Chapter

11 *The PostMaster*

This chapter is a reference to the communication mechanism in the Telelogic Tau tools, the PostMaster. The functionality of the PostMaster makes it possible to integrate applications by using a well-defined means of communication.

Introduction to the PostMaster

Caution!

The PostMaster was originally designed and implemented for integrating tools in the SDL suite environment. Currently it integrates tools in the Telelogic Tau environment. Our experience is that the PostMaster is also suitable for integrating external tools and applications with the Telelogic Tau applications; one application area is for instance quick prototyping.

However, Telelogic does not support using the PostMaster as a communication mechanism between real-time applications, in a run-time environment.

The PostMaster is the mechanism used for communication between the different tools in Telelogic Tau. A C program generated by the Telelogic Tau tools can also take advantage of this communication mechanism. It can communicate with any application connected to the PostMaster that send messages according to a defined format. This makes it possible for an SDL simulator to communicate with, for instance, a user interface process for the Simulator.

The PostMaster also provides the basic means for the Open Telelogic Tau concept, see chapter 12, *The Telelogic Tau Public Interface*.

The PostMaster provides the following functionality:

- Starting an application and connecting to it
- Letting an application connect itself
- Letting an application send messages to a given recipient
- Making a “broadcast” of a message.

The PostMaster is a message passing service based on a selective broadcasting mechanism. It will distribute a copy of each message it receives to the tools subscribing to that type of message. By this, the PostMaster provides an integration mechanism between tools without the hard coupling between them that follows from conventional two part communication mechanisms.

Figure 163 illustrates some of the PostMaster concepts. The PostMaster maintains a list of which messages each tool subscribes to. Each tool has a PostMaster part for sending and receiving messages. In the figure, tool F broadcasts a message, i.e. the message is sent to the PostMaster. The

subscription lists of tool A and C (but not the lists of B and F) contains the message type. Accordingly the PostMaster broadcasts the message to tool A and C.

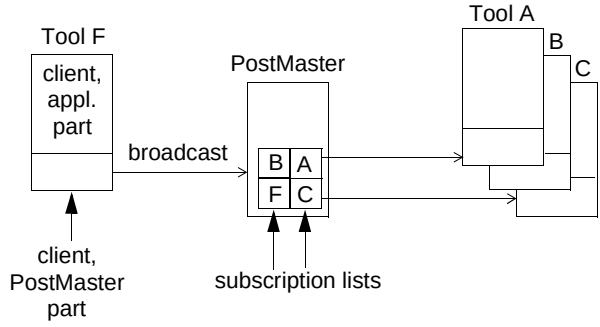

Figure 163: Example of a PostMaster broadcast

The PostMaster configuration is a file that informs the PostMaster about what tools and messages exist in the current context, i.e. it contains the message subscription lists. To include new tools or add new messages, the configuration must be edited.

For detailed information on the configuration, see "The PostMaster Configuration" on page 490.

PostMaster Reference

This section describes the external interface to the PostMaster, including messages to send and their format, contents of the configuration, and functions to call.

> **Note:**
>
> In order to promote a high throughput, it is strongly recommended that the PostMaster messages are consumed as soon as they are available.

PostMaster Messages

The PostMaster configuration defines a large number of messages that can be utilized in a Telelogic Tau system. A broad range of these messages are public, that is, they can be used externally. A description of these are found in *chapter 12, The Telelogic Tau Public Interface*.

All messages contain information about the identity of the sender, the time it was sent, the message type, and the size of an optional data part.

The optional data part can be seen as parameters to the message and is not interpreted by the PostMaster.

> **Note:**
>
> It is the responsibility of the tools using the PostMaster to define the format of the data part and to interpret it correctly.

The PostMaster Configuration

The PostMaster can be configured either statically at start up or dynamically during runtime. When a dynamic configuration is performed, the services Add Tool or Add Tool Subscription is used. The PostMaster is configured statically using the PostMaster configuration file(s)

The PostMaster configuration file(s) informs the PostMaster about what tools and messages exist in the current context. These files are often referred to as simply the *configuration*.

At start-up the PostMaster reads an environment variable `POSTPATH` which is a list of directories separated by colons **(on UNIX)** or semicolons **(in Windows)**. In these directories, the PostMaster searches for configuration files.

