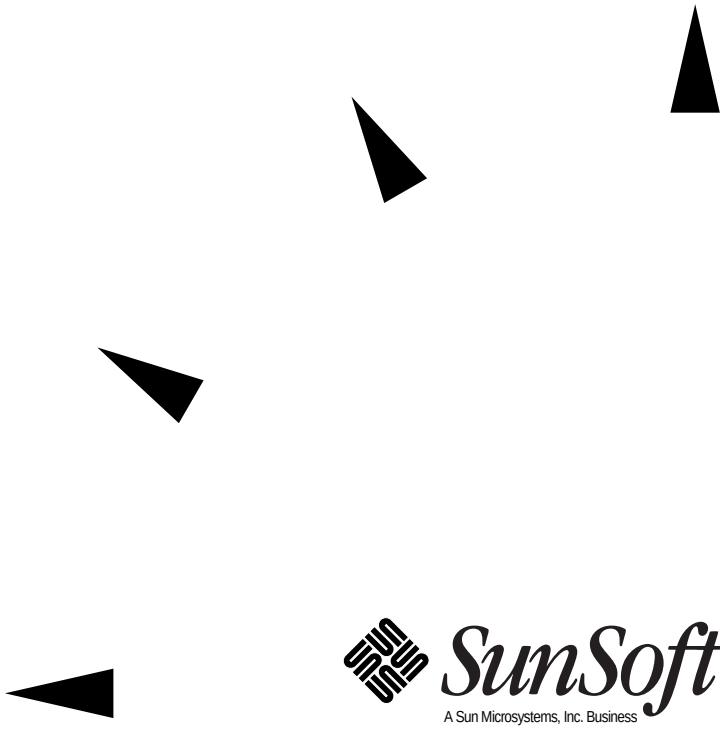
ToolTalk Reference Manual



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

SunSoft
A Sun Microsystems, Inc. Business

Table of Contents

Tables

Preface

This book describes the Application Interface (API) components, commands, and error messages of the ToolTalk® service.

Who Should Use this Book

This manual is for developers who create or maintain applications that use the ToolTalk service to inter-operate with other applications. This manual assumes familiarity with operating system commands, system administrator commands, and system terminology.

How This Book Is Organized

This book is organized as follows:

Chapter 1, “An Overview of the ToolTalk Service,” gives an overview of this manual, including highlights of new api calls in this release.

Chapter 2, “The ToolTalk Enumerated Types,” describes each of the ToolTalk enumerated types.

Chapter 3, “The ToolTalk Functional Groupings,” lists the ToolTalk functions by the specific operations they perform.

Chapter 4, “The ToolTalk Functions,” describes each of the ToolTalk functions.

Chapter 5, “ToolTalk Commands,” describes each of the ToolTalk commands.

Chapter 6, “Initialization Error Messages,” describes the ToolTalk errors that may occur during initialization or startup.

Chapter 7, “ToolTalk Error Messages,” describes the ToolTalk error messages found in the message catalog.

Appendix A, “The ToolTalk Desktop Services Message Set,” describes the ToolTalk message set developed for desktop applications.

Appendix B, “The ToolTalk Document and Media Exchange Message Set,” describes the ToolTalk message set developed for document and media applications.

Related Documentation and Books

The following is a list of related ToolTalk documentation and books:

- *ToolTalk User’s Guide*
- ToolTalk Message Sets
  - CASE Inter-Operability Message Sets
- ToolTalk and Open Protocols, ISBN 013-031055-7
  Published by SunSoft Press/Prentice Hall

An Overview of the ToolTalk Service 1

The ToolTalk service supports several messaging styles. A sender can address a ToolTalk message to a particular process, to any interested process, to an object, or to an object type. Message senders are not concerned with the locations of processes and objects in any network; the ToolTalk service finds receiving processes and objects.

This manual describes the following components of the ToolTalk application programming interface (API):

- Enumerated Types
- Functions

It also describes the following:

- ToolTalk-enhanced operating system shell commands
- Error Messages
- Standard ToolTalk Messaging Sets

1

The ToolTalk Enumerated Types 2

This chapter provides reference information for the enumerated types component of the ToolTalk application programming interface (API).

The ToolTalk enumerated types fall into these categories:

- Tt_address
- Tt_callback
- Tt_category
- Tt_class
- Tt_disposition
- Tt_filter
- Tt_mode
- Tt_scope
- Tt_state
- Tt_status

They are listed in alphabetical order in each section.

Tt_address

Tt_address indicates which message attributes form the address to which the message will be delivered. Table 2-1 describes the possible values.

Table 2-1 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

These values are used to specify the action taken by the callback attached to messages or patterns. If no callback returns TT_CALLBACK_PROCESSED, tt_message_receive() will return the message. Table 2-2 describes the possible values.

Table 2-2 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

`Tt_category` values for the category attribute of a pattern indicate the receiver's intent. Table 2-3 describes the possible values.

Table 2-3 Possible Values for Tt_category

Value	Description
TT_OBSERVE	Just looking at the message. No feedback will be given to the sender.
TT_HANDLE	Will process the message, including filling in return values if any.

Tt_class

These values for the class attribute of a message or pattern indicate whether the sender wants an action to take place after the message has been received. Table 2-4 describes the possible values.

Table 2-4 Possible Values for Tt_class

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

`Tt_disposition` values indicate whether the receiving application should be started to receive the message or if the message should be queued until the receiving process is started at a later time. The message can also be discarded if the receiver is not started.

Note that `Tt_disposition` values can be added together, so that `TT_QUEUE+TT_START` means both to queue the message and to try to start a process. This can be useful if the start can fail (or be vetoed by the user), to ensure the message is processed as soon as an eligible process does start.

Table 2-5 describes the possible values.

Table 2-5 Possible Values for Tt_disposition

Value	Description
TT_DISCARD = 0	No receiver for this message. Message is returned to sender with the Tt_status field containing `TT_FAILED`.
TT_QUEUE = 1	Queue the message until a process of the proper ptype receives the message.
TT_START = 2	Attempt to start a process of the proper ptype if none is running.

Tt_filter

`Tt_filter_action` is the return value from a query callback filter procedure. Table 2-6 describes the possible values.

Table 2-6 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

Tt_mode values specify whether the sender, handler, or observers writes a message argument. Table 2-7 describes the possible values.

Table 2-7 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

Tt_scope values for the Scope attribute of a message or pattern indicate the set of processes eligible to receive the message. Table 2-8 describes the possible values.

Table 2-8 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

Tt_state values indicate a message's delivery status. Table 2-9 describes the possible values.

Table 2-9 Possible Values for Tt_state

Value	Description
TT_CREATED	Message has been created but not yet sent. (Only the sender of a message will see a message in this state.)
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_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.

Tt_status

A Tt_status code is returned by all functions, sometimes directly and sometimes encoded in an error return value. See the *ToolTalk 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.

Chapter 7, "ToolTalk Error Messages," lists the Tt_status codes. The following information is provided for each status code:

- Message id
- Error message string
- Description

- Solution

2

The ToolTalk Functional Groupings 3

This chapter describes the ToolTalk functions component of the ToolTalk application programming interface (API). The functions are grouped to perform specific operations; for example, the functions required to initialize the ToolTalk Service. They are grouped under the following headings:

- Initialization Functions
- Message Patterns
- Ptypes
- Sessions
- Files
- Messages
- Objects
- ToolTalk Storage Management
- ToolTalk Error Status
- Exiting
- ToolTalk Error-Handling Macros

See Chapter 4, “The ToolTalk Functions,” for a detailed description of these functions.

3

3.1 Initialization Functions

Table 3-1 Initializing and Registering with the ToolTalk Service

Return Type	ToolTalk Function
char *	tt_X_session(const char *xdisplay)
Tt_status	tt_default_session_set(const char *sessid)
char *	tt_open(void)
char *	tt_default_procid(void)
Tt_status	tt_default_procid_set(const char *procid)
Tt_status	tt_ptype_declare(const char *ptid)
Tt_status	tt_ptype_undeclare(const char *ptid)
Tt_status	tt_ptype_exists(const char *ptid)
char *	tt_default_ptype(void)
Tt_status	tt_default_ptype_set(const char *ptid)
int	tt_fd(void)

3.2 Message Patterns

Table 3-2 Creating, Filling In, Registering, and Destroying Message Patterns

Return Type	ToolTalk Function
Tt_pattern	tt_pattern_create(void)
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_iarg_add(Tt_pattern m, Tt_mode n, const char *vtype, int value)
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_address_add(Tt_pattern p, Tt_address d)
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_bcontext_add(Tt_pattern p, const char *slotname, const unsigned char *value, int length);
Tt_status	tt_pattern_context_add(Tt_pattern p, const char *slotname, const char *value);
Tt_status	tt_pattern_icontext_add(Tt_pattern p, const char *slotname, int value);
Tt_status	tt_pattern_xcontext_add(Tt_pattern p, const char *slotname, xdrproc_t xdr_proc, void *value)
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_object_add(Tt_pattern p, const char *objid)
Tt_status	tt_pattern_op_add(Tt_pattern p, const char *opname)
Tt_status	tt_pattern_otype_add(Tt_pattern p, const char *otype)
Tt_status	tt_pattern_scope_add(Tt_pattern p, Tt_scope s)

Table 3-2 Creating, Filling In, Registering, and Destroying Message Patterns (Continued)

Return Type	ToolTalk Function
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)
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_register(Tt_pattern p)
Tt_status	tt_pattern_unregister(Tt_pattern p)
Tt_status	tt_bcontext_join(const char *slotname, const unsigned *char value, int length);
Tt_status	tt_context_join(const char *slotname, const char *value);
Tt_status	tt_icontext_join(const char *slotname, int value);
Tt_status	tt_xcontext_join(const char *slotname, xdrproc_t xdr_proc, void *value);
Tt_status	tt_bcontext_quit(const char *slotname, const unsigned *char value, int length);
Tt_status	tt_xcontext_quit(const char *slotname, xdrproc_t xdr_proc, void *value)
Tt_status	tt_context_quit(const char *slotname, const char *value);
Tt_status	tt_icontext_quit(const char *slotname, int value);

3.3 Ptypes

Table 3-3 Declaring, Undeclaring, and Checking Ptypes

Return Type	ToolTalk Function
Tt_status	tt_ptype_declare(const char *ptid)
Tt_status	tt_ptype_exists(const char *ptid)
Tt_status	tt_ptype_undeclare(const char *ptid)

3.4 Sessions

Table 3-4 Expressing Interest in Sessions

Return Type	ToolTalk Function
char *	tt_default_session(void)
Tt_status	tt_default_session_set(const char *sessid)
char *	tt_initial_session(void)
Tt_status	tt_session_join(const char *sessid)
Tt_status	tt_session_quit(const char *sessid)

Table 3-5 Managing Session Information

Return Type	ToolTalk Function
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)
Tt_status	tt_session_bprop(const char *sessid, const char *propname, int i, unsigned char **value, int *length)

Table 3-5 Managing Session Information (Continued)

Return Type	ToolTalk Function
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)
char *	tt_session_propname(const char *sessid, int n)
int	tt_session_propnames_count(const char *sessid)
Tt_status	tt_session_types_load(const char *session, const char *filename)

3.5 Files

Table 3-6 Expressing Interest in Files

Return Type	ToolTalk Function
Tt_status	tt_file_join(const char *filepath)
Tt_status	tt_file_quit(const char *filepath)
char *	tt_default_file(void)
Tt_status	tt_default_file_set(const char *docid)

Table 3-7 Managing Files

Return Type	ToolTalk Function
Tt_status	tt_file_move(const char *oldfilepath, const char *newfilepath)
Tt_status	tt_file_copy(const char *oldfilepath, const char *newfilepath)
Tt_status	tt_file_destroy(const char *filepath)

3.6 Messages

Table 3-8 Creating Messages

Return Type	ToolTalk Function
Tt_message	tt_onotice_create(const char *objid, const char *op)
Tt_message	tt_orequest_create(const char *objid, const char *op)
Tt_message	tt_pnotice_create(Tt_scope scope, const char *op)
Tt_message	tt_prequest_create(Tt_scope scope, const char *op)
Tt_message	tt_message_create(void)
Tt_message	tt_message_create_super(Tt_message m)

Table 3-9 Filling In Messages and Replies

Return Type	ToolTalk Function
Tt_status	tt_message_address_set(Tt_message m, Tt_address p)
Tt_status	tt_message_accept(Tt_message m);
Tt_status	tt_message_arg_add(Tt_message m, Tt_mode n, const char *vtype, const char *value)
Tt_status	tt_message_arg_bval_set(Tt_message m, int n, unsigned char *value, int len)
Tt_status	tt_message_arg_ival_set(Tt_message m, int n, int value)
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)
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, unsigned char *value, int length);
Tt_status	tt_message_callback_add(Tt_message m, Tt_message_callback f)
Tt_status	tt_message_class_set(Tt_message m, Tt_class c)
int	tt_message_contexts_count(Tt_message m);

Table 3-9 Filling In Messages and Replies (Continued)

Return Type	ToolTalk Function
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);
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);
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);
Tt_status	tt_message_disposition_set(Tt_message m, Tt_disposition r)
Tt_status	tt_message_file_set(Tt_message m, const char *file)
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);
Tt_status	tt_message_object_set(Tt_message m, const char *objid)
Tt_status	tt_message_op_set(Tt_message m, const char *opname)
Tt_status	tt_message_otype_set(Tt_message m, const char *otype)
Tt_status	tt_message_scope_set(Tt_message m, Tt_scope s)
Tt_status	tt_message_send_on_exit(Tt_message m);
Tt_status	tt_message_sender_ptype_set(Tt_message m, const char *ptid)
Tt_status	tt_message_session_set(Tt_message m, const char *sessid)
Tt_status	tt_message_status_set(Tt_message m, int status)
Tt_status	tt_message_status_string_set(Tt_message m, const char *status_str)
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)

Table 3-9 Filling In Messages and Replies (Continued)

Return Type	ToolTalk Function
Tt_status	tt_message_xcontext_set(Tt_message m, const char *slotname, xdrproc_t xdr_proc, void *value)
Tt_status	tt_otype_opnum_callback_add(const char *otid,int opnum, Tt_message_callback f)
Tt_status	tt_ptype_opnum_callback_add(const char *ptid, int opnum, Tt_message_callback f)

*Table 3-10*Examining Messages

Return Type	ToolTalk Function
Tt_address	tt_message_address(Tt_message m)
Tt_status	tt_message_arg_bval(Tt_message m, int n, unsigned char **value, int *len)
Tt_status	tt_message_arg_ival(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_xval(Tt_message m, int n, xdrproc_t xdr_proc, void *value)
int	tt_message_args_count(Tt_message m)
Tt_class	tt_message_class(Tt_message m)
Tt_disposition	tt_message_disposition(Tt_message m)
char *	tt_message_file(Tt_message m)
gid_t	tt_message_gid(Tt_message m)
char *	tt_message_handler(Tt_message m)
char *	tt_message_handler_ptype(Tt_message m)
char *	tt_message_id(Tt_message m)
char *	tt_message_object(Tt_message m)
char *	tt_message_op(Tt_message m)
int	tt_message_opnum(Tt_message m)
char *	tt_message_otype(Tt_message m)

Table 3-10 Examining Messages (Continued)

Return Type	ToolTalk Function
Tt_pattern	tt_message_pattern(Tt_message m)
Tt_scope	tt_message_scope(Tt_message m)
char *	tt_message_sender(Tt_message m)
char *	tt_message_sender_ptype(Tt_message m)
char *	tt_message_session(Tt_message m)
Tt_state	tt_message_state(Tt_message m)
int	tt_message_status(Tt_message m)
char *	tt_message_status_string(Tt_message m)
uid_t	tt_message_uid(Tt_message m)
void *	tt_message_user(Tt_message m, int key)
Tt_status	tt_message_send(Tt_message m)
Tt_status	tt_message_destroy(Tt_message m)

Table 3-11 Sending and Destroying Messages

Return Type	ToolTalk Function
Tt_status	tt_message_send(Tt_message m)
Tt_status	tt_message_destroy(Tt_message m)

Table 3-12 Receiving, Replying to, Rejecting, and Destroying Messages

Return Type	ToolTalk Function
Tt_message	tt_message_receive(void)
Tt_status	tt_message_reply(Tt_message m)
Tt_status	tt_message_reject(Tt_message m)
Tt_status	tt_message_fail(Tt_message m)
int	tt_message_status(Tt_message m)

*Table 3-12*Receiving, Replying to, Rejecting, and Destroying Messages (Continued)

Return Type	ToolTalk Function
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)
Tt_status	tt_message_destroy(Tt_message m)

3.7 Objects

*Table 3-13*Creating, Moving, and Destroying Objects

Return Type	ToolTalk Function
char *	tt_spec_create(const char *filepath)
Tt_status	tt_spec_prop_add(const char *objid, const char *propname, const char *value)
Tt_status	tt_spec_prop_set(const char *objid, const char *propname, const char *value)
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)
Tt_status	tt_spec_type_set(const char *objid, const char *otid)
Tt_status	tt_spec_write(const char *objid)
char *	tt_spec_move(const char *objid, const char *newfilepath)
Tt_status	tt_spec_destroy(const char *objid)

*Table 3-14*Using ToolTalk Storage

Return Type	ToolTalk Function
char *	tt_spec_prop(const char *objid, const char *propname, int i)
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)
Tt_status	tt_spec_bprop(const char *objid, const char *propname, int i, unsigned char **value, int *length)
char *	tt_spec_propname(const char *objid, int n)
int	tt_spec_propnames_count(const char *objid)
char *	tt_spec_type(const char *objid)
char *	tt_spec_file(const char *objid)
Tt_status	tt_spec_write(const char *objid)
Tt_status	tt_file_objects_query(const char *filepath, Tt_filter_function filter, void *context, void *accumulator)
int	tt_objid_equal(const char *objid1, const char *objid2)
char *	tt_objid_objkey(const char *objid)

*Table 3-15*Examining Object Type Information

Return Type	ToolTalk Function
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)

*Table 3-15*Examining Object Type Information (Continued)

Return Type	ToolTalk Function
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)

3.8 ToolTalk Storage Management

*Table 3-16*Managing ToolTalk Storage

Return Type	ToolTalk Function
int	tt_mark(void)
void	tt_release(int mark)
void	tt_free(caddr_t p)
caddr_t	tt_malloc(size_t s)

3.9 ToolTalk Error Status

Table 3-17 Retrieving ToolTalk Error Information

Return Type	ToolTalk Function
Tt_status	tt_int_error(int return_val)
Tt_status	tt_pointer_error(void *pointer)
char *	tt_status_message(Tt_status ttrc)

Table 3-18 Encoding Error Values

Return Type	ToolTalk Function
int	tt_error_int(Tt_status ttrc)
void *	tt_error_pointer(Tt_status ttrc)

3.10 Exiting

Table 3-19 Leaving the ToolTalk Session

Return Type	ToolTalk Function
Tt_status	tt_close(void)

3.11 ToolTalk Error-Handling Macros

Table 3-20 ToolTalk Error-Handling Macros

Return Type	ToolTalk Macro
int	tt_is_err(Tt_status s)
Tt_status	tt_ptr_error(pointer)

3

tt_bcontext_join

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

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.

Arguments

`const char *slotname`
The name of the context.

`const unsigned char *value`
The value to be added.

`int length`
The length of the value.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-1.

Table 4-1 Possible Status of tt_bcontext_join Call

Value Returned	Description
TT_OK	Operation successful.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_bcontext_quit

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

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.

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

Arguments

`const char *slotname`
The name of the context.

`const unsigned char *value`
The value to be added.

`int length`
The length of the value.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-2.

Table 4-2 Possible Status of tt_bcontext_quit Call

Value Returned	Description
TT_OK	Operation successful.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_SLOTNME	The slotname is not valid.

4

tt_close

Tt_status tt_close(void)

Closes the current procid.

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

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Related Functions

`tt_open()`

tt_context_join

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

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.

Arguments

`const char *slotname`
The name of the context.

`const char *value`
The value to be added.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-3.

Table 4-3 Possible Status of tt_context_join Call

Value Returned	Description
TT_OK	Operation successful.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_context_quit

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

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, the given string value is removed from the list of values for that slot.

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

Arguments

`const char *slotname`
The name of the context.

`const char *value`
The value to be added.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-4.

Table 4-4 Possible Status of tt_context_quit Call

Value Returned	Description
TT_OK	Operation successful.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_SLOTNME	The slotname is not valid.

tt_default_file

```
char        *tt_default_file(void)
```

Returns the current default file.

When you join a file, it becomes the default file.

Returned Value

`char *`
The pointer to a character string that specifies the current default file. If the pointer is `NULL`, no default file is set.

Use `tt_ptr_error()` to determine whether the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

`tt_file_join()`

tt_default_file_set

Tt_status tt_default_file_set(const char *docid)

Sets the default file to the specified file.

Arguments

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

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_PROCID`
- `TT_ERR_FILE`

tt_default_procid

char *tt_default_procid(void)

Retrieves the current default procid for your process.

Returned Value

`char *`

The pointer to a character string that uniquely identifies the current default process.

Use `tt_ptr_error()` to determine whether the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_default_procid_set

Tt_status tt_default_procid_set(const char *procid)

Sets the current default procid.

Arguments

`const char *procid`
The name of process that is to be the default process.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Related Functions

`tt_open()`

tt_default_ptype

```
char        *tt_default_ptype(void)
```

Retrieves the current default ptype.

When you declare a ptype, it becomes the default ptype.

Returned Value

`char *`
The pointer to a character string that uniquely identifies the current default process type. If the pointer is `NULL`, no default ptype is set.

