TIBCO SmartSockets™

API Quick Reference

Software Release 6.8
July 2006

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE *TIBCO SMARTSOCKETS INSTALLATION GUIDE*). USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND / OR CHANGES IN THE PRODUCT(S) AND / OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

Contents

Preface

TIBCO SmartSockets™ is a message-oriented middleware product that enables programs to communicate quickly, reliably, and securely across:

- local area networks (LANs)
- wide area networks (WANs)
- the Internet

TIBCO SmartSockets takes care of network interfaces, guarantees delivery of messages, handles communications protocols, and directs recovery after system or network problems. This enables you to focus on higher-level requirements rather than the underlying complexities of the network.

This quick reference provides the detailed information you need to understand and use the SmartSockets system. Before using this quick reference, it is helpful to read the *TIBCO SmartSockets Tutorial* and work through the lessons in that book. The *TIBCO SmartSockets Tutorial* begins at a more basic level and is an excellent introduction to SmartSockets.

For an overview of the new features, changes, and enhancements in this Version 6.8 release, see the *TIBCO SmartSockets Installation Guide.*

Topics

Related Documentation

This section lists documentation resources you may find useful.

TIBCO Product Documentation

The following documents form the TIBCO SmartSockets documentation set:

- *TIBCO SmartSockets API Quick Reference*
- *TIBCO SmartSockets Application Programming Interface*
- *TIBCO SmartSockets C++ User's Guide*
- *TIBCO SmartSockets cxxipc Class Library*
- *TIBCO SmartSockets Installation Guide*
- *TIBCO SmartSockets Java Library User's Guide and Tutorial*
- *TIBCO SmartSockets .NET User's Guide and Tutorial*
- *TIBCO SmartSockets Tutorial*
- *TIBCO SmartSockets User's Guide*
- *TIBCO SmartSockets Utilities*
- *TIBCO SmartSockets C++ and Java Class Libraries*

  C++ class library and Java application programming interface (API) materials are available in HTML format only. Access the references through the TIBCO HTML documentation interface.

Using the Online Documentation

The SmartSockets documentation files are available for you to download separately, or you can request a copy of the TIBCO Documentation CD.

Conventions Used in This Manual

This manual uses the following conventions.

Typeface Conventions

This manual uses the following typeface conventions

Example	Use
`monospace`	This monospace font is used for program output and code example listing and for file names, commands, configuration file parameters, and literal programming elements in running text.
`monospace bold`	This bold monospace font indicates characters in a command line that you must type exactly as shown. This font is also used for emphasis in code examples.
Italic	Italic text is used as follows: • In code examples, file names etc., for text that should be replaced with an actual value. For example: "Select *install-dir* / `runexample.bat`." • For document titles. • For emphasis.
Bold	Bold text indicates actions you take when using a GUI, for example, click **OK**, or choose **Edit** from the menu. It is intended to help you skim through procedures when you are familiar with them and just want a reminder. Submenus and options of a menu item are indicated with an angle bracket, for example, **Menu > Submenu.**
	Warning. The accompanying text describes a condition that severely affects the functioning of the software.
	Note. Be sure you read the accompanying text for important information.
	Tip. The accompanying text may be especially helpful.

Notational Conventions

The notational conventions in the table below are used for describing command syntax. When used in this context, do not type the brackets listed in the table as part of a command line.

Notation	Description	Use
[ ]	Brackets	Used to enclose an optional item in the command syntax.
< >	Angle Brackets	Used to enclose a name (usually in *Italic*) that represents an argument for which you substitute a value when you use the command. This convention is not used for XML or HTML examples or other situations where the angle brackets are part of the code.
{ }	Curly Brackets	Used to enclose two or more items among which you can choose only one at a time. Vertical bars (|) separate the choices within the curly brackets.
...	Ellipsis	Indicates that you can repeat an item any number of times in the command line.

Identifiers

The term identifier is used to refer to a valid character string that names entities created in a SmartSockets application. The string starts with an underscore (_) or alphabetic character and is followed by zero or more letters, digits, percent signs (%), or underscores. No other special characters are valid. The maximum length of the string is 63 characters. Identifiers are not case-sensitive.

These are examples of valid identifiers:

```
EPS
battery_11
K11
__
_all
```

These are invalid identifiers:

```
20
battery-11
@com
$amount
```

Case

Function names are case-sensitive, and must use the mixed-case format you see in the text. For example, TipcMsgCreate, TipcSrvStop, and TipcMonClientMsgTrafficPoll are SmartSockets functions and must use the case as shown.

Monitoring messages are also case-sensitive, and should be all upper case, such as T_MT_MON_SERVER_NAMES_POLL_CALL. This makes it easy to distinguish them from option or function names.

Although option names are not case-sensitive, they are usually presented in text with mixed case, to help distinguish them from commands or other items. For example, Server_Names, Unique_Subject, and Project are all SmartSockets options.

Identifiers used with the products in the SmartSockets family are not case-sensitive. For example, the identifiers `thermal` and `THERMAL` are equivalent in all processes.

In UNIX, shell commands and filenames are case-sensitive, though they might not be in other operating systems, such as Windows. To make it easier to port applications between operating systems, always specify filenames in lower case.

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, please contact TIBCO Support as follows.

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:

  http://support.tibco.com

  Entry to this site requires a user name and password. If you do not have a user name, you can request one.

Chapter 1 Data Structures

This chapter summarizes the TIBCO SmartSockets architecture-independent data types, data structures, and typedefs.

Topics

Architecture-Independent Data Types

Instead of using native C/C++ scalar data types, function parameters, and return values in the TIBCO SmartSockets API, use SmartSockets data types that are independent of system architecture, as listed in Table 1. These data types make the SmartSockets API more portable and consistent across different 32-bit and 64-bit architectures.

Table 1 Summary of SmartSockets Data Types

SmartSockets Data Type	Description	Example 32-bit C Data Type	Example 64-bit C Data Type
T_STR	character string	`char *`	`char *`
T_PTR	generic pointer	`void *`	`void *`
T_BOOL	boolean	`int`	`int`
T_CHAR	character	`char`	`char`
T_UCHAR	unsigned character	`unsigned char`	`unsigned char`
T_INT1	one-byte integer	`signed char`	`signed char`
T_UINT1	unsigned one-byte integer	`unsigned char`	`unsigned char`
T_INT2	two-byte integer	`short`	`short`
T_UINT2	unsigned two-byte integer	`unsigned short`	`unsigned short`
T_INT4	four-byte integer	`long`	`int`
T_UINT4	unsigned four-byte integer	`unsigned long`	`unsigned int`
T_INT8	eight-byte integer	`long long` `__int64` (Windows)	`long` `__int64` (Windows)
T_REAL4	four-byte real	`float`	`float`
T_REAL8	eight-byte real	`double`	`double`
T_REAL16	sixteen-byte real	`char [16]`	`long double`

Data Structures

A number of SmartSockets inter-process communication (IPC) data structures are accessible from user-written programs. This is how these structures are defined (using typedef) in <rtworks/ipc.h>:

```
typedef struct T_IPC_MT_STRUCT T_IPC_MT_STRUCT, *T_IPC_MT;
typedef struct T_IPC_MSG_STRUCT T_IPC_MSG_STRUCT, *T_IPC_MSG;
typedef struct T_IPC_MSG_FIELD_STRUCT T_IPC_MSG_FIELD_STRUCT,
   *T_IPC_MSG_FIELD;
typedef struct T_IPC_MSG_FILE_STRUCT T_IPC_MSG_FILE_STRUCT,
   *T_IPC_MSG_FILE;
typedef struct T_IPC_CONN_STRUCT T_IPC_CONN_STRUCT, *T_IPC_CONN;
```

These structure definitions are incomplete in the sense that none of the fields in the structure are defined here. This is one way of creating an opaque data type in C. Pointers to these structures cannot be de-referenced to look at what they point to. These pointers can only be manipulated through access functions. This protects users from changes to the underlying data structures.

Messages

Messages are packets of information sent from one process to one or more other processes providing instructions or data for the receiving process. Messages are described in detail in the *TIBCO SmartSockets User's Guide*.

The fields in a message can be traversed with the function TipcMsgTraverse. It uses a traversal function defined in this way:

```
typedef T_PTR (*T_IPC_MSG_TRAV_FUNC)
   (T_IPC_MSG msg, T_IPC_MSG_TRAV field, T_PTR arg);
```

For more information on traversal functions, see the *TIBCO SmartSockets Utilities* reference. The traversal function for TipcMsgTraverse must be defined with these arguments. The second argument to the traversal function contains information for a field in a message. The the data structure for this argument is defined:

```
typedef struct T_IPC_MSG_TRAV_STRUCT T_IPC_MSG_TRAV_STRUCT,
   *T_IPC_MSG_TRAV;
struct T_IPC_MSG_TRAV_STRUCT {
   T_INT2 *type_ptr;              /* stored as 2 bytes in message */
   T_PTR value_ptr;               /* can be cast to appropriate type */
   T_INT4 *array_size_ptr;        /* NULL if not applicable */
   T_IPC_MSG_FIELD field;         /* actual field */
};
```

The field type in a message is stored as a 2-byte integer (to save space), but should be referenced in code with the enumerated type T_IPC_FT. This is how the T_IPC_FT type is defined:

```
typedef enum {
  T_IPC_FT_INVALID,
  T_MSG_FT_CHAR,
  T_MSG_FT_BINARY,
  T_MSG_FT_STR,
  T_IPC_FT_STR_ARRAY,
  T_IPC_FT_INT2,
  T_IPC_FT_INT2_ARRAY,
  T_IPC_FT_INT4,
  T_IPC_FT_INT4_ARRAY,
  T_IPC_FT_INT8,
  T_IPC_FT_INT8_ARRAY,
  T_IPC_FT_REAL4,
  T_IPC_FT_REAL4_ARRAY,
  T_IPC_FT_REAL8,
  T_IPC_FT_REAL8_ARRAY,
  T_IPC_FT_REAL16,
  T_IPC_FT_REAL16_ARRAY,
  T_IPC_FT_MSG,
  T_IPC_FT_MSG_ARRAY
  T_MSG_FT_XML,
  T_MSG_FT_TIMESTAMP,
  T_MSG_FT_TIMESTAMP_ARRAY,
  T_MSG_FT_UTF8,
```

```
  T_MSG_FT_UTF8_ARRAY,
  T_MSG_FT_BOOL,
  T_MSG_FT_BOOL_ARRAY,
  T_MSG_FT_BYTE

}  T_IPC_FT;
```

The T_IPC_PROP_* enumerated values are not field types, but instead allow a message property to be accessed with the convenience functions TipcMsgRead, TipcMsgReadVa, TipcMsgWrite, TipcMsgWriteVa, TipcConnMsgWrite, TipcConnMsgWriteVa, TipcSrvMsgWrite, and TipcSrvMsgWriteVa.

Both messages and message types have a delivery mode property that controls what level of guarantee is used when a message is sent through a connection with TipcConnMsgSend. This is how the enumerated type used to identify this property is defined:

```
typedef enum {
   T_IPC_DELIVERY_BEST_EFFORT,  /* network failure can cause loss */
   T_IPC_DELIVERY_SOME,         /* guaranteed to deliver to at least one */
   T_IPC_DELIVERY_ALL,          /* guaranteed to deliver to all receivers */
   T_IPC_DELIVERY_ORDERED       /* despite failure, delivers in order published */
}  T_IPC_DELIVERY_MODE;
```

Both messages and message types have a load balancing mode property that controls how a publish-subscribe message is load balanced. This is how the enumerated type used to identify this property is defined:

```
typedef enum {
   T_IPC_LB_NONE,          /* normal publish subscribe delivery mode */
   T_IPC_LB_ROUND_ROBIN,   /* RoundRobin load balancing */
   T_IPC_LB_WEIGHTED,      /* weighted RoundRobin load balancing(based on GMD acks)*/
   T_IPC_LB_SORTED         /* sorted load balancing */
}  T_IPC_LB_MODE;
```

Message Types

Each message has a type property which defines the purpose of the message. Message types are described in the *TIBCO SmartSockets User's Guide*. All of the standard message types that SmartSockets supports are defined in <rtworks/mt.h>, which is included by <rtworks/ipc.h>.

The list of known message types can be traversed with the function TipcMtTraverse. This is how the traversal function used by TipcMtTraverse is defined:

```
typedef T_PTR (*T_IPC_MT_TRAV_FUNC)
   (T_STR name, T_IPC_MT mt, T_PTR arg);
```

For more information on traversal functions, see the *TIBCO SmartSockets Utilities* reference. The specified traversal function for TipcMtTraverse must be defined with these arguments.

Message Files

A message file is a text or binary file containing one or more messages. Message files are described in detail in the *TIBCO SmartSockets User's Guide*. A message file is created with the function TipcMsgFileCreate or TipcMsgFileCreateFromFile using a creation mode that is defined in this way:

```
typedef enum {
   T_IPC_MSG_FILE_CREATE_READ,          /* read (auto-detect text or binary) */
   T_IPC_MSG_FILE_CREATE_WRITE,         /* write in text mode */
   T_IPC_MSG_FILE_CREATE_WRITE_BINARY,  /* write in binary mode */
   T_IPC_MSG_FILE_CREATE_APPEND         /* append (auto-detect text or binary) */
}  T_IPC_MSG_FILE_CREATE_MODE;
```

Connections

A connection is an endpoint of a communication link used to send and receive messages between two processes. Connections are described in detail in the *TIBCO SmartSockets User's Guide*.

A connection has several timeout properties which are used to check for possible network failures. These timeouts are accessed with the functions TipcConnGetTimeout and TipcConnSetTimeout. This is how the enumerated type used to identify these properties is defined:

```
typedef enum {
   T_IPC_TIMEOUT_READ,
   T_IPC_TIMEOUT_WRITE,
   T_IPC_TIMEOUT_KEEP_ALIVE,
   T_IPC_TIMEOUT_DELIVERY
}  T_IPC_TIMEOUT;
```

The function TipcConnMsgSearch can be used to search a connection's message queue for a specific message. This is how its search function is defined:

```
typedef T_BOOL (*T_IPC_CONN_MSG_SEARCH_FUNC)
   (T_IPC_CONN conn, T_IPC_MSG msg, T_PTR arg);
```

General Connection Callbacks

Connections have several types of callbacks. Each callback type has a callback function type and callback data type.