Such files should be named `post.cfd` and are read by the PostMaster whenever it is invoked, for instance when the SDL suite or the TTCN suite is started.

It is not possible to use C preprocessor statements or symbols in the configuration file.

Caution!

The file `post.cfd` residing in `$telelogic/sdt/bin` and `$telelogic/itex/bin` **(on UNIX)**, or in the Telelogic Tau installation directory **(in Windows)**, defines the existing tools in the environment and must not be edited; otherwise unpredictable behavior may occur.

It is possible to have local configurations which extends the standard PostMaster configuration set up by the Telelogic Tau tools, by simply defining the `POSTPATH` variable with a directory holding the extended configuration.

File Contents and Syntax

The configuration is a list of tool identities. Each tool identity is bound to an executable, a subscription list of messages and optionally a limit of instances of that tool. Each tool in the file is described in the following way:

```
Tool <tool number>:<executable>:<instance limit>
<message number>;
<message number>;
...
```

Description

- `<tool number>`

  is an integer defined by `SET_` symbols in the file `sdt.h`, and by `IET_` symbols in the file `itex.h`. The number 27000 (`SET_SDLENV`) is used for tools acting as an SDL environment.

- <executable>

  is the name of the executable file associated with the tool. It is used by the PostMaster after receiving an SESTART message when a tool is to be started.

  A file path may prefix the filename of the executable in order to explicitly tell where to find the tool. In this case the complete file description (path + filename) should be surrounded by a pair of quotes (i.e. "/home/sdt/demo/demotool" **(on UNIX)** or "c:\sdt\demo\demotool" **(in Windows)**).

- <instance limit>

  is the maximum number of concurrent running instances of the tool. It may be wildcarded with a '*' to a default value (SPMAXNOOFINSTANCES).

- <message number>

  is an integer defined by the symbols starting with SE in sdt.h. It is normally specified as <tool number> + <nr>. Symbols which are used in the TTCN suite have the prefix IE and are defined in itex.h.

Adding Tools and Messages

To include new tools or add new messages, the configuration file must be edited. However, the tools and subscription lists existing in the original configuration must not be changed.

To **include a new tool**, follow the steps below. Note that it is not required to include a new tool in the configuration if it only serves as an SDL/TTCN environment that does not need to be started from other tools by using the Start service.

1. Select a tool number not conflicting with the ones already existing in the configuration, preferably a number greater than 100,000. The number chosen must be equally divisible by 1,000.
2. Make a new tool description in the configuration, using the syntax described above.
3. Define a new SET_ (IET_) symbol in the file sdt.h (itex.h) so that the tool number can be easily accessed in the code.

Example 24 ----------

```
#define SET_MYTOOL   110000
```

To **add a new message**, follow these steps:

1. Decide upon a tool that the new message “belongs to” or “is defined by,” which should not be one of the pre-defined tools in the original configuration.
2. Define a new `SE` (`IE`) symbol in the file `sdt.h` (`itex.h`) as the tool symbol plus an ordinal number, not conflicting with any other message symbol.

Example 25 ----------

```
#define SEMYMESSAGE   SP_MESSAGE(SET_MYTOOL+1)
```

----

3. Use the numeric value of the message symbol when adding the new message to a subscription list in the configuration.

Example 26 ----------

```
110000+1;
```

Environment Variables

The PostMaster recognizes a number of environment variables setting the context in which the PostMaster is to operate. They are read when the PostMaster starts.

PostMaster Environment Variables

The following environment variables are recognized by the PostMaster:

- `POSTPORT`

  UNIX only: Should be set to a valid TCP port number. Causes the PostMaster to try to listen to this port number for possible connections. Makes it possible to connect applications running on other hosts than the PostMaster is running on. Is used in combination with the client side environment variables `POSTHOST` and `POSTPORT`.

- `POSTPATH`

A list of directories separated by colons **(on UNIX)** or semicolons **(in Windows)**, where to search for configuration files named `post.cfd`.