Use `tt_ptr_error()` to determine whether the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

`tt_ptype_declare()`

tt_default_ptype_set

Tt_status tt_default_ptype_set(const char *ptid)

Sets the default ptype.

Arguments

`const char *ptid`
Use the character string that uniquely identifies the process that is to be the default process.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

tt_default_session

```
char        *tt_default_session(void)
```

Retrieves the current default session identifier.

Note – A session can have more than one session identifier. This means that you *cannot* compare the result of `tt_default_session` with the result of `tt_message_session` to verify that the message was sent in your default session.

Returned Value

`char *`

The pointer to the unique identifier for the current session. If the pointer is `NULL`, no default session is set.

Use `tt_ptr_error()` to determine whether the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_default_session_set

Tt_status tt_default_session_set(const char *sessid)

Sets the current default session identifier.

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

Note – To join other sessions, your process 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()
```

You can call `tt_open` to create additional ToolTalk processes; however, the current implementation of the ToolTalk service allows only one ToolTalk session per process (although multiple processes are allowed in a client). The ToolTalk service does not currently support API calls to determine to which session a particular process is connected if multiple processes are running. If you are running multiple sessions and it is important for your application to know the session to which it is connected, it must make the following calls in the indicated order:

```
tt_open
tt_default_session
```

You can then store the information by indexing it by the procid returned by the `tt_open` call.

Note – To change to another opened session, you must use the `tt_default_procid_set` call.

Arguments

`const char *sessid`
The pointer to the unique identifier for the session in which your process is interested.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`
- `TT_ERR_SESSION`

Related Functions

`tt_open()`
`tt_default_procid()`
`tt_default_session`

tt_error_int

int tt_error_int(Tt_status ttrc)

Returns an integer error object that encodes the code.

Note – The integer error objects are negative integers; use this call only when the valid integer values are non-negative.

Arguments

`Tt_status ttrc`
The `Tt_status` code you want to encode.

Returned Value

`int`
The encoded `Tt_status` code.

tt_error_pointer

void *tt_error_pointer(Tt_status ttrc)

Returns a pointer to an error object that encodes the code.

Arguments

`Tt_status ttrc`
The `Tt_status` code that is to be encoded.

Returned Value

`void *`
The pointer to the encoded `Tt_status` code.

tt_fd

int tt_fd(void)

Returns a file descriptor.

The returned file descriptor alerts your process that a message has arrived for the default procid in the default session.

File descriptors are either active or inactive. When your file descriptor becomes active, your process needs to call `tt_message_receive` to receive the message.

Note – You must have a separate file descriptor for each procid. To get an associated file descriptor, use `tt_fd` each time you call `tt_open`.

Returned Value

`int`
The file descriptor for the current procid.

Use `tt_int_error()` to determine whether the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Related Functions

`tt_open()`
`tt_message_receive()`

tt_file_copy

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

Copies all objects that exist on the specified file to a new file.

Note – If any objects already exist on *newfilepath*, they are not overwritten by the copy (that is, they are not removed.)

Arguments

`const char *oldfilepath`
The pointer to the name of the file whose objects are to be copied.

`const char *newfilepath`
The pointer to the name of the file on which to create the copied objects.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_ACCESS`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_FILE`
- `TT_ERR_NOMEN`
- `TT_ERR_NOMP`
- `TT_ERR_PATH`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

`tt_file_move()`
`tt_file_destroy()`

tt_file_destroy

Tt_status tt_file_destroy(const char *filepath)

Removes all objects that exist on the files and directories rooted at *filepath*.

Call this function when you unlink a file or remove a directory.

Arguments

`const char *filepath`
The pointer to the pathname of the file or directory to be removed.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_ACCESS`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_FILE`
- `TT_ERR_NOMP`
- `TT_ERR_PATH`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

`tt_file_copy()`
`tt_file_move()`
`rmdir(2)`
`unlink(2)`

tt_file_join

Tt_status tt_file_join(const char *filepath)

Informs the ToolTalk service that the process is interested in messages which involve the specified file.

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

Note – 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 recognized as having interest in the specified file.

Arguments

`const char *filepath`
The pointer to the pathname of the file in which your process is interested.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_PATH`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_file_move

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

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 filesystem, `tt_file_move()` replaces *oldfilepath* with *newfilepath* in the path associated with every object in that filesystem; that is, all the objects in the directory tree rooted at *oldfilepath* are overlayed onto *newfilepath*. In this mode, the behavior of `tt_file_move()` is similar to the system call `rename(2)`.

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

Arguments

`const char *oldfilepath`
The name of the file or directory whose objects are to be moved.

`const char *newfilepath`
The name of the file or directory to which the objects are to be moved.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_ACCESS`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_FILE`
- `TT_ERR_NOMP`
- `TT_ERR_PATH`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

```
tt_file_copy()
tt_file_destroy()
rename(2)
```

tt_file_objects_query

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

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 you initially specify is 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`
- `TT_FILTER_STOP`

Arguments

`const char *filepath`
The name of the file to be searched for objects.

`Tt_filter_function filter`
The filter function to which the objids are to be passed.

`void *context`
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.

`void *accumulator`
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.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_PATH`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_WRN_STOPPED`

tt_file_quit

Tt_status tt_file_quit(const char *filepath)

Informs the ToolTalk service that the process is no longer interested in messages which involve the specified file.

The ToolTalk service removes this file value from any currently registered patterns. The default file is nulled.

Arguments

`const char *filepath`
The name of the file in which the process is no longer interested.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_FILE`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PATH`
- `TT_ERR_PROCID`

tt_free

```
void        tt_free(caddr_t p)
```

Frees storage from the ToolTalk API allocation stack.

Use the `tt_free` function instead of the `tt_mark` and `tt_release` if, for example, your process is in a loop (that is, it obtains strings from the ToolTalk service and processes each in turn).

Arguments

`caddr_t p`
The address of the storage in the ToolTalk API allocation stack to be freed.

Related Functions

`tt_malloc()`

tt_icontext_join

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

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.

Arguments

`const char *slotname`
The name of the context.

`int value`
The value to be added.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-5.

Table 4-5 Possible Status of tt_icontext_join Call

Value Returned	Description
TT_OK	Operation successful.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_icontext_quit

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

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.

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

Arguments

`const char *slotname`
The name of the context.

`int value`
The value to be added.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-6.

Table 4-6 Possible Status of tt_icontext_quit Call

Value Returned	Description
TT_OK	Operation successful.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_SLOTNME	The slotname is not valid.

tt_initial_session

```
char        *tt_initial_session(void)
```

Returns the initial session identifier of the ttsession with which the current process identifier is associated.

The `tt_initial_session` call returns the initial session identifier of the ttsession with which the current process identifier is associated. The current process identifier is obtained by calling `tt_open`.

Returned Value

`char *`

The identifier for the current ToolTalk session.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_int_error

Tt_status tt_int_error(int return_val)

Returns the status of an error object.

When given an integer, this call 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.

Arguments

`int return_val`
The integer returned by a ToolTalk function.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_`*xxx*

tt_is_err

int tt_is_err(Tt_status s)

Checks whether status is a warning or an error.

This macro informs whether the Tt_status enum provided is a warning or an error.

Arguments

`Tt_status s`
The Tt_status code to check.

Returned Value

`int`

If 1 is returned, the `Tt_status` enum is an error; if 0 is returned, the `Tt_status` enum is either a warning or `TT_OK`.

tt_malloc

caddr_t tt_malloc(size_t s)

Allocates storage on the ToolTalk API allocation stack.

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

Arguments

`size_t s`
The amount of storage to be allocated in bytes.

Returned Value

`caddr_t`
The address of the storage in the ToolTalk API allocation stack that is to be allocated. If `NULL` is returned, no storage is available.

Related Functions

`tt_free()`

tt_mark

int tt_mark(void)

Marks a storage position in the ToolTalk API allocation stack.

Returned Value

`int`
The integer that marks the storage position in the ToolTalk API allocation stack.

Related Functions

`tt_release()`

tt_message_accept

Tt_status tt_message_accept(Tt_message m)

Declares that the process has been initialized and can accept messages.

This call is invoked for start messages.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_UNIMP`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_message_address

Tt_address tt_message_address(Tt_message m)

Retrieves the address attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_address`
Specifies which message attributes form the address of this message. Possible values are:

- `TT_PROCEDURE`
- `TT_OBJECT`
- `TT_HANDLER`
- `TT_OTYPE`

Use `tt_int_error()` to determine if the `Tt_address` integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_POINTER`

tt_message_address_set

```
Tt_status       tt_message_address_set(Tt_message m,
                Tt_address a)
```

Sets the address attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`Tt_address a`
Specifies which message attributes form the address to which the message will be delivered. Possible values are:

- `TT_PROCEDURE`
- `TT_OBJECT`
- `TT_HANDLER`
- `TT_OTYPE`

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_POINTER`

tt_message_arg_add

Tt_status tt_message_arg_add(Tt_message m, Tt_mode n,
const char *vtype, const char *value)

Adds a new argument to a message object.

You must add all arguments before the message is sent. To change existing argument values, only use mode TT_OUT or TT_INOUT.

Note – Do *not* add arguments when you reply to a message.

Arguments

Tt_message m
The opaque handle for the message involved in this operation.

Tt_mode n
Specifies who (sender, handler, observers) writes and reads a message argument. Possible modes are:

- TT_IN
- TT_OUT
- TT_INOUT

const char *vtype
Describes the type of argument data being added.

const char *value
The contents for the message argument attribute. 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
- tt_message_barg_val_set
- tt_message_iarg_val_set

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

Related Functions

`tt_message_arg_val_set()`
`tt_message_barg_add()`
`tt_message_iarg_add()`

tt_message_arg_bval

```
Tt_status       tt_message_arg_bval(Tt_message m, int n,
                unsigned char **value, int *len)
```

Retrieves the byte-array value of the *nth* message argument.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to be retrieved. The first argument is 0.

`unsigned char **value`
The address of a character pointer to which the ToolTalk service is to point a string that contains the contents of the argument.

`int *len`
The address of an integer to which the ToolTalk service is to set the length of the value in bytes.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_arg_bval_set

```
Tt_status       tt_message_arg_bval_set(Tt_message m, int n,
                const unsigned char *value, int len)
```

Sets the byte-array value and the type of the *nth* message argument.

This function also changes the value of the *nth* message argument to a byte string. The sending process can use `tt_message_arg_bval_set` to fill in opaque data.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to set. The first argument is 0.

`const unsigned char *value`
The byte string with the contents for the message argument.

`int len`
The length of the value in bytes.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

```
tt_message_barg_add()
tt_message_arg_val_set()
tt_message_iarg_val_set()
```

tt_message_arg_ival

```
Tt_status       tt_message_arg_ival(Tt_message m, int n,
                int *value)
```

Retrieves the integer value of the *nth* message argument.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to be retrieved. The first argument is 0.

`int *value`
The pointer to an integer where the ToolTalk service is to store the contents of the argument.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`

`int value`
The value of the *nth* argument.

tt_message_arg_ival_set

```
Tt_status       tt_message_arg_ival_set(Tt_message m, int n,
                int value)
```

Adds an integer value in the *nth* message argument.

This function also changes the value of the *nth* message argument to an integer.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to be set. The first argument is 0.

`int value`
The contents (in integer form) for the message argument.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`

Related Functions

```
tt_message_arg_ival_add()
tt_message_arg_val_set()
tt_message_barg_val_set()
```

tt_message_arg_mode

Tt_mode tt_message_arg_mode(Tt_message m, int n)

Returns the mode of the *nth* message argument.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to be returned. The first argument is 0.

Returned Value

`Tt_mode`
Specifies who (sender, handler, observers) writes and reads a message argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

Use `tt_int_error()` to determine if the `Tt_mode` integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`

tt_message_arg_type

```
char        *tt_message_arg_type(Tt_message m, int n)
```

Retrieves the type of the *nth* message argument.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to be retrieved. The first argument is 0.

Returned Value

`char *`
The type of the *nth* message argument.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_arg_val

```
char        *tt_message_arg_val(Tt_message m, int n)
```

Returns a pointer to the value of the *nth* message argument.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to be returned. The first argument is 0.

Returned Value

`char *`
The contents for the message argument.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_arg_val_set

```
Tt_status       tt_message_arg_val_set(Tt_message m, int n,
                const char *value)
```

Changes the value of the *nth* message argument.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to be changed. The first argument is 0.

`const char *value`
The contents for the message argument.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_arg_xval

Tt_status tt_message_arg_xval(Tt_message m, int n,
xdrproc_t xdr_proc, void *value)

Retrieves and deserializes the data from a message argument.

This function uses an XDR routine that is supplied by the client.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to be returned. The first argument is 0.

`xdrproc_t xdr_proc`
Points to the XDR procedure to be used to deserialize the data in the nth argument into the storage pointed to by value.

Note – The allocation calls are made by the XDR procedure; therefore, any storage allocated is *not* allocated from the ToolTalk allocation stack. You must use the `xdr_free` call to free this storage.

`void *value`
The data to be deserialized.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-7.

Table 4-7 Possible Status of tt_message_arg_xval Call

Value Returned	Description
TT_OK	The operation completed successfully.
TT_ERR_MODE	Invalid mode value.
TT_ERR_NOMP	ToolTalk is not initialized
TT_ERR_NUM	Invalid argument number
TT_ERR_POINTER	An invalid message handle, XDR proc pointer, or data pointer.
TT_ERR_XDR	The XDR procedure failed on the given data, or evaluated to a 0 length structure.

tt_message_arg_xval_set

Tt_status tt_message_arg_xval_set(Tt_message m, int n, xdrproc_t xdr_proc, void *value)

Serializes and sets data into an existing message argument.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the argument to be changed. The first argument is 0.

`xdrproc_t xdr_proc`
Serialize the data pointed to by value and stores it as a byte string value of the nth argument of the message.

`void *value`
The data to be serialized.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-8.

Table 4-8 Possible Status of tt_message_arg_xval_set Call

Value Returned	Description
TT_OK	The operation was successful.
TT_ERR_MODE	Invalid mode value
TT_ERR_NOMP	ToolTalk is not initialized
TT_ERR_POINTER	Invalid message handle, XDR proc pointer, or data pointer
TT_ERR_NUM	Invalid argument number
TT_ERR_XDR	The XDR procedure failed on the given data, or evaluated to a 0 length structure.

tt_message_args_count

```
int             tt_message_args_count(Tt_message m)
```

Returns the number of arguments in the message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`int`
The total number of arguments in the message.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_barg_add

```
Tt_status       tt_message_barg_add(Tt_message m, Tt_mode n,
                const char *vtype, const unsigned char *value,
                int len)
```

Adds an argument to a pattern that may have a byte-array value which contains imbedded nulls.

To change existing argument values, only use mode `TT_OUT` or `TT_INOUT`.

Note – Do *not* add arguments to a reply.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`Tt_mode n`
Specifies who (sender, handler, observers) writes and reads a message argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

`const char *vtype`
Describes the type of argument data being added.

The ToolTalk service treats the value as an opaque byte string. To pass structured data, your application and the receiving application must encode and decode these opaque byte strings. The most common method to do this is XDR.

`const unsigned char *value`
The value to be filled in by the ToolTalk service.

`int len`
The length of the value in bytes.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

```
tt_message_barg_val_set()
tt_message_arg_add()
tt_message_iarg_add()
```

tt_message_bcontext_set

```
Tt_status       tt_message_bcontext_set (Tt_message m,
                const char *slotname, unsigned char *value, int length);
```

Sets the byte-array value of a message's context.

This function overwrites any previous value associated with *slotname*.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *slotname`
Describes the slotname in this message.

`const unsigned char *value`
The byte string with the contents for the message argument.

`int length`
The length of the value in bytes.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-9.

Table 4-9 Possible Status of tt_message_bcontext_set Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_POINTER	The Tt_message handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_message_callback_add

Tt_status tt_message_callback_add(Tt_message m,
Tt_message_callback f)

Registers a callback function to be automatically invoked by `tt_message_receive` whenever a reply or other state-change to this message is returned.

Note – Callbacks are called in reverse order of registration (for example, the most recently added callback is called first).

`Tt_callback_action` is an enum that contains the values `TT_CALLBACK_CONTINUE` and `TT_CALLBACK_PROCESSED`.

- 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.

Use this function 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.

Note – 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.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`Tt_message_callback f`
Passes the specified message and the pattern that matched it to the callback.

Note – The pattern handle will be null if the message did not match a dynamic pattern. This is usually the case for message callbacks.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_class

Tt_class tt_message_class(Tt_message m)

Retrieves the class attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_class`
Indicates whether the sender wants an action to take place after the message is received. Possible values are:

- `TT_NOTICE`
- `TT_REQUEST`

Use `tt_int_error()` to determine if the `Tt_class` integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_message_class_set

Tt_status tt_message_class_set(Tt_message m,
Tt_class c)

Sets the class attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`Tt_class c`
Indicates whether an action is to take place after the message is received. Possible values are:

- `TT_NOTICE`
- `TT_REQUEST`

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_context_bval

```
Tt_status       tt_message_context_bval (Tt_message m,
                const char *slotname, unsigned char **value, int *len);
```

Retrieves the byte-array value and length of a message's context.

If there is no context slot associated with *slotname*, a null pointer and zero length is returned.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *slotname`
Describes the context of this message.

`unsigned char **value`
The value to be filled in by the ToolTalk service.

`int *len`
The length of the value in bytes.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-10.

Table 4-10 Possible Status of tt_message_context_bval Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_POINTER	The Tt_message handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_message_context_ival

```
Tt_status       tt_message_context_ival (Tt_message m,
                const char *slotname, int *value);
```

Retrieves the integer value of a message's context.

If there is no context slot associated with *slotname*, a zero value is returned.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *slotname`
Describes the context of this message.

`int *value`
The value to be filled in by the ToolTalk service.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-11.

Table 4-11 Possible Status of tt_message_context_ival Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_NUM	The integer value that was passed is not valid.
TT_ERR_POINTER	The Tt_message handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

tt_message_context_set

```
Tt_status       tt_message_context_set (Tt_message m,
                const char *slotname, const char *value);
```

Sets the character string value of a message's context.

This function overwrites any previous value associated with *slotname*.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *slotname`
Describes the context of this message.

`const char *value`
The value to be filled in by the ToolTalk service.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-12.

Table 4-12 Possible Status of tt_message_context_set Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_UNIMP	The function called is not implemented.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_POINTER	The Tt_message handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_message_context_slotname

char * tt_message_context_slotname(Tt_message m, int n)

Returns the name of a message's *nth* context.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int n`
The number of the context to be retrieved. The first context is 0.

Returned Value

`char *`
The slotname of the *nth* message context.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_UNIMP`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_context_val

```
char *          tt_message_context_val(Tt_message m, const char
*slotname);
```

Retrieves the character string of a message's context.

If there is no context slot associated with *slotname*, a null pointer is returned.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *slotname`
Describes the context of this message.

Returned Value

`char *`
The value of the context. Use `tt_ptr_error()` to determine if the pointer is valid. Possible values are shown in Table 4-13.

Table 4-13 Possible Status of tt_message_context_val Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_UNIMP	The function called is not implemented.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_POINTER	The Tt_message handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

tt_message_context_xval

Tt_status * tt_message_context_xval(Tt_message m, const char *slotname,
xdrproc_t xdr_proc, void *value);

Retrieves and deserializes the data from a message's context.

Arguments

`_message m`
The opaque handle for the message involved in this operation.

`const char *slotname`
Describes the context of this message.

`xdrproc_t xdr_proc`
Points to the XDR procedure to be used to deserialize the data in the nth argument into the storage pointed to by value.

Note – The allocation calls are made by the XDR procedure; therefore, any storage allocated is *not* allocated from the ToolTalk allocation stack. You must use the `xdr_free` call to free this storage.

`void *value`
The data to be deserialized.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-14.

*Table 4-14*Possible Status of tt_message_context_xval Call

Returned Value	Description
TT_OK	Operation was successful.
TT_ERR_MODE	Invalid mode value
TT_ERR_NOMP	ToolTalk is not initialized
TT_ERR_POINTER	Invalid message handle, XDR proc pointer, or data pointer.
TT_ERR_NUM	Invalid argument number
TT_ERR_XDR	The XDR procedure failed on the given data, or evaluated to a 0 length structure.

tt_message_contexts_count

```
int          tt_message_contexts_count(Tt_message m)
```

Returns the number of contexts in a message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`int`
The total number of contests in the message.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_UNIMP`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_message_create

Tt_message tt_message_create(void)

Creates a new message object.

The ToolTalk service returns a message handle that is an opaque pointer to a ToolTalk structure.

Returned Value

`Tt_message`
The unique opaque handle that identifies the message object.

If the ToolTalk service is unable to create a message when requested, an invalid handle is returned. When you attempt to use this handle, the ToolTalk service report an error. Use `tt_pointer_error` to determine why the ToolTalk service was not able to create the message.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

`tt_message_send()`
`tt_message_destroy()`

tt_message_create_super

Tt_message tt_message_create_super(Tt_message m)

Creates a copy of the specified message and re-addresses the copy of the message to the parent of the specified otype.

The handle to the new message is returned.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_message`
The opaque unique handle for the re-addressed message.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_ADDRESS`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_OTYPE`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

`tt_message_send()`
`tt_message_destroy()`

tt_message_destroy

Tt_status tt_message_destroy(Tt_message m)

Destroys the message.

Destroying a message has no effect on the delivery of a message already sent.

If you sent a request and are expecting a reply with return values, destroy the message *after* you have received the reply. If you sent a notice, you can destroy the message immediately after you send the notice.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

`tt_message_create()`
`tt_message_create_super()`

tt_message_disposition

Tt_disposition tt_message_disposition(Tt_message m)

Retrieves the disposition attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_disposition`
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. Possible values are:

- `TT_QUEUE`
- `TT_START`
- `TT_QUEUE+TT_START`

Use `tt_int_error()` to determine if the `Tt_disposition` integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_POINTER`

tt_message_disposition_set

Tt_status tt_message_disposition_set(Tt_message m,
Tt_disposition r)

Sets the disposition attribute for the specified message.

You need to set the disposition of a message you send only when the handler_ptype of the message has been set. However, this disposition is over-ridden by the disposition of any static signatures that the message matches. If the message your application is sending must start an instance of the ptype, you must set both the disposition and the handler_ptype; in addition, you must make sure that the message does not match any non-start signature of that ptype. For example, the message:

