Message Patterns 3

This chapter describes how to provide message pattern information to the ToolTalk service. The ToolTalk service uses message patterns to determine message recipients. After receiving a message, the ToolTalk service compares the message to all current message patterns to find a matching pattern. Once a match is made, the message is delivered to the application that registered the message pattern.

You can provide message pattern information to the ToolTalk service using either dynamic or static methods, or both. The method you choose depends on the type of messages you want to receive.

- If the types of messages you want to receive will vary while your application is running, the *dynamic method* allows you to add, change, or remove message pattern information after your application has started.
- If you want a message to start your application or to be queued if your application is not running, the *static method* provides an easy way to specify these instructions. The static method also provides an easy way to specify the message pattern information if you want to receive a defined set of messages. For more information, see Chapter 10, “Static Message Patterns.”

Regardless of the method you choose to provide message patterns to the ToolTalk service, you will want to update these patterns with each current session and file information so that you receive all messages that reference the session or file in which you are interested.

3

3.1 Message Pattern Attributes

The attributes in your message pattern specify the type of messages you want to receive. Although some attributes are set and have only one value, you can supply multiple values for most of the attributes you add to a pattern.

Table 3-1 provides a complete list of attributes you can put in your message patterns.

Table 3-1 ToolTalk Message Pattern Attributes

Pattern Attribute	Value	Description
Category	TT_OBSERVE TT_HANDLE	Declares whether you want to perform the operation listed in a message or only observe a message.
Scope	TT_SESSION TT_FILE TT_FILE_IN_SESSION TT_BOTH	Declares interest in messages about a session or a file, or both; Join a session or file after the message pattern is registered to update the sessid and filename.
Arguments	arguments or results	Declares the positional arguments for the operation in which you are interested.
Context	<name, value>	Declares the keyword or non-positional arguments for the operation in which you are interested
Class	TT_NOTICE TT_REQUEST	Declares whether you want to receive notices or requests, or both.
File	char *pathname	Declares the files in which you are interested. If the scope of the pattern does not require a file, the file is an attribute only.
Object	char *objid	Declares what objects in which you are interested.
Operation	char *opname	Declares the operations in which you are interested.
Otype	char *otype	Declares the type of objects in which you are interested.
address	TT_PROCEDURE TT_OBJECT TT_HANDLER TT_OTYPE	Declares the type of address in which you are interested.
disposition	TT_DISCARD TT_QUEUE TT_START TT_START+TT_QUEUE	Instructs the ToolTalk service how to handle messages to your application if an instance is not currently running.

Table 3-1 ToolTalk Message Pattern Attributes (Continued)

Pattern Attribute	Value	Description
sender	char *procid	Declares the sender in which you are interested.
sender_ptype	char *ptype	Declares the type of sending process in which you are interested.
session	char *sessid	Declares the session in which you are interested.
state	TT_CREATED TT_SENT TT_HANDLED TT_FAILED TT_QUEUED TT_STARTED TT_REJECTED	Declares the state of the message in which you are interested.

All your message patterns must at least specify:

- **Category** — Whether the application wants to perform operations listed in messages or only view messages.
  - Use `TT_OBSERVE` if you only want to observe messages.
  - Use `TT_HANDLE` if you want to perform operations requested by the messages.
- **Scope** — Whether the application is interested in messages about a particular session or file.
  - Use `TT_SESSION` to receive messages from other processes in your session.
  - Use `TT_FILE` to receive messages about the file you have joined.
  - Use `TT_FILE_IN_SESSION` to receive messages for the file you have joined while in this session.
  - Use `TT_BOTH` to receive both messages for the file, the session, or the file and the session you have joined.

The ToolTalk service compares message attributes to pattern attributes as follows:

- The ToolTalk service counts the message attribute as matched if:
  - No pattern attribute is specified.
  - The pattern does not name a context slot.

- The pattern has an empty context slot.

  The fewer pattern attributes you specify, the more messages you become eligible to receive.

- If there are multiple values specified for a pattern attribute, one of the values must match the message attribute value. If no value matches, the ToolTalk service will not consider your application as a receiver.
- If context slots are contained in the message, the ToolTalk service will not consider your application as a receiver unless:
  - A value specified in a context slot of a pattern matches the value specified in the message context slot.
  - When multiple context slots are specified in a message, each context slot value in the message matches a corresponding context slot value in the pattern.

3.2 Scope Attributes

You can specify the following types of scopes in your message patterns:

1. Scope to a session only.
2. Scope to a file only.
3. Scope only to a file in a particular session.
4. Scope to either or both a file and a session.

Note – File scopes are restricted to NFS and UFS file systems; you cannot scope a file across other types of file systems (for example, a tmpfs file system).

Scoping to a Session Only

The type `TT_SESSION` scopes to a session only. Static session-scoped patterns require an explicit `tt_session_join` call to set the scope value; dynamic session-scoped patterns can be set with either the `tt_session_join` call or the `tt_pattern_session_add` call.

Note – The session specified by these calls must be the default session.

Code Example 3-1 shows a static session-scoped pattern; Code Example 3-2 shows a dynamic session-scoped pattern.

Code Example 3-1 Static Session-Scoped Pattern

```
Obtain procid                  tt_open();

Ptype is scoped to session     tt_ptype_declare(ptype);
Join session                   tt_session_join(tt_default_session());
```

Code Example 3-2 Dynamic Session-Scoped Pattern with a File Attribute

```
Obtain procid                  tt_open();

Create pattern                 Tt_pattern pat = tt_create_pattern();
Add scope to pattern           tt_pattern_scope_add(pat, TT_SESSION);
Add session to pattern         tt_pattern_session_add (tt_default_session());
Register pattern               tt_pattern_register(pat);
```

Scoping to a File Only

The type `TT_FILE` scopes to a file only. Code Example 3-3 shows a static file-scoped pattern; Code Example 3-4 shows a dynamic file-scoped pattern.