- `POSTDEBUG`

  If set, the PostMaster will log active tools and sent messages as an MSC log. The environment variable takes two optional parameters. The first tells the filename to put the log on. If the parameter is set to `'-e'` the log is put on `stdout`. The second parameter tells the log level. Normally only public messages are logged. But if set to a value > 2 all messages will be logged.

  If no parameters are submitted, the log will be stored on the file `post.mpr`

  See also "Start MSC Log" on page 594

- `STARTTIMEOUT`

  When the PostMaster executes its start service it assumes that the started client will connect, via SPInit, within a certain amount of time. This timeout (in seconds) sets the limit when the PostMaster considers the start having failed. Default is 60 seconds.

  If the Telelogic Tau tools environment is running on slow computer or on a network which is heavily loaded, start-up of a Telelogic Tau tool might fail due to this timeout. In such case, this environment variable should be increased to an appropriate value.

PostMaster Application Library Environment Variables

The following environment variables are recognized by the PostMaster application library. These environment variables are read in the SPInit function.

- `POSTHOST`

  UNIX only: Should be set to a hostname, on which a PostMaster runs. This PostMaster should be started with the environment variable `POSTPORT` set.

- `POSTPORT`

  UNIX only: Should be set to a valid TCP port which is allocated to a PostMaster. Corresponds to `POSTPORT` set for the PostMaster.

- POSTPID

  This environment variable is only of interest if multiple PostMasters are simultaneously active. If it is desired to make a connection to one particular PostMaster, this environment variable should be set to the process id corresponding to that PostMaster.

  If only one PostMaster is active, the application library automatically finds the PostMaster.

  If more than one PostMaster is running and this environment variable is not set, a connection is made to the PostMaster instance having the highest process id value.

Functional Interface

Communication with the PostMaster is based on a small set of fundamental functions:

SPInit	initialization
SPExit	termination
SPBroadcast	output message, broadcast
SPSendToTool	output message, sent to a certain tool
SPSendToPid	output message, sent to a certain PId
SPRead	input message
SPFree	frees memory allocated by SPRead or SPFindActivePostMasters
SPConvert	translate a symbolic message to an id
SPErrorString	Error string conversion
SPRegisterPMCallback	Message notification **(Windows only)**
SPQuoteString	Quoting strings
SPUnquoteString	Unquoting strings
SPFindActivePostMasters	find all PostMasters

Every function returns a value denoting success or failure of the associated operation.

Calling Conventions

The PostMaster functions should be called in the following ways from the external tool:

- The first thing to do is to initiate the connection with the PostMaster itself by calling SPInit. This is preferably done early in the main program. This initiation is private to the client and the PostMaster.
- When having successfully called SPInit, the tool should listen to the PostMaster message port and read the incoming messages by SPRead. The tool should then act upon the messages it subscribes on.
- If so desired, the tool should send messages by calling SPSendToTool, SPSendToPid or SPBroadcast.
- When the tool decides to terminate, it should broadcast the standard message SESTOPNOTIFY by calling SPBroadcast. Then it should disconnect from the PostMaster by calling SPExit.
- Every call to a PostMaster function should check the return value and act properly if an error is detected.

Variables

The following variables are defined.

```
extern INT16 sperrno;

extern INT16 spPortNO; (on UNIX)
extern int spPortNO; (in Windows)
```

sperrno contains an error code when a call to a PostMaster function failed. The error codes are defined in sdt.h and should be self-explanatory.

On UNIX, spPortNO contains a descriptor to the port where incoming messages from the PostMaster are found. **In Windows**, spPortNO contains a process identifier associated with the current process. The state of the variable prior to the call of SPInit is undefined.

These variables are found in the file post.h.

Caution!

Windows only: When linking with the PostMaster's dynamically linked libraries (`post.lib` and `post.dll`), the environment variable `USING_DLL` must be defined before including `post.h`. Example:

```
#define USING_DLL
#include "post.h"
#undef USING_DLL
```

Error Codes

When a function returns `SPERROR` which indicates an error or an not expected result of the function. In this case `sperrno` is set to one of the following values.