```
file Static( in string val )  => queue opnum = 0;
```

matches the signature

```
ptype Rec {
      start "$DTHOME/bin/xview/xterm -e rec -title Rec";
      handle :
          file Static( in string val )  => queue opnum = 0;
```

The disposition, therefore, would be TT_QUEUE.

Note – Queued requests and file-scoped queued requests are not implemented.

Arguments

Tt_message m
The opaque handle for the message involved in this operation.

`Tt_disposition r`
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. Possible values are:

- `TT_QUEUE`
- `TT_START`
- `TT_QUEUE+TT_START`

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_fail

Tt_status tt_message_fail(Tt_message m)

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.

To distinguish this case from the case where a message failed because no matching handler could be found, place an explanatory message code in the status attribute of the message with tt_message_status_set and tt_message_status_string_set before calling tt_message_fail.

Note – The status value must be greater than 2047 (TT_ERR_LAST) to avoid confusion with the ToolTalk service status values.

Arguments

Tt_message m
The opaque handle for the message involved in this operation.

Returned Value

Tt_status
The status of the operation. Possible values are:

- TT_OK
- TT_ERR_NOMP
- TT_ERR_NOTHANDLER
- TT_ERR_POINTER
- TT_ERR_PROCID

Related Functions

tt_message_status_set()
tt_message_status_string_set()

tt_message_file

```
char        *tt_message_file(Tt_message m)
```

Retrieves the file attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
The file attribute of the specified message.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_file_set

```
Tt_status       tt_message_file_set(Tt_message m,
                const char *file)
```

Sets the file attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *file`
The name of the file involved in this operation.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_FILE`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_gid

gid_t tt_message_gid(Tt_message m)

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.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`gid_t`
The group identifier of the message. If the group *65534* is returned, the message handle is not valid.

Related Functions

`tt_message_uid()`

tt_message_handler

```
char        *tt_message_handler(Tt_message m)
```

Retrieves the handler attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
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`).

Use `tt_ptr_error()`, which returns `Tt_status`, to determine if the pointer is valid. Possible `Tt_status` values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_handler_ptype

```
char        *tt_message_handler_ptype(Tt_message m)
```

Retrieves the handler ptype attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
Type of process that should handle this message.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_handler_ptype_set

```
Tt_status       tt_message_handler_ptype_set(Tt_message m,
                const char *ptid)
```

Sets the handler ptype attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *ptid`
The type of process which is to handle this message.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_handler_set

```
Tt_status       tt_message_handler_set(Tt_message m,
                const char *procid)
```

Sets the handler attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *procid`
The character value that uniquely identifies the process which is to handle the message.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_message_iarg_add

```
Tt_status       tt_message_iarg_add(Tt_message m, Tt_mode n,
                const char *vtype, int value)
```

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, only use mode `TT_OUT` or `TT_INOUT`.

Note – Do *not* add arguments to a reply.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`Tt_mode n`
Specifies who (sender, handler, observers) writes and reads a message argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

`const char *vtype`
Describes the type of argument data being added.

`int value`
The value to be added.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_POINTER`
- `TT_ERR_VTYPE`

Related Functions

```
tt_message_arg_ival_set()
tt_message_arg_add()
tt_message_barg_add()
```

tt_message_icontext_set

```
Tt_status       tt_message_icontext_set (Tt_message m,
                const char *slotname, int value);
```

Sets the integer value of a message's context.

This function overwrites any previous value associated with *slotname*.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *slotname`
Describes the context of this message.

`int value`
The value to be filled in by the ToolTalk service.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-15.

Table 4-15 Possible Status of tt_message_icontext_set Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_UNIMP	The function called is not implemented.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_POINTER	The Tt_message handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_message_id

```
char            *tt_message_id(Tt_message m)
```

Retrieves the identifier of the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
The character string value that uniquely identifies the message across all running ToolTalk sessions. The id of the message is set at its creation and never changes.

Use `tt_ptr_error()`, which returns `Tt_status`, to determine if the pointer is valid. Possible `Tt_status` values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_object

```
char        *tt_message_object(Tt_message m)
```

Retrieves the object attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
The object involved in this message.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_object_set

```
Tt_status       tt_message_object_set(Tt_message m,
                const char *objid)
```

Sets the object attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *objid`
The object involved in this message.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_op

```
char            *tt_message_op(Tt_message m)
```

Retrieves the operation attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
The operation which the receiving process is to perform.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_op_set

```
Tt_status       tt_message_op_set(Tt_message m,
                const char *opname)
```

Sets the operation attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *opname`
The operation that the receiving process is to perform.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_opnum

int tt_message_opnum(Tt_message m)

Retrieves the operation number attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`int`
The number of the operation involved in this message.

Use `tt_int_error()` to determine if the `opnum` integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_ERR_POINTER`

tt_message_otype

```
char        *tt_message_otype(Tt_message m)
```

Retrieves the object type attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
The type of the object involved in this message.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_otype_set

```
Tt_status       tt_message_otype_set(Tt_message m,
                const char *otype)
```

Sets the object type (otype) attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *otype`
The type of the object involved in this message.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OTYPE`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_pattern

```
Tt_pattern      tt_message_pattern(Tt_message m)
```

Returns the pattern that the specified message matched.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_pattern`
The opaque handle for a message pattern.

Use `tt_ptr_error()` to determine if the handle is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_receive

Tt_message tt_message_receive(void)

Returns a handle for the next message queued to be delivered to the process.

The `tt_message_receive()` function also runs any message or pattern callbacks applicable to the queued message.

- If the return value is 0, no message is available.
  This value can occur if a message or pattern callback processes the message.
- If the return value is `TT_WRN_START_MESSAGE`, the ToolTalk service started the process to deliver the queued message and the process must reply to this message even if it is a notice.

  To verify that the return value is `TT_WRN_START_MESSAGE`, use `tt_message_status()`.

Returned Value

`Tt_message`
The handle for the message object.

Use `tt_ptr_error()` to determine if the handle is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_message_reject

Tt_status tt_message_reject(Tt_message m)

Informs the ToolTalk service that the process cannot handle this message.

The ToolTalk service will attempt to deliver the message to other handlers.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_INVALID`
- `TT_ERR_NOMP`
- `TT_ERR_NOTHANDLER`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_reply

Tt_status tt_message_reply(Tt_message m)

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.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- TT_OK
- TT_ERR_CLASS
- TT_ERR_NOMP
- TT_ERR_NOTHANDLER
- TT_ERR_POINTER
- TT_ERR_PROCID

tt_message_scope

Tt_scope tt_message_scope(Tt_message m)

Retrieves the scope attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_scope`
Identifies the set of processes eligible to receive the message. Possible values are:

- `TT_SESSION`
- `TT_FILE`
- `TT_BOTH`
- `TT_FILE_IN_SESSION`

Use `tt_int_error()` to determine if the `Tt_scope` integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_POINTER`

tt_message_scope_set

```
Tt_status       tt_message_scope_set(Tt_message m,
                Tt_scope s)
```

Sets the scope attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`Tt_scope s`
Identifies the set of processes eligible to receive the message. Possible values are:

- `TT_SESSION`
- `TT_FILE`
- `TT_BOTH`
- `TT_FILE_IN_SESSION`

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_message_send

Tt_status tt_message_send(Tt_message m)

Sends the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_ADDRESS`
- `TT_ERR_CLASS`
- `TT_ERR_FILE`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_OTYPE`
- `TT_ERR_OVERFLOW`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_SESSION`
- `TT_WRN_STALE_OBJID`
- `TT_ERR_STATE`
- `TT_ERR_SCOPE`

tt_message_send_on_exit

```
Tt_status       tt_message_send_on_exit(Tt_message m);
```

Requests that the ToolTalk service send this message if process exits unexpectedly.

This message is sent to the ToolTalk service, which queues the message internally until either of two events occur:

1. The process that sent the tt_message_send_on_exit message to the ToolTalk service calls `tt_close`.

   In this case, the queued message is deleted.

2. The connection between the ttsession 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`, both the normal termination message and the on_exit message are delivered.

Note – This message *must* be a session-scoped notice. Requests or file-scoped messages *cannot* be sent with this call.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_ADDRESS`

- TT_ERR_CLASS
- TT_ERR_FILE
- TT_ERR_NOMP
- TT_ERR_OBJID

tt_message_sender

```
char *          tt_message_sender(Tt_message m)
```

Retrieves the sender attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
The character value that uniquely identifies the sending process.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_sender_ptype

```
char *          tt_message_sender_ptype(Tt_message m)
```

Retrieves the sender ptype attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
The sending process.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_sender_ptype_set

```
Tt_status       tt_message_sender_ptype_set(Tt_message m,
                const char *ptid)
```

Sets the sender ptype attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *ptid`
The type of process that is sending this message.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

tt_message_session

```
char        *tt_message_session(Tt_message m)
```

Retrieves the session attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
The identifier of the session to which this message applies.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_message_session_set

```
Tt_status       tt_message_session_set(Tt_message m,
                const char *sessid)
```

Sets the session attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *sessid`
The identifier of the session in which the process is interested.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_SESSION`

tt_message_state

Tt_state tt_message_state(Tt_message m)

Retrieves the state attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`Tt_state`
Indicates the current delivery state of the message. Possible values are:

- `TT_CREATED`
- `TT_SENT`
- `TT_HANDLED`
- `TT_FAILED`
- `TT_QUEUED`
- `TT_STARTED`
- `TT_REJECTED`

Use `tt_int_error()` to determine if the `Tt_state` integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_POINTER`

tt_message_status

```
int             tt_message_status(Tt_message m)
```

Retrieves the status attribute from the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`int`
An integer that describes the status stored in the status attribute of this message.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_POINTER`

Related Functions

`tt_message_status_string()`

tt_message_status_set

```
Tt_status       tt_message_status_set(Tt_message m,
                int status)
```

Sets the status attribute for the specified message.

Note – The status value must be greater than 2047 (TT_ERR_LAST) to avoid confusion with the ToolTalk service status values.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int status`
The status to be stored in this message.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_POINTER`

tt_message_status_string

```
char        *tt_message_status_string(Tt_message m)
```

Retrieves the character string stored with the status attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`char *`
The status string stored in this message.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_POINTER`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

`tt_message_status()`

tt_message_status_string_set

Tt_status tt_message_status_string_set(Tt_message m,
const char *status_str)

Sets a character string with the status attribute for the specified message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *status_str`
The status string stored in this message.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_POINTER`

Related Functions

`tt_message_status_set()`

tt_message_uid

uid_t tt_message_uid(Tt_message m)

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.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

Returned Value

`uid_t`
The user identifier of the message. If the group *65534* is returned, the message handle is not valid.

Related Functions

`tt_message_gid()`

tt_message_user

```
void        *tt_message_user(Tt_message m, int key)
```

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. User data can only be retrieved by the same process, using the same procid, that put the data on the message.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int key`
The user data cell to be retrieved. The user data cell must be unique for this message.

Returned Value

`void *`
A piece of arbitrary user data that is one word in size.

tt_message_user_set

```
Tt_status       tt_message_user_set(Tt_message m, int key,
                void *v)
```

Stores user information in data cells associated with the specified message object.

Note – 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. For data that needs to be seen by handlers or observers, use arguments for that data.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`int key`
The user data cell in which user information is to be stored.

`void *v`
A piece of arbitrary user data that is one word in size.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

`tt_message_arg_add()`

tt_message_xarg_add

```
Tt_status       tt_message_xarg_add(Tt_message m, Tt_mode n,
                const char *vtype, xdrproc_t xdr_proc,
                void *value)
```

Adds an argument with an XDR-interpreted value to a message object.

To change existing argument values, only use mode `TT_OUT` or `TT_INOUT`.

Note – Do *not* add arguments to a reply.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`Tt_mode n`
Specifies who (sender, handler, observers) writes and reads a message argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

`const char *vtype`
Describes the type of argument data being added.

`xdrproc_t xdr_proc`
Points to the XDR value to be used to serialize the data pointed to by value.

`void *value`
The data to be serialized.

Returned Values

`Tt_status`
The status of the operation. Possible values are shown in Table 4-16.

*Table 4-16*Possible Status of tt_message_xarg_add Call

Returned Value	Description
TT_OK	Operation was successful.
TT_ERR_MODE	Invalid mode value.
TT_ERR_NOMP	ToolTalk is not initialized.
TT_ERR_POINTER	Invalid message handle, XDR proc pointer, or data pointer.
TT_ERR_XDR	The XDR procedure failed on the given data, or evaluated to a 0 length structure.

tt_message_xcontext_set

```
Tt_status       tt_message_xcontext_set(Tt_message m, const char
*slotname,
                xdrproc_t xdr_proc, void *value)
```

Sets the XDR-interpreted byte-array value of a message's context.

Arguments

`Tt_message m`
The opaque handle for the message involved in this operation.

`const char *slotname`
Describes the slotname in this message.

`const char *value`
The byte string with the contents for the message argument.

`xdrproc_t xdr_proc`
Points to the XDR value to be used to serialize the data pointed to by value.

`void *value`
The data to be serialized.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-17.

*Table 4-17*Possible Status of tt_message_xcontext_set Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_NOMP	The ToolTalk service is not initialized.

*Table 4-17*Possible Status of tt_message_xcontext_set Call

Value Returned	Description
TT_ERR_POINTER	The Tt_message handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.
TT_ERR_XDR	The XDR procedure failed on the given data, or evaluated to a 0 length structure.

tt_objid_equal

```
int         tt_objid_equal(const char *objid1,
            const char *objid2)
```

Tests whether two objids are equal.

The `tt_objid_equal()` function is recommended rather than `strcmp` for this purpose because the `tt_objid_equal()` function returns 1 even in the case where one objid is a forwarding pointer for the other.

Arguments

`const char *objid1`
The identifier of the first object involved in this operation.

`const char *objid2`
The identifier of the second object involved in this operation.

Returned Value

`int`
The integer that indicates whether the objids are equal. Possible values are:

- `0 – no`
- `1 – yes`

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_PROCID`

tt_objid_objkey

```
char        *tt_objid_objkey(const char *objid)
```

Returns the unique key of a objid.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

Returned Value

`char *`
The unique key of the objid. No two objids have the same unique key.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_OBJID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_onotice_create

```
Tt_message    tt_onotice_create (const char *objid,
                                 const char *op)
```

Creates a message.

The message created contains the following:

- `Tt_address = TT_OBJECT`
- `Tt_class = TT_NOTICE`

Use the returned handle to add arguments and other attributes, and to send the message.

Arguments

`const char *objid`
The identifier of the specified object.

`const char *op`
The operation to be performed by the receiving process.

Returned Value

`Tt_message`
The unique handle that identifies the message.

Use `tt_ptr_error()` to determine if the handle is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_open

char * tt_open(void)

Returns the process identifier for the calling process.

This function also sets this identifier as the default procid for the process. `tt_open()` is typically the first ToolTalk function which a process calls.

Caution – Your application must call `tt_open()` before other `tt_` calls are made; otherwise, errors may occur. However, there are two exceptions: `tt_default_session_set()` and `tt_X_session()` can be called before `tt_open()`.

A process may call `tt_open()` more than once to obtain multiple procids. To open another session, make the following calls in the order specified:

- `tt_default_session_set()`
- `tt_open()`

Each procid has its own associated file descriptor, and can join another session. To switch to another procid, call `tt_default_procid_set()`.

Returned Value

`char *`
The character value that uniquely identifies the process.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

```
tt_fd()
tt_default_procid
tt_default_procid_set()
tt_default_session()
tt_default_session_set()
```

tt_orequest_create

```
Tt_message    tt_orequest_create(const char *objid,
                                 const char *op)
```

Creates a message.

The message created contains the following:

- `Tt_address = TT_OBJECT`
- `Tt_class = TT_REQUEST`

Use the returned handle to add arguments and other attributes, and to send the message.

Arguments

`const char *objid`
The identifier of the specified object.

`const char *op`
The operation to be performed by the receiving process.

Returned Value

`Tt_message`
The unique handle that identifies the message.

Use `tt_ptr_error()` to determine if the handle is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_otype_base

```
char        *tt_otype_base(const char *otype)
```

Returns the base otype of the given otype.

`NULL` is returned if the given otype is not derived.

Arguments

`char *otype`
The object type involved in this operation.

Returned Value

`char *`
The name of the base otype; if the given otype is not derived, this value is `NULL`.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

`tt_otype_is_derived()`
`tt_otype_derived()`
`tt_otype_deriveds_count()`
`tt_spec_type()`
`tt_message_otype()`

tt_otype_derived

```
char        *tt_otype_derived(const char *otype, int i)
```

Returns the *ith* otype derived from the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

`int i`
The zero-based index into the otypes derived from the given otype.

Returned Value

`char *`
The name of the *ith* otype derived from the given otype.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

`tt_otype_is_derived()`
`tt_otype_base()`
`tt_otype_deriveds_count()`
`tt_spec_type()`
`tt_message_otype()`

tt_otype_deriveds_count

```
int          tt_otype_deriveds_count(const char *otype)
```

Returns the number of otypes derived from the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

Returned Value

`int`
The number of otypes derived from the given otype.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Related Functions

`tt_otype_is_derived()`
`tt_otype_base()`
`tt_otype_derived()`
`tt_spec_type()`
`tt_message_otype()`

tt_otype_hsig_arg_mode

```
Tt_mode     tt_otype_hsig_arg_mode(const char *otype,
            int sig, int arg)
```

Returns the mode of the arg'th argument of the sig'th request signature of the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

`int sig`
The zero-based index into the request signatures of the specified otype.

`int arg`
The zero-based index into the arguments of the specified signature.

Returned Value

`Tt_mode`
Determines who (sender or handler) writes and reads a message argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Related Functions

```
tt_otype_hsig_arg_type()
tt_otype_hsig_count()
tt_otype_hsig_args_count()
tt_otype_hsig_op()
```

tt_otype_hsig_arg_type

```
char        *tt_otype_hsig_arg_type(const char *otype,
            int sig, int arg)
```

Returns the data type of the arg'th argument of the sig'th request signature of the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

`int sig`
The zero-based index into the request signatures of the specified otype.

`int arg`
The zero-based index into the arguments of the specified signature.

Returned Value

`char *`
The data type of the specified argument.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

```
tt_otype_hsig_arg_mode()
tt_otype_hsig_count()
tt_otype_hsig_args_count()
tt_otype_hsig_op()
```

tt_otype_hsig_args_count

```
int     tt_otype_hsig_args_count(const char *otype,
        int sig)
```

Returns the number of arguments of the sig'th request signature of the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

`int sig`
The zero-based index into the request signatures of the specified otype.

Returned Value

`int`
The number of arguments of the sig'th request signature of the given otype.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Related Functions

`tt_otype_hsig_arg_type()`
`tt_otype_hsig_arg_mode()`
`tt_otype_hsig_count()`
`tt_otype_hsig_op()`

tt_otype_hsig_count

```
int             tt_otype_hsig_count(const char *otype)
```

Returns the number of request signatures for the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

Returned Value

`int`
The number of request signatures for the given otype.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Related Functions

```
tt_otype_hsig_arg_type()
tt_otype_hsig_arg_mode()
tt_otype_hsig_args_count()
tt_otype_hsig_op()
```

tt_otype_hsig_op

```
char        *tt_otype_hsig_op(const char *otype, int sig)
```

Returns the operation name of the sig'th request signature of the give otype.

Arguments

`const char *otype`
The object type involved in this operation.

`int sig`
The zero-based index into the request signatures of the given otype.

Returned Value

`char *`
The operation attribute of the specified request signature.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

```
tt_otype_hsig_arg_type()
tt_otype_hsig_arg_mode()
tt_otype_hsig_args_count()
tt_otype_hsig_count()
```

tt_otype_is_derived

```
int     tt_otype_is_derived(const char *derivedotype,
        const char *baseotype)
```

Specifies whether derived otype is derived directly or indirectly from base otype.

Arguments

`const char *derivedotype`
The specified derived otype.

`const char *baseotype`
The specified base otype.

Returned Value

`int`
Returns 1 only if *derivedotype* is derived directly or indirectly from *baseotype*.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Related Functions

`tt_otype_deriveds_count()`
`tt_otype_base()`
`tt_otype_derived()`
`tt_spec_type()`
`tt_message_otype()`

tt_otype_opnum_callback_add

```
Tt_status       tt_otype_opnum_callback_add(const char *otid,
                int opnum, Tt_message_callback f)
```

Automatically returns a callback if the specified opnums are equal.

Note – Callbacks are called in reverse order of registration (for example, the most recently added callback is called first).

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 given callback is run in the usual ToolTalk way. See the *ToolTalk User's Guide* for more information about callbacks.

Note – This function works only with handler signatures because the observer_ptype is not part of the message.

Arguments

`const char *otid`
The identifier of the object type involved in this operation.

`int opnum`
The opnum of the specified otype.

`Tt_message_callback f`
The message callback to be run.

Returned Value

`Tt_status`
Status of the operation. Possible values are shown in Table 4-18.

*Table 4-18*Possible Status tt_otype_opnum_callback_add Call

Value Returned	Description
TT_OK	The operation was successful.
TT_ERR_OTYPE	Invalid type name.
TT_ERR_POINTER	Invalid (null) callback function pointer.
TT_ERR_NOMP	ToolTalk is not initialized.

tt_otype_osig_arg_mode