Process, Default, Read, and Write Callbacks

The connection process, default, read, and write callbacks all use similar data structures. This is how the data structures are defined:

```
typedef void (*T_IPC_CONN_MSG_CB_FUNC)
   (T_IPC_CONN conn, T_IPC_CONN_MSG_CB_DATA data, T_CB_ARG arg);

typedef T_IPC_CONN_MSG_CB_FUNC T_IPC_CONN_PROCESS_CB_FUNC;

typedef T_IPC_CONN_MSG_CB_FUNC T_IPC_CONN_DEFAULT_CB_FUNC;

typedef T_IPC_CONN_MSG_CB_FUNC T_IPC_CONN_WRITE_CB_FUNC;

typedef void (*T_IPC_CONN_READ_CB_FUNC)
   (T_IPC_CONN conn, T_IPC_CONN_READ_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_CONN_MSG_CB_DATA_STRUCT
   T_IPC_CONN_MSG_CB_DATA_STRUCT, *T_IPC_CONN_MSG_CB_DATA;
```

```
struct T_IPC_CONN_MSG_CB_DATA_STRUCT {
   T_CB cb;
   T_IPC_MSG msg;
};

typedef T_IPC_CONN_MSG_CB_DATA_STRUCT
   T_IPC_CONN_WRITE_CB_DATA_STRUCT, *T_IPC_CONN_WRITE_CB_DATA;

typedef T_IPC_CONN_MSG_CB_DATA_STRUCT
   T_IPC_CONN_PROCESS_CB_DATA_STRUCT, *T_IPC_CONN_PROCESS_CB_DATA;

typedef T_IPC_CONN_MSG_CB_DATA_STRUCT
   T_IPC_CONN_DEFAULT_CB_DATA_STRUCT, *T_IPC_CONN_DEFAULT_CB_DATA;

typedef struct T_IPC_CONN_READ_CB_DATA_STRUCT
   T_IPC_CONN_READ_CB_DATA_STRUCT, *T_IPC_CONN_READ_CB_DATA;
struct T_IPC_CONN_READ_CB_DATA_STRUCT {
   T_CB cb;
   T_IPC_MSG msg;
   T_INT4 packet_size;     /* make available for monitoring */
};
```

Error Callbacks

This is how the connection error callback data structures are defined:

```
typedef void (*T_IPC_CONN_ERROR_CB_FUNC)
   (T_IPC_CONN conn, T_IPC_CONN_ERROR_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_CONN_ERROR_CB_DATA_STRUCT
   T_IPC_CONN_ERROR_CB_DATA_STRUCT, *T_IPC_CONN_ERROR_CB_DATA;
struct T_IPC_CONN_ERROR_CB_DATA_STRUCT {
   T_CB cb;
   T_INT4 c_err_num;         /* C error number (errno) */
   T_INT4 os_err_num;        /* OS error number (vaxc$errno, GetLastError(), etc.) */
   T_INT4 socket_err_num;    /* socket error number (WSAGetLastError(), */
                             /* sock_errno(), etc.) */
};
```

Queue Callbacks

This is how the connection queue callback data structures are defined:

```
typedef void (*T_IPC_CONN_QUEUE_CB_FUNC)
   (T_IPC_CONN conn, T_IPC_CONN_QUEUE_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_CONN_QUEUE_CB_DATA_STRUCT
T_IPC_CONN_QUEUE_CB_DATA_STRUCT, *T_IPC_CONN_QUEUE_CB_DATA;
struct T_IPC_CONN_QUEUE_CB_DATA_STRUCT {
   T_CB cb;
   T_IPC_MSG msg;          /* message inserted into or deleted from queue */
   T_BOOL insert_flag;     /* TRUE if inserted, FALSE if deleted */
   T_INT4 position;        /* position of message in queue (0 is front) */
};
```

Global Connections

RTserver routes publish-subscribe messages to RTclient processes through local and global connections. RTserver and RTclient are discussed in detail in the *TIBCO SmartSockets User's Guide*. Most RTclient applications require only a single connection to an RTserver. This single global connection is created and managed using the TipcSrv API. RTclient applications that require multiple connections to RTservers use the TipcSrvConn API, covered in Multiple Connections on page 11.

The functions TipcSrvCreate, TipcSrvDestroy, and TipcSrvGetConnStatus manipulate the status of the connection to RTserver using an enumerated type. This is how the enumerated type is defined:

```
typedef enum {
   T_IPC_SRV_CONN_NONE,       /* not connected at all */
   T_IPC_SRV_CONN_WARM,       /* warm connection to RTserver */
   T_IPC_SRV_CONN_FULL        /* fully connected to RTserver */
}  T_IPC_SRV_CONN_STATUS;
```

The functions TipcSrvLogAddMt and TipcSrvLogRemoveMt add a message type to or remove a message type from a message file logging type, using an enumerated type. This is how the enumerated type is defined:

```
typedef enum {
   T_IPC_SRV_LOG_DATA,        /* slot values and time */
   T_IPC_SRV_LOG_STATUS,      /* status messages */
   T_IPC_SRV_LOG_INTERNAL     /* all other standard message types */
}  T_IPC_SRV_LOG_TYPE;
```

The function TipcSrvStop stops RTserver and possibly more processes, using an enumerated type. This is how the enumerated type is defined:

```
typedef enum {
   T_IPC_SRV_STOP_NORMAL = 1, /* just stop one RTserver */
   T_IPC_SRV_STOP_ALL,        /* stop all RTservers and RTclients */
   T_IPC_SRV_STOP_CLIENTS,    /* stop one RTserver and its RTclients */
   T_IPC_SRV_STOP_SERVERS     /* stop all RTservers */
}  T_IPC_SRV_STOP_TYPE;
```

The function TipcSrvSubjectTraverseSubscribe can be used to traverse the subjects that an RTclient process is subscribing to, using a traversal function. This is how the traversal function is defined:

```
typedef T_PTR (*T_IPC_SRV_SUBJECT_TRAV_FUNC)
   (T_STR name, T_STR not_used, T_PTR arg);
```

Global Connection Callbacks

An RTclient process can have server create callbacks and server destroy callbacks. Each callback type has a callback function type and callback data type.

Server Create Callbacks

This is how the server create callback data structures are defined:

```
typedef void (*T_IPC_SRV_CREATE_CB_FUNC)
   (T_IPC_CONN conn, T_IPC_SRV_CREATE_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_SRV_CREATE_CB_DATA_STRUCT
   T_IPC_SRV_CREATE_CB_DATA_STRUCT, *T_IPC_SRV_CREATE_CB_DATA;
struct T_IPC_SRV_CREATE_CB_DATA_STRUCT {
   T_CB cb;
   T_IPC_SRV_CONN_STATUS old_conn_status; /* old connection status */
   T_IPC_SRV_CONN_STATUS new_conn_status; /* new connection status */
};
```

Server Destroy Callbacks

This is how the server destroy callback data structures are defined:

```
typedef void (*T_IPC_SRV_DESTROY_CB_FUNC)
   (T_IPC_CONN conn, T_IPC_SRV_DESTROY_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_SRV_DESTROY_CB_DATA_STRUCT
   T_IPC_SRV_DESTROY_CB_DATA_STRUCT, *T_IPC_SRV_DESTROY_CB_DATA;
struct T_IPC_SRV_DESTROY_CB_DATA_STRUCT {
   T_CB cb;
   T_IPC_SRV_CONN_STATUS old_conn_status; /* old connection status */
   T_IPC_SRV_CONN_STATUS new_conn_status; /* new connection status */
};
```

Subject Callbacks

This is how the server subject callback data structures are defined:

```
typedef void (*T_IPC_CONN_MSG_CB_FUNC)
   (T_IPC_CONN conn, T_IPC_CONN_MSG_CB_DATA data, T_CB_ARG arg);

typedef T_IPC_CONN_MSG_CB_FUNC T_IPC_SRV_SUBJECT_CB_FUNC;

struct T_IPC_CONN_MSG_CB_DATA_STRUCT {
   T_CB cb;
   T_IPC_MSG msg;
};

typedef T_IPC_CONN_MSG_CB_DATA_STRUCT
   T_IPC_SRV_SUBJECT_CB_DATA_STRUCT, *T_IPC_SRV_SUBJECT_CB_DATA;
```

Multiple Connections

For applications whose architecture requires an RTclient to have connections to more than one RTserver, RTclients can use multiple connections instead of a single global connection. Multiple connections use the TipcSrvConn API, which allows distinct connections to multiple RTservers. Each connection has its own set of subscriptions and callbacks. Do not mix types of connections within an RTclient. An RTclient should connect to RTservers either with a single global connection or with multiple connections. For information on the enumerated types and callbacks used by global connections, see Global Connections on page 9.

Multiple Connection Callbacks

An RTclient process can have server create callbacks and server close callbacks. Each callback type has a callback function type and callback data type.

Server Open Callbacks

When using multiple connections, the server open callback may be used. This is how the server open callback data structures are defined:

```
typedef void (*T_IPC_SRV_OPEN_CB_FUNC)
   (T_IPC_SRV srv, T_IPC_SRV_OPEN_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_SRV_OPEN_CB_DATA_STRUCT
   T_IPC_SRV_OPEN_CB_DATA_STRUCT, *T_IPC_SRV_OPEN_CB_DATA;
struct T_IPC_SRV_OPEN_CB_DATA_STRUCT {
   T_CB cb;
   T_IPC_SRV_CONN_STATUS old_conn_status;
   T_IPC_SRV_CONN_STATUS new_conn_status;
};
```

Server Close Callbacks

When using multiple connections, the server close callback may be used. This is how the server close callback data structures are defined:

```
typedef void (*T_IPC_SRV_CLOSE_CB_FUNC)
   (T_IPC_SRV srv, T_IPC_SRV_CLOSE_CB_DATA data, T_CB_ARG arg);

typedef struct T_IPC_SRV_CLOSE_CB_DATA_STRUCT
   T_IPC_SRV_CLOSE_CB_DATA_STRUCT, *T_IPC_SRV_CLOSE_CB_DATA;
struct T_IPC_SRV_CLOSE_CB_DATA_STRUCT {
   T_CB cb;
   T_IPC_SRV_CONN_STATUS old_conn_status;
   T_IPC_SRV_CONN_STATUS new_conn_status;
};
```

Events

Each event has an associated dispatch function defined as:

```
typedef void (*T_IPC_EVENT_FUNC)
   (T_IPC_EVENT event, T_IPC_EVENT_DATA data, T_PTR arg);

typedef struct T_IPC_EVENT_DATA_STRUCT T_IPC_EVENT_DATA_STRUCT,
   *T_IPC_EVENT_DATA;
struct T_IPC_EVENT_DATA_STRUCT {
   T_IPC_EVENT_TYPE type;
   T_PTR data;
   T_INT4 socket;
   T_IO_CHECK_MODE check_mode;
   T_REAL8 interval;
   T_IPC_CONN conn;
   T_IPC_MSG msg;
};

typedef enum {
   T_IPC_EVENT_USER,
   T_IPC_EVENT_TIMER,
   T_IPC_EVENT_SOCKET,
   T_IPC_EVENT_CONN,
   T_IPC_EVENT_MSG,
   T_IPC_EVENT_MSG_TYPE
}  T_IPC_EVENT_TYPE;
```

Chapter 2 Summary of Basic API

This chapter is a summary of the basic TIBCO SmartSockets API. It is designed for both novice and experienced TIBCO SmartSockets developers. If you are a novice to TIBCO SmartSockets, you can scan the entire chapter to gain an understanding of all that is available to you in the SmartSockets API. If you are more experienced, you can use this chapter to quickly find a function to perform a particular task or to check the arguments of a function.

For each basic function available in the API, this chapter lists a short description of what the function does and a synopsis of its arguments and return values. For more details on these functions and examples, see the *TIBCO SmartSockets Application Programming Interface*. For a summary of the advanced SmartSockets API, see Chapter 3, Summary of Advanced API.

Topics

Buffer Functions (TipcBuf*)

The TipcBuf* functions are used to append or get a message.

TipcBufMsgAppend— append a message to a buffer

```
T_BOOL TipcBufMsgAppend(buf, msg)
T_BUF buf;
T_IPC_MSG msg;
```

TipcBufMsgNext— get a message from a buffer

```
T_IPCMSG TipcBufMsgNext(buf)
T_BUF buf;
```

Connection Functions (TipcCbConn* and TipcConn*)

The TipcCbConn* and TipcConn* functions are used when you need to do direct peer-to-peer communication between two programs, without going through RTserver.

Callback Functions

These functions are used to work with callbacks on connections. For subject callbacks, see Global Connection Callback Functions (TipcCbSrv* and TipcSrv*Cb*) on page 81.

TipcCbConnProcessGmdFailure — callback on all connections to process guaranteed message delivery failure messages when GMD fails

```
void TipcCbConnProcessGmdFailure(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
```

TipcCbConnProcessKeepAliveCall — callback to process KEEP_ALIVE_CALL messages from the other end of a connection

```
void TipcCbConnProcessKeepAliveCall(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
```

TipcConnDefaultCbCreate — create a default callback in a connection

```
T_CB TipcConnDefaultCbCreate(conn, func, arg)
T_IPC_CONN conn;
T_IPC_CONN_DEFAULT_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnDefaultCbLookup — look up a default callback in a connection

```
T_CB TipcConnDefaultCbLookup(conn, func, arg)
T_IPC_CONN conn;
T_IPC_CONN_DEFAULT_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnErrorCbCreate — create an error callback in a connection

```
T_CB TipcConnErrorCbCreate(conn, func, arg)
T_IPC_CONN conn;
T_IPC_CONN_ERROR_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnErrorCbLookup — look up an error callback in a connection

```
T_CB TipcConnErrorCbLookup(conn, func, arg)
T_IPC_CONN conn;
T_IPC_CONN_ERROR_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnProcessCbCreate — create a process callback in a connection

```
T_CB TipcConnProcessCbCreate(conn, mt, func, arg)
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_CONN_PROCESS_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnProcessCbLookup — look up a process callback in a connection

```
T_CB TipcConnProcessCbLookup(conn, mt, func, arg)
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_CONN_PROCESS_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnQueueCbCreate — create a queue callback in a connection

```
T_CB TipcConnQueueCbCreate(conn, mt, func, arg)
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_CONN_QUEUE_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnQueueCbLookup — look up a queue callback in a connection

```
T_CB TipcConnQueueCbLookup(conn, mt, func, arg)
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_CONN_QUEUE_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnReadCbCreate — create a read callback in a connection

```
T_CB TipcConnReadCbCreate(conn, mt, func, arg)
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_CONN_READ_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnReadCbLookup — look up a read callback in a connection

```
T_CB TipcConnReadCbLookup(conn, mt, func, arg)
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_CONN_READ_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnWriteCbCreate — create a write callback in a connection

```
T_CB TipcConnWriteCbCreate(conn, mt, func, arg)
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_CONN_WRITE_CB_FUNC func;
T_CB_ARG arg;
```

TipcConnWriteCbLookup — look up a write callback in a connection

```
T_CB TipcConnWriteCbLookup(conn, mt, func, arg)
T_IPC_CONN conn;
T_IPC_MT mt;
T_IPC_CONN_WRITE_CB_FUNC func;
T_CB_ARG arg;
```

Accept, Create and Destroy Functions

These functions are used to accept connections from a client, create connections of different types, and destroy connections.