Error Code	Explanation
`SPNOSESSION`	The function `SPInit` has not been called successfully
`SPALREADYCONNECTED`	When calling `SPInit` more than once without disconnecting
`SPNOPOSTMASTER`	No PostMaster could be found when trying to connect
`SPNOCHANNEL`	The contact with the PostMaster is lost.
`SPNOMESSAGE`	No message available when trying to read by polling or after a timeout.
`SPTIMEDOUT`	The connection to the PostMaster timed out.
`SPNOSUCHPID`	Sending to a PId with a non positive value (<=0)
`SPNOMEMORY`	Cannot allocate anymore dynamic memory (rare)
`SPTOOLNOTFOUND`	When sending to a tool, this tool is not found in PostMaster configuration list

Error Code	Explanation
SPINVALIDMESSAGE	Sending a message with a NULL parameter but specifying the length greater than 0.
SPBADMESSAGE	A not supported message was to be sent.
SPMANYPOSTMASTERS	Many PostMasters running. Could not decide which one to connect to. **(Windows only)**
SPOLDPOSTMASTER	Old PostMaster running. **(Windows only)**

Functional Description

The available PostMaster functions are described on the following pages. The descriptions use the following format.

First, the function declaration is shown, including data types of parameters, followed by a short explanation of what the function does.

After that, in- and out parameters are described, together with possible return values and error codes.

SPInit

```
int SPInit(int toolType,
           char * argv0,
           *SPMList list);
```

`SPInit` initiates a session and establishes a connection with the PostMaster. See "Multiple PostMaster Instances" on page 518 for information on how to connect to a specific instance of the PostMaster.

Inparameters

- `toolType`

  The tool number identifies the tool and should be a value available in the PostMasters subscription list. For tools acting as an SDL/TTCN environment, it should be set to `SET_SDLENV`. See "Run-Time Considerations" on page 515 for more information.

- `argv0`

  The name of the executable (specified with its path). Normally argv[0] as passed to the program could be used.

- `list`

  Defines a list of predefined messages allowed to send. It also provides a set of mappings between textual strings and integer values for tools and messages. Normally the list provided in `sdt.h` (`itex.h`) is used. This list is later used by the function `SPConvert`, which translates between a textual string and the corresponding identifier.

Returns

- `>0`

  On UNIX, the PId of the calling process. Normally this PId corresponds to an UNIX PId. But if the PostMaster is started with the environment variable `POSTPORT` set, the PostMaster decides what PId value each client should get. In this case the numbering scheme gives the first started client 1, the next 2 and so on.

  In Windows, an identifier ("PId") which is used internally by the PostMaster in order to uniquely identify the calling process.

  This value could be used when sending messages and in comparisons with PIds contained in received messages.

- SPERROR

  SPInit failed, sperrno is set.

Errors

SPNOPOSTMASTER
SPTIMEDOUT
SPALREADYCONNECTED
SPNOMEMORY
SPMANYPOSTMASTERS
SPOLDPOSTMASTER

SPExit

```
int SPExit(void);
```

SPExit exits a session and disconnects the connection with the PostMaster. Subsequent calls to PostMaster functions will return the error code SPNOSESSION until a new SPInit is performed.

Returns

- SPOK

  Status OK.

- SPERROR

  SPExit failed, sperrno is set.

Errors

SPNOCHANNEL
SPNOSESSION

SPSendToTool

```
int SPSendToTool(int tool,
                 int event,
                 void * data,
                 int size);
```

`SPSendToTool` sends a message to the process of kind `tool`.

Inparameters

- `tool`

  Type of tool identifying the tool to send the message to. If such a tool is not running a service reply is sent by the PostMaster.

- `event`

  Type of message

- `data`

  Handle to an information block

- `size`

  Size of data.

Returns

- `SPOK`

  Status OK.

- `SPERROR`

  `SPSendToTool` failed, `sperrno` is set.

Errors

`SPNOCHANNEL`
`SPNOSESSION`
`SPNOMEMORY`
`SPBADMESSAGE`
`SPINVALIDMESSAGE`
`SPTOOLNOTFOUND`
`SPBADMESSAGE`
`SPNOSUCHPID`

SPSendToPid

```
int SPSendToPid(int pid,
                int event,
                void * data,
                int size);
```

`SPSendToPid` sends a message to the process which has process id `toPid`.