```
Tt_mode     tt_otype_osig_arg_mode(const char *otype,
            int sig, int arg)
```

Returns the mode of the arg'th argument of the sig'th notice signature of the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

`int sig`
The zero-based index into the notice signatures of the specified otype.

`int arg`
The zero-based index into the arguments of the specified signature.

Returned Value

`Tt_mode`
Determines who (sender or handler) writes and reads a message argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

Use `tt_int_error()` to determine if the `Tt_mode` value is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Related Functions

```
tt_otype_osig_arg_type()
tt_otype_osig_count()
tt_otype_osig_args_count()
tt_otype_osig_op()
```

tt_otype_osig_arg_type

```
char        *tt_otype_osig_arg_type(const char *otype,
            int sig, int arg)
```

Returns the data type of the arg'th argument of the sig'th notice signature of the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

`int sig`
The zero-based index into the notice signatures of the specified otype.

`int arg`
The zero-based index into the arguments of the specified signature.

Returned Value

`char *`
The data type of the specified argument.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

```
tt_otype_osig_arg_mode()
tt_otype_osig_count()
tt_otype_osig_args_count()
tt_otype_osig_op()
```

tt_otype_osig_args_count

```
int     tt_otype_osig_args_count(const char *otype,
        int sig)
```

Returns the number of arguments of the sig'th notice signature of the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

`int sig`
The zero-based index into the notice signatures of the specified otype.

Returned Value

`int`
The number of arguments of the sig'th notice signature of the given otype.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Related Functions

`tt_otype_osig_arg_type()`
`tt_otype_osig_arg_mode()`
`tt_otype_osig_count()`
`tt_otype_osig_op()`

tt_otype_osig_count

```
int          tt_otype_osig_count(const char *otype)
```

Returns the number of notice signatures for the given otype.

Arguments

`const char *otype`
The object type involved in this operation.

Returned Value

`int`
The number of notice signatures for the given otype.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Related Functions

```
tt_otype_osig_arg_type()
tt_otype_osig_arg_mode()
tt_otype_osig_args_count()
tt_otype_osig_op()
```

tt_otype_osig_op

```
char        *tt_otype_osig_op(const char *otype, int sig)
```

Returns the op name of the sig'th notice signature of the give otype.

Arguments

`const char *otype`
The object type involved in this operation.

`int sig`
The zero-based index into the notice signatures of the given otype.

Returned Value

`char *`
The operation attribute of the specified notice signature.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

```
tt_otype_osig_arg_type()
tt_otype_osig_arg_mode()
tt_otype_osig_args_count()
tt_otype_osig_count()
```

tt_pattern_address_add

```
Tt_status       tt_pattern_address_add(Tt_pattern p,
                Tt_address d)
```

Adds a value to the address field for the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after a `tt_pattern_create()` call has been made.

`Tt_address d`
Specifies which pattern attributes form the address that messages will be matched against. Possible values are:

- `TT_PROCEDURE`
- `TT_OBJECT`
- `TT_HANDLER`
- `TT_OTYPE`

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_pattern_arg_add

```
Tt_status       tt_pattern_arg_add(Tt_pattern p, Tt_mode n,
                const char *vtype, const char *value)
```

Adds an argument to a pattern.

Add pattern arguments before you register the pattern with the ToolTalk service.

Arguments

`Tt_pattern p`
The opaque handle for the pattern involved in this operation

`Tt_mode n`
Specifies who (sender, handler, observers) writes and reads a message argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

`const char *vtype`
Describes the type of argument data being added. To match any argument value type, use type `ALL`.

`const char *value`
The value to fill in. This value must be an unsigned character string. To specify that any value matches, fill in the value as `NULL`.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

Related Functions

```
tt_pattern_register()
tt_pattern_barg_add()
tt_pattern_iarg_add()
```

tt_pattern_barg_add

Tt_status tt_pattern_barg_add(Tt_pattern m, Tt_mode n,
const char *vtype, const unsigned char *value,
int len)

Adds an argument with a value that contains imbedded nulls to a pattern.

Arguments

`Tt_pattern m`
The opaque handle for the pattern involved in this operation.

`Tt_mode n`
Specifies who (sender, handler, observers) writes and reads a message argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

`const char *vtype`
Describes the type of argument data being added. To match any argument value type, use type `ALL`.

The ToolTalk service treats the value as an opaque byte string. To pass structured data, your application and the receiving application must encode and decode these unique values. The most common method to use is XDR.

`const unsigned char *value`
The value to be filled in. To specify that any value matches, fill in the value as `NULL`.

`int len`
The length of the value in bytes.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

Related Functions

`tt_pattern_register()`
`tt_pattern_arg_add()`
`tt_pattern_iarg_add()`

tt_pattern_bcontext_add

```
Tt_status       tt_pattern_bcontext_add(Tt_pattern p,
                const char *slotname, const unsigned char *value,
                int length);
```

Adds a byte-array value to the values in this pattern's named context.

Arguments

`Tt_pattern p`
The opaque handle for the pattern involved in this operation.

`const char *slotname`
Describes the context for this pattern.

`const unsigned char *value`
The byte string with the contents for the message context. To specify that any value matches, use the form

```
tt_pattern_arg_add(p, mode, type, NULL)
```

`int length`
The length of the value in bytes.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-19.

*Table 4-19*Possible Status of tt_pattern_bcontext_add Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_UNIMP	The function called is not implemented.

*Table 4-19*Possible Status of tt_pattern_bcontext_add Call

Value Returned	Description
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_POINTER	The Tt_pattern handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_pattern_callback_add

Tt_status tt_pattern_callback_add(Tt_pattern m,
Tt_message_callback f)

Registers a callback function that will be automatically invoked by `tt_message_receive()` whenever a message matches the pattern.

Note – Callbacks are called in reverse order of registration (for example, the most recently added callback is called first).

`Tt_callback_action` is an `enum` that contains the values `TT_CALLBACK_CONTINUE` and `TT_CALLBACK_PROCESSED`.

- 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 will be invoked for this event; `tt_message_receive()` does not return the message.

Arguments

`Tt_pattern m`
The opaque handle for the pattern involved in this operation.

`Tt_message_callback f`
Passes the specified message and the pattern that matched it to the callback.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

Related Functions

`tt_pattern_register()`

tt_pattern_category

```
Tt_category     tt_pattern_category(Tt_pattern p)
```

Returns the category value of the specified pattern.

Arguments

`Tt_pattern p`
The opaque handle for a message pattern.

Returned Value

`Tt_category`
Indicates whether the receiving process will observe or handle messages. Possible values are:

- `TT_OBSERVE`
- `TT_HANDLE`

Use `tt_int_error()` to determine if the `Tt_category` integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

Related Functions

`tt_pattern_category_set()`

tt_pattern_category_set

```
Tt_status       tt_pattern_category_set(Tt_pattern p,
                Tt_category c)
```

Fills in the category field for the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`Tt_category c`
Indicates whether the receiving process will observe or handle messages. Possible values are:

- `TT_OBSERVE`
- `TT_HANDLE`

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_CATEGORY`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

Related Functions

`tt_pattern_category()`

tt_pattern_class_add

Tt_status tt_pattern_class_add(Tt_pattern p,
Tt_class c)

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.

Arguments

Tt_pattern p
A unique handle for a message pattern. This handle is returned after tt_pattern_create() is called.

Tt_class c
Indicates whether the receiving process is to take action after the message is received. Possible values are:

- TT_NOTICE
- TT_REQUEST

Returned Value

Tt_status
The status of the operation. Possible values are:

- TT_OK
- TT_ERR_NOMP
- TT_ERR_POINTER

tt_pattern_context_add

```
Tt_status       tt_pattern_context_add(Tt_pattern p,
                const char *slotname, const char *value);
```

Adds a string value to the values of this pattern's context.

Note – If the value pointer is null, a slot is created with the specified name but no value is added.

Arguments

`Tt_pattern p`
The opaque handle for the pattern involved in this operation.

`const char *slotname`
Describes the context of this pattern.

`const char *value`
The value to be added.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-20.

Table 4-20 Possible Status of tt_pattern_context_add Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_UNIMP	The function called is not implemented.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_POINTER	The Tt_pattern handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_pattern_create

Tt_pattern tt_pattern_create(void)

Requests a new pattern object.

After receiving the pattern object, fill in the message pattern fields to indicate what type of messages to your process wants to receive and then register the pattern with the ToolTalk service.

Note – You can supply multiple values for each attribute you add 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.

Returned Value

`Tt_pattern`
The opaque handle for a message pattern. Use this handle in future calls to identify the pattern object.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

`tt_pattern_register()`

tt_pattern_destroy

Tt_status tt_pattern_destroy(Tt_pattern p)

Destroys a pattern object.

Destroying a pattern object automatically unregisters the pattern with the ToolTalk service.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

`tt_pattern_register()`

tt_pattern_disposition_add

```
Tt_status       tt_pattern_disposition_add(Tt_pattern p,
                Tt_disposition r)
```

Adds a value to the disposition field for the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

Tt_disposition r
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. Possible values are:

- `TT_DISCARD`
- `TT_QUEUE`
- `TT_START`
- `TT_QUEUE+TT_START`

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_pattern_file_add

```
Tt_status       tt_pattern_file_add(Tt_pattern p,
                const char *file)
```

Adds a value to the file field of the specified pattern.

Note – Use this call to set individual files on individual patterns. However, this call does *not* cause the pattern's ToolTalk session to be stored in the database.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`const char *file`
The name of the file of the specified pattern.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_FILE`

tt_pattern_iarg_add

```
Tt_status       tt_pattern_iarg_add(Tt_pattern m, Tt_mode n,
                const char *vtype, int value)
```

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.

Arguments

`Tt_pattern m`
The opaque handle for the pattern involved in this operation.

`Tt_mode n`
Specifies who (sender, handler, observers) writes and reads a message argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

`const char *vtype`
Describes the type of argument data being added. To match any argument value type, use type `ALL`.

`int value`
The value to fill in.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_MODE`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_VTYPE`

Related Functions

`tt_pattern_register()`

tt_pattern_icontext_add

```
Tt_status       tt_pattern_icontext_add(Tt_pattern p,
                const char *slotname, int value);
```

Adds an integer value to the values of this pattern's context.

Arguments

`Tt_pattern p`
The opaque handle for the pattern involved in this operation.

`const char *slotname`
Describes the slotname in this pattern.

`int value`
The value to be added.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-21.

Table 4-21 Possible Status of tt_pattern_icontext_add Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_UNIMP	The function called is not implemented.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_POINTER	The Tt_pattern handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.

tt_pattern_object_add

```
Tt_status       tt_pattern_object_add(Tt_pattern p,
                const char *objid)
```

Adds a value to the object field of the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`const char *objid`
The identifier for the specified object. Both `tt_spec_create()` and `tt_spec_move()` return objids.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_POINTER`
- `TT_WRN_STALE_OBJID`

tt_pattern_op_add

```
Tt_status       tt_pattern_op_add(Tt_pattern p,
                const char *opname)
```

Adds a value to the operation field of the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`const char *opname`
The name of the operation your process can perform.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_pattern_opnum_add

Tt_status tt_pattern_opnum_add(Tt_pattern p, int opnum)

Adds an operation number to the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`int opnum`
The operation number to be added.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_pattern_otype_add

```
Tt_status       tt_pattern_otype_add(Tt_pattern p,
                const char *otype)
```

Adds a value to the object type field for the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`const char *otype`
The name of the object type the application manages.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_OTYPE`
- `TT_ERR_POINTER`

tt_pattern_register

Tt_status tt_pattern_register(Tt_pattern p)

Registers your pattern with the ToolTalk service.

When your 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.

Note – When your process joins a session or file, the ToolTalk service updates the file and session field of its registered patterns.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_CATEGORY`
- `TT_ERR_INVALID`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

`tt_pattern_unregister()`

tt_pattern_scope_add

```
Tt_status       tt_pattern_scope_add(Tt_pattern p,
                Tt_scope s)
```

Adds a value to the scope field for the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`Tt_scope s`
Specifies what process are eligible to receive the message. Possible values are:

- `TT_SESSION`
- `TT_FILE`
- `TT_BOTH`
- `TT_FILE_IN_SESSION`

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_pattern_sender_add

```
Tt_status       tt_pattern_sender_add(Tt_pattern p,
                const char *procid)
```

Adds a value to the sender field for the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`const char *procid`
The character value that uniquely identifies the process of interest.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_pattern_sender_ptype_add

```
Tt_status       tt_pattern_sender_ptype_add(Tt_pattern p,
                const char *ptid)
```

Adds a value to the sending process's ptype field for the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`const char *ptid`
The character string that uniquely identifies the type of process in which you are interested.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_pattern_session_add

```
Tt_status       tt_pattern_session_add(Tt_pattern p,
                const char *sessid)
```

Adds a value to the session field for the specified pattern.

Note – When your process joins a session, the ToolTalk service updates the session field of its registered patterns.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`const char *sessid`
The session of interest.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_SESSION`

tt_pattern_state_add

Tt_status tt_pattern_state_add(Tt_pattern p, Tt_state s)

Adds a value to the state field for the specified pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`Tt_state s`
Indicates the current delivery state of a message. Possible values are:

- `TT_CREATED`
- `TT_SENT`
- `TT_HANDLED`
- `TT_FAILED`
- `TT_QUEUED`
- `TT_STARTED`
- `TT_REJECTED`

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_pattern_unregister

Tt_status tt_pattern_unregister(Tt_pattern p)

Unregisters the specified pattern from the ToolTalk service.

The process will stop receiving messages that match this pattern.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_INVALID`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

`tt_pattern_register()`

tt_pattern_user

void *tt_pattern_user(Tt_pattern p, int key)

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. Your tool can use these keys in any manner to associate arbitrary data with a pattern object.

Note – 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.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`int key`
The specified user data cell. To assign the keys to the user data cells which are part of the pattern object, use `tt_pattern_user_set()`. The value of each data cell must be unique for this pattern.

Returned Value

`void *`
A piece of arbitrary user data that is one word in size.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

`tt_pattern_user_set()`

tt_pattern_user_set

```
Tt_status       tt_pattern_user_set(Tt_pattern p, int key,
                void *v)
```

Stores information in the user data cells associated with the specified pattern object.

Arguments

`Tt_pattern p`
A unique handle for a message pattern. This handle is returned after `tt_pattern_create()` is called.

`int key`
The specified user data cell. The value for each data cell must be unique for this pattern.

`void *v`
A piece of arbitrary user data that is one word in size.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

Related Functions

`tt_pattern_user()`

tt_pattern_xarg_add

```
Tt_status       tt_pattern_xarg_add(Tt_pattern m, Tt_mode n,
                const char *vtype, xdrproc_t xdr_proc,
                void *value)
```

Adds a new argument with an interpreted XDR value to a pattern object.

Arguments

`Tt_pattern m`
The opaque handle for the pattern involved in this operation.

`Tt_mode n`
Specifies who (sender, handler, observers) writes and reads a pattern argument. Possible modes are:

- `TT_IN`
- `TT_OUT`
- `TT_INOUT`

`const char *vtype`
Describes the type of argument data being added.

`xdrproc_t xdr_proc`
Points to the XDR procedure to be used to serialize the data pointed to by value.

`void *value`
The data to be serialized.

Returned Values

`Tt_status`
The status of the operation. Possible values are shown in Table 4-22.

*Table 4-22*Possible Status of tt_pattern_xarg_add Call1

Returned Value	Description
TT_OK	Operation was successful.
TT_ERR_MODE	Invalid mode value.
TT_ERR_NOMP	ToolTalk is not initialized.
TT_ERR_POINTER	Invalid message handle, XDR proc pointer, or data pointer.
TT_ERR_XDR	The XDR procedure failed on the given data, or evaluated to a 0 length structure.

tt_pattern_xcontext_add

Tt_status tt_pattern_xcontext_add(Tt_pattern p, const char *slotname, xdrproc_t xdr_proc, void *value)

Adds an XDR-interpreted byte-array value to the values in this pattern's named context.

Arguments

`Tt_pattern p`
The opaque handle for the pattern involved in this operation.

`const char *slotname`
Describes the context for this pattern.

`xdrproc_t xdr_proc`
Points to the XDR procedure to be used to serialize the data pointed to by value.

`void *value`
The data to be serialized.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-23.

Table 4-23 Possible Status of tt_pattern_xcontext_add Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_UNIMP	The function called is not implemented.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_POINTER	The Tt_pattern handle is not valid.
TT_ERR_SLOTNAME	The slotname is not valid.
TT_ERR_XDR	The XDR procedure failed on the given data, or evaluated to a 0 length structure.

tt_pnotice_create

```
Tt_message    tt_pnotice_create(Tt_scope scope,
              const char *op)
```

Creates a message.

The message created contains the following:

- Tt_address = TT_PROCEDURE
- Tt_class = TT_NOTICE

Use the returned handle to add arguments and other attributes, and to send the message.

Arguments

`Tt_scope scope`
Determine which processes are eligible to receive the message. Possible values are:

- `TT_SESSION`
- `TT_FILE`
- `TT_BOTH`
  `TT_FILE_IN_SESSION`

- If the scope is `TT_SESSION`, the session is set to the current default session.
- If the scope is `TT_FILE`, the file is set to the current default file.
- If the scope is `BOTH` or `FILE_IN_SESSION`, both file and session are set to the defaults.

`const char *op`
The operation to be performed by the receiving process.

Returned Value

`Tt_message`
The unique handle that identifies this message.

If the ToolTalk service is unable to create a message when requested, an invalid handle is returned. If you attempt to use this handle, the ToolTalk service reports an error.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_pointer_error

Tt_status tt_pointer_error(void *pointer)

Returns the status of 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.

Note – A macro `tt_ptr_error(p)` is provided that expands to `tt_pointer_error((void *)(p))`.

Arguments

`void *pointer`
The opaque pointer or character pointer to be checked.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_prequest_create

```
Tt_message    tt_prequest_create(Tt_scope scope,
              const char *op)
```

Creates a message.

The message created contains the following:

- Tt_address = TT_PROCEDURE
- Tt_class = TT_REQUEST

Use the returned handle to add arguments and other attributes, and to send the message.

Arguments

`Tt_scope scope`
Determine which processes are eligible to receive the message. Possible values are:

- `TT_SESSION`
- `TT_FILE`
- `TT_BOTH`
  `TT_FILE_IN_SESSION`

- If the scope is `TT_SESSION`, the session is set to the current default session.
- If the scope is `TT_FILE`, the file is set to the current default file.
- If the scope is `BOTH` or `FILE_IN_SESSION`, both file and session are set to the defaults.

`const char *op`
The operation to be performed by the receiving process.

Returned Value

`Tt_message`
The unique handle that identifies this message.

If the ToolTalk service is unable to create a message when requested, an invalid handle is returned. If you attempt to use this handle, the ToolTalk service reports an error.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PROCID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_ptr_error

Tt_status tt_ptr_error(pointer)

Returns the status of specified pointer.

This function is a macro that expands to `tt_pointer_error`.

Arguments

`pointer`
The opaque pointer or character pointer to be checked.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`

tt_ptype_declare

Tt_status tt_ptype_declare(const char *ptid)

Registers your process type with the ToolTalk service.

Arguments

`const char *ptid`
The character string specified in the ptype that uniquely identifies this process.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PTYPE`

tt_ptype_exists

Tt_status tt_ptype_exists(const char *ptid)

Returns whether indicated ptype is already installed.

Arguments

`const char *ptid`
The character string specifying the ptype.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PTYPE`

tt_ptype_opnum_callback_add

```
Tt_status       tt_ptype_opnum_callback_add(const char *ptid,
                int opnum, Tt_message_callback f)
```

Automatically returns a callback if the specified opnums are equal.

Note – Callbacks are called in reverse order of registration (for example, the most recently added callback is called first).

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. See the *ToolTalk User's Guide* for more information about callbacks.

Note – This function works only with handler signatures because the observer_ptype is not part of the message.

Arguments

`const char *ptid`
The identifier of the ptype involved in this operation.

`int opnum`
The opnum of the specified ptype.

`Tt_message_callback f`
The message callback to be run.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-24.

*Table 4-24*Possible Status of tt_optype_opnum_callback_add Call

Value Returned	Description
TT_OK	Operation was successful.
TT_ERR_PTYPE	Invalid type name.
TT_ERR_POINTER	Invalid (or null) callback function pointer.
TT_ERR_NOMP	ToolTalk is not initialized.

tt_ptype_undeclare

Tt_status tt_ptype_undeclare(const char *ptid)

Undeclares the indicated ptype.

This function unregisters the patterns associated with the indicated ptype from the ToolTalk service.

Arguments

`const char *ptid`
The character string specifying the ptype.

Returned Value

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_PTYPE`

tt_release

void tt_release(int mark)

Frees storage allocated on the ToolTalk API allocation stack.

This function frees storage allocated since the given mark was made, and is typically called at the end of a procedure to release all storage allocated within the procedure.

Arguments

`int mark`
An integer that marks the application's storage position in the ToolTalk API allocation stack.

Related Functions

`tt_mark()`

tt_session_bprop