TipcConnAccept — accept a connection from a client

```
T_IPC_CONN TipcConnAccept(server_conn)
T_IPC_CONN server_conn;
```

TipcConnCreate — create a new connection

```
T_IPC_CONN TipcConnCreate()
```

TipcConnCreateClient — create the client side of a connection

```
T_IPC_CONN TipcConnCreateClient(logical_conn_name)
T_STR logical_conn_name;
```

TipcConnCreateServer — create the server side of a connection

```
T_IPC_CONN TipcConnCreateServer(logical_conn_name)
T_STR logical_conn_name;
```

TipcConnDestroy — destroy a connection

```
T_BOOL TipcConnDestroy(conn)
T_IPC_CONN conn;
```

GMD Functions (Tipc*Gmd*)

These functions are used to work with the guaranteed delivery message services of SmartSockets for connections.

TipcConnGmdFileCreate — create a GMD area on a connection

```
T_BOOL TipcConnGmdFileCreate(conn)
T_IPC_CONN conn;
```

TipcConnGmdFileDelete — delete GMD files for a connection

```
T_BOOL TipcConnGmdFileDelete(conn)
T_IPC_CONN conn;
```

TipcConnGmdMsgDelete — delete a message from the GMD area after a GMD failure on a connection

```
T_BOOL TipcConnGmdMsgDelete(conn, msg)
T_IPC_CONN conn;
T_IPC_MSG msg;
```

TipcConnGmdMsgResend — resend a message after a GMD failure on a connection

```
T_BOOL TipcConnGmdMsgResend(conn, msg)
T_IPC_CONN conn;
T_IPC_MSG msg;
```

TipcConnGmdResend — resend all guaranteed messages after a delivery failure on a connection

```
T_BOOL TipcConnGmdResend(conn)
T_IPC_CONN conn;
```

TipcGetGmdDir — get name of the directory where files are written for GMD

```
T_STR TipcGetGmdDir()
```

Miscellaneous Functions

These functions do not fit with any of the other established categories:

TipcConnCheck — check if data can be read from or written to a connection

```
T_BOOL TipcConnCheck(conn, check_mode, timeout)
T_IPC_CONN conn;
T_IO_CHECK_MODE check_mode;
T_REAL8 timeout;
```

TipcConnKeepAlive — check if the process at the other end of a connection is still alive

```
T_BOOL TipcConnKeepAlive(conn)
T_IPC_CONN conn;
```

TipcConnLock — acquire exclusive access to a connection

```
T_BOOL TipcConnLock(conn)
T_IPC_CONN conn;
```

TipcConnRead — read data from a connection and queue messages in priority order

```
T_BOOL TipcConnRead(conn, timeout)
T_IPC_CONN conn;
T_REAL8 timeout;
```

TipcConnUnlock — release exclusive access to a connection

```
T_BOOL TipcConnUnlock(conn)
T_IPC_CONN conn;
```

Property Get Functions (TipcConnGet* and TipcConn*Get*)

The TipcConnGet* and TipcConn*Get* functions are used to retrieve the properties of a connection. A connection property is a piece of information about the connection, such as how often messages are expected to be available for reading.

TipcConnBufferGetReadSize — get the total number of bytes in a connection's read buffer

```
T_BOOL TipcConnBufferGetReadSize(conn, read_size_return)
T_IPC_CONN conn;
T_INT4 *read_size_return;
```

TipcConnBufferGetWriteSize — get the total number of bytes in a connection's write buffer

```
T_BOOL TipcConnBufferGetWriteSize(conn, write_size_return)
T_IPC_CONN conn;
T_INT4 *write_size_return;
```

TipcConnGetArch — determine the architecture name of a connection's peer process

```
T_BOOL TipcConnGetArch(conn, arch_return)
T_IPC_CONN conn;
T_STR *arch_return;
```

TipcConnGetAutoFlushSize — determine the automatic flush size of a connection

```
T_BOOL TipcConnGetAutoFlushSize(conn, auto_flush_size_return)
T_IPC_CONN conn;
T_INT4 *auto_flush_size_return;
```

TipcConnGetBlockMode — determine the block mode of a connection

```
T_BOOL TipcConnGetBlockMode(conn, block_mode_return)
T_IPC_CONN conn;
T_BOOL *block_mode_return;
```

TipcConnGetGmdMaxSize — determine the GMD area maximum size of a connection

```
T_BOOL TipcConnGetGmdMaxSize(conn, gmd_max_size_return)
T_IPC_CONN conn;
T_UINT4 *gmd_max_size_return;
```

TipcConnGetGmdNumPending — determine the number of outgoing GMD messages still pending on a connection

```
T_BOOL TipcConnGetGmdNumPending(conn, gmd_num_pending_return)
T_IPC_CONN conn;
T_INT4 *gmd_num_pending_return;
```

TipcConnGetNode — determine the node name of a connection's peer process

```
T_BOOL TipcConnGetNode(conn, node_return)
T_IPC_CONN conn;
T_STR *node_return;
```

TipcConnGetNumQueued — determine the number of queued messages from a connection

```
T_BOOL TipcConnGetNumQueued(conn, num_queued_return)
T_IPC_CONN conn;
T_INT4 *num_queued_return;
```

TipcConnGetPid — determine the process ID of a connection's peer process

```
T_BOOL TipcConnGetPid(conn, pid_return)
T_IPC_CONN conn;
T_INT4 *pid_return;
```

TipcConnGetSocket — determine the socket of a connection

```
T_BOOL TipcConnGetSocket(conn, socket_return)
T_IPC_CONN conn;
T_INT4 *socket_return;
```

TipcConnGetTimeout — determine the timeout property of a connection

```
T_BOOL TipcConnGetTimeout(conn, timeout, value_return)
T_IPC_CONN conn;
T_IPC_TIMEOUT timeout;
T_REAL8 *value_return;
```

TipcConnGetUniqueSubject — determine the unique subject of a connection's peer process

```
T_BOOL TipcConnGetUniqueSubject(conn, unique_subject_return)
T_IPC_CONN conn;
T_STR *unique_subject_return;
```

TipcConnGetUser — determine the user name of a connection's peer process

```
T_BOOL TipcConnGetUser(conn, user_return)
T_IPC_CONN conn;
T_STR *user_return;
```

TipcConnGetXtSource — get the source suitable for XtAppAddInput from a connection

```
T_BOOL TipcConnGetXtSource(conn, source_return)
T_IPC_CONN conn;
T_INT4 *source_return;
```

TipcConnTrafficGetBytesRecv8 — get the total number of bytes received on a connection

```
T_BOOL TipcConnTrafficGetBytesRecv8(conn, bytes_recv_return)
T_IPC_CONN conn;
T_INT8 *bytes_recv_return;
```

TipcConnTrafficGetBytesSent8 — get the total number of bytes sent on a connection

```
T_BOOL TipcConnTrafficGetBytesSent8(conn, bytes_sent_return)
T_IPC_CONN conn;
T_INT8 *bytes_sent_return;
```

TipcConnTrafficGetMsgsRecv8 — get the total number of messages received on a connection

```
T_BOOL TipcConnTrafficGetMsgsRecv8(conn, msgs_recv_return)
T_IPC_CONN conn;
T_INT8 *msgs_recv_return;
```

TipcConnTrafficGetMsgsSent8 — get the total number of messages sent on a connection

```
T_BOOL TipcConnTrafficGetMsgsSent8(conn, msgs_sent_return)
T_IPC_CONN conn;
T_INT8 *msgs_sent_return;
```

The previous functions supersede their older counterparts. Do not use the following older functions; we have retained them only for backward source compatibility.

TipcConnTrafficGetBytesRecv — get the total number of bytes received on a connection

```
T_BOOL TipcConnTrafficGetBytesRecv(conn, bytes_recv_return)
T_IPC_CONN conn;
T_INT4 *bytes_recv_return;
```

TipcConnTrafficGetBytesSent — get the total number of bytes sent on a connection

```
T_BOOL TipcConnTrafficGetBytesSent(conn, bytes_sent_return)
T_IPC_CONN conn;
T_INT4 *bytes_sent_return;
```

TipcConnTrafficGetMsgsRecv — get the total number of messages received on a connection

```
T_BOOL TipcConnTrafficGetMsgsRecv(conn, msgs_recv_return)
T_IPC_CONN conn;
T_INT4 *msgs_recv_return;
```

TipcConnTrafficGetMsgsSent — get the total number of messages sent on a connection

```
T_BOOL TipcConnTrafficGetMsgsSent(conn, msgs_sent_return)
T_IPC_CONN conn;
T_INT4 *msgs_sent_return;
```

Property Set Functions (TipcConnSet*)

The TipcConnSet* functions are used to set the properties of a connection. The property of a connection is a piece of information about the connection, such as how often messages are expected to be available for reading.

TipcConnSetAutoFlushSize — set the automatic flush size of a connection

```
T_BOOL TipcConnSetAutoFlushSize(conn, auto_flush_size)
T_IPC_CONN conn;
T_INT4 auto_flush_size;
```

TipcConnSetBlockMode — set the block mode of a connection

```
T_BOOL TipcConnSetBlockMode(conn, block_mode)
T_IPC_CONN conn;
T_BOOL block_mode;
```

TipcConnSetGmdMaxSize — set a connection's GMD area maximum size

```
T_BOOL TipcConnSetGmdMaxSize(conn, gmd_max_size)
T_IPC_CONN conn;
T_UINT4 gmd_max_size;
```

TipcConnSetSocket — set the socket of a connection

```
T_BOOL TipcConnSetSocket(conn, socket)
T_IPC_CONN conn;
T_INT4 socket;
```

TipcConnSetTimeout — set a timeout property of a connection

```
T_BOOL TipcConnSetTimeout(conn, timeout, value)
T_IPC_CONN conn;
T_IPC_TIMEOUT timeout;
T_REAL8 value;
```

Reading and Processing Messages from a Connection

These functions are used to read a message from a connection, search for a message on a connection, and process a message which has been read in from a connection.

TipcConnMainLoop — read and process messages on a connection

```
T_BOOL TipcConnMainLoop(conn, timeout)
T_IPC_CONN conn;
T_REAL8 timeout;
```

TipcConnMsgInsert — insert a message into the queue of a connection

```
T_BOOL TipcConnMsgInsert(conn, msg, pos)
T_IPC_CONN conn;
T_IPC_MSG msg;
T_INT4 pos;
```

TipcConnMsgNext — determine the next message from a connection

```
T_IPC_MSG TipcConnMsgNext(conn, timeout)
T_IPC_CONN conn;
T_REAL8 timeout;
```

TipcConnMsgProcess — process a message in a connection

```
T_BOOL TipcConnMsgProcess(conn, msg)
T_IPC_CONN conn;
T_IPC_MSG msg;
```

TipcConnMsgSearch — search the message queue of a connection for a specific message

```
T_IPC_MSG TipcConnMsgSearch(conn, timeout, func, arg)
T_IPC_CONN conn;
T_REAL8 timeout;
T_IPC_CONN_MSG_SEARCH_FUNC func;
T_PTR arg;
```

TipcConnMsgSearchType — search the message queue of a connection for a message with a specific type

```
T_IPC_MSG TipcConnMsgSearchType(conn, timeout, mt)
T_IPC_CONN conn;
T_REAL8 timeout;
T_IPC_MT mt;
```

Sending Messages on a Connection

These functions are used to send a message to a connection.

TipcConnFlush — flush buffered outgoing messages to a connection's socket

```
T_BOOL TipcConnFlush(conn)
T_IPC_CONN conn;
```

TipcConnMsgSend — send a message through a connection

```
T_BOOL TipcConnMsgSend(conn, msg)
T_IPC_CONN conn;
T_IPC_MSG msg;
```

TipcConnMsgSendRpc — make a remote procedure call (RPC) with messages on a connection

```
T_IPC_MSG TipcConnMsgSendRpc(conn, call_msg, timeout)
T_IPC_CONN conn;
T_IPC_MSG call_msg;
T_REAL8 timeout;
```

TipcConnMsgWrite — construct a message and send it through a connection

```
T_BOOL TipcConnMsgWrite(conn, mt, ...)
T_IPC_CONN conn;
T_IPC_MT mt;
```

TipcConnMsgWriteVa — construct a message and send it through a connection (`va_list` version)

```
T_BOOL TipcConnMsgWriteVa(conn, mt, var_arg_list)
T_IPC_CONN conn;
T_IPC_MT mt;
va_list var_arg_list;
```

Monitoring Functions (TipcMon*)

There are two types of monitoring data that SmartSockets supports when monitoring your distributed application:

- data generated by SmartSockets, such as project names, RTclient names, RTserver names, subjects, message queues, message types, option values, and so on. This type of data is retrieved by the TipcMon* functions.

  You can also use these functions to locate processes on the network. For example, you can find all the RTclients subscribed to subject `/system/thermal`. In general, the TipcMon* functions use the RTserver to retrieve the monitoring information.

- data generated by an RTclient, referred to as extension data. Extension data is retrieved by another RTclient with the TipcMonClientExtPoll function. TipcMonExt* functions are used to define and update the extension data in an RTclient's MON_CLIENT_EXT_POLL_RESULT message before the data is retrieved. The RTserver is not involved in creating extension data.

In general, two types of monitoring are supported:

- Poll — synchronous monitoring, which retrieves a snapshot of information and can be set up to poll at regular intervals
- Watch — asynchronous monitoring, which notifies you when something changes

Polling Functions (TipcMon*Poll)

The TipcMon*Poll functions are used to do a one-time poll for information about various aspects of a SmartSockets project or to retrieve data generated by an RTclient, referred to as extension data.

