Managing Information Storage 13

To simplify your application storage management, the ToolTalk service copies all information your application provides to the ToolTalk service and also provides you with a copy of the information it returns to your application.

13.1 Information Provided to the ToolTalk Service

When you provide a pointer to the ToolTalk service, the information referenced by the pointer is copied. You can then dispose of the information you provided; the ToolTalk service will not use the pointer again to retrieve the information.

13.2 Information Provided by the ToolTalk Service

The ToolTalk service provides an allocation stack in the ToolTalk API library to store information it gives to you. For example, if you ask for the sessid of the default session with `tt_default_session`, the ToolTalk service returns the address of the character string in the allocation stack (a `char *` pointer) that contains the sessid. After you retrieve the sessid, you can dispose of the character string to clean up the allocation stack.

Note – Do not confuse the API allocation stack with your program's runtime stack. The API stack will not discard information until instructed to do so.

13.3 Calls Provided to Manage the Storage of Information

The ToolTalk service provides the calls listed in Table 13-1 to manage the storage of information in the ToolTalk API allocation stack:

Table 13-1 Managing ToolTalk Storage

Return Type	ToolTalk Function	Description
int	tt_mark(void)	Marks information returned by a series of functions.
void	tt_release(int mark)	Frees information returned by a series of functions.
caddr_t	tt_malloc(size_t s)	Reserves a specified amount of storage in the allocation stack for your use.
void	tt_free(caddr_t p)	Frees storage set aside by `tt_malloc`. This function takes an address returned by the ToolTalk API and frees the associated storage.

Marking and Releasing Information

The `tt_mark` and `tt_release` functions are a general mechanism to help you easily manage information storage. The `tt_mark` and `tt_release` functions are typically used at the beginning and end of a routine where the information returned by the ToolTalk service is no longer necessary once the routine has ended.

Marking Information for Storage

To ask the ToolTalk service to mark the beginning of your storage space, use `tt_mark`. The ToolTalk service returns a mark, an integer that represents a location on the API stack. All the information that the ToolTalk service subsequently returns to you will be stored in locations that come after the mark.

Releasing Information No Longer Needed

When you no longer need the information contained in your storage space, use `tt_release` and specify the mark that signifies the beginning of the information you no longer need.

Example of Marking and Releasing Information

The following code sample calls `tt_mark` at the beginning of a routine that examines the information in a message. When the information examined in the routine is no longer needed and the message has been destroyed, `tt_release` is called with the mark to free storage on the stack.

```
/*
 * Get a storage mark so we can easily free all the data
 * ToolTalk returns to us.
 */

mark = tt_mark();

if (0==strcmp("ttsample1_value", tt_message_op(msg_in))) {
       tt_message_arg_ival(msg_in, 0, &val_in);
       xv_set(gauge, PANEL_VALUE, val_in, NULL);
}

tt_message_destroy(msg_in);
tt_release(mark);
return;
```

Allocating and Freeing Storage Space

The `tt_malloc` and `tt_free` functions are a general mechanism to help you easily manage allotted storage allocation.

Allocating Storage Space

`tt_malloc` reserves a specified amount of storage in the allocation stack for your use. For example, you can use `tt_malloc` to create a storage location and copy the sessid of the default session into that location.

Freeing Allocated Storage Space

To free storage of individual objects that the ToolTalk service provides you pointers to, use `tt_free`. For example, you can free up the space in the API allocation stack that stores the sessid after you have examined the sessid. `tt_free` takes an address in the allocation stack (a `char *` pointer or an address returned from `tt_malloc`) as an argument.

13.4 Special Case: Callback and Filter Routines

The way that the ToolTalk service behaves toward information passed into filter functions and callbacks is a special case. Callback and filter routines called by the ToolTalk service are called with two kinds of arguments:

- Context arguments — the arguments you passed into the API call that triggered the callback. These arguments point to items owned by your application.
- Pointers to API objects — the address of message or pattern attributes in storage.

The context arguments are passed from the ToolTalk service to your application. The API objects referenced by pointers are freed by the ToolTalk service as soon as your callback or filter function returns. If you want to keep any of these objects, you must copy the objects before your function returns.

Note – The way that the ToolTalk service behaves toward information passed into filter functions and callbacks is a special case. In all other instances, the ToolTalk service stores the information in the API allocation stack until you free it.

Callback Routines

One of the features of the ToolTalk service is callback support for messages, patterns, and filters. Callbacks are routines in your program that ToolTalk calls when a particular message arrives (*message callback*) or when a message matches a particular pattern you registered (*pattern callback*).

To tell the ToolTalk service about these callbacks, add the callback to a message or pattern before you send the message or register the pattern.

Filter Routines

When you call file query functions such as `tt_file_objects_query`, you point to a filter routine that the ToolTalk service calls as it returns items from the query. For example, you could use filter routine used by the ToolTalk file query function to find a specific object. The `tt_file_objects_query` function returns all the objects in a file and runs the objects through a filter routine that you provide. Once your filter routine finds the specified object, you can use `tt_malloc` to create a storage location and copy the object into the location. When your filter function returns, the ToolTalk service will free all storage used by the objects in the file but the object you stored with the `tt_malloc` call will be available for further use.

13