```
Tt_status       tt_session_bprop(const char *sessid,
                const char *propname, int i,
                unsigned char **value, int *length)
```

Retrieves the *ith* 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.

Arguments

`const char *sessid`
The session joined. Use the sessid value returned when `tt_default_session()` is called.

`const char *propname`
The name of the property from which values are to be obtained.

`int i`
The number of the item in the property list from which the value is to be obtained. The list numbering begins with 0.

`unsigned char **value`
The address of a character pointer to which the ToolTalk service is to point a string that contains the contents of the property.

`int *len`
The address of an integer to which the ToolTalk service is to set the length of the value in bytes.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`

- TT_ERR_NUM
- TT_ERR_POINTER
- TT_ERR_PROCID
- TT_ERR_PROPNAME
- TT_ERR_SESSION

`unsigned char **value`
The address of a character pointer to which the ToolTalk service is to point a string that contains the contents of the property.

`int *len`
The address of an integer to which the ToolTalk service is to set the length of the value in bytes.

tt_session_bprop_add

```
Tt_status       tt_session_bprop_add(const char *sessid,
                const char *propname,
                const unsigned char *value, int length)
```

Adds a new byte-string value to the end of the list of values for the named property of the specified session.

Arguments

`const char *sessid`
The name of the session joined. Use the sessid value returned when `tt_default_session()` is called.

`const char *propname`
The name of the property to which to add values.

`const unsigned char *value`
The value to add to the session property.

`int length`
The size of the value in bytes.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PROPLEN`
- `TT_ERR_PROPNAME`
- `TT_ERR_SESSION`

tt_session_bprop_set

```
Tt_status       tt_session_bprop_set(const char *sessid,
                const char *propname,
                const unsigned char *value, int length)
```

Replaces any current values stored under the named property of the specified session with the given byte-string value.

Arguments

`const char *sessid`
The name of the session joined. Use the sessid value returned when `tt_default_session()` is called.

`const char *propname`
The name of the property whose value is to be replaced.

`const unsigned char *value`
The value to which the session property is set. If value is `NULL`, the property is removed entirely.

`int length`
The size of the value in bytes.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PROPLEN`
- `TT_ERR_PROPNAME`
- `TT_ERR_SESSION`

tt_session_join

Tt_status tt_session_join(const char *sessid)

Joins the session named and makes it the default session.

Arguments

`const char *sessid`
The name of the session to join. Use the sessid value returned by `tt_default_session()`, `tt_X_session()`, or `tt_initial_session()`.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_PATH`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`

Related Functions

`tt_default_session()`

tt_session_prop

```
char        *tt_session_prop(const char *sessid,
            const char *propname, int i)
```

Returns the *ith* value of the specified session property.

Note – If this value has embedded nulls, you will not be able to determine how long it is. Use `tt_session_bprop()` for values with embedded nulls.

Arguments

`const char *sessid`
The name of the session joined. Use the sessid value returned when `tt_default_session()` is called.

`const char *propname`
The name of the property from which a value is to be retrieved. The name must be less than 64 characters.

`int i`
The number of the item in the property name list for which the value is to be obtained. The list numbering begins with 0.

Returned Value

`char *`
The value of the requested property. If there are *i* values or fewer, `NULL` is returned.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_PROPNAME`
- `TT_ERR_SESSION`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

tt_session_prop_add

```
Tt_status       tt_session_prop_add(const char *sessid,
                const char *propname, const char *value)
```

Adds a new character-string value to the end of the list of values for the property of the specified session.

Arguments

`const char *sessid`
The name of the session joined. Use the sessid value returned when `tt_default_session()` is called.

`const char *propname`
The name of the property to which a value is to be added. The name must be less than 64 characters.

`const char *value`
The character string to add to the property name list.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PROPLEN`
- `TT_ERR_PROPNAME`
- `TT_ERR_SESSION`

tt_session_prop_count

```
int     tt_session_prop_count(const char *sessid,
        const char *propname)
```

Returns the number of values stored under the named property of the specified session.

Arguments

`const char *sessid`
The name of the session joined. Use the sessid value returned when `tt_default_session()` is called.

`const char *propname`
The name of the property to be examined.

Returned Value

`int`
The number of values in the specified property list.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PROPNAME`
- `TT_ERR_SESSION`

tt_session_prop_set

```
Tt_status       tt_session_prop_set(const char *sessid,
                const char *propname, const char *value)
```

Replaces all current values stored under the named property of the specified session with the given character-string value.

Arguments

`const char *sessid`
The name of the session joined. Use the sessid value returned when `tt_default_session()` is called.

`const char *propname`
The name of the property to be examined.

`const char *value`
The new value to be inserted. To remove a value from the property list, specify the value as `NULL`.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PROPLEN`
- `TT_ERR_PROPNAME`
- `TT_ERR_SESSION`

tt_session_propname

```
char        *tt_session_propname(const char *sessid,
            int n)
```

Returns the *nth* element of the list of currently-defined property names for the specified session.

Arguments

`const char *sessid`
The name of the session joined. Use the sessid value returned when `tt_default_session()` is called.

`int n`
The number of the item in the property name list for which a name is to be obtained. The list numbering begins with 0.

Returned Value

`char *`
The name of the specified property from the session property list. If there are *n* properties or fewer, `NULL` is returned.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_SESSION`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_session_propnames_count

```
int         tt_session_propnames_count(
            const char *sessid)
```

Returns the number of currently-defined property names for the session.

Arguments

`const char *sessid`
Name of the session joined. Use the sessid value returned when `tt_default_session()` is called.

Returned Value

`int`
The number of property names for the session.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_SESSION`

tt_session_quit

Tt_status tt_session_quit(const char *sessid)

Informs the ToolTalk service that the process is no longer interested in this ToolTalk session.

The ToolTalk service will stop delivering messages scoped to this session.

Arguments

`const char *sessid`
The name of the session to quit.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_NOMP`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_SESSION`
- `TT_WRN_NOTFOUND`

tt_session_types_load

Tt_status tt_session_types_load(const char *session, const char *filename)

Merges a compiled ToolTalk types file into the running ttsession.

Arguments

`const char *session`
The name of the running session.

`const char *filename`
The name of the compiled ToolTalk types file.

Returned Values

`Tt_status`
The status of the operation. Possible values are shown in Table 4-25.

Table 4-25 Possible Status of tt_session_types_load Call

Value Returned	Description
TT_OK	Operation was successful.
TT_ERR_NOMP	ToolTalk is not initialized.
TT_ERR_SESSION	The named session is not the name of the session for the current default procid.
TT_ERR_FILE	The file either could not be opened for reading or it is not a compiled ToolTalk types file.
TT_ERR_UNIMP	The requested operation is not implemented

tt_spec_bprop

```
Tt_status       tt_spec_bprop (const char *objid,
                const char *propname, int i,
                unsigned char **value, int *length)
```

Retrieves the *ith* value of the specified property.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`const char *propname`
The name of the property whose value is to be retrieved. The name must be less than 64 characters.

`int i`
The item of the list for which a value is to be obtained. The list numbering begins with 0.

`unsigned char **value`
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 0.

`int *len`
The address of an integer to which the ToolTalk service is to set the length of the value in bytes.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OBJID`
- `TT_ERR_PROPNAME`

`unsigned char **value`
The address of a character pointer to which the ToolTalk service is to point a string that contains the contents of the property. If there are *i* values or fewer, the pointer is set to 0.

`int *len`
The address of an integer to which the ToolTalk service is to set the length of the value in bytes. If there are *i* values or fewer, the length is 0.

tt_spec_bprop_add

```
Tt_status       tt_spec_bprop_add(const char *objid,
                const char *propname,
                const unsigned char *value, int length)
```

Adds a new byte-string to the end of the list of values associated with the specified spec property.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`const char *propname`
The name of the property to which the byte-string is to be added.

`const unsigned char *value`
The byte-string to be added to the property value list.

`int length`
The length of the byte-string.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PROPLEN`
- `TT_ERR_PROPNAME`
- `TT_WRN_STALE_OBJID`

tt_spec_bprop_set

```
Tt_status       tt_spec_bprop_set(const char *objid,
                const char *propname,
                const unsigned char *value, int length)
```

Replaces any current values stored under this spec property with a new byte-string.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`const char *propname`
The name of the property which stores the values.

`const unsigned char *value`
The byte-string to be added to the property value list.

Note – If the value is `NULL`, the property is removed entirely.

`int length`
The length of the value in bytes.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PROPLEN`
- `TT_ERR_PROPNAME`
- `TT_WRN_STALE_OBJID`

tt_spec_create

```
char            *tt_spec_create(const char *filepath)
```

Creates a spec (in memory) for an object.

Use the objid returned in future calls to manipulate the object.

Note – To make the object a permanent ToolTalk item or visible to other processes, the creating process must call `tt_spec_write()`.

Arguments

`const char *filepath`
The name of the file.

Returned Value

`char *`
The identifier for this object.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OTYPE`
- `TT_ERR_PATH`
- `TT_ERR_PROCID`
- `TT_WRN_STALE_OBJID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

Related Functions

```
tt_spec_type_set()
tt_spec_write()
```

tt_spec_destroy

Tt_status tt_spec_destroy(const char *objid)

Destroys an object's spec immediately.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_PROCID`
- `TT_WRN_STALE_OBJID`

tt_spec_file

```
char            *tt_spec_file(const char *objid)
```

Retrieves the name of the file that contains the object described by the spec.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

Returned Value

`char *`
The absolute pathname of the file.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_PROCID`
- `TT_WRN_STALE_OBJID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_spec_move

```
char        *tt_spec_move(const char *objid,
            const char *newfilepath)
```

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.

Note – If a new objid is not required (for example, because the new and old files are in the same file system), `TT_WRN_SAME_OBJID` is returned.

For efficiency and reliability, replace any references in your application to the old objid with references to the new one.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`const char *newfilepath`
The new file name.

Returned Value

`char *`
The new unique identifier of the object involved in this operation.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_PATH`
- `TT_ERR_PROCID`
- `TT_WRN_STALE_OBJID`

- TT_WRN_SAME_OBJID

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

tt_spec_prop

```
char        *tt_spec_prop(const char *objid,
            const char *propname, int i)
```

Retrieves the *ith* value of the property associated with this object spec.

Note – If this value has embedded nulls, its length cannot be determined.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`const char *propname`
The name of the property associated with the object spec.

`int i`
The item of the list whose value is to be retrieved. The list numbering begins with 0.

Returned Value

`char *`
The contents of the property value. If there are *i* values or less, a value of `NULL` is returned.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OBJID`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PROPNAME`

- TT_WRN_STALE_OBJID

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

tt_spec_prop_add

```
Tt_status       tt_spec_prop_add(const char *objid,
                const char *propname, const char *value)
```

Adds a new item to the end of the list of values associated with this spec property.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`const char *propname`
The property to which the item is to be added.

`const char *value`
The new character-string to be added to the property value list.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_POINTER`
- `TT_ERR_PROPNAME`
- `TT_ERR_PROPLEN`
- `TT_ERR_PROCID`

Related Functions

`tt_spec_prop_set()`

tt_spec_prop_count

```
int         tt_spec_prop_count(const char *objid,
            const char *propname)
```

Returns the number of values listed in this spec property.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`const char *propname`
The name of the property which contains the value to be returned.

Returned Value

`int`
The number of values listed in the spec property.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_PROCID`
- `TT_ERR_PROPNAME`

tt_spec_prop_set

```
Tt_status       tt_spec_prop_set(const char *objid,
                const char *propname, const char *value)
```

Replaces any values currently stored under this property of the object spec with a new value.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`const char *propname`
The name of the property which stores the values.

`const char *value`
The value to be placed in the property value list. If value is `NULL`, the property is removed entirely.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_POINTER`
- `TT_ERR_PROCID`
- `TT_ERR_PROPNAME`
- `TT_ERR_PROPLEN`
- `TT_WRN_STALE_OBJID`

Related Functions

`tt_spec_prop_add()`

tt_spec_propname

```
char        *tt_spec_propname(const char *objid, int n)
```

Returns the *nth* element of the property name list for this object spec.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`int n`
The item of the list whose element is to be returned. The list numbering begins with 0.

Returned Value

`char *`
The property name. If there are *n* properties or less, `NULL` is returned.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_NUM`
- `TT_ERR_OBJID`
- `TT_ERR_PROCID`
- `TT_WRN_STALE_OBJID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_spec_propnames_count

```
int             tt_spec_propnames_count(const char *objid)
```

Returns the number of property names for this object.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

Returned Value

`int`
The number of values listed in the spec property.

Use `tt_int_error()` to determine if the integer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_PROCID`
- `TT_WRN_STALE_OBJID`

tt_spec_type

```
char            *tt_spec_type(const char *objid)
```

Returns the name of the object type.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

Returned Value

`char *`
The type of this object.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_PROCID`
- `TT_WRN_STALE_OBJID`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_spec_type_set

```
Tt_status       tt_spec_type_set(const char *objid,
                const char *otid)
```

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 changed thereafter.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

`const char *otid`
The otype to be assigned to the spec.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_PROCID`
- `TT_ERR_READONLY`
- `TT_WRN_STALE_OBJID`

Related Functions

`tt_spec_create()`
`tt_spec_write()`

tt_spec_write

Tt_status tt_spec_write(const char *objid)

Writes the spec and any associated properties to the ToolTalk database.

It is not necessary to perform a write operation after a destroy operation.

Note – The type must be set *before* the spec is written for the first time.

Several changes can be batched between write calls; for example, you can create an object spec, set some properties, and then write all the changes at once with one write call.

Arguments

`const char *objid`
The identifier of the object involved in this operation.

Returned Values

`Tt_status`
The status of the operation. Possible values are:

- `TT_OK`
- `TT_ERR_DBAVAIL`
- `TT_ERR_DBEXIST`
- `TT_ERR_NOMP`
- `TT_ERR_OBJID`
- `TT_ERR_OTYPE`
- `TT_ERR_PROCID`
- `TT_WRN_STALE_OBJID`

Related Functions

`tt_spec_create()`
`tt_spec_type_set()`

tt_status_message

```
char        *tt_status_message(Tt_status ttrc)
```

Returns a pointer to a message that describes the problem indicated by this status code.

Arguments

`Tt_status ttrc`
The status code received during an operation.

Returned Value

`char *`
The pointer to character string that describes the status code.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_`*xxx*

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_X_session

```
char        *tt_X_session(const char *xdisplaystring)
```

Returns the session associated with the named X window system display.

Your application can make this call before it calls `tt_open()`.

Arguments

`const char *xdisplaystring`
The name of an X display server; for example, somehost:0, :0.

Returned Value

`char *`
The identifier for the current ToolTalk session.

Use `tt_ptr_error()` to determine if the pointer is valid. Possible `Tt_status` values that can be returned are:

- `TT_OK`
- `TT_ERR_SESSION`

Note – Use `tt_free()` to free any data stored in the address returned by the ToolTalk API.

---

tt_xcontext_join

```
Tt_status       tt_message_xcontext_join (const char *slotname,
                                          xdrproc_t xdr_proc, void *value);
```

Adds the given XDR-interpreted byte-array value to the list of values for the named contexts of all patterns.

Arguments

`const char *slotname`
Describes the slotname in this message.

`xdrproc_t xdr_proc`
Points to the XDR value to be used to serialize the data pointed to by value.

`void *value`
The data to be serialized.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-26.

Table 4-26 Possible Status of tt_xcontext_join Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_SLOTNAME	The slotname is not valid.
TT_ERR_XDR	The XDR procedure failed on the given data, or evaluated to a 0 length structure.

tt_xcontext_quit

Tt_status tt_xcontext_quit(const char *slotname,
xdrproc_t xdr_proc, void *value)

Removes the given XDR-interpreted byte-array value from the list of values for the contexts of all patterns.

Arguments

`const char *slotname`
Describes the slotname in this message.

`xdrproc_t xdr_proc`
Points to the XDR value to be used to serialize the data pointed to by value.

`void *value`
The data to be serialized.

Returned Value

`Tt_status`
The status of the operation. Possible values are shown in Table 4-27.

Table 4-27 Possible Status of tt_xcontext_quit Call

Value Returned	Description
TT_OK	Value returned is OK.
TT_ERR_NOMP	The ToolTalk service is not initialized.
TT_ERR_SLOTNAME	The slotname is not valid.
TT_ERR_XDR	The XDR procedure failed on the given data, or evaluated to a 0 length structure.

ToolTalk Commands 5

This chapter provides reference information for the ToolTalk-enhanced operating system commands.

ttce2xdr

ttce2xdr [-xn] -d *user*| *system*
ttce2xdr [-xn] -d *network* [*DTHOME-from* [*DTHOME-to*]]
ttce2xdr [-h]
ttce2xdr [-v]

Converts the ToolTalk Classing Engine type tables to XDR format.

Description

The `ttce2xdr` command converts the ToolTalk types stored in the Classing Engine database used by ToolTalk Versions 1.0.x to the XDR-format database used in ToolTalk Version 1.1 and later. This command generally only needs to be used to update types databases common to systems or the network; it is run automatically for user type databases.

The format to convert user or system data bases for the current user or current system is:

```
ttce2xdr [ -xn ] -d user| system
```

The format to convert the network-wide data base is:

```
ttce2xdr [ -xn ] -d network [ DTHOME-from [ DTHOME-to ]]
```

This format provides additional options allowing types to be taken from one database and stored into another.

Options

-d

Specifies the database (use, system, or network) to be converted. The types are read from the Classing Engine database (shown int Table 5-1) and written to the XDR database (shown in Table 5-2).

If the network database is specified, the optional arguments `DTHOME-from` and `DTHOME-to` may be specified.

- If neither argument is specified, ToolTalk uses the current value of the environment variable `DTHOME` to locate the data bases to be read from and written to.
- If only the `DTHOME-from` argument is specified, ToolTalk reads from and writes to the databases under the directory named by `DTHOME-from`.
- If both arguments are specified, ToolTalk reads from the database under `DTHOME-from` and writes to the database under `DTHOME-to`.

Table 5-1 Classing Engine Database

Database	Directory
user	~/.cetables/cetables
system	/etc/cetables/cetables
network	$DTHOME/lib/cetables/cetables

Table 5-2 XDR Database

Name	Database
user	~/.tt/types.xdr
system	/etc/tt/types.xdr
network	$DTHOME/etc/tt/types.xdr

-h
Describes the ttce2xdr options and exits.

-n
Displays the underlying commands that are to be executed by the `ttce2xdr` command.

-v
Prints version and exits.

-x
Displays the underlying commands executed by the `ttce2xdr` command.

Environment

CEPATH
In Classing Engine mode, `tt_type_comp` uses this variable for its definition of where the databases are located.

DTHOME
Location of network databases.

Files

~/.tt/types.xdr
The user's ToolTalk XDR format types file.

/etc/tt/types.xdr
The system ToolTalk XDR format types file.

$DTHOME/etc/tt/types.xdr
The network-wide ToolTalk XDR format types file.

$DTHOME/lib/cetables/cetables
The Classing Engine database that contains the ToolTalk type definitions.

ttcp

ttcp [–pL] *filename1 filename2*
ttcp -rR [–pL] *directory1 directory2*
ttcp [–prRL] *filename directory*
ttcp -v
ttcp -h

Copies files and directories in a ToolTalk-safe way.

Description

The `ttcp` command invokes the standard operating system command `cp` to copy files and directories, and informs the ToolTalk service about its actions. This command ensures that the ToolTalk objects associated with the named files and directories are copied at the same time.

Options

-h
Prints the usage information for the `ttcp` command.

-i
Interactive copy option.

Note – This option of the standard operating command `cp` is not supported.

-L
Copies the ToolTalk objects of the specified files but does not invoke the standard operating system command `cp` to copy the actual files.

–p
Preserves the modification time and permission modes when the contents of the original file or directory are copied.

Note – To preserve the modification time of ToolTalk objects, you must copy the file or directory as *superuser*.

–r
–R
Recursively copies any files and directories (including any subdirectories and their files) associated with the specified source file.

Note – The destination must be a directory.

-v
Prints the version of the `ttcp` command.

Files

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

ttdbck

ttdbck [-v] [*selection*<*plainopts*] [*diagnosis*<*plainopts*] [*display*<*plainopts*] [*repair*<*plainopts*] [*data-base-directory*]...

Display, check, or repair ToolTalk databases.

Description

The `ttdbck` command is the ToolTalk database maintenance tool. Use this command to directly inspect ToolTalk spec data, detect inconsistencies, and repair the problems found. Run this command on the same machine on which the ToolTalk database files that are being inspected and repaired physically exist; that is, do not attempt to access the ToolTalk database files using the Network File System (NFS).

Note – Since ToolTalk databases are typically accessible only to *root*, this command is normally run as *root*.

Options

data-base-directory

Names the directory that contains the ToolTalk database to be inspected or repaired.

- If no directory is named, the current directory is assumed.
- If a directory path does not end in `TT_DB`, the `TT_DB` is appended.

The user must have read access to the files in the directory in order to inspect the data and write access to the files in the directory in order to repair the data.

-v

Prints the version of the `ttdbck` command.

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.

Note – To prevent accidental changes to the ToolTalk databases, only repair option `-I` is allowed unless a selection or diagnosis option is given.

-f *filename*
Restricts the set of specs to be inspected or modified. When this option is specified, only specs whose files describe objects can be inspected or modified.

Note – If you use shell-style wildcards in *filename*, precede them with an escape (\ ) symbol to prevent the shell from expanding them.

-k *objidkey*
Specifies a particular spec to be displayed or modified. The object id key can be obtained from a previous invocation of the `ttdbck` command; for example, you can display a set of specs, determine the spec that needs repair, and specify its key with this option.

-t *type*
Restricts the set of specs to be inspected or modified. When this option is specified, only specs whose otype is *type* can be inspected or modified.

Note – If you use shell-style wildcards in *type*, precede them with an escape (\ ) symbol to prevent the shell from expanding them.

Diagnosis Options

The diagnosis options check for and report on inconsistencies in the selected specs. Only specs specified 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.

Note – To prevent accidental changes to the ToolTalk databases, only repair option `-I` is allowed unless a selection or diagnosis option is given.

-b
Checks for badly formed specs; that is, specs that do not contain a filename or type, or specs that have types not defined in the ToolTalk Types Database.

–x
Checks for specs which reference files that no longer exist.

Display Options

The display options determine the data to be printed for each selected spec.

–a
Displays all data. This option is equivalent to specifying:

```
ttdbck -imp
```

–i
Displays the object identifier (including the object id key).

–m
Displays the mandatory data that must appear in every spec; that is, the otype of the object described by the spec and the file in which the spec is stored.

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

Repair Options

The repair options modify the selected specs.

Note – To prevent accidental changes to the ToolTalk databases, only repair option `-I` is allowed unless a selection or diagnosis option is given.

–F filename
Changes the filename for the selected specs to the filename specified by this option.

–I
Invokes the NetISAM `isrepair()` function for all files accessed. The NetISAM function is applied before any other inspection or repair action. Use this option when normal operations return error `EBADFILE`.

–T *obtypeid*
Changes the type of the selected specs to the otype specified by this option.

–Z
Removes the selected specs entirely.

Examples

The examples in this section illustrate three uses of the `ttdbck` command.

Example 1

In the directory `/home/TT_DB`, find all badly formed specs and all specs that reference non-existent files and prints their identifiers.