TipcMonClientBufferPoll — poll once for RTclient message-related buffer information

```
T_BOOL TipcMonClientBufferPoll(client_name)
T_STR client_name;
```

TipcMonClientCbPoll — poll once for RTclient callback information

```
T_BOOL TipcMonClientCbPoll(client_name)
T_STR client_name;
```

TipcMonClientCpuPoll — poll for the percentage of CPU time used by an RTclient

```
T_BOOL TipcMonClientCpuPoll(client_name)
T_STR client_name;
```

TipcMonClientExtPoll — poll for extension data of an RTclient

```
T_BOOL TipcMonClientExtPoll(client_name)
T_STR client_name;
```

TipcMonClientGeneralPoll — poll once for RTclient general information

```
T_BOOL TipcMonClientGeneralPoll(client_name)
T_STR client_name;
```

TipcMonClientInfoPoll — poll for RTclient information

```
T_BOOL TipcMonClientInfoPoll(client_name)
T_STR client_name;
```

TipcMonClientMsgTrafficPoll — poll once for RTclient message traffic information

```
T_BOOL TipcMonClientMsgTrafficPoll(client_name)
T_STR client_name;
```

TipcMonClientMsgTypeExPoll — poll once for RTclient message type information

```
T_BOOL TipcMonClientMsgTypeExPoll(client_name, msg_type_name)
T_STR client_name;
T_STR msg_type_name;
```

The previous Ex function supersedes its older counterpart for all new development. Use the following older method only when operating with clients prior to release 6.7.

TipcMonClientMsgTypePoll — poll once for RTclient message type information

```
T_BOOL TipcMonClientMsgTypePoll(client_name, msg_type_name)
T_STR client_name;
T_STR msg_type_name;
```

TipcMonClientNamesNumPoll — poll for the number of RTclients in the current project that match the value of the Monitor_Scope option

```
T_BOOL TipcMonClientNamesNumPoll()
```

TipcMonClientNamesPoll — poll once for RTclient names

```
T_BOOL TipcMonClientNamesPoll()
```

TipcMonClientOptionPoll — poll once for RTclient option information

```
T_BOOL TipcMonClientOptionPoll(client_name, option_name)
T_STR client_name;
T_STR option_name;
```

TipcMonClientSubjectExPoll — poll once for RTclient subject information

```
T_BOOL TipcMonClientSubjectExPoll(client_name, subject_name)
T_STR client_name;
T_STR subject_name;
```

The previous Ex function supersedes its older counterpart for all new development. Use the following older method only when operating with clients prior to release 6.7.

TipcMonClientSubjectPoll — poll once for RTclient subject information

```
T_BOOL TipcMonClientSubjectPoll(client_name, subject_name)
T_STR client_name;
T_STR subject_name;
```

TipcMonClientSubscribeNumPoll — poll for the number of subjects subscribed to by an RTclient

```
T_BOOL TipcMonClientSubscribeNumPoll(client_name)
T_STR client_name;
```

TipcMonClientSubscribePoll — poll once for subjects to which an RTclient subscribes

```
T_BOOL TipcMonClientSubscribePoll(client_name)
T_STR client_name;
```

TipcMonClientTimePoll — poll once for RTclient time information

```
T_BOOL TipcMonClientTimePoll(client_name)
T_STR client_name;
```

TipcMonClientVersionPoll — poll once for RTclient version

```
T_BOOL TipcMonClientVersionPoll(client_name)
T_STR client_name;
```

TipcMonProjectNamesPoll — poll once for project names

```
T_BOOL TipcMonProjectNamesPoll()
```

TipcMonServerBufferPoll — poll once for RTserver message-related buffer information

```
T_BOOL TipcMonServerBufferPoll(server_name, connected_process_name)
T_STR server_name;
T_STR connected_process_name;
```

TipcMonServerConnPoll — poll once for RTserver connections

```
T_BOOL TipcMonServerConnPoll()
```

TipcMonServerCpuPoll — poll for the percentage of CPU time used by an RTserver

```
T_BOOL TipcMonServerCpuPoll(server_name)
T_STR server_name;
```

TipcMonServerGeneralPoll — poll once for RTserver general information

```
T_BOOL TipcMonServerGeneralPoll(server_name)
T_STR server_name;
```

TipcMonServerMsgTrafficExPoll — poll once for RTserver message traffic information

```
T_BOOL TipcMonServerMsgTrafficExPoll(server_name,
connected_process_name)
T_STR server_name;
T_STR connected_process_name;
```

The previous Ex function supersedes its older counterpart for all new development. Use the following older method only when operating with servers prior to release 6.7.

TipcMonServerMsgTrafficPoll — poll once for RTserver message traffic information

```
T_BOOL TipcMonServerMsgTrafficPoll(server_name,
connected_process_name)
T_STR server_name;
T_STR connected_process_name;
```

TipcMonServerNamesPoll — poll once for RTserver names

```
T_BOOL TipcMonServerNamesPoll()
```

TipcMonServerOptionPoll — poll once for RTserver option information

```
T_BOOL TipcMonServerOptionPoll(server_name, option_name)
T_STR server_name;
T_STR option_name;
```

TipcMonServerRoutePoll — poll once for RTserver route information

```
T_BOOL TipcMonServerRoutePoll(server_name, dest_server_name)
T_STR server_name;
T_STR dest_server_name;
```

TipcMonServerStartTimePoll — poll once for RTserver start time and elapsed time

```
T_BOOL TipcMonServerStartTimePoll(server_name)
T_STR server_name;
```

TipcMonServerTimePoll — poll once for RTserver time information

```
T_BOOL TipcMonServerTimePoll(server_name)
T_STR server_name;
```

TipcMonServerVersionPoll — poll once for RTserver version information

```
T_BOOL TipcMonServerVersionPoll(server_name)
T_STR server_name;
```

TipcMonSubjectNamesPoll — poll once for subject names

```
T_BOOL TipcMonSubjectNamesPoll()
```

TipcMonSubjectSubscribePoll — poll once for RTclients that are subscribing to a subject

```
T_BOOL TipcMonSubjectSubscribePoll(subject_name)
T_STR subject_name;
```

Watch Functions (TipcMon*Watch)

The TipcMon*Watch functions are used to set up asynchronous notification when specified information in a SmartSockets project changes.

TipcMonClientBufferGetWatch — determine whether this RTclient is watching message-related buffer information in an RTclient

```
T_BOOL TipcMonClientBufferGetWatch(client_name, watch_status_return)
T_STR client_name;
T_BOOL *watch_status_return;
```

TipcMonClientBufferSetWatch — start or stop watching RTclient message-related buffer information

```
T_BOOL TipcMonClientBufferSetWatch(client_name, watch_status)
T_STR client_name;
T_BOOL watch_status;
```

TipcMonClientCongestionGetWatch — determine if the RTclient is watching a client's write buffer for congestion

```
T_BOOL TipcMonClientCongestionGetWatch(client_name,
watch_status_return)
T_STR client_name;
T_BOOL *watch_status_return;
```

TipcMonClientCongestionSetWatch — start or stop watching for congestion in an RTclient's write buffer

```
T_BOOL TipcMonClientCongestionSetWatch(client_name, high_water,
low_water, watch_status)
T_STR client_name;
T_INT4 high_water;
T_INT4 low_water;
T_BOOL watch_status;
```

TipcMonClientMsgRecvGetWatch — determine whether this RTclient is watching received messages in an RTclient

```
T_BOOL TipcMonClientMsgRecvGetWatch(client_name, msg_type_name,
watch_status_return)
T_STR client_name;
T_STR msg_type_name;
T_BOOL *watch_status_return;
```

TipcMonClientMsgRecvSetWatch — start or stop watching RTclient received messages

```
T_BOOL TipcMonClientMsgRecvSetWatch(client_name, msg_type_name,
watch_status)
T_STR client_name;
T_STR msg_type_name;
T_BOOL watch_status;
```

TipcMonClientMsgSendGetWatch — determine whether this RTclient is watching sent messages in an RTclient

```
T_BOOL TipcMonClientMsgSendGetWatch(client_name, msg_type_name,
watch_status_return)
T_STR client_name;
T_STR msg_type_name;
T_BOOL *watch_status_return;
```

TipcMonClientMsgSendSetWatch — start or stop watching RTclient sent messages

```
T_BOOL TipcMonClientMsgSendSetWatch(client_name, msg_type_name,
watch_status)
T_STR client_name;
T_STR msg_type_name;
T_BOOL watch_status;
```

TipcMonClientNamesGetWatch — determine whether this RTclient is watching RTclient names

```
T_BOOL TipcMonClientNamesGetWatch(watch_status_return)
T_BOOL *watch_status_return;
```

TipcMonClientNamesSetWatch — start or stop watching RTclient names

```
T_BOOL TipcMonClientNamesSetWatch(watch_status)
T_BOOL watch_status;
```

TipcMonClientSubscribeGetWatch — determine whether this RTclient is watching the subjects that an RTclient is subscribing to

```
T_BOOL TipcMonClientSubscribeGetWatch(client_name,
watch_status_return)
T_STR client_name;
T_BOOL *watch_status_return;
```

TipcMonClientSubscribeSetWatch — start or stop watching the subjects that an RTclient is subscribing to

```
T_BOOL TipcMonClientSubscribeSetWatch(client_name, watch_status)
T_STR client_name;
T_BOOL watch_status;
```

TipcMonClientTimeGetWatch — determine whether this RTclient is watching time information in an RTclient

```
T_BOOL TipcMonClientTimeGetWatch(client_name, watch_status_return)
T_STR client_name;
T_BOOL *watch_status_return;
```

TipcMonClientTimeSetWatch — start or stop watching RTclient time information

```
T_BOOL TipcMonClientTimeSetWatch(client_name, watch_status)
T_STR client_name;
T_BOOL watch_status;
```

TipcMonPrintWatch — print all monitoring categories being watched

```
T_BOOL TipcMonPrintWatch(func)
T_OUT_FUNC func;
```

TipcMonProjectNamesGetWatch — determine whether this RTclient is watching project names

```
T_BOOL TipcMonProjectNamesGetWatch(watch_status_return)
T_BOOL *watch_status_return;
```

TipcMonProjectNamesSetWatch — start or stop watching project names

```
T_BOOL TipcMonProjectNamesSetWatch(watch_status)
T_BOOL watch_status;
```

TipcMonServerCongestionGetWatch — determine if the server is watching for congestion in the RTserver write buffer of a connected process

```
T_BOOL TipcMonServerCongestionGetWatch(server_name,
connected_process_name, watch_status_return)
T_STR server_name;
T_STR connected_process_name;
T_BOOL *watch_status_return;
```

TipcMonServerCongestionSetWatch — start or stop watching for congestion in the RTserver write buffer of a connected process

```
T_BOOL TipcMonServerCongestionSetWatch(server_name,
connected_process_name, high_water, low_water, watch_status)
T_STR server_name;
T_STR connected_process_name;
T_INT4 high_water;
T_INT4 low_water;
T_BOOL watch_status;
```

TipcMonServerConnGetWatch — determine whether this RTclient is watching RTserver connections

```
T_BOOL TipcMonServerConnGetWatch(watch_status_return)
T_BOOL *watch_status_return;
```

TipcMonServerConnSetWatch — start or stop watching RTserver connections

```
T_BOOL TipcMonServerConnSetWatch(watch_status)
T_BOOL watch_status;
```

TipcMonServerMaxClientLicensesGetWatch — determine if the RTserver is watching for the server cloud to exceed the licensed number of RTclient connections

```
T_BOOL TipcMonServerMaxClientLicensesGetWatch(server_name,
watch_status_return)
T_STR server_name;
T_BOOL *watch_status_return;
```

TipcMonServerMaxClientLicensesSetWatch — start or stop watching for a server to exceed the licensed number of RTclient connections

```
T_BOOL TipcMonServerMaxClientLicensesSetWatch(server_name,
watch_status)
T_STR server_name;
T_BOOL watch_status;
```

TipcMonServerNamesGetWatch — determine whether this RTclient is watching RTserver names

```
T_BOOL TipcMonServerNamesGetWatch(watch_status_return)
T_BOOL *watch_status_return;
```

TipcMonServerNamesSetWatch — start or stop watching RTserver names

```
T_BOOL TipcMonServerNamesSetWatch(watch_status)
T_BOOL watch_status;
```

TipcMonSubjectNamesGetWatch — determine whether this RTclient is watching subject names

```
T_BOOL TipcMonSubjectNamesGetWatch(watch_status_return)
T_BOOL *watch_status_return;
```

TipcMonSubjectNamesSetWatch — start or stop watching subject names

```
T_BOOL TipcMonSubjectNamesSetWatch(watch_status)
T_BOOL watch_status;
```

TipcMonSubjectSubscribeGetWatch — determine whether this RTclient is watching the RTclients that are subscribing to a subject

```
T_BOOL TipcMonSubjectSubscribeGetWatch(subject_name,
watch_status_return)
T_STR subject_name;
T_BOOL *watch_status_return;
```

TipcMonSubjectSubscribeSetWatch — start or stop watching the RTclients that are subscribing to a subject

```
T_BOOL TipcMonSubjectSubscribeSetWatch(subject_name, watch_status)
T_STR subject_name;
T_BOOL watch_status;
```

Extension Data Functions (TipcMonExt*)

The TipcMonExt* functions are used to delete, define, or update a field in an RTclient's MON_CLIENT_EXT_POLL_RESULT message. This message is returned to another RTclient that polls for RTclient information by issuing TipcMonClientExtPoll.