Inparameters

- `pid`

  PId of the message's receiver. If the specified PId does not exist, an `SEOPFAILED` message or a service reply is sent by the PostMaster.

- `event`

  Type of message.

- `data`

  Handle to an information block.

- `size`

  Size of data.

Returns

- `SPOK`

  Status OK.

- `SPERROR`

  `SPSendToPid` failed, `sperrno` is set.

Errors

`SPNOCHANNEL`
`SPNOSESSION`
`SPNOMEMORY`
`SPBADMESSAGE`
`SPINVALIDMESSAGE`
`SPTOOLNOTFOUND`
`SPNOSUCHPID`

SPBroadcast

```
int SPBroadcast(int event,
                void * data,
                int size);
```

`SPBroadcast` sends a message to all processes that subscribes on the message type.

Inparameters

- `event`

  Type of message.

- `data`

  Handle to an information block.

- `size`

  Size of data.

Returns

- `SPOK`

  Status OK.

- `SPERROR`

  `SPBroadcast` failed, `sperrno` is set.

Errors

SPNOCHANNEL
SPNOSESSION
SPNOMEMORY
SPBADMESSAGE
SPINVALIDMESSAGE

SPRead

```
int SPRead(int timeOut,
           int * pid,
           int * message,
           void ** data,
           int * len);
```

SPRead reads a message from the queue of unread messages that the PostMaster has sent. When the message is read, it is also consumed, i.e. removed from the queue. The function allocates the necessary amount of memory needed for the data component in the message. The application using this function is responsible for freeing the allocated memory with SPFree when it is no longer needed.

Inparameters

- timeOut

  Maximum amount of time (in milliseconds) that the function waits for a message. If a message has not arrived when timeOut expires, the function returns with return value SPERROR and sperrno is set to SPNOMESSAGE. If a message arrives, the function reads the message and returns immediately. If the desired behavior is to wait until a message arrives, which could mean forever, timeOut should be set to SPWAITFOREVER.

Outparameters

- pid

  PId of the tool sending the message.

- message

  Message identifier.

- data

  Pointer to data associated with the received message. SPFree should be used to free memory.

  Length of allocated data. If ASCII data is received, it is not assured that data is terminated by a ASCII NUL (‘\0’). The application should test if data[len-1] is ‘\0’.

Returns

- SPOK

Status OK.

- SPERROR

  SPRead failed, sperrno is set.

Errors

SPNOCHANNEL
SPNOMESSAGE
SPNOSESSION
SPNOMEMORY

SPFree

```
void SPFree (void * ptr)
```

SPFree should be used to free the memory allocated by SPRead. This is necessary when different compilers with different memory management are used.

Inparameters

- ptr

  A pointer to the memory block to be freed.

Returns

N/A.

SPErrorString

```
char * SPErrorString(int code);
```

Converts an error code into a textual string description of a tool or a message from the corresponding integer value.

Inparameters

- code

  Error code. Typically set by sperrno.

Returns

An descriptive error string corresponding to the error code.

SPConvert

```
int SPConvert(char * str);
```

Converts a textual description of a tool or a message to the corresponding integer value as provided by the parameter `list` in `SPInit`.

Inparameters

- `str`

  Textual description of the tool or message. The list provided in `SPInit` is used when searching for a mapping.

Returns

An integer value for the tool or message. If no mapping is found, `SPERROR` is returned.

SPRegisterPMCallback

```
typedef void (* SP_PM_MessageCallback) (void);
void SPRegisterPMCallback (SP_PM_MessageCallback
cb);
```

Windows only: Registers a callback function that gets called every time a new PostMaster message arrives. Registering this callback enables (32-bit) Windows applications to function correctly. Console applications need not use this function.

Inparameters

- `cb`

  The function to be called when a new PostMaster message is present. If `cb` is `NULL` the current callback is removed; otherwise the callback is replaced.

Returns

N/A.

SPQuoteString

```
int SPQuoteString(char *stringToQuote,
                  char *buffer,
                  int bufferLength,
                  int append)
```

`SPQuoteString` quotes a string. The following operations are performed:

- A quote is added to the beginning and end of the string.
- All quotes and backslashes in the string are escaped, i.e. a backslash character is added before them.