```
ttdbck -bxi /home
```

Example 2

In the directory `/home/TT_DB`, finds all specs that reference objects in file `/home/sample/data` and change the references to `/home/sample/data1`.

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

Example 3

In the directory `/export/TT_DB`, find all specs that reference objects of type *Sample_Chain_Link* and delete the specs.

```
ttdbck -t Sample_Chain_Link -Z /export/TT_DB
```

Files

/path/TT_DB
The ToolTalk database.

ttdbserver, rpc.ttdbserver (ttdbserverd, rpc.ttdbserverd)

rpc.ttdbserver[-n] [-v]

Remote Procedure Call (RPC)-based ToolTalk database server

Description

`rpc.ttdbserver` is the ToolTalk database server daemon. This process, normally started by inetd, performs all database operations for databases stored on the host.

Note – `rpc.ttdbserverd` is the name of the ToolTalk database server daemon prior to this release and is maintained for backwards compatibility.

Options

–n

Turns off permission checking. The protection of the file that contains the spec determines who may read and write that particular spec; however, this option disables the checking for permission and allows anyone to read and write the spec.

Caution – This option allows any file to be over-written. Use with caution.

–v

Prints out the version number for this program and then exits.

Files

TT_DB/*

The NetISAM database files are kept in the this directory under each disk partition mount point.

ttmv

ttmv [–] [–fL] *pathname1 pathname2*
ttmv [–] [–fL] *pathname directory*
ttmv -v
ttmv -h

Move or rename files in a ToolTalk-safe way.

Description

The ttmv command invokes the standard operating system command mv to move files and directories, and informs the ToolTalk service about its actions. This command ensures that the ToolTalk objects associated with the named files and directories are moved at the same time.

Note – The ttmv command moves the ToolTalk objects before it moves the files; however, it does not check whether the object move operation is successful before it moves the files.

Options

-
Treats the arguments that follow as filenames. This option allows you to specify filenames that begin with a minus sign.

-f
Forces a move operation. This option does not report errors and passes the force option to the standard operating system command mv.

-h
Prints usage information for the ttmv command.

-i
Interactive copy option.

Note – This option of the standard operating command cp is not supported.

-L
Moves the ToolTalk objects of the specified files but does not invoke the standard operating system command `mv` to move the actual files.

-v
Prints the version of the `ttmv` command.

Files

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

ttrm, ttrmdir

ttrm [–] [–frL] *pathname*
rmdir *directory*
ttrm[dir] -v
ttrm[dir] -h

Remove files or directories in a ToolTalk-safe way.

Description

The `ttrm` command invokes the standard operating system command `rm`; the `ttrm`dir command invokes the standard operating system command `rmdir`. The specified files and directories are removed, and the ToolTalk service is informed about the actions. These commands ensure that the ToolTalk objects associated with the deleted files and directories are removed at the same time.

Note – The `ttrm` and `ttrmdir` commands removes the ToolTalk objects before they remove the files; however, these commands perform only a minimal check to verify whether the object remove operation is successful before they remove the files.

Options

-
Treats the arguments that follow as filenames. This option allows you to specify filenames that begin with a minus sign.

-f
Forces a move operation. This option does not report errors and passes the force option to the standard operating system command `rm` or `rmdir`.

-h
Print usage information for the `ttrm` or `ttrmdir` command.

-i
Interactive copy option.

Note – This option of the standard operating command `cp` is not supported.

-L
Removes the ToolTalk objects of the specified files but does not invoke the standard operating system command `rm` or `rmdir` to remove the actual files.

-r
Recursively deletes the ToolTalk objects of any directories specified, and pass the recursive option to the standard operating system command `rm` or `rmdir`.

-v
Prints the version of the `ttrm` or `ttrmdir` command.

Files

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

ttsession

ttsession [-a *level*] [-d *display*] [-spStvh] [-{E | X}] [-c *[command]*]

The ToolTalk message server.

Description

The `ttsession` command invokes the ToolTalk message server. Each message server defines a *session*. A session is a group of processes that have an instance of the ToolTalk message server in common.

The message server does not have a user interface and typically runs in the background. It is started either by the user's `.xinitrc` file, or automatically by any program which needs to send a message. The message server must be running before any ToolTalk messages can be sent or received.

The message server reacts to two signals.

- If it receives the *USR1* signal, it toggles trace mode on or off.
- If it receives the *USR2* signal, it rereads the ToolTalk Types Database.

Table 5-3 describes the ttsession exit codes.

Table 5-3 ttsession Exit Codes

Code	Description
0	Normal termination. If the `-c` or `-S` option has not been specified, a zero exit status means that `ttsession` has successfully forked an instance of itself that has begun serving the session.
1	Abnormal termination. `ttsession` was given invalid command line options, was interrupted by SIGINT, or encountered an internal error.
2	Collision. Another `ttsession` is already serving the session.

Options

-a level
Sets the server authentication level. The level must be either *unix*, *xauth*, or *des*.

-c [command]
Starts a process tree session and runs the specified command. The special environment variable `_TT_SESSION` is set to the name of this session. The default session of any process started with this special environment variable will be in this session. If *command* is omitted, the value of `$SHELL` is used. When this process tree session exits, `ttsession` exits with its exit code.

Note – The `-c` option must be the last option on the command line; any options or arguments that follow the `-c` option are read as the command to be executed.

-d [display]
Directs `ttsession` to start an X session for the specified display. (The `ttsession` command normally uses the `DISPLAY` environment variable.)

–E
Reads in the types from the Classing Engine database.

–h
Prints help on how to invoke `ttsession` and exits.

–p
Prints the name of a new process tree session to stdout, then directs `ttsession` to fork a background instance to manage this new session.

–S
Directs `ttsession` to not fork a background instance to manage its session.

–s
Suppresses the printing of warning messages.

–t

Turns on trace mode. When trace mode is on, the state of a message when it is first seen by `ttsession` is displayed, and then the lifetime of the message is shown as follows:

- Dispatch stage: The result of matching the message against type signatures.
- Delivery stage: The result of matching the message against any registered message patterns.
- Any attempt to send the message to a given process and the success or failure of that attempt.

–v

Prints out the version number of `ttsession` and exits.

–X

Reads in the types from the ToolTalk Types Database. This option is the default.

Environment Variables

Table 5-4 describes the environment variables that can be set.

Table 5-4 Environment Variables for ttsession

Environment Variable	Description
DT_TTSESSION_CMD	If set, all ToolTalk clients use this variable as the command to auto-start `ttsession`. Although this variable can be set manually, it is typically set by `ttsession` when running in process tree mode.
TTPATH	Tells the ToolTalk service where to find the ToolTalk Types Databases.The format of this variable is: userDB[:systemDB[:networkDB]]
CEPATH	If the `-E` option is specified, tells the Classing Engine where to find the databases that contain ToolTalk types.
_DT_TT_ARG_TRACE_WIDTH	Specifies how many characters of argument and context values to print when in trace mode. The default is to print the first 40 characters.
_TT_SESSION	`ttsession` uses this variable to communicate its session ID to the tools that it starts. If set, the ToolTalk client library uses its value as the default session ID.

Table 5-4 Environment Variables for ttsession (Continued)

Environment Variable	Description
DISPLAY	If this variable is set and the `_TT_SESSION` variable is not set, all ToolTalk clients will use this variable as the command to auto-start `ttsession`. This variable is typically used when ttsession is auto-started while running under the desktop environment.
_DT_TT_TOKEN	Notifies the ToolTalk client library that it has been invoked by `ttsession` so that the client can confirm to `ttsession` that it started successfully.
_DT_TT_FILE	`ttsession` places a pathname in this variable when it invokes a tool as a consequence of a message scoped to that file.

tt_type_comp

tt_type_comp [E] [-d {user | system | network}]
-{p | O | P | h | v | {r type1..typeN} | {m | M | x} [-o ofile] file}

The ToolTalk otype and ptype compiler.

Description

The `tt_type_comp` command invokes the ToolTalk types compiler, which compiles the otypes and ptypes in the specified typefile. The *typefile* is first run through the standard operating system function `cpp`, and then parsed and checked for correctness. The types compiler produces files for an XDR format.

By default, the tt_type_comp function merges the types of the given file into the existing user ToolTalk Types Database file.

Options

–d
Specifies the database (*user*, *system*, or *network*) which contains the file to be compiled. The default is *user*.

Note – The three databases form a hierarchy where the definition of a type in the user database overrides the definition of the type in the system database, which overrides the definition of the type in the network database.

Table 5-5 ToolTalk Types Database Definitions

Database	Definition
user	~/.tt/types.xdr
system	/etc/tt/types.xdr
network	$TTHOME/tt/types.xdr

–E
Resets the default format XDR to Classing Engine.

Note – Maintained for backward compatiblity.

–h
Prints help when the `tt_type_comp` function is invoked and exits.

–M
Merges types into the specified database only if they do not already exist in that database.

–m
Merges types into the specified database and updates any existing type with the new definition. This option is the default.

Note – The -m option is no longer supported for Classing Engine mode.

–O
Prints a list of known otypes.

–o *outputfile*
Outputs to the specified file name.

–P
Prints a list of known ptypes.

–p
Prints the ToolTalk types in the specified database to a file. The types are output in a source format that can be recompiled with the `tt_type_comp` command.

–r type1 ... typeN
Removes the given ptypes or otypes from the database.

–s
Suppresses printing.

–v
Prints the version number and exits.

–x
Produces a compiled XDR format type file.

5

tttar

tttar [EfhpSv] [*tarfile*] *pathname1pathname2*
tttar [EhpRSv] *tttarfile* [[-rename *oldname newname*] . . .]
pathname1pathname2 . . .
tttar -v
tttar -h[elp]

Archives or de-archives files and ToolTalk objects.

The `tttar` command has two fundamentally different modes.

1. If the `L` function modifier is not specified, the `tttar` command invokes the standard operating system command `tar` to archive or extract multiple files and their ToolTalk objects onto or from, respectively, a single archive (called a *tarfile*) in a ToolTalk-safe way.

2. If the `L` function modifier is specified, the `tttar` command does *not* invoke the standard operating system command `tar` to archive or extract actual files. Instead, this command archives or extracts only ToolTalk objects onto or from, respectively, a single archive (called a *tttarfile*).

Note – This section discusses the `ttar` command with the `L` function modifier specified; that is, it references *tttarfiles* instead of *tarfiles* and discusses archiving and de-archiving only the ToolTalk objects of the named file rather than archiving and de-archiving both the named file and its ToolTalk objects.

The first (or *key*) argument controls the actions of the `ttar` command. The key argument is a string of characters that contain one *function letter* and one or more *function modifiers*. Other arguments are file or directory names that specify from which files ToolTalk objects are to be archived or extracted. By default, a directory name recursively references the files and subdirectories of that directory.

Note – A file does not need to exist in order for a ToolTalk object to be associated with its pathname. The `tttar` command does not attempt to archive the objects associated with any files that do not exist in the directory.

Note – When you extract a *tttarfile* from a tar archive, the current working directory must be writable so that the *tttarfile* can be placed in it temporarily.

Function Letters

c

Creates a new *tttarfile* and writes the ToolTalk objects of the specified files onto it.

r

This function letter of the standard operating system command `tar` is not supported.

t

Lists the table of contents of the *tttarfile*.

u

This function letter of the standard operating system command `tar` is not supported.

x

Extracts the ToolTalk objects of the specified files from the *tttarfile*. If a specified file matches a directory with contents written onto the tape, this directory is (recursively) extracted. The owner and modification time of the ToolTalk objects are restored. If a *filename* is not specified, the ToolTalk objects of all files named in the archive are extracted.

Function Modifiers

F

This function modifier of the standard operating system command `tar` is not supported.

f

Uses the next argument as the name of the *tttarfile*. If *tttarfile* is specified as a minus (–) sign, the `ttar` command either writes to the standard output or reads from the standard input, whichever is appropriate.

h

Treats symbolic links as normal files or directories. (The `tttar` command normally does not follow symbolic links.)

L

Do not invoke the standard operating system command `tar`.

Note – This function must be used with the `f` function modifier because reading and writing an archived *tttarfile* directly to or from magnetic tape is unimplemented.

---

p

Preserves the original mode of the specified files when used with the `x` function letter. You can also extract setUID and sticky information if you are the superuser.

Note – If the `L` function letter is also specified, this function modifier is disabled.

R

Do not recurse into directories. You must specify the `L` function modifier with this function modifier.

v

Verbose mode. This function modifier displays the name of each file, preceded by the function letter. (The `ttar` command normally does not display this information.)

w

This function modifier of the standard operating system command `tar` is not supported.

X

This function modifier of the standard operating system command `tar` is not supported.

Options

-rename
Reads the next two arguments as *oldname* and *newname*, respectively, and renames any entry archived as *oldname* to *newname*.

- If *oldname* is a directory, its entries are recursively renamed.
- If more than one `-rename` option applies to an entry (for example, because one or more parent directories are being renamed), the most specific `-rename` option applies.

Note – You must use the `L` function modifier with the `-rename` option.

-C
This option of the standard operating system command `tar` is not supported.

-h
Prints usage information for the `ttar` command.

-I
This option of the standard operating system command `tar` is not supported.

-v
Prints the version of the `ttar` command.

Files

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

5

Initialization Error Messages 6

The ToolTalk error messages described in Table 6-1 can occur either when the ToolTalk service, or an application that uses the ToolTalk service, is attempting to start up.

Table 6-1 Errors that may Occur During Initialization

Error Message	Description	Solution
ld.so: libtt.so.1.1: not found	The run-time linker could not find the dynamic library, `libtt.so.1.1`.	Place a directory that contains `libtt.so.1.1` in `$LD_LIBRARY_PAT`.
/bin/sh: application_name: not found	The start string as installed in the ToolTalk Types Database does not correspond to an executable file in `$PATH`.	To correct this error: a. First, start the application as it would be started without the ToolTalk service. b. After the application has started, retry the operation that should have started the application with the ToolTalk service. To prevent the error from occurring again, verify that the start string in the relevant ptype corresponds to an executable file in `$PATH`.

Table 6-1 Errors that may Occur During Initialization (Continued)

Error Message	Description	Solution
Cannot open display	`ttsession` could not contact the server named with the `-d` *`display`* option or the `$DISPLAY` variable.	To start the session: a. Verify that the named display is running. b. Verify that the host on which `ttsession` is running has permission to connect to the named display.
ttsession: Illegal environment (-c or -d not specified and DISPLAY variable not set)	Neither the `-d` *`display`* option or the `$DISPLAY` variable is set. This error typically occurs when you or your client attempt to start `ttsession` after you have either switched to another user name or become superuser.	Set the `-d` *`display`* option or the `$DISPLAY` variable.

ToolTalk Error Messages 7

The ToolTalk error and warning identifiers are allocated as follows:

TT_OK	TT_WRN_*	APP_WRN_*	TT_WRN_LAST	TT_ERR_*	APP_ERR_*	TT_ERR_LAST
0	1	512	1024	1025	1536	2047

Table 7-1 is an alphabetical listing of the ToolTalk error messages and their corresponding message ids.

Table 7-1 Alphabetical List of ToolTalk Error Messages

Error Message	Message ID
TT_ERR_ACCESS	TTERR-1032
TT_ERR_ADDRESS	TTERR-1039
TT_ERR_APPFIRST	TTERR-1536
TT_ERR_CATEGORY	TTERR-1057
TT_ERR_CLASS	TTERR-1025
TT_ERR_DBAVAIL	TTERR-1026
TT_ERR_DBCONSIST	TTERR-1060

Table 7-1 Alphabetical List of ToolTalk Error Messages

Error Message	Message ID
TT_ERR_DBEXIST	TTERR-1027
TT_ERR_DBFULL	TTERR-1059
TT_ERR_DBUPDATE	TTERR-1058
TT_ERR_DISPOSITION	TTERR-1046
TT_ERR_FILE	TTERR-1028
TT_ERR_INTERNAL	TTERR-1051
TT_ERR_LAST	TTERR-2047
TT_ERR_MODE	TTERR-1031
TT_ERR_NO_MATCH	TTERR-1053
TT_ERR_NOMEM	TTERR-1062
TT_ERR_NOMP	TTERR-1033
TT_ERR_NOTHANDLER	TTERR-1034
TT_ERR_NO_VALUE	TTERR-1050
TT_ERR_NUM	TTERR-1035
TT_ERR_OBJID	TTERR-1036
TT_ERR_OP	TTERR-1037
TT_ERR_OTYPE	TTERR-1038
TT_ERR_OVERFLOW	TTERR-1055
TT_ERR_PATH	TTERR-1040
TT_ERR_POINTER	TTERR-1041
TT_ERR_PROCID	TTERR-1042
TT_ERR_PROPLEN	TTERR-1043
TT_ERR_PROPNAME	TTERR-1044

Table 7-1 Alphabetical List of ToolTalk Error Messages

Error Message	Message ID
TT_ERR_PTYPE	TTERR-1045
TT_ERR_PTYPE_START	TTERR-1056
TT_ERR_READONLY	TTERR-1052
TT_ERR_SCOPE	TTERR-1047
TT_ERR_SESSION	TTERR-1048
TT_ERR_SLOTNAME	TTERR-1063
TT_ERR_STATE	TTERR-1061
TT_ERR_UNIMP	TTERR-1054
TT_ERR_VTYPE	TTERR-1049
TT_ERR_XDR	TTERR-1064
TT_OK	TTERR-0
TT_STATUS_LAST	TTERR-2048
TT_WRN_APPFIRST	TTERR-512
TT_WRN_LAST	TTERR-1024
TT_WRN_NOTFOUND	TTERR-1
TT_WRN_SAME_OBJID	TTERR-4
TT_WRN_STALE_OBJID	TTERR-2
TT_WRN_START_MESSAGE	TTERR-5
TT_WRN_STOPPED	TTERR-3

Table 7-2 describes the ToolTalk error messages; the error messages are listed in order of their message id.

Table 7-2 ToolTalk Error Messages

Error Message	Message ID	Error Message String	Description	Solution
TT_OK	TTERR-0	TT_OK Request successful.	The call was completed successfully.	
TT_WRN_NOTFOUND	TTERR-1	TT_WRN_NOTFOUND The object was not removed because it was not found.	The ToolTalk service could not find the specified object in the ToolTalk database. The destroy operation did not succeed.	
TT_WRN_STALE_OBJID	TTERR-2	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.	When the ToolTalk service looked up the specified object in the ToolTalk database, it found a forwarding pointer to the object.	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_STOPPED	TTERR-3	TT_WRN_STOPPED The query was halted by the filter procedure.	The query operation being performed was halted by the Tt_filter_function.	
TT_WRN_SAME_OBJID	TTERR-4	TT_WRN_SAME_OBJID The moved object retains the same objid.	The object moved stayed within the same file system. The ToolTalk service will retain the same objid and update the location.	

Table 7-2 ToolTalk Error Messages (Continued)

Error Message	Message ID	Error Message String	Description	Solution
TT_WRN_START_MESSAGE	TTERR-5	TT_WRN_START_MESSAGE This message caused this process to be started. This message should be replied to even if it is a notice.	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.	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_APPFIRST	TTERR-512	TT_WRN_APPFIRST This code should be unused.	This code marks the beginning of the messages allocated for ToolTalk application warnings.	
TT_WRN_LAST	TTERR-1024	TT_WRN_LAST This code should be unused.	This code marks the last of the messages allocated for ToolTalk warnings.	
TT_ERR_CLASS	TTERR-1025	TT_ERR_CLASS The Tt_class value passed is invalid.	The ToolTalk service does not recognize the class value specified.	The Tt_class values are TT_NOTICE and TT_REQUEST. Retry the call with one of these values.
TT_ERR_DBAVAIL	TTERR-1026	TT_ERR_DBAVAIL A required database is not available. The condition may be temporary, trying again later may work.	The ToolTalk service could not access the ToolTalk database needed for this operation.	a. Check if the file server or workstation that contains the database is available. b. Try the operation again later.
TT_ERR_DBEXIST	TTERR-1027	TT_ERR_DBEXIST A required database does not exist. The database must be created before this action will work.	The ToolTalk service did not find the specified ToolTalk database in the expected place.	Install the rpc.ttdbserved program on the machine that stores the file or object involved in this operation.

Table 7-2 ToolTalk Error Messages (Continued)

Error Message	Message ID	Error Message String	Description	Solution
TT_ERR_FILE	TTERR-1028	TT_ERR_FILE File object could not be found.	The file specified does not exist or is not accessible.	a. Check the file path name and retry the operation. b. Check if the machine where the file is stored is accessible.
TT_ERR_MODE	TTERR-1031	TT_ERR_MODE The Tt_mode value is not valid.	The ToolTalk service does not recognize the specified mode value.	The Tt_mode values are TT_IN, TT_OUT, and TT_INOUT. Retry the call with one of these values.
TT_ERR_ACCESS	TTERR-1032	TT_ERR_ACCESS An attempt was made to access a ToolTalk object in a way forbidden by the protection system.	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.	a. Obtain proper access to the object. b. Retry the operation.
TT_ERR_NOMP	TTERR-1033	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.	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.	a. Verify that the ToolTalk service is installed. b. Verify that ttsession is installed on the machine in use.
TT_ERR_NOTHANDLER	TTERR-1034	TT_ERR_NOTHANDLE Only the handler of the message can do this.	Only the handler of a message can perform this operation. This application is not the handler for this message.	

Table 7-2 ToolTalk Error Messages (Continued)

Error Message	Message ID	Error Message String	Description	Solution
TT_ERR_NUM	TTERR-1035	TT_ERR_NUM The integer value passed is not valid.	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.	Check the integer specified.
TT_ERR_OBJID	TTERR-1036	TT_ERR_OBJID The object id passed does not refer to any existing object spec.	The objid does not reference an existing objectt.	Update the spec property that contains the objid specified.
TT_ERR_OP	TTERR-1037	TT_ERR_OP The operation name passed is not syntactically valid.	The specified operation name is null or contains non-alphanumeric characters.	a. Remove any non-alphanumeric characters. b. Retry the operation.
TT_ERR_OTYPE	TTERR-1038	TT_ERR_OTYPE The object type passed is not the name of an installed object type.	The ToolTalk service could not locate the specified otype.	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>*

Table 7-2 ToolTalk Error Messages (Continued)

Error Message	Message ID	Error Message String	Description	Solution
TT_ERR_ADDRESS	TTERR-1039	TT_ERR_ADDRESS The Tt_address value passed is not valid.	The ToolTalk service does not recognize the address value specified.	The Tt_address values are TT_PROCEDURE, TT_OBJECT, TT_HANDLER, and TT_OTYPE. Retry the call with one of these values.
TT_ERR_PATH	TTERR-1040	TT_ERR_PATH One of the directories in the file path passed does not exist or cannot be read.	The ToolTalk service was not able to read a directory in the specified file path name.	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	TTERR-1041	TT_ERR_POINTER The opaque pointer (handle) passed does not indicate an object of the proper type.	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.	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	TTERR-1042	TT_ERR_PROCID The process id passed is not valid.	The process identifier specified is out of date or invalid.	Retrieve the default procid with tt_default_procid().
TT_ERR_PROPLEN	TTERR-1043	TT_ERR_PROPLEN The property value passed is too long.	The ToolTalk service accepts property values of up to 64 characters.	Shorten the property value to less than 64 characters.
TT_ERR_PROPNAME	TTERR-1044	TT_ERR_PROPNAME The property name passed is syntactically invalid.	The property name is too long, contains non-alphanumeric characters, or is null.	Check the property name, modify if necessary, and retry the operation.

Table 7-2 ToolTalk Error Messages (Continued)

Error Message	Message ID	Error Message String	Description	Solution
TT_ERR_PTYPE	TTERR-1045	TT_ERR_PTYPE The process type passed is not the name of an installed process type.	The ToolTalk service could not locate the specified ptype.	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_DISPOSITION	TTERR-1046	TT_ERR_DISPOSITION The Tt_disposition value passed is not valid.	The disposition passed is not recognized by the ToolTalk service.	The Tt_disposition values are TT_DISCARD, TT_QUEUE, and TT_START. Retry the call with one of these values.
TT_ERR_SCOPE	TTERR-1047	TT_ERR_SCOPE The Tt_scope value passed is not valid.	The scope passed is not recognized by the ToolTalk service.	The Tt_scope values are TT_SESSION and TT_FILE. Retry the call with one of these values.
TT_ERR_SESSION	TTERR-1048	TT_ERR_SESSION The session id passed is not the name of an active session.	An out-of-date or invalid ToolTalk session was specified.	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()

Table 7-2 ToolTalk Error Messages (Continued)

Error Message	Message ID	Error Message String	Description	Solution
TT_ERR_VTYPE	TTERR-1049	TT_ERR_VTYPE The value type name passed is not valid.	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.	a. Change the type of the value to either int or string. b. Retry the operation.
TT_ERR_NO_VALUE	TTERR-1050	TT_ERR_NO_VALUE No property value with the given name and number exists.	The ToolTalk service could not locate a value for the property specified in the ToolTalk database.	Retrieve the current list of properties to find the property.
TT_ERR_INTERNAL	TTERR-1051	TT_ERR_INTERNAL Internal error (bug)	The ToolTalk service has suffered an internal error.	a. Restart all applications that are using the ToolTalk service. b. Report the error to the your system vendor support center.
TT_ERR_READONLY	TTERR-1052	TT_ERR_READONLY The attribute cannot be changed.	The application does not have ownership or write permissions for the attribute. Therefore, this operation cannot be performed.	
TT_ERR_NO_MATCH	TTERR-1053	TT_ERR_NO_MATCH No handler could be found for this message, and the disposition was not queue or start.	The message the application sent could not be delivered. No applications that are running have registered interest in this type of message.	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.

Table 7-2 ToolTalk Error Messages (Continued)

Error Message	Message ID	Error Message String	Description	Solution
TT_ERR_UNIMP	TTERR-1054	TT_ERR_UNIMP Function not implemented.	The ToolTalk function called is not implemented.	
TT_ERR_OVERFLOW	TTERR-1055	TT_ERR_OVERFLOW Too many active messages (try again later).	The ToolTalk service has received the maximum amount of active messages (2000) it can properly handle.	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_PTYPE_START	TTERR-1056	TT_ERR_PTYPE_START Attempt to launch a client specified in the start attribute of a ptype failed.	The ToolTalk service could not start the type of process specified.	Verify that the application that the ptype represents is properly installed and has execute permission.
TT_ERR_CATEGORY	TTERR-1057	TT_ERR_CATEGORY Pattern object has no category set.	The category was not set.	
TT_ERR_DBUPDATE	TTERR-1058	TT_ERR_DBUPDATE The database is inconsistent: another tt_spec_write updated object first.	The ToolTalk service could not update the database because the specified object was already updated by a previous tt_spec_write call.	
TT_ERR_DBFULL	TTERR-1059	ToolTalk database is full.	The ToolTalk service could not write to the database because it is full.	Create more space on the file system in which the database is stored.

Table 7-2 ToolTalk Error Messages (Continued)

Error Message	Message ID	Error Message String	Description	Solution
TT_ERR_DBCONSIST	TTERR-1060	Database is access information is incomplete or database is corrupt (run ttdbck).	The ToolTalk service could not write to the database because it is either corrupt, or the access information is incomplete.	Run the ttdbck utility to repair the database.
TT_ERR_STATE	TTERR-1061	The Tt_message is in a state that is not valid for the attempted operation.	The state of the message is invalid for the type of operation being requested.	
TT_ERR_NOMEM	TTERR-1062	No more memory.	There is not enough available memory to perform the operation.	Check the swap space, then retry the operation.
TT_ERR_SLOTNAME	TTERR-1063	The slot name is syntactically invalid.	The syntax for the slot name is not valid.	Correct the syntax for the slot name.
TT_ERR_XDR	TTERR-1065	The XDR procedure failed on the given data, or evaluated to a 0 length structure.	The XDR procedure failed on the given data, or evaluated to a 0 length structure.	
TT_ERR_APPFIRST	TTERR-1536	TT_ERR_APPFIRST This code should be unused.	This code marks the beginning of the messages allocated for ToolTalk application errors.	
TT_ERR_LAST	TTERR-2047	TT_ERR_LAST This code should be unused.	This code marks the last of the messages allocated for ToolTalk errors.	
TT_STATUS_LAST	TTERR-2048	TT_STATUS_LAST This code should be unused.	This code marks the last of the messages allocated for ToolTalk status.	

The ToolTalk Desktop Services Message Set

A

A.1 General Description of the ToolTalk Desktop Services Message Set

The ToolTalk Desktop Services Message Set conventions apply to any tools in a POSIX or X11 environment. In addition to standard messages for these environments, the Desktop conventions define data types and error codes that apply to all of the ToolTalk inter-client conventions. The request and notification messages which comprise the ToolTalk Desktop Services Message Set are listed in Table A-1.

Table A-1 The ToolTalk Desktop Services Message Set

Requests	**Notifications**
Do_Command	Created, Deleted
Get_Modified	Modified, Reverted
Get_Status	Moved
Get_Sysinfo	Saved
Pause, Resume	Started, Stopped
Quit	Staus
Raise, Lower	
Save, Revert	
Set_Environment, Get_Environment	

Table A-1 The ToolTalk Desktop Services Message Set (Continued)

Requests	Notifications
Set_Geometry, Get_Geometry	
Set_Iconified, Get_Iconified	
Set_Locale, Get_Locale	
Set_Mapped, Get_Mapped	
Set_Situation, Get_Situation	
Set_XInfo, Get_XInfo	
Signal	

A.2 Desktop Definitions and Conventions

This section defines terms and error messages unique to the Desktop Services message set. Specific to the desktop services messages are values associated with fields as described in Table A-2.

Table A-2 Values Associated with Fields

Field	Associated Value
boolean	A vtype for logical values. The underlying data type of boolean is integer; manipulate arguments of this vtype with `tt_*_arg_ival[_set]()` and `tt_*_iarg_add()`. A zero value means false; a non-zero value means true.
buffer	A volatile, non-shared (for example, in-memory) representation of persistent data.
bufferID	A vtype that uniquely identifies buffers. The underlying data type of bufferID is string. To guarantee bufferID uniqueness, use the form `<internal_counter> <procID>`
messageID	A vtype that uniquely identifies messages. The underlying data type of messageID is string; manipulate arguments of this vtype with `tt_*_arg_val[_set]()` and `tt_*_arg_add()`. To guarantee messageID uniqueness, use the form `<internal_counter> <procID>` tt_message_id() returns an opaque string of similar uniqueness. Use tt_message_id() to generate a message's messageID; however, the inter-client conventions explicitly include the messageID as a message argument to support inter-operation with other versions of the ToolTalk service.
type	Any vtype that is the name of the kind of objects in a particular persistent-object system. For example, the vtype for the kind of objects in filesystems is File; the vtype for ToolTalk objects is ToolTalk_Object.

Table A-2 Values Associated with Fields (Continued)

Field	Associated Value
vendor toolName toolVersion	Names of arguments. These strings appear in several of the Desktop Service messages. These strings are not defined rigorously; they are intended to present to the user descriptions of these three attributes of the relevant procID.
view	A screen display, such as a (portion of a) window, that presents to the user part or all of a document.
viewID	A vtype that uniquely identifies views. The underlying data type of viewID is string. To guarantee viewID uniqueness, use the form `<internal_counter> <procID>`

Errors

Table A-3 describes the Desktop Services error messages; the error messages are listed in order of their message id.

Table A-3 Desktop Services Error Messages

Message ID	Error Message	Error Message String	Description
1538	TT_DESKTOP_ENOENT	No such file or directory	
1549	TT_DESKTOP_EACCES	Permission Denied	
1558	TT_DESKTOP_EINVAL	Invalid argument	An argument's value was not valid; for example, a locale in `Set_Locale` that is not valid on the handler's host. Use this error status only when a more-specific error status does not apply.
1571	TT_DESKTOP_ENOMSG	No message of desired type	A messageID does not refer to any message currently known by the handler.

Table A-3 Desktop Services Error Messages (Continued)

Message ID	Error Message	Error Message String	Description
1610	TT_DESKTOP_EPROTO	Protocol error	A message was not understood because: a. A required argument was omitted. b. An argument had the wrong vtype, or the vtype is not allowed in this message; for example, the vtype `boolean` in the `Get_Geometry` message. c. An argument's value was not legal for its vtype; for example, negative values for width in the `Set_Geometry` message. d. An argument's value was not legal for this message; for example, the `PATH=/foo` variable in `Get_Environment` message. In general, this error status indicates that the message is malformed.
1688	TT_DESKTOP_CANCELED	Operation was canceled	The operation was canceled because of direct or indirect user intervention. An example of indirect intervention is termination of the handling process caused by the user, or receipt of a Quit() request. (All messages should be taken as authentically representing the wishes of the user whose uid is indicated by tt_message_uid().)
1689	TT_DESKTOP_ENOTSUP	Operation not supported	The requested operation is not supported by this handler. This error indicates that a handler assumes that, if it rejects a request, no other handler will be able to perform the operation. For example, a request such as Set_Iconified() or a request that refers to a state (such as a bufferID) that is managed by this handler alone. A request failed with this error distinguishes the case of an incompletely-implemented handler from the case of the absence of a handler. **Note:** Do not use TT_ERR_UNIMP in place of TT_DESKTOP_ENOTSUP as TT_ERR_UNIMP means that a particular feature of ToolTalk itself is not implemented.
1699	TT_DESKTOP_UNMODIFIED	Operation does not apply to unmodified entities	

Warnings

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

A.3 The ToolTalk Desktop Services Message Set

This section contains a description of each of the generic messages which constitute the ToolTalk Desktop Services Message Set.

Created, Deleted (Notice)

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

Synopsis

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

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

Description

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

Required Arguments

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

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

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

Optional Arguments

type ID
: Extra instances of this argument may be included.

Do_Command (Request)

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

Synopsis

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

Description

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

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

Required Arguments

string command
: The command being requested to be performed.

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

Optional Arguments

messageID counterfoil

Unique string created by the message sender (typically by concatenating a counter and a procID) to give both sender and receiver a way to refer to this request in other correspondence. Include this argument 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, then the handler should immediately send at least one Status notice point-to-point back to the requestor to identify itself to the requestor.

Warnings

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

Get_Modified (Request)

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

Synopsis