TipcMonExtDelete — remove a field from an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtDelete(field_name)
T_STR field_name;
```

TipcMonExtSetBinary — add or update a field of type BINARY in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetBinary(field_name, data, size)
T_STR field_name;
T_PTR data;
T_INT4 size;
```

TipcMonExtSetBool — add or update a field of type BOOL in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetBool(field_name, data)
T_STR field_name;
T_BOOL data;
```

TipcMonExtSetBoolArray — add or update a field containing an array of type BOOL in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetBoolArray(field_name, data, size)
T_STR field_name;
T_BOOL *data;
T_INT4 size;
```

TipcMonExtSetInt2 — add or update a field of type INT2 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetInt2(field_name, data)
T_STR field_name;
T_INT2 data;
```

TipcMonExtSetInt2Array — add or update a field containing an array of type INT2 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetInt2Array(field_name, data, size)
T_STR field_name;
T_INT2 *data;
T_INT4 size
```

TipcMonExtSetInt4 — add or update a field of type INT4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetInt4(field_name, data)
T_STR field_name;
T_INT4 data;
```

TipcMonExtSetInt4Array — add or update a field containing an array of type INT4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetInt4Array(field_name, data, size)
T_STR field_name;
T_INT4 *data;
T_INT4 size;
```

TipcMonExtSetInt8 — add or update a field of type INT8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetInt8(field_name, data)
T_STR field_name;
T_INT8 data;
```

TipcMonExtSetInt8Array — add or update a field containing an array of type INT8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetInt8Array(field_name, data, size)
T_STR field_name;
T_INT8 *data;
T_INT4 size;
```

TipcMonExtSetReal4 — add or update a field of type REAL4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetReal4(field_name, data)
T_STR field_name;
T_REAL4 data;
```

TipcMonExtSetReal4Array — add or update a field containing an array of type REAL4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetReal4Array(field_name, data, size)
T_STR field_name;
T_REAL4 *data;
T_INT4 size;
```

TipcMonExtSetReal8 — add or update a field of type REAL8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetReal8(field_name, data)
T_STR field_name;
T_REAL8 data;
```

TipcMonExtSetReal8Array — add or update a field containing an array of type REAL8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetReal8Array(field_name, data, size)
T_STR field_name;
T_REAL8 *data;
T_INT4 size;
```

TipcMonExtSetReal16 — add or update a field of type REAL16 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetReal16(field_name, data)
T_STR field_name;
T_REAL16 data;
```

TipcMonExtSetReal16Array — add or update a field containing an array of type REAL16 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetReal16Array(field_name, data, size)
T_STR field_name;
T_REAL16 *data;
T_INT4 size;
```

TipcMonExtSetStr — add or update a field of type STR in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetStr(field_name, data)
T_STR field_name;
T_STR data;
```

TipcMonExtSetStArray — add or update a field containing an array of type STR in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetStrArray(field_name, data, size)
T_STR field_name;
T_STR *data;
T_INT4 size;
```

TipcMonExtSetUtf8 — add or update a field of type UTF8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetUtf8(field_name, data)
T_STR field_name;
T_UTF8 data;
```

TipcMonExtSetUtf8Array — add or update a field containing an array of type UTF8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcMonExtSetUtf8Array(field_name, data, size)
T_STR field_name;
T_UTF8 *data;
T_INT4 size;
```

Miscellaneous Functions

These functions do not fit with any of the other established categories:

TipcMonGetIdentStr — retrieve the monitoring identification string of this process

```
T_BOOL TipcMonGetIdentStr(ident_str_return)
T_STR *ident_str_return;
```

TipcMonSetIdentStr — set the monitoring identification string of this process

```
T_BOOL TipcMonSetIdentStr(ident_str)
T_STR ident_str;
```

Message Functions (TipcMsg*)

The TipcMsg* functions are used to operate on messages. Typical operations include creating, cloning, and destroying messages, as well as constructing and accessing the fields in the data part of a message.

Construction Functions (TipcMsgAddNamed*)

The TipcMsgAdd* functions are used to write a field or fields into the data part of a message, referencing it by name.

TipcMsgAddNamedBinary — add a named BINARY field to a message

```
T_BOOL TipcMsgAddNamedBinary(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_PTR value;
T_INT4 size;
```

TipcMsgAddNamedBinaryPtr — add a named BINARY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedBinaryPtr(msg, name, ptr, size, field_return)
T_IPC_MSG msg;
T_STR name;
T_PTR ptr;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedBool — add a BOOL field to a message, referencing it by name

```
T_BOOL TipcMsgAddNamedBool(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_BOOL value
```

TipcMsgAddNamedBoolArray — add a BOOL_ARRAY field to a message, referencing it by name

```
T_BOOL TipcMsgAddNamedBoolArray(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_BOOL *value;
T_INT4 size;
```

TipcMsgAddNamedBoolArrayPtr — add a named BOOL_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedBoolArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_BOOL *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedByte — add a BYTE field to a message, referencing it by name

```
T_BOOL TipcMsgAddNamedByte(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_UCHAR value;
```

TipcMsgAddNamedChar — add a named CHAR field to a message

```
T_BOOL TipcMsgAddNamedChar(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_CHAR value;
```

TipcMsgAddNamedInt2 — add a named INT2 field to a message

```
T_BOOL TipcMsgAddNamedInt2(msg, name, value, field_return)
T_IPC_MSG msg;
T_STR name;
T_INT2 value;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedInt2Array — add a named INT2_ARRAY field to a message

```
T_BOOL TipcMsgAddNamedInt2Array(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_INT2 *value;
T_INT4 size;
```

TipcMsgAddNamedInt2ArrayPtr — add a named INT2_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedInt2ArrayPtr(msg, name, value, size,
field_return)
T_IPC_MSG msg;
T_STR name;
T_INT2 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedInt4 — add a named INT4 field to a message

```
T_BOOL TipcMsgAddNamedInt4(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_INT4 value;
```

TipcMsgAddNamedInt4Array — add a named INT4_ARRAY field to a message

```
T_BOOL TipcMsgAddNamedInt4Array(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_INT4 *value;
T_INT4 size;
```

TipcMsgAddNamedInt4ArrayPtr — add a named INT4_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedInt4ArrayPtr(msg, name, value, size,
field_return)
T_IPC_MSG msg;
T_STR name;
T_INT4 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedInt8 — add a named INT8 field to a message

```
T_BOOL TipcMsgAddNamedInt8(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_INT8 value;
```

TipcMsgAddNamedInt8Array — add a named INT8_ARRAY field to a message

```
T_BOOL TipcMsgAddNamedInt8Array(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_INT8 *value;
T_INT4 size;
```

TipcMsgAddNamedInt8ArrayPtr — add a named INT8_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedInt8ArrayPtr(msg, name, value, size,
field_return)
T_IPC_MSG msg;
T_STR name;
T_INT8 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedMsg — add a named MSG field to a message

```
T_BOOL TipcMsgAddNamedMsg(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_IPC_MSG value;=
```

TipcMsgAddNamedMsgArray — add a named MSG_ARRAY field to a message

```
T_BOOL TipcMsgAddNamedMsgArray(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_IPC_MSG *value;
T_INT4 size;
```

TipcMsgAddNamedMsgArrayPtr — add a named MSG_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedMsgArrayPtr(msg, name, value, size,
field_return)
T_IPC_MSG msg;
T_STR name;
T_IPC_MSG *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedMsgPtr — add a MSG pointer field to a message, referencing it by name

```
T_BOOL TipcMsgAddNamedStrPtr(msg, name, value, size, field_return)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedReal4 — add a named REAL4 field to a message

```
T_BOOL TipcMsgAddNamedReal4(msg, name, real4_data)
T_IPC_MSG msg;
T_STR name;
T_REAL4 real4_data;
```

TipcMsgAddNamedReal4Array — add a named REAL4_ARRAY field to a message

```
T_BOOL TipcMsgAddNamedReal4Array(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL4 *value;
T_INT4 size;
```

TipcMsgAddNamedReal4ArrayPtr — add a named REAL4_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedReal4ArrayPtr(msg, name, value, size,
field_return)
T_IPC_MSG msg;
T_STR name;
T_REAL4 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedReal8 — add a named REAL8 field to a message

```
T_BOOL TipcMsgAddNamedReal8(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_REAL8 value;
```

TipcMsgAddNamedReal8Array — add a named REAL8_ARRAY field to a message

```
T_BOOL TipcMsgAddNamedReal8Array(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value;
T_INT4 size;
```

TipcMsgAddNamedReal8ArrayPtr — add a named REAL8_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedReal8ArrayPtr(msg, name, value, size,
field_return)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedReal16 — add a named REAL16 field to a message

```
T_BOOL TipcMsgAddNamedReal16(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_REAL16 value;
```

TipcMsgAddNamedReal16Array — add a named REAL16_ARRAY field to a message

```
T_BOOL TipcMsgAddNamedReal16Array(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL16 *value;
T_INT4 size;
```

TipcMsgAddNamedReal16ArrayPtr — add a named REAL16_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedReal16ArrayPtr(msg, name, value, size,
field_return)
T_IPC_MSG msg;
T_STR name;
T_REAL16 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedStr — add a named STR field to a message

```
T_BOOL TipcMsgAddNamedStr(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_STR value;
```

TipcMsgAddNamedStrArray — add a named STR_ARRAY field to a message

```
T_BOOL TipcMsgAddNamedStrArray(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
```

TipcMsgAddNamedStrArrayPtr — add a named STR_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAddNamedStrArrayPtr(msg, name, value, size,
field_return)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedStrPtr — add a named STR field to a message using a pointer

```
T_BOOL TipcMsgAddNamedStrPtr(msg, name, value, size, field_return)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedTimestamp — add a TIMESTAMP field to a message using a name

```
T_BOOL TipcMsgAddNamedTimestamp(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_REAL8 value;
```

TipcMsgAddNamedTimestampArray — add a field containing an array of TIMESTAMP fields to a message using a name

```
T_BOOL TipcMsgAddNamedTimestampArray(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value;
T_INT4 size;
```

TipcMsgAddNamedTimestampArrayPtr — add a field containing an array of TIMESTAMP fields to a message using a name, and a pointer to the field value, rather than a copy

```
T_BOOL TipcMsgAddNamedTimestampArrayPtr(msg, name, value, size,
field_return)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedUnknown — add a field with an unknown or non-existent value to a message using a name

```
T_BOOL TipcMsgAddNamedUnknown(msg, name, type)
T_IPC_MSG msg;
T_STR name;
T_IPC_FT type;
```

TipcMsgAddNamedUtf8 — add a UTF8 field to a message, referencing it by name

```
T_BOOL TipcMsgAddNamedUtf8(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_STR value;
```

TipcMsgAddNamedUtf8Array — add a UTF8_ARRAY field to a message, referencing it by name

```
T_BOOL TipcMsgAddNamedUtf8Array(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
```

TipcMsgAddNamedUtf8ArrayPtr — add a UTF8_ARRAY pointer field to a message, referencing it by name

```
T_BOOL TipcMsgAddNamedUtf8ArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedUtf8Ptr — add a UTF8 pointer field to a message, referencing it by name

```
T_BOOL TipcMsgAddNamedUtf8Ptr(msg, name, value, *field_return)
T_IPC_MSG msg;
T_STR name;
T_STR value;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAddNamedXml — add a named XML field to a message

```
T_BOOL TipcMsgAddNamedXml(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_XML value;
```

TipcMsgAddNamedXmlPtr — add a named XML pointer field to a message

```
T_BOOL TipcMsgAddNamedXmlPtr(msg, name, value, field_return)
T_IPC_MSG msg;
T_STR name;
T_XML value;
T_IPC_MSG_FIELD *field_return;
```

Construction Functions (TipcMsgAppend*)

The TipcMsgAppend* functions are used to write a field or fields into the data part of a message.

TipcMsgAppendBinary — append a BINARY field to a message

```
T_BOOL TipcMsgAppendBinary(msg, binary_data, binary_size)
T_IPC_MSG msg;
T_PTR binary_data;
T_INT4 binary_size;
```

TipcMsgAppendBinaryPtr — append a BINARY field to a message using a pointer

```
T_BOOL TipcMsgAppendBinaryPtr(msg, binary_data, binary_size,
field_return)
T_IPC_MSG msg;
T_PTR binary_data;
T_INT4 binary_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendBool — append a BOOL field to a message

```
T_BOOL TipcMsgAppendBool(msg, bool_data)
T_IPC_MSG msg;
T_BOOL bool_data;
```

TipcMsgAppendBoolArray — append a field containing an array of BOOL fields to a message

```
T_BOOL TipcMsgAppendBoolArray(msg, bool_array_data, array_size)
T_IPC_MSG msg;
T_BOOL *bool_array_data;
T_INT4 array_size;
```

TipcMsgAppendBoolArrayPtr — append a field containing an array of BOOL fields to a message using a pointer

```
T_BOOL TipcMsgAppendBoolArrayPtr(msg, bool_array_data, array_size,
field_return)
T_IPC_MSG msg;
T_BOOL *bool_array_data;
T_INT4 array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendByte — append a BYTE field to a message

```
T_BOOL TipcMsgAppendByte(msg, byte_data, byte_size)
T_IPC_MSG msg;
T_PTR byte_data;
T_INT4 byte_size;
```

TipcMsgAppendChar — append a CHAR field to a message

```
T_BOOL TipcMsgAppendChar(msg, char_data)
T_IPC_MSG msg;
T_CHAR char_data;
```

TipcMsgAppendInt2 — append an INT2 field to a message

```
T_BOOL TipcMsgAppendInt2(msg, int2_data)
T_IPC_MSG msg;
T_INT2 int2_data;
```

TipcMsgAppendInt2Array — append an INT2_ARRAY field to a message

```
T_BOOL TipcMsgAppendInt2Array(msg, int2_array_data, int2_array_size)
T_IPC_MSG msg;
T_INT2 *int2_array_data;
T_INT4 int2_array_size;
```

TipcMsgAppendInt2ArrayPtr — append an INT2_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAppendInt2ArrayPtr(msg, int2_array_data,
int2_array_size, field_return)
T_IPC_MSG msg;
T_INT2 *int2_array_data;
T_INT4 int2_array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendInt4 — append an INT4 field to a message

```
T_BOOL TipcMsgAppendInt4(msg, int4_data)
T_IPC_MSG msg;
T_INT4 int4_data;
```

TipcMsgAppendInt4Array — append an INT4_ARRAY field to a message

```
T_BOOL TipcMsgAppendInt4Array(msg, int4_array_data, int4_array_size)
T_IPC_MSG msg;
T_INT4 *int4_array_data;
T_INT4 int4_array_size;
```

TipcMsgAppendInt4ArrayPtr — append an INT4_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAppendInt4ArrayPtr(msg, int4_array_data,
int4_array_size, field_return)
T_IPC_MSG msg;
T_INT4 *int4_array_data;
T_INT4 int4_array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendInt8 — append an INT8 field to a message

```
T_BOOL TipcMsgAppendInt8(msg, int8_data)
T_IPC_MSG msg;
T_INT8 int8_data;
```

TipcMsgAppendInt8Array — append an INT8_ARRAY field to a message

```
T_BOOL TipcMsgAppendInt8Array(msg, int8_array_data, int8_array_size)
T_IPC_MSG msg;
T_INT8 *int8_array_data;
T_INT4 int8_array_size;
```

TipcMsgAppendInt8ArrayPtr — append an INT8_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAppendInt8ArrayPtr(msg, int8_array_data,
int8_array_size, field_return)
T_IPC_MSG msg;
T_INT8 *int8_array_data;
T_INT4 int8_array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendMsg — append a MSG field to a message

```
T_BOOL TipcMsgAppendMsg(msg, msg_data)
T_IPC_MSG msg;
T_IPC_MSG msg_data;
```