The quoted string can later be unquoted with a call to SPUnquoteString.

Inparameters

- `stringToQuote`

  Pointer to a null-terminated string. This is the string that should be quoted.

- `bufferLength`

  The size of the buffer (see the outparameter `buffer` below).

- `append`

  If the value of append is non-zero, `buffer` is supposed to already contain a null-terminated string. The result of the quoting operation will be appended to this string at the end.

  If several quoted strings are concatenated, they can be extracted and unquoted one at a time with calls to the SPUnquoteString function.

Outparameters

- `buffer`

  Pointer to a buffer where the resulting, quoted string will be returned. The buffer must be large enough to contain all the characters of the quoted string plus a trailing null character. The maximum buffer size needed can be quoted with the following formula:

  maximum buffer size = unquoted string size * 2 + 3

This includes space for escaping every character in the string, plus three bytes for quotes and null character. A larger buffer might of course be needed when appending (see `append` above).

The size of the buffer is given in `bufferLength`. If the quoted string does not fit in the buffer, the function will fail and return zero (further explained in the next section).

Returns

- `1`

  The call was successful.

- `0`

  The function call failed. The buffer was not large enough to contain the quoted representation of the string. The contents of the buffer are undefined.

SPUnquoteString

```
int SPUnquoteString(char *quotedString,
                    int inputLength,
                    char *buffer,
                    int bufferLength,
                    int position)
```

`SPUnquoteString` unquotes a string previously quoted with SPQuoteString, ignoring leading white-space characters. The following operations are performed:

- Any white-space characters in the beginning of the string are ignored. If the string is empty or contains only white-space characters, the function call fails and returns zero (see "Returns" on page 510).
- If the first character after white-spaces is a quote, the function assumes that this quote starts a string quoted with SPQuoteString. In this quoted string, backslashes escape the following character. The string is ended with a non-escaped quote.

  The returned string will have the escaping backslashes and the leading and closing quotes removed.
- If the first character after white-spaces is **not** a quote, the function will simply extract and return a substring up to but not including the next white-space character. If no further white-space characters are found, the rest of the string is returned. Backslashes and quotes have no special meaning when unquoting strings in this way.

Several concatenated quoted strings can be extracted with subsequent calls to this function by using the value returned in the `position` out parameter, see below.

Inparameters

- `quotedString`

  Pointer to a string containing the string previously quoted with SPQuoteString. The string should either be null-terminated or `inputLength` below should have a meaningful value.
- `inputLength`

  The maximum number of characters of `quotedString` that will be scanned. If `quotedString` is known to be null-terminated, a very large number should be supplied here.

- `bufferLength`

  The size of the buffer (see the outparameter `buffer` below).

Outparameters

- `buffer`

  A pointer to a buffer where the unquoted string will be returned. The buffer must be large enough to contain the resulting, unquoted string, including null character. A buffer one character larger than `inputLength` will always suffice. If the buffer is too small, the function call will fail and return zero (see "Returns" on page 510). The size of the buffer is given in `bufferLength`.

  `buffer` can also be null, in which case no unquoting will be performed. The value of `position` can still be interesting, though.

- `position`

  If this pointer is non-null it should point to an integer that will be filled in with the index of the character in the input string immediately following the extracted substring. This will be the character following a closing quote or a whitespace character or the terminating null character.

Returns

- `1`

  The function call was successful.

- `0`

  The function call failed. The supplied string might not be a valid quoted string according to the rules given above. The buffer might not be large enough to contain the result. The contents of the buffer and `position` are undefined.

SPFindActivePostMasters

```
int SPFindActivePostMaster (int *bufferPid,
                            char** bufferText,
                            int maxBufferlength);
```

Finds all PostMasters available for the application on the computer. Information retrieved is process id and a description in plain text.

Inparameters

- `maxBufferlength`

  The size of the arrays `bufferPid` and `bufferText`.

Outparamters

- `bufferPid`

  Pointer to an array where all process ids will be stored.

- `bufferText`

  Pointer to an array of strings telling when the PostMaster was started. `SPFindActivePostMaster` will allocate memory for all strings and the application must call SPFree in order to free the memory.

Returns

Number of PostMasters found.

