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.

4

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()`

4

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.

4

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()`

4

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.

4

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`

4

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()`

4

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)`

4

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.

4

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.

4

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()`

4

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`

4

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`

4

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.

4

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`

4

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.

4

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.

4

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.

4

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`

4

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`

4

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.

4

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.

4

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`

4

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()`

4

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`

4

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 "$OPENWINHOME/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`

4

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.

4

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()`

4

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.

4

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`

4

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()
```

4

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.

4

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`

4

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`

4

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.

4

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.

4

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`

4

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`

4

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`

4

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

4

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.

4

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.

4

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`

4

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.

4

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.
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.

4

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.

4

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.

4

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()`

4

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()`

4

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()`

4

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()`

4

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.

`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.
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.

4

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()`

4

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()`

4

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.

4

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()`

4

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`

4

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`

4

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`

4

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()`

4

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`

4

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`

4

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()`

4

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()`

4

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.

4

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.

4

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`

4

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()`

4

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.

4

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`

4

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`

4

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`

4

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`

4

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

4

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`

4

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.

4

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.

4

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.

4

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`

4

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.

4

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.

4

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()`

4

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.

4

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.

4