TipcMsgAppendMsgArray — append a MSG_ARRAY field to a message

```
T_BOOL TipcMsgAppendMsgArray(msg, msg_array_data, msg_array_size)
T_IPC_MSG msg;
T_IPC_MSG *msg_array_data;
T_INT4 msg_array_size;
```

TipcMsgAppendMsgArrayPtr — append a MSG_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAppendMsgArrayPtr(msg, msg_array_data,
msg_array_size, field_return)
T_IPC_MSG msg;
T_IPC_MSG *msg_array_data;
T_INT4 msg_array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendMsgPtr — append a MSG field to a message using a pointer

```
T_BOOL TipcMsgAppendMsgPtr(msg, msg_data, field_return)
T_IPC_MSG msg;
T_IPC_MSG msg_data;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendReal4 — append a REAL4 field to a message

```
T_BOOL TipcMsgAppendReal4(msg, real4_data)
T_IPC_MSG msg;
T_REAL4 real4_data;
```

TipcMsgAppendReal4Array — append a REAL4_ARRAY field to a message

```
T_BOOL TipcMsgAppendReal4Array(msg, real4_array_data,
real4_array_size)
T_IPC_MSG msg;
T_REAL4 *real4_array_data;
T_INT4 real4_array_size;
```

TipcMsgAppendReal4ArrayPtr — append a REAL4_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAppendReal4ArrayPtr(msg, real4_array_data,
real4_array_size, field_return)
T_IPC_MSG msg;
T_REAL4 *real4_array_data;
T_INT4 real4_array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendReal8 — append a REAL8 field to a message

```
T_BOOL TipcMsgAppendReal8(msg, real8_data)
T_IPC_MSG msg;
T_REAL8 real8_data;
```

TipcMsgAppendReal8Array — append a REAL8_ARRAY field to a message

```
T_BOOL TipcMsgAppendReal8Array(msg, real8_array_data,
real8_array_size)
T_IPC_MSG msg;
T_REAL8 *real8_array_data;
T_INT4 real8_array_size;
```

TipcMsgAppendReal8ArrayPtr — append a REAL8_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAppendReal8ArrayPtr(msg, real8_array_data,
real8_array_size, field_return)
T_IPC_MSG msg;
T_REAL8 *real8_array_data;
T_INT4 real8_array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendReal16 — append a REAL16 field to a message

```
T_BOOL TipcMsgAppendReal16(msg, real16_data)
T_IPC_MSG msg;
T_REAL16 real16_data;
```

TipcMsgAppendReal16Array — append a REAL16_ARRAY field to a message

```
T_BOOL TipcMsgAppendReal16Array(msg, real16_array_data,
real16_array_size)
T_IPC_MSG msg;
T_REAL16 *real16_array_data;
T_INT4 real16_array_size;
```

TipcMsgAppendReal16ArrayPtr — append a REAL16_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAppendReal16ArrayPtr(msg, real16_array_data,
real16_array_size, field_return)
T_IPC_MSG msg;
T_REAL16 *real16_array_data;
T_INT4 real16_array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendStr — append a STR field to a message

```
T_BOOL TipcMsgAppendStr(msg, str_data)
T_IPC_MSG msg;
T_STR str_data;
```

TipcMsgAppendStrArray — append a STR_ARRAY field to a message

```
T_BOOL TipcMsgAppendStrArray(msg, str_array_data, str_array_size)
T_IPC_MSG msg;
T_STR *str_array_data;
T_INT4 str_array_size;
```

TipcMsgAppendStrArrayPtr — append a STR_ARRAY field to a message using a pointer

```
T_BOOL TipcMsgAppendStrArrayPtr(msg, str_array_data, str_array_size,
field_return)
T_IPC_MSG msg;
T_STR *str_array_data;
T_INT4 str_array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendStrPtr — append a STR field to a message using a pointer

```
T_BOOL TipcMsgAppendStrPtr(msg, str_data, field_return)
T_IPC_MSG msg;
T_STR str_data;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendStrReal8 — append a STR field and a REAL8 field to a message

```
T_BOOL TipcMsgAppendStrReal8(msg, str_data, real8_data)
T_IPC_MSG msg;
T_STR str_data;
T_REAL8 real8_data;
```

TipcMsgAppendTimestamp — append a TIMESTAMP field to a message

```
T_BOOL TipcMsgAppendTimestamp(msg, timestamp_data)
T_IPC_MSG msg;
T_REAL8 timestamp_data;
```

TipcMsgAppendTimestampArray — append a field containing an array of TIMESTAMP fields to a message

```
T_BOOL TipcMsgAppendTimestampArray(msg, timestamp_array_data,
array_size)
T_IPC_MSG msg;
T_REAL8 *timestamp_array_data;
T_INT4 array_size;
```

TipcMsgAppendTimestampArrayPtr — use a pointer to append a field containing an array of TIMESTAMP fields to a message

```
T_BOOL TipcMsgAppendTimestampArrayPtr(msg, timestamp_array_data,
array_size, field_return)
T_IPC_MSG msg;
T_REAL8 *timestamp_array_data;
T_INT4 array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendUnknown— append a field with value marked unknown to a message

```
T_BOOL TipcMsgAppendUnknown(msg, field_type)
T_IPC_MSG msg;
T_IPC_FT field_type;
```

TipcMsgAppendUtf8 — append a UTF8 field to a message

```
T_BOOL TipcMsgAppendUtf8(msg, utf8_data)
T_IPC_MSG msg;
T_STR utf8_data;
```

TipcMsgAppendUtf8Array — append a field containing an array of UTF8 fields to a message

```
T_BOOL TipcMsgAppendUtf8Array(msg, utf8_array_data, array_size)
T_IPC_MSG msg;
T_UTF8 *utf8_array_data;
T_INT4 array_size;
```

TipcMsgAppendUtf8ArrayPtr — append a field containing an array of UTF8 fields to a message using a pointer

```
T_BOOL TipcMsgAppendUtf8ArrayPtr(msg, utf8_array_data, array_size,
field_return)
T_IPC_MSG msg;
T_UTF8 *utf8_array_data;
T_INT4 array_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendUtf8Ptr — append a UTF8 field to a message using a pointer

```
T_BOOL TipcMsgAppendUtf8Ptr(msg, utf8_data, utf8_size, field_return)
T_IPC_MSG msg;
T_PTR utf8_data;
T_INT4 utf8_size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgAppendXml — append an XML field to the end of a message

```
T_BOOL TipcMsgAppendXml(msg, xml_data)
T_IPC_MSG msg;
T_XML xml_data;
```

TipcMsgAppendXmlPtr — append an XML field to a message using a pointer

```
T_BOOL TipcMsgAppendXmlPtr(msg, xml_data, field_return)
T_IPC_MSG msg;
T_XML xml_data;
T_IPC_MSG_FIELD *field_return;
```

Update Functions (TipcMsgUpdateNamed*)

The TipcMsgUpdateNamed* functions are used to update a field or fields in the data part of a message, referencing it by name.

TipcMsgUpdateNamedBinary — update a named BINARY field in a message

```
T_BOOL TipcMsgUpdateNamedBinary(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_PTR value;
T_INT4 size;
```

TipcMsgUpdateNamedBinaryPtr — update a named BINARY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedBinaryPtr(msg, name, value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_PTR value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedBool — update a named BOOL field in a message with new data

```
T_BOOL TipcMsgUpdateNamedBool(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_BOOL value;
```

TipcMsgUpdateNamedBoolArray — update a named BOOL_ARRAY field in a message with new data

```
T_BOOL TipcMsgUpdateNamedBoolArray(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_BOOL *value;
T_INT4 size;
```

TipcMsgUpdateNamedBoolArrayPtr — update a named BOOL_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedBoolArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_BOOL *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedByte — update a named BYTE field in a message with new data

```
T_BOOL TipcMsgUpdateNamedByte(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_UCHAR value;
```

TipcMsgUpdateNamedChar — update a named CHAR field in a message

```
T_BOOL TipcMsgUpdateNamedChar(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_CHAR value;
```

TipcMsgUpdateNamedInt2 — update a named INT2 field in a message

```
T_BOOL TipcMsgUpdateNamedInt2(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_INT2 value;
```

TipcMsgUpdateNamedInt2Array — update a named INT2_ARRAY field in a message

```
T_BOOL TipcMsgUpdateNamedInt2Array(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_INT2 *value;
T_INT4 size;
```

TipcMsgUpdateNamedInt2ArrayPtr — update a named INT2_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedInt2ArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_INT2 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedInt4 — update a named INT4 field in a message

```
T_BOOL TipcMsgUpdateNamedInt4(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_INT4 value;
```

TipcMsgUpdateNamedInt4Array — update a named INT4_ARRAY field in a message

```
T_BOOL TipcMsgUpdateNamedInt4Array(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_INT4 *value;
T_INT4 size;
```

TipcMsgUpdateNamedInt4ArrayPtr — update a named INT4_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedInt4ArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_INT4 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedInt8 — update a named INT8 field in a message

```
T_BOOL TipcMsgUpdateNamedInt8(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_INT8 value;
```

TipcMsgUpdateNamedInt8Array — update a named INT8_ARRAY field in a message

```
T_BOOL TipcMsgUpdateNamedInt8Array(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_INT8 *value;
T_INT4 size;
```

TipcMsgUpdateNamedInt8ArrayPtr — update a named INT8_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedInt8ArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_INT8 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedMsg — update a named MSG field in a message

```
T_BOOL TipcMsgUpdateNamedMsg(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_IPC_MSG value;
```

TipcMsgUpdateNamedMsgArray — update a named MSG_ARRAY field in a message

```
T_BOOL TipcMsgUpdateNamedMsgArray(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_IPC_MSG *value;
T_INT4 size;
```

TipcMsgUpdateNamedMsgArrayPtr — update a named MSG_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedMsgArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_IPC_MSG *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedMsgPtr — update a named MSG field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedMsgPtr(msg, name, value, *field_return)
T_IPC_MSG msg;
T_STR name;
T_REAL4 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedReal4 — update a named REAL4 field in a message

```
T_BOOL TipcMsgUpdateNamedReal4(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_REAL4 value;
```

TipcMsgUpdateNamedReal4Array — update a named REAL4_ARRAY field in a message

```
T_BOOL TipcMsgUpdateNamedReal4Array(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL4 *value;
T_INT4 size;
```

TipcMsgUpdateNamedReal4ArrayPtr — update a named REAL4_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedReal4ArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_REAL4 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedReal8 — update a named REAL8 field in a message

```
T_BOOL TipcMsgUpdateNamedReal8(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_REAL8 value;
```

TipcMsgUpdateNamedReal8Array — update a named REAL8_ARRAY field in a message

```
T_BOOL TipcMsgUpdateNamedReal8Array(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value;
T_INT4 size;
```

TipcMsgUpdateNamedReal8ArrayPtr — update a named REAL8_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedReal8ArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedReal16 — update a named REAL16 field in a message

```
T_BOOL TipcMsgUpdateNamedReal16(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_REAL16 value;
```

TipcMsgUpdateNamedReal16Array — update a named REAL16_ARRAY field in a message

```
T_BOOL TipcMsgUpdateNamedReal16Array(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL16 *value;
T_INT4 size;
```

TipcMsgUpdateNamedReal16ArrayPtr — update a named REAL16_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedReal16ArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_REAL16 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedStrArray — update a named STR_ARRAY field in a message

```
T_BOOL TipcMsgUpdateNamedStrArray(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
```

TipcMsgUpdateNamedStrArrayPtr — update a named STR_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedStrArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedStrPtr — update a named STR field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedStrPtr(msg, name, value, *field_return)
T_IPC_MSG msg;
T_STR name;
T_STR value;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedTimestamp — update a TIMESTAMP field in a message with new data

```
T_BOOL TipcMsgUpdateNamedTimestamp(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_REAL8 value;
```

TipcMsgUpdateNamedTimestampArray — update a field containing an array of TIMESTAMP fields in a message with new data

```
T_BOOL TipcMsgUpdateNamedTimestampArray(msg, name, value, size)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value;
T_INT4 size;
```

TipcMsgUpdateNamedTimestampArrayPtr — update a pointer to the value of a named field containing an array of TIMESTAMP fields

```
T_BOOL TipcMsgUpdateNamedTimestampArrayPtr(msg, name, value,
size, field_return)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedUtf8 — update a named UTF8 field in a message with new data

```
T_BOOL TipcMsgUpdateNamedUtf8(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_STR value;
```

TipcMsgUpdateNamedUtf8Array — update a named UTF8_ARRAY field in a message with new data

```
T_BOOL TipcMsgUpdateNamedUtf8Array(msg, name, *value, size)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
```

TipcMsgUpdateNamedUtf8ArrayPtr — update a named UTF8_ARRAY field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedUtf8ArrayPtr(msg, name, *value, size,
*field_return)
T_IPC_MSG msg;
T_STR name;
T_STR *value;
T_INT4 size;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedUtf8Ptr — update a named UTF8 field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedUtf8Ptr(msg, name, value, *field_return)
T_IPC_MSG msg;
T_STR name;
T_STR value;
T_IPC_MSG_FIELD *field_return;
```

TipcMsgUpdateNamedXml — update a named XML field in a message

```
T_BOOL TipcMsgUpdateNamedXml(msg, name, value)
T_IPC_MSG msg;
T_STR name;
T_XML value;
```

TipcMsgUpdateNamedXmlPtr — update a named XML field in a message using a pointer

```
T_BOOL TipcMsgUpdateNamedXmlPtr(msg, name, value, *field_return)
T_IPC_MSG msg;
T_STR name;
T_XML value;
T_IPC_MSG_FIELD *field_return;
```

Update Functions (TipcMsgFieldUpdate*)

The TipcMsgFieldUpdate* functions are used to update a field or fields in the data part of a message.