Code Example 3-3 Static File-Scoped Pattern

```
Obtain procid                  tt_open();

Ptype is scoped to file        tt_ptype_declare(ptype);
Join file                      tt_file_join(file);
```

Code Example 3-4 Dynamic File-Scoped Pattern

```
Obtain procid                tt_open();

Create pattern               Tt_pattern pat = tt_create_pattern();
Add scope to pattern         tt_pattern_scope_add(pat, TT_FILE);
Add file to pattern          tt_pattern_file_add (pat, file);
Register pattern             tt_pattern_register(pat);
```

Scoping to a File in a Session

The type `TT_FILE_IN_SESSION` scopes to the specified file in the specified session only. A pattern with this scope set will only match messages that are scoped to both the file and the session. Code Example 3-5 adds the session and then registers the pattern.

Code Example 3-5 Adding a Session to the TT_FILE_IN_SESSION-Scoped Pattern

```
Obtain procid                tt_open();

Create pattern               Tt_pattern pat = tt_create_pattern();
Add scope to pattern         tt_pattern_scope_add(pat, TT_FILE_IN_SESSION);
Add file to pattern          tt_pattern_file_add(pat, file);
Add session to pattern       tt_pattern_session_add(pat, tt_default_session());
Register pattern             tt_pattern_register(pat);
```

Code Example 3-6 registers the pattern and then joins a session.

Code Example 3-6 Joining a Session to Set the Session of a TT_FILE_IN_SESSION-Scoped Pattern

```
Obtain procid                tt_open();

Create pattern               Tt_pattern pat = tt_create_pattern();
Add scope to pattern         tt_pattern_scope_add(pat, TT_FILE_IN_SESSION);
Add file to pattern          tt_pattern_file_add(pat, file);
Register pattern             tt_pattern_register(pat);
Join session                 tt_session_join(tt_default_session());
```

Code Example 3-7 sets the scope value for a static pattern.

Code Example 3-7 Setting the Scope Value for a TT_FILE_IN_SESSION Static Pattern

```
Obtain procid                tt_open();

Declare Ptype                Tt_ptype_declare(ptype);
Join File                    tt_file_join(file);
Join session                 tt_session_join(tt_default_session());
```

Scoping to a File and/or a Session

A `TT_BOTH`-scoped pattern will match messages that are scoped to the file, the session, or the file and the session. However, when you use this scope, you must explicitly make a `tt_file_join` call; otherwise, the ToolTalk service will only match messages that are scoped to both the file and session of the registered pattern. Code Example 3-8 and Code Example 3-9 show examples of how to use this scope.

Code Example 3-8 A Dynamic Pattern that Uses the TT_BOTH Scope

```
Obtain procid                 tt_open();

Create pattern                Tt_pattern pat = tt_create_pattern();
Add scope to pattern          tt_pattern_scope_add(pat, TT_BOTH);
Add session to pattern        tt_pattern_session_add(pat, tt_default_session());
Add file to pattern           tt_pattern_file_add (pat, file);
Register pattern              tt_pattern_register(pat);
```

Code Example 3-9 A Static Pattern that Uses the TT_BOTH Scope

```
Obtain procid                 tt_open();

Declare Ptype                 Tt_ptype_declare(ptype);
Join file                     tt_file_join(file);
Join session                  tt_session_join(tt_default_session());
```

Adding Files to Scoped Patterns

To match `TT_SESSION`-scoped messages and `TT_SESSION`-scoped patterns that have the same file attributes, you can add file attributes to `TT_SESSION`-scoped patterns with the `tt_pattern_file_add` call, as shown in Code Example 3-10.

Note – The file attribute values do not affect the scope of the pattern.

Code Example 3-10 Adding Two File Attributes to a Session-Scoped Pattern

```
Obtain procid                          tt_open();

Create pattern                         Tt_pattern pat = tt_create_pattern();
Add scope to pattern                   tt_pattern_scope_add(pat, TT_SESSION);
Add session to pattern                 tt_pattern_session_add(tt_default_session());
Add first file attribute to pattern    tt_pattern_file_add(pat, file1);
Add second file attribute to pattern   tt_pattern_file_add(pat, file2);
Register pattern                       tt_pattern_register(pat);
```

3.3 Context Attributes

ToolTalk *context*s are sets of <name, value> pairs explicitly included in both messages and patterns. ToolTalk contexts allow fine-grain matching.

You can use contexts to associate arbitrary pairs with ToolTalk messages and patterns, and to restrict the set of possible recipients of a message. One common use of the restricted pattern matching provided by ToolTalk context attributes is to create sub-sessions. For example, two different programs could be debugged simultaneously with tools such as a browser, an editor, a debugger, and a configuration manager active for each program. The message and pattern context slots for each set of tools contain different values; the normal ToolTalk pattern matching of these values keep the two sub-sessions separate.

Another use for the restricted pattern matching provided by ToolTalk context attributes is to provide information in environment variables and command line arguments to tools started by the ToolTalk service.

3.4 Disposition Attributes

Disposition attributes instruct the ToolTalk service how to handle messages to your application if an instance of the application is not currently running.

The disposition value specified in the static type definition of a pattern is the default disposition; however, if the message deposition specifies the handler ptype the default disposition value is over-ridden. For example, a message disposition specifies a static type definition for the ptype *UWriteIt* which includes the message signature *Display*. This message signature does not match any of the static signatures in the pattern. The ToolTalk service will follow the instructions for the disposition set in the message; for example, if the message disposition is `TT_START` and the *UWriteIt* ptype specifies a start string, the ToolTalk service will start an instance of the application if one is not running.