Java Interface

The file `postmaster.java` contains a java class that encapsulates the postmaster interface for java programmers. The class contains a few fundamental methods.

`Init`	initialization
`SendToTool`	output message, sent to a certain tool
`SendToPid`	output message, sent to a certain Pid
`Broadcast`	output message, broadcast
`Read`	read a message
`Exit`	termination

Init

```
int Init(int toolType);
```

`SPInit` initiates a session and establishes a connection with the PostMaster. See "Multiple PostMaster Instances" on page 518 for information on how to connect to a specific instance of the PostMaster.

Inparameters

- `toolType`

  The tool number identifies the tool and should be a value available in the PostMasters subscription list.

Returns

See functional description for the C interface

SendToTool

```
int SendToTool(int toolType, int message,
               String data);
```

`SPSendToTool` sends a message to the process of kind `tool`.

Inparameters

- `toolType`

  Type of tool identifying the tool to send the message to. If such a tool is not running a service reply is sent by the PostMaster.

- `message`

  Type of message

- `data`

  Data to send

Returns

See functional description for the C interface

SendToPid

```
int SendToPid(int pId, int message, String data);
```

`SPSendToPid` sends a message to the process which has process id `toPid`.

Inparameters

- `pid`

  PId of the message's receiver. If the specified PId does not exist, an `SEOPFAILED` message or a service reply is sent by the PostMaster.

- `message`

  Type of message

- `data`

  Data to send

Returns

See functional description for the C interface

Broadcast

```
int Broadcast(int message, String data);
```

`SPBroadcast` sends a message to all processes that subscribes on the message type.

Inparameters

- `message`

  Type of message

- `data`

  Data to send

Returns

See functional description for the C interface

Read

```
int Read(int timeOut);
```

`SPRead` reads a message from the queue of unread messages that the PostMaster has sent. When the message is read, it is also consumed, i.e. removed from the queue. The data read by the last `Read` is copied to the `Sender`, `Message` and `Data` members of the class `postmaster`.

Inparameters

- `timeOut`

Maximum amount of time (in milliseconds) that the function waits for a message. If a message has not arrived when `timeOut` expires, the function returns with return value `SPERROR` and `sperrno` is set to `SPNOMESSAGE`. If a message arrives, the function reads the message and returns immediately. If the desired behavior is to wait until a message arrives, which could mean forever, `timeOut` should be set to `SPWAITFOREVER`.

Returns

See functional description for the C interface

Exit

```
int Exit();
```

`SPExit` exits a session and disconnects the connection with the PostMaster. Subsequent calls to PostMaster functions will return the error code `SPNOSESSION` until a new `SPInit` is performed.

Returns

See functional description for the C interface

Run-Time Considerations

Starting Up the PostMaster (in Windows)

When Telelogic Tau is started **in Windows**, an instance of the PostMaster is automatically started. No additional commands are needed for the PostMaster. There might however be situations when it is necessary to start the PostMaster stand-alone. This is described in the following sections. The examples only handle how the SDL suite is started from the "DOS" command prompt, but in many cases a shortcut with the analogous parameters can be created.

Start-Up

There are several possible ways to start the PostMaster itself and the tools that wish to communicate via the PostMaster. In principle, there are two main alternatives:

- Starting the tools when the PostMaster is running
- Starting them without having the PostMaster running.

We will exemplify the start-up methods by using the DemonGame simulator and a User Interface to the DemonGame simulator.

> **Note:**
>
> The PostMaster **must be started first**, but the communicating tools may be started in any order. An SDL simulator must also execute the commands Start-SDL-Env and Go **before communication** with another tool can start.

Starting When the PostMaster Is Present

When the SDL suite is started from the "DOS" or UNIX prompt, an instance of the PostMaster is automatically started. No additional commands are needed for the PostMaster in this case.

The DemonGame simulator is preferably started from the SDL suite, thus giving access to all simulation features. It can also be started directly from the "DOS" or UNIX prompt.

Starting from the "DOS" Prompt

Use the following command from the "DOS" prompt:

```
DemonGame.exe -post
```

> **Note:**
>
> **In Windows**, the simulator is a Windows GUI application and not a Console application. It is therefore not possible to run the simulator stand-alone, i.e. without **`-post`**.