TipcMsgFieldUpdateBinaryPtr — update the pointer of the MSG field returned by TipcMsgAppendBinaryPtr

```
T_BOOL TipcMsgFieldUpdateBinaryPtr(field, binary_data, array_size)
T_IPC_MSG_FIELD field;
T_PTR binary_data;
T_INT4 array_size;
```

TipcMsgFieldUpdateBoolArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendBoolArrayPtr

```
T_BOOL TipcMsgFieldUpdateBoolArrayPtr(field, *bool_array,
array_size)
T_IPC_MSG_FIELD field;
T_BOOL *bool_array;
T_INT4 array_size;
```

TipcMsgFieldUpdateInt2ArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendInt2ArrayPtr

```
T_BOOL TipcMsgFieldUpdateInt2ArrayPtr(field, *int2_array,
array_size)
T_IPC_MSG_FIELD field;
T_INT2 *int2_array;
T_INT4 array_size;
```

TipcMsgFieldUpdateInt4ArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendInt4ArrayPtr

```
T_BOOL TipcMsgFieldUpdateInt4ArrayPtr(field, *int4_array,
array_size)
T_IPC_MSG_FIELD field;
T_INT4 *int4_array;
T_INT4 array_size;
```

TipcMsgFieldUpdateInt8ArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendInt8ArrayPtr

```
T_BOOL TipcMsgFieldUpdateInt8ArrayPtr(field, *int8_array,
array_size)
T_IPC_MSG_FIELD field;
T_INT8 *int8_array;
T_INT4 array_size;
```

TipcMsgFieldUpdateMsgArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendMsgArrayPtr

```
T_BOOL TipcMsgFieldUpdateMsgArrayPtr(field, *msg_array_data,
array_size)
T_IPC_MSG_FIELD field;
T_MSG *msg_array_data;
T_INT4 array_size;
```

TipcMsgFieldUpdateMsgPtr — update the pointer of the MSG field returned by TipcMsgAppendMsgPtr

```
T_BOOL TipcMsgFieldUpdateMsgPtr(field, msg_data)
T_IPC_MSG_FIELD field;
T_MSG msg_data;
```

TipcMsgFieldUpdateReal4ArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendReal4ArrayPtr

```
T_BOOL TipcMsgFieldUpdateReal4ArrayPtr(field, *real4_array,
array_size)
T_IPC_MSG_FIELD field;
T_REAL4 *real4_array;
T_INT4 array_size;
```

TipcMsgFieldUpdateReal8ArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendReal8ArrayPtr

```
T_BOOL TipcMsgFieldUpdateReal8ArrayPtr(field, *real8_array,
array_size)
T_IPC_MSG_FIELD field;
T_REAL8 *real8_array;
T_INT4 array_size;
```

TipcMsgFieldUpdateReal16ArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendReal16ArrayPtr

```
T_BOOL TipcMsgFieldUpdateReal16ArrayPtr(field, *real16_array,
array_size)
T_IPC_MSG_FIELD field;
T_REAL16 *real16_array;
T_INT4 array_size;
```

TipcMsgFieldUpdateStrArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendStrArrayPtr

```
T_BOOL TipcMsgFieldUpdateStrArrayPtr(field, *str_array_data,
array_size)
T_IPC_MSG_FIELD field;
T_STR *str_array_data;
T_INT4 array_size;
```

TipcMsgFieldUpdateStrPtr — update the pointer of the MSG field returned by TipcMsgAppendStrPtr

```
T_BOOL TipcMsgFieldUpdateStrPtr(field, str_data)
T_IPC_MSG_FIELD field;
T_STR str_data;
```

TipcMsgFieldUpdateTimestampArrayPtr — update the pointer of the message field returned by TipcMsgAppendTimestampArrayPtr

```
T_BOOL TipcMsgFieldUpdateTimestampArrayPtr(field,
*timestamp_array, array_size)
T_IPC_MSG_FIELD field;
T_REAL8 *timestamp_array;
T_INT4 array_size;
```

TipcMsgFieldUpdateUtf8ArrayPtr — update the pointer of the MSG field returned by TipcMsgAppendUtf8ArrayPtr

```
T_BOOL TipcMsgFieldUpdateUtf8ArrayPtr(field, *str_array_data,
array_size)
T_IPC_MSG_FIELD field;
T_STR *str_array_data;
T_INT4 array_size;
```

TipcMsgFieldUpdateUtf8Ptr — update the pointer of the MSG field returned by TipcMsgAppendUtf8Ptr

```
T_BOOL TipcMsgFieldUpdateUtf8Ptr(field, str_data)
T_IPC_MSG_FIELD field;
T_STR str_data;
```

TipcMsgFieldUpdateXmlPtr — update the pointer of the MSG field returned by TipcMsgAppendXmlPtr

```
T_BOOL TipcMsgFieldUpdateXmlPtr(field, xml_data)
T_IPC_MSG_FIELD field;
T_XML xml_data;
```

Create, Destroy and Clone Functions (TipcMsg*)

These functions are used to copy a message, create a new message of a specified type, and destroy a message.

TipcMsgClone — make an identical copy of a message

```
T_IPC_MSG TipcMsgClone(msg)
T_IPC_MSG msg;
```

TipcMsgCreate — create a new message

```
T_IPC_MSG TipcMsgCreate(mt)
T_IPC_MT mt;
```

TipcMsgDeleteCurrent — delete the current field from a message

```
T_BOOL TipcMsgDeleteCurrent(msg)
T_IPC_MSG msg;
```

TipcMsgDeleteField — delete a field, specified by field number, from a message

```
T_BOOL TipcMsgDeleteField(msg, field_num)
T_IPC_MSG msg;
T_INT4 field_num;
```

TipcMsgDeleteNamedField — delete a field, specified by field name, from a message

```
T_BOOL TipcMsgDeleteNamedField(msg, name)
T_IPC_MSG msg;
T_STR name;
```

TipcMsgDestroy — destroy a message

```
T_BOOL TipcMsgDestroy(msg)
T_IPC_MSG msg;
```

TipcMsgExistsNamed — determine if a field exists in a message, using a name

```
T_BOOL TipcMsgExistsNamed(msg, name, exists)
T_IPC_MSG msg;
T_STR name;
T_BOOL *exists;
```

File Functions (TipcMsgFile*)

The TipcMsgFile* functions are used to work with message files.

TipcMsgFileCreate — create a message file

```
T_IPC_MSG_FILE TipcMsgFileCreate(file_name, create_mode)
T_STR file_name;
T_IPC_MSG_FILE_CREATE_MODE create_mode;
```

TipcMsgFileCreateFromFile — create a message file from an existing file

```
T_IPC_MSG_FILE TipcMsgFileCreateFromFile(file, create_mode)
FILE *file;
T_IPC_MSG_FILE_CREATE_MODE create_mode;
```

TipcMsgFileDestroy — destroy a T_IPC_MSG_FILE

```
T_BOOL TipcMsgFileDestroy(msg_file)
T_IPC_MSG_FILE msg_file;
```

TipcMsgFileRead — read a message from a message file

```
T_BOOL TipcMsgFileRead(msg_file, msg_return)
T_IPC_MSG_FILE msg_file;
T_IPC_MSG *msg_return;
```

TipcMsgFileWrite — write a message to a message file

```
T_BOOL TipcMsgFileWrite(msg_file, msg)
T_IPC_MSG_FILE msg_file;
T_IPC_MSG msg;
```

Property Get Functions (TipcMsgGet*)

The TipcMsgGet* functions are used to retrieve the properties of a message. A property is a piece of information about a message, typically stored in the message header and the message data structure. They include items such as the destination of the message, message priority, and message type.

TipcMsgGetArrivalTimestamp — determine the arrival timestamp property of a message

```
T_BOOL TipcMsgGetArrivalTimestamp(msg, arrival_timestamp_return)
T_IPC_MSG msg;
T_REAL8 *arrival_timestamp_return;
```

TipcMsgGetCompression — get the compression setting of a message

```
T_BOOL TipcMsgGetCompression(msg, compress_return)
T_IPC_MSG msg;
T_BOOL *compress_return;
```

TipcMsgGetCorrelationId — determine the correlation identifier property of a message

```
T_BOOL TipcMsgGetCorrelationId(msg, correlation_id_return)
T_IPC_MSG msg;
T_STR *correlation_id_return;
```

TipcMsgGetCurrent — determine the current field of a message

```
T_BOOL TipcMsgGetCurrent(msg, field_num_return)
T_IPC_MSG msg;
T_INT4 *field_num_return;
```

TipcMsgGetCurrentFieldKnown — check if the current field of a message is of known value

```
T_BOOL TipcMsgGetCurrentFieldKnown(msg, field_valid_return)
T_IPC_MSG msg;
T_BOOL *field_valid_return;
```

TipcMsgGetDeliveryMode — determine the delivery mode of a message

```
T_BOOL TipcMsgGetDeliveryMode(msg, delivery_mode_return)
T_IPC_MSG msg;
T_IPC_DELIVERY_MODE *delivery_mode_return;
```

TipcMsgGetDeliveryTimeout — determine the delivery timeout property of a message

```
T_BOOL TipcMsgGetDeliveryTimeout(msg, delivery_timeout_return)
T_IPC_MSG msg;
T_REAL8 *delivery_timeout_return;
```

TipcMsgGetDest — determine the destination property of a message

```
T_BOOL TipcMsgGetDest(msg, dest_return)
T_IPC_MSG msg;
T_STR *dest_return;
```

TipcMsgGetExpiration — determine the expiration property of a message

```
T_BOOL TipcMsgGetExpiration(msg, expiration_return)
T_IPC_MSG msg;
T_REAL8 *expiration_return;
```

TipcMsgGetHeaderStrEncode — determine the header string encode property of a message

```
T_BOOL TipcMsgGetHeaderStrEncode(msg, header_str_encode_return)
T_IPC_MSG msg;
T_BOOL *header_str_encode_return;
```

TipcMsgGetLbMode — determine the load balancing mode of a message

```
T_BOOL TipcMsgGetLbMode(msg, lb_mode_return)
T_IPC_MSG msg;
T_IPC_LB_MODE *lb_mode_return;
```

TipcMsgGetMessageId — determine the message identifier property of a message

```
T_BOOL TipcMsgGetMessageId(msg, message_id_return)
T_IPC_MSG msg;
T_STR *message_id_return;
```

TipcMsgGetNumFields — determine the number of fields in a message

```
T_BOOL TipcMsgGetNumFields(msg, num_fields_return)
T_IPC_MSG msg;
T_INT4 *num_fields_return;
```

TipcMsgGetPacketSize — determine the packet size of a message

```
T_BOOL TipcMsgGetPacketSize(msg, packet_size_return)
T_IPC_MSG msg;
T_UINT4 *packet_size_return;
```

TipcMsgGetPriority — determine the priority of a message

```
T_BOOL TipcMsgGetPriority(msg, priority_return)
T_IPC_MSG msg;T_INT2 *priority_return;
```

TipcMsgGetReadOnly — get the read-only property of message

```
T_BOOL TipcMsgGetReadOnly(msg, read_only_return)
T_IPC_MSG msg;
T_BOOL *read_only_return;
```

TipcMsgGetRefCount — determine the reference count of a message

```
T_BOOL TipcMsgGetRefCount(msg, ref_count_return)
T_IPC_MSG msg;
T_INT4 *ref_count_return;
```

TipcMsgGetReplyTo — determine the reply to destination property of a message

```
T_BOOL TipcMsgGetReplyTo(msg, reply_to_dest_return)
T_IPC_MSG msg;
T_STR *reply_to_dest_return;
```

TipcMsgGetSender — determine the sender property of a message

```
T_BOOL TipcMsgGetSender(msg, sender_return)
T_IPC_MSG msg;
T_STR *sender_return;
```

TipcMsgGetSenderTimestamp — determine the sender timestamp property of a message

```
T_BOOL TipcMsgGetSenderTimestamp(msg, sender_timestamp_return)
T_IPC_MSG msg;
T_REAL8 *sender_timestamp_return;
```

TipcMsgGetSeqNum — determine the sequence number of a message

```
T_BOOL TipcMsgGetSeqNum(msg, seq_num_return)
T_IPC_MSG msg;
T_INT4 *seq_num_return;
```

TipcMsgGetType — determine the type of a message

```
T_BOOL TipcMsgGetType(msg, mt_return)
T_IPC_MSG msg;
T_IPC_MT *mt_return;
```

TipcMsgGetTypeNamed — determine the type of a message using a name

```
T_BOOL TipcMsgGetTypeNamed(msg, name, type)
T_IPC_MSG msg;
T_STR name;
T_IPC_MT *type;
```

TipcMsgGetTypeNum — get the message type of a message as an integer

```
T_BOOL TipcMsgGetTypeNum (msg, type_num_return)
T_IPC_MSG msg;
T_INT4 *type_num_return;
```

TipcMsgGetUserProp — determine the user-defined property of a message

```
T_BOOL TipcMsgGetUserProp(msg, user_prop_return)
T_IPC_MSG msg;
T_INT4 *user_prop_return;
```

Property Set Functions (TipcMsgSet*)

The TipcMsgSet* functions are used to set the properties of a message. A property is a piece of information about a message, typically stored in the message header and the message data structure. They include items such as the destination of the message, message priority, and message type.

TipcMsgSetArrivalTimestamp — set the arrival timestamp property of a message

```
T_BOOL TipcMsgSetArrivalTimestamp(msg, arrival_timestamp)
T_IPC_MSG msg;
T_REAL8 arrival_timestamp;
```

TipcMsgSetCompression — enable or disable message compression

```
T_BOOL TipcMsgSetCompression(msg, compress)
T_IPC_MSG msg;
T_BOOL compress;
```

TipcMsgSetCorrelationId — set the correlation identifier property of a message

```
T_BOOL TipcMsgSetCorrelationId(msg, correlation_id)
T_IPC_MSG msg;
T_STR correlation_id;
```

TipcMsgSetCurrent — set the current field of a message

```
T_BOOL TipcMsgSetCurrent(msg, field_num)
T_IPC_MSG msg;
T_INT4 field_num;
```

TipcMsgSetDeliveryMode — set the delivery mode of a message

```
T_BOOL TipcMsgSetDeliveryMode(msg, delivery_mode)
T_IPC_MSG msg;
T_IPC_DELIVERY_MODE delivery_mode;
```

TipcMsgSetDeliveryTimeout — set the delivery timeout property of a message

```
T_BOOL TipcMsgSetDeliveryTimeout(msg, delivery_timeout)
T_IPC_MSG msg;
T_REAL8 delivery_timeout;
```

TipcMsgSetDest — set the destination of a message

```
T_BOOL TipcMsgSetDest(msg, dest)
T_IPC_MSG msg;
T_STR dest;
```

TipcMsgSetExpiration — set the expiration property of a message

```
T_BOOL TipcMsgSetExpiration(msg, expiration)
T_IPC_MSG msg;
T_REAL8 expiration;
```

TipcMsgSetHeaderStrEncode — set the header string encode property of a message

```
T_BOOL TipcMsgSetHeaderStrEncode(msg, header_str_encode)
T_IPC_MSG msg;
T_BOOL header_str_encode;
```

TipcMsgSetLbMode — set the load balancing mode of a message

```
T_BOOL TipcMsgSetLbMode(msg, lb_mode)
T_IPC_MSG msg;
T_IPC_LB_MODE lb_mode;
```

TipcMsgSetNameCurrent — set the name of the current field

```
T_BOOL TipcMsgSetNameCurrent(msg, name)
T_IPC_MSG msg;
T_STR name;
```

TipcMsgSetNumFields — set the number of fields in a message

```
T_BOOL TipcMsgSetNumFields(msg, num_fields)
T_IPC_MSG msg;
T_INT4 num_fields;
```

TipcMsgSetPriority — set the priority of a message

```
T_BOOL TipcMsgSetPriority(msg, priority)
T_IPC_MSG msg;
T_INT2 priority;
```

TipcMsgSetReplyTo — set the reply to destination property of a message

```
T_BOOL TipcMsgSetReplyTo(msg, reply_to_dest)
T_IPC_MSG msg;
T_STR reply_to_dest;
```

TipcMsgSetSender — set the sender of a message

```
T_BOOL TipcMsgSetSender(msg, sender)
T_IPC_MSG msg;
T_STR sender;
```

TipcMsgSetSenderTimestamp — set the sender timestamp property of a message

```
T_BOOL TipcMsgSetSenderTimestamp(msg, sender_timestamp)
T_IPC_MSG msg;
T_REAL8 sender_timestamp;
```

TipcMsgSetType — set the type of a message

```
T_BOOL TipcMsgSetType(msg, mt)
T_IPC_MSG msg;
T_IPC_MT mt;
```

TipcMsgSetUserProp — set the user-defined property of a message

```
T_BOOL TipcMsgSetUserProp(msg, user_prop)
T_IPC_MSG msg;
T_INT4 user_prop;
```

Access Functions (TipcMsgGetName*)

The TipcMsgGetName* functions are used to read a field or fields from the data part of a message using a name.