```
[file] Get_Modified(intype          ID,
                    out     boolean modified);
```

Description

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

Required Arguments

type ID
: The identity of the entity that may have been modified.

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

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

Errors

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

Get_Status (Request)

Requests that a tool's current status be returned.

Synopsis

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

Description

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

Required Arguments

string status
: The status to be retrieved.

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

string toolName
: The name of the receiving tool.

string toolVersion
: The version of the receiving tool.

Optional Arguments

messageID operation2Query
: The ID of the request that initiated the operation the status of which is being requested.

Get_Sysinfo (Request)

Retrieves information about a tool's host.

Synopsis

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

Description

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

Required Arguments

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

string nodename
: The name of the host.

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

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

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

string provider
: The name of the hardware manufacturer.

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

See Also

sysinfo(2), umane(2)

Modified, Reverted (Notice)

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

Synopsis

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

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

Description

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

Required Arguments

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

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

Moved (Notice)

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

Synopsis

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

Description

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

Required Arguments

type newID
: The new identity of the moved entity.

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

type oldID
: The old identity of the moved entity.

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

Pause, Resume (Request)

Requests the specified tool, operation, or data performance to pause or resume.

Synopsis

```
Pause(      [in     messageID     operation] );

Pause(      in      bufferID      docBuf );

Resume(     [in     messageID     operation] );

Resume(     in      bufferID      docBuf
            [in     locator       whither
            |in     vector        duration] );
```

Description

The Pause and Resume messages request that the specified tool, operation, or data performance pause or resume, respectively.

- If the optional *operation* argument is included, the handler should pause or resume the operation that was invoked by the specified request. Use a Tt_address of TT_HANDLER to send this form of the request.
- If the optional *docBuf* argument is included, performance of the data in the specified buffer should be paused or resumed. Use a Tt_address of TT_PROCEDURE to send this form of the request.
- If both of the optional arguments are omitted, the handling procid should pause or resume its operations. Use a Tt_address of TT_HANDLER to send this form of the request.

Caution – The Pause and Resume requests may also be sent as a multicast notices; however, the consequences can be severe and unexpected.

Optional Arguments

bufferID docBuf
The buffer in which data performance is to be paused or resume.

messageID operation
The request to be paused.

locator whither
The buffer location to which performance is to be resumed.

vector duration
The duration for which performance is to be resumed.

Note – If neither the *whither* nor the *duration* argument is included in this message, the performance is resumed indefinitely.

Errors

TT_ERR_NOMATCH
The bufferID may not be valid; no editor has a pattern handling this request for docBuf.

TT_DESKTOP_EINVAL
The value for the *whither* is not a legal locator for the media type of the document in docBuf.

TT_DESKTOP_EINVAL
The destination is not a legal vector for the media type of the document in docBuf.

TT_DESKTOP_EFAULT
The value for the *whither* argument is not a valid locator for the document in docBuf.

TT_DESKTOP_EFAULT
The value for the *duration* argument is not a valid vector for the document in docBuf.

TT_DESKTOP_ENOMSG
The operation does not refer to any message currently known by the handler.

Quit (Request)

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

Synopsis

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

Description

The Quit message requests that the specified operation or tool terminate.

- If the *operation2Quit* argument is included, this request asks the recipient to terminate the indicated request. (Whether the terminated request must be failed depends on its semantics. Often, termination can be considered to indicate that the requested operation has been carried out to the requestor's satisfaction.)
- If the *operation2Quit* argument is omitted, this request asks the recipient procID to quit.
  If the request succeeds, one or more ToolTalk procID's should call tt_close, and zero or more processes should exit. ("Zero or more process" are indicated because a single process can instantiate multiple independent procID's, and a single procID can conceivably be implemented by a set of cooperating processes.)

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

Caution – The Quit request may also be sent as a multicast notice; however, the consequences can be severe and unexpected.

Required Arguments

boolean silent
: Boolean value that indicates whether the recipient tool is allowed to block on user input before terminating itself, or the indicated operation. If this value is *false*, the handler is not required to seek user input.

boolean force
: Boolean value that indicates whether the recipient tool should terminate itself even if circumstances are such that the tool ordinarily would not terminate under them.

  For example, a tool's policy is to not quit with unsaved changes unless the user has been asked whether the changes should be saved. When this argument is true, this tool should terminate athough the user has not been asked whether changes should be saved and those changes will be lost.

Optional Arguments

messageID operation2Quit
: The request that should be terminated. For a request to be terminable, an (optional) counterfoil messageID shall have been included in the request, and the handler shall have sent a Status notice back to the requestor (thus identifying itself to the requestor).

Errors

TT_DESKTOP_ECANCELED
: The Quit request was over-ridden by the user.

TT_DESKTOP_ENOMSG
: The operation2Quit argument does not refer to any message currently known by the handler.

Raise, Lower (Request)

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

Synopsis

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

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

Description

The Raise and Lower messages raise or lower, respectively, the window(s) associated with the recipient's procid. If any optional arguments are present, only the indicated window(s) are raise or lowered.

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

Optional Arguments

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

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

Save, Revert (Request)

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

Synopsis

```
[file]       Save(    in            typeID);
[file]       Revert(  in            typeID);
```

Description

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

Required Arguments

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

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

Errors

TT_DESKTOP_UNMODIFIED
: The entity had no pending, unsaved modifications.

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

Saved(Notice)

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

Synopsis

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

Description

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

Required Arguments

type ID
The identity of the saved entity.

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

Set_Environment, Get_Environment (Request)

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

Synopsis

```
Set_Environment(     in            stringvariable,
                     in            stringvalue
                     [...]);

Get_Environment(     in            stringvariable,
                     out           stringvalue
                     [...]);
```

Description

The Set_Environment and Get_Environment messages request that the value of the indicated environment variable(s) either be replaced or reported, respectively.

Caution – The Set_Environment request may also be sent as a multicast notice; however, the consequences can be severe and unexpected.

Required Arguments

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

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

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

Optional Arguments

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

Set_Geometry, Get_Geometry (Request)

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

Synopsis

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

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

Description

The Set_Geometry and Get_Geometry messages request that the value of the on-screen geometry of the optionally-specified window, or the value of the on-screen geometry of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved (respectively).

Required Arguments

width w
height h
xOffset x
yOffset y

The integer geometry values in pixels.

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

Note – Negative offset values are interpreted according to X11 rules.

Optional Arguments

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

viewID view2Set
viewID view2Get
: The identifier of any view associated with the window(s) that is (are) to be set or retrieved.

Set_Iconified, Get_Iconified(Request)

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

Synopsis

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

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

Description

The Set_Iconified and Get_Iconified messages request that the value of the iconic state of the optionally-specified window, or the iconic state of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved (respectively).

Caution – The Set_Iconified and Get_Iconified requests may also be sent as a multicast notice; however, the consequences can be severe and unexpected.

Required Arguments

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

Optional Arguments

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

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

Set_Locale, Get_Locale (Request)

Sets or retrieves a tool's locale.

Synopsis

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

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

Description

The Set_Locale and Get_Locale messages replace or report (respectively) the locale of the POSIX locale categories.

Caution – The Set_Locale request may also be sent as a multicast notice; however, the consequences can be severe and unexpected.

Required Arguments

string category
: The locale category to set or retrieve.

  A locale category is a group of data types whose formatting varies according to locale; for example, ANSI C and X/OPEN locale categories include:

- LC_CTYPE
- LC_NUMERIC
- LC_TIME
- LC_COLLATE
- LC_MONETARY
- LC_ALL
- LC_MESSAGES (Solaris-specific)

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

Optional Arguments

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

Set_Mapped,Get_Mapped(Request)

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

Synopsis

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

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

Description

The Set_Mapped and Get_Mapped messages request that value of the mapped state of the optionally-specified window, or the mapped state of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved (respectively).

Caution – The Set_Mapped request may also be sent as a multicast notice; however, the consequences can be severe and unexpected.

Required Arguments

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

Optional Arguments

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

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

Set_Situation, Get_Situation (Request)

Requests that a tool's current working directory be set or reported.

Synopsis