Starting from the UNIX Prompt

Use the following command from the UNIX prompt:

```
DemonGame.sct -post
```

(Without the parameter, the simulator runs stand-alone, which is not desired in this case.) Starting the simulator this way restricts the possibilities of the simulation since there is no connection to the SDL suite tools. For instance, graphical trace is disabled.

The UI is preferably started directly from the UNIX prompt. It can also be started from the DemonGame simulator with the command Start-ITEX-Com, but this **requires** that the executable is named `sdtenv`. This is the name specified in the configuration for tool number 27000.

Starting Without the PostMaster

The SDL suite must be started directly from the "DOS" or UNIX prompt with the command:

```
$telelogic/bin/sdtpm <arg> &
```
(on UNIX)

```
<Installation Directory>\sdt <arg>
```
(in Windows)

(using one of the start-up arguments, see below).

The DemonGame simulator must also be started directly from the "DOS" or UNIX prompt, as described above.

> **Note:**
>
> **On UNIX:** If activating the PostMaster this way, the environment variable `POSTPATH` **must** be set to include the directory where the executables resides, typically `$sdtbin`.

The UI can be started in the same way as when the PostMaster is running (see above). It can also be started indirectly when starting the PostMaster by using the “DOS” or UNIX command:

```
$telelogic/bin/sdtpm -clients 27000 & (on UNIX)

<Installation Directory>\sdt -clients 27000 (in Windows)
```

This **requires** that the executable is named `sdtenv`, the name specified in the configuration for tool number 27000.

> **Note:**
>
> A tool communicating through the PostMaster can also be started from another tool by using the `SESTART` message. This requires that the tool to be started is specified in the configuration.

Start-Up Arguments

The PostMaster recognizes the following arguments at start-up. Normally they are not needed, but could be used for special purposes

- `-clients <toolid>` Used if the PostMaster should invoke a certain tool at start-up. `<toolid>` is set to the logical tool number of a tool to start as defined in `post.cfd`.
- `-noclients` Used if only the PostMaster is to be invoked without starting any clients. In this case the PostMaster enters an idle state where it waits for a client to connect

All other arguments are passed to the tool started.

SDT-2 Connections (UNIX only)

Applications or tools linked with an SDT 2.X PostMaster application library will not be compatible with SDT 3.X PostMaster. Trying to connect such an old application to an SDT 3.X PostMaster will result in an error message:

```
Postmaster cannot connect SDT2 tool:<tool> with pid:
<pid>
```

on standard output, if externally started, or as a service reply, if started via the start service, and the connection is aborted.

Version 3.4 PostMaster (Windows only)

If an old PostMaster (version 3.4 or older) is used the SPInit function will fail. The error code is SPOLDPOSTMASTER.

Applications or tools linked with an SDT 3.4 PostMaster application library will not be compatible with the SDT 3.6 PostMaster. Trying to connect such an old application to an SDT 3.6 PostMaster with the SPInit function will fail. The error code is SPNOPOSTMASTER.

Multiple PostMaster Instances

It is possible to have multiple instances of the PostMaster running, for instance when more than one Telelogic Tau tool has been started. In this case it should be made sure that the communicating tools are connected to the correct instance of the PostMaster.

On UNIX, the function `SPInit` by default looks for the PostMaster instance with the **highest process id** (PId) number and connects to it.

On Windows, the function `SPInit` by default fails if more than one PostMaster is found.

To connect to another PostMaster instance, you can set the environment variable `POSTPID` to the PId number of the desired PostMaster. If this variable is set, `SPInit` connects to the PostMaster instance with that PId number.

Configuration and Tool Search

The PostMaster configuration is read whenever a PostMaster instance is invoked. The environment variable `POSTPATH` defines a list of directories where the PostMaster looks for a configuration file named `post.cfd`.

The user can extend the normal configuration by defining the `POSTPATH` variable to a directory containing an extended configuration. When a Telelogic Tau tool is started normally, the directories containing the binaries are put as the first directories in the `POSTPATH` variable.

The same search order is applied when the start service is used. The supplied tool number is matched against the name of an executable in the configuration, which is then searched for as above.