tt_pointer_error(void *pointer);
Tt_message tt_prequest_create(Tt_scope scope,
                              const char *op);
Tt_status tt_ptr_error(pointer);
Tt_status tt_ptype_declare(const char *ptid);
Tt_status tt_ptype_exists(const char *ptid);
Tt_status tt_ptype_opnum_callback_add(const char *ptid,
                                      int opnum,
                                      Tt_message_callback f);
Tt_status tt_ptype_undeclare(const char *ptid);
void tt_release(int mark);
Tt_status tt_session_bprop(const char *sessid,
                           const char *propname,
                           int i,
                           unsigned char **value,
                           int *length);
Tt_status tt_session_bprop_add(const char *sessid,
                               const char *propname,
                               const unsigned char *value,
                               int length);
Tt_status tt_session_bprop_set(const char *sessid,
                               const char *propname,
                               const unsigned char *value,
                               int length);
```

```
Tt_status tt_session_join(const char *sessid);
char *tt_session_prop(const char *sessid,
                      const char *propname,
                      int i);
Tt_status tt_session_prop_add(const char *sessid,
                              const char *propname,
                              const char *value);
int tt_session_prop_count(const char *sessid,
                          const char *propname);
Tt_status tt_session_prop_set(const char *sessid,
                              const char *propname,
                              const char *value);
char *tt_session_propname(const char *sessid,
                          int n);
int tt_session_propnames_count(const char *sessid);
Tt_status tt_session_quit(const char *sessid);
Tt_status tt_session_types_load(const char *session,
                                const char *filename);
Tt_status tt_spec_bprop(const char *objid,
                        const char *propname,
                        int i,
                        unsigned char **value,
                        int *length);
Tt_status tt_spec_bprop_add(const char *objid,
                            const char *propname,
                            const unsigned char *value,
                            int length);
Tt_status tt_spec_bprop_set(const char *objid,
                            const char *propname,
                            const unsigned char *value,
                            int length);
char *tt_spec_create(const char *filepath);
Tt_status tt_spec_destroy(const char *objid);
char *tt_spec_file(const char *objid);
char *tt_spec_move(const char *objid,
                   const char *newfilepath);
char *tt_spec_prop(const char *objid,
                   const char *propname,
                   int i);
```

```
Tt_status tt_spec_prop_add(const char *objid,
                           const char *propname,
                           const char *value);
int tt_spec_prop_count(const char *objid,
                       const char *propname);
Tt_status tt_spec_prop_set(const char *objid,
                           const char *propname,
                           const char *value);
char *tt_spec_propname(const char *objid,
                       int n);
int tt_spec_propnames_count(const char *objid);
char *tt_spec_type(const char *objid);
Tt_status tt_spec_type_set(const char *objid,
                           const char *otid);
Tt_status tt_spec_write(const char *objid);
char *tt_status_message(Tt_status ttrc);
int tt_trace_control(int onoff);
Tt_status tt_xcontext_quit(const char *slotname,
                           xdrproc_t xdr_proc,
                           void *value);
Tt_status tt_feature_enabled(Tt_feature *f);
Tt_status tt_feature_required(Tt_feature *f);
int tt_message_accepters_count(Tt_message *m);
char * tt_message_accepter(Tt_message *m,
                           int n);
int tt_message_rejecters_count(Tt_message *m);
char * tt_message_rejecter(Tt_message *m,
                           int n);
int tt_message_abstainers_count(Tt_message *m);
char * tt_message_abstainer(Tt_message *m,
                            int n);
```

NAME

ttsample1, broadcast – demonstrate simple use of ToolTalk

SYNOPSIS

broadcast

DESCRIPTION

The **broadcast** utility is provided as demo code for the ToolTalk product. This program is compiled by copying the files in the directory /usr/dt/share/examples/tt to a convenient spot and invoking the **make**(1) utility. This will pop up an application with a single button, slider, and a scale reflecting the value last received. By starting several instances of the application, setting the slider in each to a different value, and pushing the button in each instance, the effect of broadcasting the value can be seen. The **ttsnoop**(1) and **tttrace**(1) utilities can be used to monitor the contents of the ToolTalk messages sent by **broadcast.**

SEE ALSO

ttsnoop(1) **tttrace**(1) **ttsession**(1)

DIAGNOSTICS

If you try and invoke broadcast (or any ToolTalk application) and you get a message saying the application could not start ToolTalk, or ttsession, make sure that you have one of the environment variables DISPLAY or TT_SESSION set, and that ttsession is in your PATH, or that the TTSESSION_CMD environment variable indicates where the ttsession program resides. For more information on ttsession and the environment variables it uses, see the ttsession man page.

Index