```
Set_Situation(in     string        path );
Get_Situation(out    string        path );
```

Description

The Set_Situation and Get_Situation messages request that value of the current working directory be either set or reported (respectively).

Caution – The Set_Situation request may also be sent as a multicast notice; however, the consequences can be severe and unexpected.

Required Arguments

string path
: The pathname of the working directory that the recipient is either using or is to use.

Set_XInfo, Get_XInfo (Request)

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

Synopsis

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

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

Description

The Set_XInfo and Get_XInfo messages request that the X11 attributes of the optionally-specified window, or the X11 attributes of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved (respectively).

Required Arguments

string display
   An X11 display.

Note – Since the handler may be running on a different host, use the value `hostname:n[.n]` rather than `:n[.n]`.

---

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

StaticGray
GrayScale
StaticColor
PseudoColor
TrueColor
DirectColor

integer depth
The number of bits in a pixel.

Optional Arguments

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

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

Signal(Request)

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

Synopsis

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

Description

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

Required Arguments

integer theSignal
The signal to be sent.

Caution – The Signal request may also be sent as a multicast notice; however, the consequences can be severe and unexpected.

Started, Stopped (Notice)

Notification that a tool has started or terminated.

Synopsis

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

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

Description

The Started and Stopped messages notify interested tools whenever a tool starts or terminates, respectively.

Required Arguments

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

string toolName
: The name of the started or terminated tool.

string toolVersion
: The version of the started or terminated tool.

Status(Notice)

Notification that a tool has status information to announce.

Synopsis

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

Description

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

Required Arguments

string status
: The status which is being announced.

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

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

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

Optional Arguments

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

A

The ToolTalk Document and Media Exchange Message Set

B

B.1 General Description of the ToolTalk Document and Media Exchange Message Set

The ToolTalk Document and Media Message Set allows a tool to be a container for arbitrary media, or to be a media player/editor that can be driven from such a container. The ToolTalk Document and Media Exchange Message Set is composed of several request messages, listed in Table B-1.

Table B-1 ToolTalk Document and Media Exchange Message Set

Requests	**Notices**
Abstract	*There are no notices in the ToolTalk Document and Media Exchange Message Set.*
Deposit	
Display	
Dsiplay, Edit	
Edit	
Interpret	
Print	
Translate	

These messages are oriented towards creating, editing, and using documents of a certain media type. The conventions for this message set 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. The ToolTalk service routes container requests to the user's preferred tool for the given media type and operation, including routing the request to an instance of the tool which is already running if that instance is best-positioned to handle the request.

B.2 Media Exchange Definitions and Conventions

Media exchange messages are sent and received by tools that display or edit some kind of media. Specific to the media exchange messages are values associated with fields. The parts of a Media Exchange message is defined as follows:

<document>
A vector of bytes with an associated mediaType.

<mediaType>
The name of a media format. The mediaType allows messages about that document to be dispatched to the right editor. Standard mediaTypes include those listed in Table B-2.

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

Table B-2 Standard Media Types

Name of Format	Description	Company
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/Microsoft
GIF	Graphics Interchange Format	CompuServe
XPM	XPM -- The X PixMap Format	Groupe Bull

Table B-2 Standard Media Types (Continued)

Name of Format	Description	Company
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

abstract mediaType
A family of similar mediaTypes, such as flat text or structured graphics.

vector
A string vtype describing a distance and a direction in a document. The syntax of vectors varies by abstract mediaType.

locator
A string describing a location in a document. The syntax of locators varies by abstract mediaType, but should usually be a superset of vector syntax.

flat text
A family of mediaTypes (such as ISO_Latin_1) which consist of a sequence of characters from some character set.

Legal vectors for flat text are:

```
lineVec  ::= Line:[-][0-9]+
charVec ::= Character:[-][0-9]+
vector    ::= <lineVec>
vector    ::= [<lineVec>,]<charVec>
```

Legal locators for flat text are vectors.

time-based media
A family of media types which consist of time-structured data; for example, Sun_Audio.

Legal vectors for time-based media include:

```
vector   ::= uSeconds:[-][0-9]+
vector   ::= Samples:[-][0-9]+
```

Legal locators for time-based media are vectors.

Errors

These definitions are common to all Document and Media Exchange messages. Any differences or additions will be noted in the man pages.

`1700 TT_MEDIA_ERR_SIZE`

The specified size was too big or too small.

`1701 TT_MEDIA_ERR_FORMAT`

The data does not conform to the specified format.

B.3 The ToolTalk Document and Media Exchange Message Set

This section contains a description of each of the messages which constitute the ToolTalk Document and Media Exchange Message Set.

Abstract(Request)

Requests a summary representation of a document.

Synopsis

```
[file] Abstract ( in          mediaType    contents,
                  out         mediaType    output
                  in          boolean      inquisitive,
                  in          boolean      covert
                  [in         messageID    counterfoil]
                  [inout      vector       size] );
```

Description

The Abstract message requests that a summary representation of a document (for example, an icon or a video frame raster) be returned. The abstraction is the best possible representation of the document within the size constraints of the sending tool.

Note – You can extract a list of the installed mediaType-to-mediaType mappings from the ToolTalk Types Database.

Required Arguments

mediaType contents
: The contents of the document.

  If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

mediaType output
: The abstracted document.

boolean inquisitive
: The boolean value that indicates whether the recipient is allowed to seek user input about interpretation options.

Note – However, even if this value is *true*, the recipient is not required to seek the input.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

boolean covert
The boolean value that indicates whether the recipient is allowed to make itself apparent to the user as it performs the interpretation.

Note – However, even if the value is *false*, the recipient is not required to make itself apparent.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

Optional Arguments

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

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

vector size

- On input, the maximum size of the abstraction. The recipient returns an abstraction as close to this size as possible without exceeding this size.

- On output, the actual size of the abstraction to be returned; or, if the error TT_MEDIA_ERR_SIZE is returned, the smallest possible size the recipient is capable of returning.

Examples

In this scenario, a container application requires a representation of some video data. To abstract a representation frame of the video tool, you could send an Abstract request such as:

```
Abstract(  in      JPEG_Movie     movie(CW,
           out     Sun_Raster     frame, ...
```

to obtain a custom raster representation; or

```
Abstract(  in  JPEG_Movie movief(CW,
           out XPM        icon, ... );
```

to obtain a generic icon representation. In either case, the container application does not need to understand the JPEG_Movie format.

Errors

TT_MEDIA_ERR_SIZE

The specified size was too big or too small.

TT_MEDIA_ERR_FORMAT

The document is not a valid instance of the specified media type.

TT_DESKTOP_ENOENT

The specified file that does not exist.

TT_DESKTOP_ENODATA

The in-mode *contents* argument had no value and the file attribute of the message was not set.

Deposit(Request)

Saves the document to its backing store.

Synopsis

```
[file] Deposit(   in          mediaType     contents
                  [in         bufferID      beingDeposited
                  |in         messageID     commission] );

[file] Deposit(   in          mediaType     contents,
                  out         bufferID      beingDeposited
                  [in         title         docName] );
```

Description

The Deposit request saves the specified document to its backing store. This request is different from the Save request because the requestor (and not the handler) has the data that needs to be written. Do not use file-scoping with the Deposit request: if the sending tool knows in which file the document belongs, it should perform the save operation itself.

Required Arguments

mediaType contents
: The contents of the document.

  If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

bufferID beingDeposited
messageID commission
The Identifier of the buffer to be deposited to backing store. The identifier is either a bufferID returned or the messageID of the edit request that created this buffer.

If the `beingDeposited` argument is an `out` parameter, a new document is created and the handling container application must save the document and return a new bufferID for it.

Optional Arguments

title docName
The name of the document.

Example

You can use the Deposit to allow the user to checkpoints (for example, via a "Save" menu item) modifications to a document that is the subject of a session-scoped Edit request in progress.

Editors can issue the second variant of this request to allow the user to create extra documents 'near' the document that was just edited; for example, when each document in the series serves as the template or starting point for the next document.

Errors

`TT_DESKTOP_ENOENT`

The specified file 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 specified media type.

TT_DESKTOP_EACCES

The document is read-only.

Display(Request)

Displays a document.

Synopsis

```
[file] Display(   in          mediaType    contents
                  [in         messageID    counterfoil]
                  [in         title        docName] );
```

Description

The Display message requests that a document be displayed. *Display* is a generic term for the operation the player performs; for example, an audiotool displays sound. The Display request invokes the requested playback mechanism (such as a video tool, or an audio tool). The receiving tool decides:

- when the display operation is complete.
- what user gesture signals that the display is completed (that is, what determines that the user has signaled “I have completed the display.”).
- the action it takes after it has replied to the request.

Note – The display request does not allow changes to be saved back to the source data; however, a tool that supports a “save as” operation may allow edits to be saved back to the document.

Required Arguments

mediaType contents
: The contents of the document. If this argument is empty (i.e., 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 of the contents argument shall be string, unless nulls are legal in the given mediaType, in which case the data type shall be bytes.

Optional Arguments

messageID counterfoil
: The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

---

title docName
: The name of the document.

Examples

1. To display a PostScript document, send a Display request with a first argument whose vtype is “PostScript” and whose value is a vector of bytes such as:

   ```
   %!^J/inch {72 mul} def...
   ```

   where “^J” is the newline character (octal 12).

2. To display a PostScript document contained in a file, send a Display request, scoped to that file with a first argument whose vtype is “PostScript” and whose value is not set.

Errors

`TT_DESKTOP_ENOENT`
: The specified file 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.

Display, Edit(Request)

Loads an X11 selection for display or edit.

Synopsis

```
Display(     in       selection    selname,
             in       integer      item,
                      in           string     target,
                      in           boolean    askMe,
                      [in          messageID  counterfoil] );

Edit(        in       selection    selname,
             in       integer      item,
             in       string       target,
             in       boolean      askMe,
             [in      messageID    counterfoil] );
```

Description

The Display and Edit messages request that the selected data be displayed or edited (respectively). Optionally, the requester may perform the display or edit operation. This request is used most often by editors into which a selection has been drag-loaded.

Required Arguments

selection selname
: The selection to be displayed or edited.

integer item
: The part of the disjoint selection to be displayed or edited. Items are numbered from zero; a value of -1 means all the items in the selection.

string target
: The target to which the selection is to be converted before it is displayed or edited.

boolean askMe
If this value is true, the handler should send a Display or Edit request with a TT_HANDLER address directly back to the requester.

If this value false, the handler should attempt to display or edit the selection.

Optional Arguments

messageID counterfoil
The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

Errors

`TT_DESKTOP_EINVAL`
The specified selname, item, or target is invalid.

Note – If the askMe argument is true and the consequent Display or Edit request fails, the ReEdit request is failed with the status code of the failed request.

`TT_DESKTOP_ENOENT`
The specified file 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.

Edit(Request)

Edits or composes a document.

Synopsis

```
[file] Edit( [in]out   mediaType     contents
             [in       messageID     counterfoil]
             [in       title         docName] );
```

Description

The Edit message requests that a document be edited and a reply containing the new contents be returned when the editing is completed. The receiving tool decides:

- when the edit operation is complete.
- what user gesture signals that the edit is completed (that is, what determines that the user has signaled “I have completed the edit.”).
- the action it takes after it has replied to the request.

If a tool supports a “save” or “checkpoint” operation during editing, it can send a Deposit request back to the tool that requested the edit.

Required Arguments

mediaType contents
: The contents of the document. If the message is file-scoped, the contents argument has no value, and the document is contained in the scoped file. The data type of the contents argument is *string* unless nulls are legal in the given mediaType; if nulls are legal, the data type is *bytes*. If the contents argument is mode `out`, a new document is to be composed and its contents to be returned in this argument.

Optional Arguments

messageID counterfoil
: The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

---

title docName
: The name of the document.

Examples

1. To edit an X11 “xbm” bitmap, send an Edit request with a first argument whose vtype is “XBM” and whose value is a string such as

   ```
   #define foo_width 44^J#define foo_height 94^J...
   ```

   where “^J” is the newline character (octal 12).

2. To edit an X11 “xbm” bitmap contained in a file, send an Edit request with a first argument whose vtype is “XBM” and whose value is not set, and scope the request to that file.

Errors

`TT_DESKTOP_ENOENT`
: The file that was alleged to contain the document does not exist.

`TT_MEDIA_NO_CONTENTS`
: The in-mode contents arg 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.

Interpret(Request)

Translates a document and displays the translation.

Synopsis

```
[file] Interpret( in         mediaType    contents,
                  in         mediaType    targetMedium,
                  in         boolean      inquisitive,
                  in         boolean      covert
                  [in        messageID    counterfoil]
                  [in        title        docName ] );
```

Description

The Interpret message translates a document from one media type to another and displays the translation.

Note – The translation is the best possible representation of the document in the target media type; however, it is possible that the resulting representation cannot be perfectly translated back into the original document.

The Interpret request is equivalent to issuing a Translate request followed by a Display request. The Interpret message is a useful optimization when the sender has no interest in retaining the translation.

Note – It is possible to extract from the ToolTalk types database a list of the installed Translate mediaType-to-mediaType mappings.

Required Arguments

mediaType contents
The contents of the document.

If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

mediaType targetMedium
An empty argument whose vtype indicates the mediaType into which the document is to be translated before it is displayed.

boolean inquisitive
The boolean value that indicates whether the recipient is allowed to seek user input about interpretation options.

Note – However, even if this value is *true*, the recipient is not required to seek the input.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

boolean covert
The boolean value that indicates whether the recipient is allowed to make itself apparent to the user as it performs the interpretation.

Note – However, even if the value is *false*, the recipient is not required to make itself apparent.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

Optional Arguments

messageID counterfoil
The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

title docName
The name of the document.

Examples

To request a string to be spoken, send an Interpret request such as the following:

```
Interpret( in ISO_Latin_1 contents, in Sun_Audio targetMedium )
```

The ToolTalk service will then pass this request to the appropriate third party server in your environment.

Errors

`TT_DESKTOP_ENOENT`
The specified file 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.

Print(Request)

Prints a document.

Synopsis

```
[file] Print(     in          mediaType     contents,
                  in          boolean       inquisitive,
                  in          boolean       covert
                  [in         messageID     counterfoil]
                  [in         title         docName ] );
```

Description

The Print message prints a document. In effect, the recipient assumes the user issued a “print...” command via the recipient's user interface. The recipient tool decides issues such as what it should do with itself after replying.

Required Arguments

mediaType contents
The contents of the document.

If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

boolean inquisitive
The boolean value that indicates whether the recipient is allowed to seek user input about interpretation options.

Note – However, even if this value is *true*, the recipient is not required to seek the input.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

boolean covert
The boolean value that indicates whether the recipient is allowed to make itself apparent to the user as it performs the interpretation.

Note – However, even if the value is *false*, the recipient is not required to make itself apparent.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

Optional Arguments

messageID counterfoil
The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

title docName
The name of the document.

Examples

Printing a PostScript Document

To print a PostScript document,

```
Print(      in PostScript          contents,
            in boolean             inquisitive,
            in boolean             covert)
```

where the first argument is vtype `PostScript` whose value is a a vector of bytes.

Printing a PostScript Document Contained in a File

To print a PostScript document contained in a file,

```
Print(      in PostScript          contents,
            in boolean             inquisitive,
            in boolean             covert)
```

where the file attribute is set to filename, and the first argument is vtype `PostScript` whose value is not set.

Errors

`TT_DESKTOP_ENOENT`

The specified file 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.

Translate(Request)

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

Synopsis

```
[file] Translate( in          mediaType    contents,
                  out         mediaType    output,
                  in          boolean      inquisitive,
                  in          boolean      covert
                  [in         messageID    counterfoil]
```

Description

The Translate message requests that a document be translated from one media type to another media type and that a reply containing the translation be returned. The translation is the best possible representation of the document in the target media type; however, it is not guaranteed that the resulting translation can be perfectly translated back into the original document.

Note – You can extract a list of the installed mediaType-to-mediaType mappings from the ToolTalk Types Database.

Required Arguments

mediaType contents
: The contents of the document.

  If this argument is empty (that is, it has a value of (char *) 0), the contents of the document are contained in the file named in the message's file attribute. If nulls are not legal in the given mediaType, the data type of the contents argument is `string`; otherwise, the data type is `bytes`.

mediaType output
: The translated document.

boolean inquisitive
The boolean value that indicates whether the recipient is allowed to seek user input about interpretation options.

Note – However, even if this value is *true*, the recipient is not required to seek the input.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

boolean covert
The boolean value that indicates whether the recipient is allowed to make itself apparent to the user as it performs the interpretation.

Note – However, even if the value is *false*, the recipient is not required to make itself apparent.

If both the *inquisitive* and `covert` values are true, the recipient should attempt to limit (for example, through iconification) its presence to the minimum required to receive any user input requested.

Optional Arguments

messageID counterfoil
The unique string created by the message sender (typically by concatenating a procID and a counter) to give both sender and receiver a reference to this request in other correspondence. Include this argument if the sender anticipates a need to communicate with the handler about this request before it is completed (for example, to cancel the request).

Note – When this argument is included and the handler determines that an immediate reply is not possible, then the handler should immediately send at least one Status notice point-to-point back to the requestor so as to identify itself to the requestor.

Examples

Speech-to-Text Translation

To translate speech to text, send a Translate request such as the following:

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

Optical Character Recognition (OCR)

To translate a GIF format bit image to text, send a Translate request such as the following:

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

Errors

`TT_DESKTOP_ENOENT`

The specified file 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.

Glossary

CAD

Computer-aided design.

CASE

Computer-aided software engineering.

Category

Attributes of a pattern that indicate whether the application wants to handle requests that match the pattern or only observe the requests.

Classing Engine (CE)

Identifies the characteristics of desktop objects; that is, it stores attributes such as print method, icons, and file opening commands of desktop objects.

Classing Engine tables

The types database read by the OpenWindows Classing Engine.

Note – The `ttce2xdr` command converts the ToolTalk types stored in the Classing Engine database used by ToolTalk Versions 1.0.x to the XDR-format database used in ToolTalk Version 1.1 and later.

---

contexts

A way to associate arbitrary pairs (that is, <name. value> pairs) with ToolTalk messages and patterns.

dynamic message patterns

Provides message pattern information while your application is running.

fail a request

Inform a sending application that the requested operation cannot be performed.

fd

File descriptor.

file

A container for data that is of interest to applications.

libtt

The ToolTalk application programming interface (API) library.

handle a message

To perform the operation requested by the sending application; to send a ToolTalk reply to a request.

initial session

The ToolTalk session in which the application was started.

mark

An integer that represents a location on the API stack.

message

A structure that the ToolTalk service delivers to processes. A ToolTalk message consists of an operation name, a vector of type arguments, a status value or string pair, and ancillary addressing information.

message callback

A client function. The ToolTalk service invokes this function to report information about the specified message back to the sending application; for example, the message failed or the message caused a tool to start.

message pattern

Defines the information your application wants to receive.

message protocol

A message protocol is a set of ToolTalk messages that describe operations the applications agree to perform.

notice

A notice is informational, a way for an application to announce an event.

object content

Object content is managed by the application that creates or manages the object and is typically a piece, or pieces, of an ordinary file: a paragraph, a source code function, or a range of spreadsheet cells.

object files

Files that contain object information. Applications can query for objects in a file and perform operations on batches of objects.

object-oriented messages

Messages addressed to objects managed by applications.

object specification (spec)

An object specification (known as a spec) contains standard properties such as the type of object, the name of the file in which the object contents are located, and the object owner.

object type (otype)

The object type (otype) for your application provides addressing information that the ToolTalk service uses when delivering object-oriented messages.

object type identifier (otid)

Identifies the object type.

observe a message

To only view a message without performing any operation that may be requested.

observe promise

Guarantees that the ToolTalk service will deliver a copy of each matching message to ptypes with an observer signature of start or queue disposition. The ToolTalk service will deliver the message either to a running instance of the ptype, by starting an instance, or by queueing the message for the ptype.

opaque pointer

A value that has meaning only when passed through a particular interface.

package

A group of components that together create some software. A package contains the executables that comprise the software, but also includes information files and scripts. Software is installed in the form of packages.

pattern callback

A client function. The ToolTalk service invokes this function when a message is received that matches the specified pattern.

process

One execution of an application, tool, or program that uses the ToolTalk service.

process-oriented messages

Messages addressed to processes.

procid

The process identifier.

ptid

The process type identifier.

ptype

The process type.

reject a request

Tells the ToolTalk service that the receiving application is unable to perform the requested operation and that the message should be given to another tool.

request

A request is a call for an action. The results of the action are recorded in the message, and the message is returned to the sender as a reply.

rpc.ttdbserverd

The ToolTalk database server process.

scope

The attribute of a message or pattern that determines how widely the ToolTalk service looks for matching messages or patterns.

sessid

Identifies the session.

session

A group of processes that are related either by the same desktop or the same process tree.

signatures

A pattern in a ptype or otype. A signature can contain values for disposition and operation numbers.

- Ptype signatures *(psignatures*) describe the procedural messages that the program wants to receive.
- Otype signatures (*osignatures*) define the messages that can be addressed to objects of the type.

spec

See object specification.

static message patterns

Provides an easy way to specify the message pattern information if you want to receive a defined set of messages.

tool manager

A program used to coordinate the development tools in the environment.

ToolTalk Types Database

The database that stores ToolTalk type information.

`ttdbck`

Check and repair utility for the ToolTalk database.

ttsession

The ToolTalk communication process.

tt_type_comp

The ToolTalk type compiler.

wrapped shell commands

ToolTalk-enhanced shell commands. These commands safely perform common file operations on ToolTalk files.

xdr format tables

The types database read when `ttsession` is invoked.

Index

A

B

C

D

E

F

G

I

J

R

S

T

U

V

W