Dynamic Message Patterns 9

9.1 Defining Dynamic Messages

The dynamic method provides message pattern information while your application is running. You create a message pattern and register it with the ToolTalk service. You can add callback routines to dynamic message patterns that the ToolTalk service will call when it matches a message to the pattern.

To create and register a dynamic message pattern, you allocate a new pattern object, fill in the proper information, and register it. When you are done with the pattern (that is, when you are no longer interested in messages that match it), either unregister or destroy the pattern. You can register and unregister dynamic message patterns as needed.

The ToolTalk functions used to create, register, and unregister dynamic message patterns are listed in Table 9-1.

Table 9-1 Functions for Creating, Updating, and Deleting Message Patterns

ToolTalk Function	Description
tt_pattern_create(void)	Create Pattern
tt_pattern_arg_add(Tt_pattern p, Tt_mode n, const char *vtype, const char *value)	Add string arguments
tt_pattern_barg_add(Tt_pattern m, Tt_mode n, const char *vtype, const unsigned char *value, int len)	Add byte array arguments

Note: The return type for all functions except tt_pattern_create is Tt_status; tt_pattern_create returns Tt_pattern.

Table 9-1 Functions for Creating, Updating, and Deleting Message Patterns (Continued)

ToolTalk Function	Description
tt_pattern_iarg_add(Tt_pattern m, Tt_mode n, const char *vtype, int value)	Add integer arguments
tt_pattern_xarg_add(Tt_pattern m, Tt_mode n, const char *vtype, xdrproc_t xdr_proc, void *value)	Adds an xdr argument to a byte array
tt_pattern_bcontext_add(Tt_pattern p, const char *slotname, const unsigned char *value, int length);	Add byte array contexts
tt_pattern_context_add(Tt_pattern p, const char *slotname, const char *value);	Add string contexts
tt_pattern_icontext_add(Tt_pattern p, const char *slotname, int value);	Add integer contexts
tt_pattern_address_add(Tt_pattern p, Tt_address d)	Add address
tt_pattern_callback_add(Tt_pattern p, Tt__message_callback_action f)	Add message callback
tt_pattern_category_set(Tt_pattern p, Tt_category c)	Set category
tt_pattern_class_add(Tt_pattern p, Tt_class c)	Add class
tt_pattern_disposition_add(Tt_pattern p, Tt_disposition r)	Add disposition
tt_pattern_file_add(Tt_pattern p, const char *file)	Add file
tt_pattern_object_add(Tt_pattern p, const char *objid)	Add object
tt_pattern_op_add(Tt_pattern p, const char *opname)	Add operation
tt_pattern_opnum_add(Tt_pattern p, int opnum)	Add operation number
tt_pattern_otype_add(Tt_pattern p, const char *otype)	Add object type
tt_pattern_scope_add(Tt_pattern p, Tt_scope s)	Ad scope
tt_pattern_sender_add(Tt_pattern p, const char *procid)	Add sending process identifier
tt_pattern_sender_ptype_add(Tt_pattern p, const char *ptid)	Add sending process type
tt_pattern_session_add(Tt_pattern p, const char *sessid)	Add session identifier
tt_pattern_state_add(Tt_pattern p, Tt_state s)	Add state
tt_pattern_user_set(Tt_pattern p, int key, void *v)	Set user
tt_pattern_register(Tt_pattern p)	Register pattern
tt_pattern_unregister(Tt_pattern p)	Unregister pattern
tt_pattern_destroy(Tt_pattern p)	Destroy message pattern

Note: The return type for all functions except tt_pattern_create is Tt_status; tt_pattern_create returns Tt_pattern.

Creating a Message Pattern

To create message patterns, use the `tt_pattern_create` function. You can use this function to get a handle or opaque pointer to a new pattern object, and then use this handle on succeeding calls to reference the pattern.

To fill in pattern information, use the `tt_pattern_`*<attribute>*`_add` and `tt_pattern_`*<attribute>*`_set` calls. You can supply multiple values for each attribute you add to a pattern. 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 you want any value to match. Some attributes are set and, therefore, can only have one value.

Adding a Message Pattern Callback

To add a callback routine to your pattern, use the `tt_pattern_callback_add` function.

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

When the ToolTalk service matches a message, it automatically calls your callback routine to examine the message and take appropriate actions. When a message that matches a pattern with a callback is delivered to you, it is processed through the callback routine. When the routine is finished, it returns `TT_CALLBACK_PROCESSED` and the API objects involved in the operation are freed. You can then use `tt_message_destroy` to destroy the message, which frees the storage used by the message, as illustrated in the following code sample.

```
Tt_callback_action
sample_msg_callback(Tt_message m, Tt_pattern p)
{
     ... process the reply msg ...

     tt_message_destroy(m);
     return TT_CALLBACK_PROCESSED;
}
```

Registering a Message Pattern

To register the completed pattern, use the `tt_pattern_register` function. After you register your pattern, you join the sessions or files of interest.

The following code sample creates and registers a pattern.

```
     /*
      * Create and register a pattern so ToolTalk knows we are
interested
      * in "ttsample1_value" messages within the session we join.
      */

     pat = tt_pattern_create();
     tt_pattern_category_set(pat, TT_OBSERVE);
     tt_pattern_scope_add(pat, TT_SESSION);
     tt_pattern_op_add(pat, "ttsample1_value");
     tt_pattern_register(pat);
```

Deleting and Unregistering a Message Pattern

Note – If delivered messages that matched the deleted pattern have not been retrieved by your application (for example, the messages might be queued), the ToolTalk service does not destroy these messages.

To delete a message pattern, use the `tt_pattern_destroy` function. This function first unregisters the pattern and then destroys the pattern object.

To stop receiving messages that match a message pattern without destroying the pattern object, use the `tt_pattern_unregister` to unregister the pattern.

The ToolTalk service will automatically unregister and destroy all message pattern objects when you call `tt_close`.

9.2 Updating Message Patterns with the Current Session or File

To update your message patterns with the session or file in which you are currently interested, join the session or file.

Joining the Default Session

When you join a session, the ToolTalk service updates your message pattern with the sessid. For example, if you have declared a ptype or registered a message pattern that specifies `TT_SESSION` or `TT_FILE_IN_SESSION`, use `tt_session_join` to join the default session. The following code sample shows how to join the default session.

```
/*
 * Join the default session
 */

tt_session_join(tt_default_session());
```

Table 9-2 lists the ToolTalk functions you use to join the session in which you are interested.

Table 9-2 ToolTalk Functions for Joining Default Sessions

Return Type	ToolTalk Function	Description
char *	tt_default_session(void)	Return default session id
Tt_status	tt_default_session_set(const char *sessid)	Set default session
char *	tt_initial_session(void)	Return initial session id
Tt_status	tt_session_join(const char *sessid)	Join this session
Tt_status	tt_session_quit(const char *sessid)	Quit session

Once your patterns are updated, you will begin to receive messages scoped to the session you joined.

Note – If you had previously joined a session and then registered a ptype or a new message pattern, you must again join the same session or a new session to update your pattern before you will receive messages that match your new pattern.

When you no longer want to receive messages that reference the default session, use the `tt_session_quit` function. This function removes the sessid from your session-scoped message patterns.

Joining Multiple Sessions

When you join multiple sessions, you will automatically get responses to requests and point-to-point messages but you will not get notices unless you explicitly join the new session. The following code sample shows how to join the multiple sessions.

```
tt_default_session_set(new_session_identifier);
tt_open();
tt_session_join(new_session);
```

In order to effectively use multiple sessions, you must store the session ids of the sessions in which you are interested in order to pass these identifiers to `tt_default_session_set` prior to opening a new session with `tt_open`; that is, you need to place the values (which `ttsession` stores in the environment variable `_SUN_TT_SESION`) in a file on the system so that other ToolTalk clients can access the value of a session id contained in that file and use it to open the non-default session. For example, you can store the session ids in a "well-known" file and then send a file-scoped message (indicating this file) to all clients which have registered an appropriate pattern. The client will then know to open the scoped-to file, read one or more session ids from it, and use these session ids (with `tt_open`) to open a non-default session. An alternative method is advertising the session ids by means of, for example, a name service or a third-party database.

Note – How `ttsession` session ids are stored and passed to interested clients is beyond the scope of the ToolTalk protocol and must be determined based on the architecture of the system.

Joining Files of Interest

When you join a file, the ToolTalk service automatically adds the name of the file to your file-scoped message patterns. For example, if you have declared a process type or registered a message pattern that specifies `TT_FILE` or `TT_FILE_IN_SESSION`, use the `tt_file_join` function to join files of interested. Table 9-3 lists the ToolTalk functions you use to express your interest in specific files.

Table 9-3 ToolTalk Functions for Joining Files of Interest

Return Type	ToolTalk Function	Description
char *	tt_default_file(void)	Join default file
Tt_status	tt_default_file_set(const char *docid)	Set default file
Tt_status	tt_file_join(const char *filepath)	Join this file
Tt_status	tt_file_quit(const char *filepath)	Quit file

When you no longer want to receive messages that reference the file, use the `tt_file_quit` function to remove the file name from your file-scoped message patterns.

9