TipcMsgGetNameCurrent — determine the name of the current field

```
T_BOOL TipcMsgGetNameCurrent(msg, name_return)
T_IPC_MSG msg;
T_STR *name_return;
```

TipcMsgGetNamedBinary — get a BINARY field from a message using a name

```
T_BOOL TipcMsgGetNamedBinary(msg, name, value_return, size_return)
T_IPC_MSG msg;
T_STR name;
T_PTR *value_return;
T_INT4 *size_return;
```

TipcMsgGetNamedChar — get a CHAR field from a message using a name

```
T_BOOL TipcMsgGetNamedChar(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_CHAR *value_return;
```

TipcMsgGetNamedInt2 — get an INT2 field from a message using a name

```
T_BOOL TipcMsgGetNamedInt2(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_INT2 *value_return;
```

TipcMsgGetNamedInt2Array — get an INT2_ARRAY field from a message using a name

```
T_BOOL TipcMsgGetNamedInt2Array(msg, name, value_return,
size_return)
T_IPC_MSG msg;
T_STR name;
T_INT2 **value_return;
T_INT4 *size_return;
```

TipcMsgGetNamedInt4 — get an INT4 field from a message using a name

```
T_BOOL TipcMsgGetNamedInt4(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_INT4 *value_return;
```

TipcMsgGetNamedInt4Array — get an INT4_ARRAY field from a message using a name

```
T_BOOL TipcMsgGetNamedInt4Array(msg, name, value_return,
size_return)
T_IPC_MSG msg;
T_STR name;
T_INT4 **value_return;
T_INT4 *size_return;
```

TipcMsgGetNamedInt8 — get an INT8 field from a message using a name

```
T_BOOL TipcMsgGetNamedInt8(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_INT8 *value_return;
```

TipcMsgGetNamedInt8Array — get an INT8_ARRAY field from a message using a name

```
T_BOOL TipcMsgGetNamedInt8Array(msg, name, value_return,
size_return)
T_IPC_MSG msg;
T_STR name;
T_INT8 **value_return;
T_INT4 *size_return;
```

TipcMsgGetNamedMsg — get a MSG field from a message using a name

```
T_BOOL TipcMsgGetNamedMsg(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_IPC_MSG *value_return;
```

TipcMsgGetNamedMsgArray — get a MSG_ARRAY field from a message using a name

```
T_BOOL TipcMsgGetNamedMsgArray(msg, name, value_return, size_return)
T_IPC_MSG msg;
T_STR name;
T_IPC_MSG **value_return;
T_INT4 *size_return;
```

TipcMsgGetNamedReal4 — get a REAL4 field from a message using a name

```
T_BOOL TipcMsgGetNamedReal4(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_REAL4 *value_return;
```

TipcMsgGetNamedReal4Array — get a REAL4_ARRAY field from a message using a name

```
T_BOOL TipcMsgGetNamedReal4Array(msg, name, value_return,
size_return)
T_IPC_MSG msg;
T_STR name;
T_REAL4 **value_return;
T_INT4 *size_return;
```

TipcMsgGetNamedReal8 — get a REAL8 field from a message using a name

```
T_BOOL TipcMsgGetNamedReal8(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value_return;
```

TipcMsgGetNamedReal8Array — get a REAL8_ARRAY field from a message using a name

```
T_BOOL TipcMsgGetNamedReal8Array(msg, name, value_return,
size_return)
T_IPC_MSG msg;
T_STR name;
T_REAL8 **value_return;
T_INT4 *_size_return;
```

TipcMsgGetNamedReal16 — get a REAL16 field from a message using a name

```
T_BOOL TipcMsgGetNamedReal16(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_REAL16 *value_return;
```

TipcMsgGetNamedReal16Array — get a REAL16_ARRAY field from a message using a name

```
T_BOOL TipcMsgGetNamedReal16Array(msg, name, value_return,
size_return)
T_IPC_MSG msg;
T_STR name;
T_REAL16 **value_return;
T_INT4 *size_return;
```

TipcMsgGetNamedStr — get a STR field from a message using a name

```
T_BOOL TipcMsgGetNamedStr(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_STR *value_return;
```

TipcMsgGetNamedStrArray — get a STR_ARRAY field from a message using a name

```
T_BOOL TipcMsgGetNamedStrArray(msg, name, value_return, size_return)
T_IPC_MSG msg;
T_STR name;
T_STR **value_return;
T_INT4 *size_return;
```

TipcMsgGetNamedTimestamp — get a field with a TIMESTAMP value from a message, using a name

```
T_BOOL TipcMsgGetNamedTimestamp(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_REAL8 *value_return;
```

TipcMsgGetNamedTimestampArray — get a field containing an array of TIMESTAMP values from a message, using a name

```
T_BOOL TipcMsgGetNamedTimestampArray(msg, name, value_return,
size_return)
T_IPC_MSG msg;
T_STR name;
T_REAL8 **value_return;
T_INT4 *size_return;
```

TipcMsgGetNamedUnknown — determine whether the next named field of a message is of unknown value

```
T_BOOL TipcMsgGetNamedUnknown(msg, name)
T_IPC_MSG msg;
T_STR name;
```

TipcMsgGetNamedXml — get an XML field from a message using a name

```
T_BOOL TipcMsgGetNamedXml(msg, name, value_return)
T_IPC_MSG msg;
T_STR name;
T_XML *value_return;
```

Access Functions (TipcMsgNext*)

The TipcMsgNext* functions are used to read a field or fields from the data part of a message.

TipcMsgNextBinary — get a BINARY field from a message

```
T_BOOL TipcMsgNextBinary(msg, binary_return, size_return)
T_IPC_MSG msg;
T_PTR *binary_return;
T_INT4 *size_return;
```

TipcMsgNextBool — get a BOOL field from a message

```
T_BOOL TipcMsgNextBool(msg, bool_return)
T_IPC_MSG msg;
T_BOOL *bool_return;
```

TipcMsgNextBoolArray — get a field containing an array of BOOL fields from a message

```
T_BOOL TipcMsgNextBoolArray(msg, bool_array_return, array_size_return)
T_IPC_MSG msg;
T_BOOL **bool_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextByte — get a BYTE field from a message

```
T_BOOL TipcMsgNextByte(msg, byte_return)
T_IPC_MSG msg;
T_BYTE *byte_return;
```

TipcMsgNextChar — get a CHAR field from a message

```
T_BOOL TipcMsgNextChar(msg, char_return)
T_IPC_MSG msg;
T_CHAR *char_return;
```

TipcMsgNextInt2 — get an INT2 field from a message

```
T_BOOL TipcMsgNextInt2(msg, int2_return)
T_IPC_MSG msg;
T_INT2 *int2_return;
```

TipcMsgNextInt2Array — get an INT2_ARRAY field from a message

```
T_BOOL TipcMsgNextInt2Array(msg, int2_array_return, array_size_return)
T_IPC_MSG msg;
T_INT2 **int2_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextInt4 — get an INT4 field from a message

```
T_BOOL TipcMsgNextInt4(msg, int4_return)
T_IPC_MSG msg;
T_INT4 *int4_return;
```

TipcMsgNextInt4Array — get an INT4_ARRAY field from a message

```
T_BOOL TipcMsgNextInt4Array(msg, int4_array_return, array_size_return)
T_IPC_MSG msg;
T_INT4 **int4_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextInt8 — get an INT8 field from a message

```
T_BOOL TipcMsgNextInt8(msg, int8_return)
T_IPC_MSG msg;
T_INT8 *int8_return;
```

TipcMsgNextInt8Array — get an INT8_ARRAY field from a message

```
T_BOOL TipcMsgNextInt8Array(msg, int8_array_return, array_size_return)
T_IPC_MSG msg;
T_INT8 **int8_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextMsg — get a MSG field from a message

```
T_BOOL TipcMsgNextMsg(msg, msg_return)
T_IPC_MSG msg;
T_IPC_MSG *msg_return;
```

TipcMsgNextMsgArray — get a MSG_ARRAY field from a message

```
T_BOOL TipcMsgNextMsgArray(msg, msg_array_return, array_size_return)
T_IPC_MSG msg;
T_IPC_MSG **msg_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextReal4 — get a REAL4 field from a message

```
T_BOOL TipcMsgNextReal4(msg, real4_return)
T_IPC_MSG msg;
T_REAL4 *real4_return;
```

TipcMsgNextReal4Array — get a REAL4_ARRAY field from a message

```
T_BOOL TipcMsgNextReal4Array(msg, real4_array_return,
array_size_return)
T_IPC_MSG msg;
T_REAL4 **real4_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextReal8 — get a REAL8 field from a message

```
T_BOOL TipcMsgNextReal8(msg, real8_return)
T_IPC_MSG msg;
T_REAL8 *real8_return;
```

TipcMsgNextReal8Array — get a REAL8_ARRAY field from a message

```
T_BOOL TipcMsgNextReal8Array(msg, real8_array_return,
array_size_return)
T_IPC_MSG msg;
T_REAL8 **real8_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextReal16 — get a REAL16 field from a message

```
T_BOOL TipcMsgNextReal16(msg, real16_return)
T_IPC_MSG msg;
T_REAL16 *real16_return;
```

TipcMsgNextReal16Array — get a REAL16_ARRAY field from a message

```
T_BOOL TipcMsgNextReal16Array(msg, real16_array_return,
array_size_return)
T_IPC_MSG msg;
T_REAL16 **real16_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextStr — get a STR field from a message

```
T_BOOL TipcMsgNextStr(msg, str_return)
T_IPC_MSG msg;
T_STR *str_return;
```

TipcMsgNextStrArray — get a STR_ARRAY field from a message

```
T_BOOL TipcMsgNextStrArray(msg, str_array_return, array_size_return)
T_IPC_MSG msg;
T_STR **str_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextStrReal8 — get a STR field and a REAL8 field from a message

```
T_BOOL TipcMsgNextStrReal8(msg, str_return, real8_return)
T_IPC_MSG msg;
T_STR *str_return;
T_REAL8 *real8_return;
```

TipcMsgNextTimestamp — get a TIMESTAMP field from a message

```
T_BOOL TipcMsgNextTimestamp(msg, timestamp_return)
T_IPC_MSG msg;
T_REAL8 *timestamp_return;
```

TipcMsgNextTimestampArray — get a field containing an array of TIMESTAMP fields from a message

```
T_BOOL TipcMsgNextTimestampArray(msg, timestamp_array_return,
array_size_return)
T_IPC_MSG msg;
T_REAL8 **timestamp_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextType — determine the field type of the current field of a message

```
T_BOOL TipcMsgNextType(msg, ft_return)
T_IPC_MSG msg;
T_IPC_FT *ft_return;
```

TipcMsgNextUnknown — check if the next field of a message is of unknown value

```
T_BOOL TipcMsgNextUnknown(msg)
T_IPC_MSG msg;
```

TipcMsgNextUtf8 — get a UTF8 field from a message

```
T_BOOL TipcMsgNextUtf8(msg, utf8_return)
T_IPC_MSG msg;
T_UTF8 *utf8_return;
```

TipcMsgNextUtf8Array — get a field containing an array of UTF8 fields from a message

```
T_BOOL TipcMsgNextUtf8Array(msg, utf8_array_return, array_size_return)
T_IPC_MSG msg;
T_UTF8 **utf8_array_return;
T_INT4 *array_size_return;
```

TipcMsgNextXml — determine the next XML string from a message

```
T_BOOL TipcMsgNextXml(msg, xml_return)
T_IPC_MSG msg;
T_XML xml_return;
```

Miscellaneous Functions

These functions do not fit with any of the other established categories.

TipcMsgAck — acknowledge delivery of a message

```
T_BOOL TipcMsgAck(msg)
T_IPC_MSG msg;
```

TipcMsgFieldSetSize — set the size of a message pointer field

```
T_BOOL TipcMsgFieldSetSize(field, size)
T_IPC_MSG_FIELD field;
T_INT4 size;
```

TipcMsgGenerateMessageId — generate the message identifier property for the message

```
T_BOOL TipcMsgGenerateMessageId(msg)
T_IPC_MSG msg;
```

TipcMsgIncrRefCount — increment the reference count of a message

```
T_BOOL TipcMsgIncrRefCount(msg)
T_IPC_MSG msg;
```

TipcMsgPrint — print all information of a message in a portable, reproducible format

```
T_BOOL TipcMsgPrint(msg, func)
T_IPC_MSG msg;
T_OUT_FUNC func;
```

TipcMsgPrintError — report an error about an unexpected message

```
T_BOOL TipcMsgPrintError(msg)
T_IPC_MSG msg;
```

TipcMsgRead — read zero or more fields from a message

```
T_BOOL TipcMsgRead(msg, ...)
T_IPC_MSG msg;
```

TipcMsgReadVa — read zero or more fields from a message (`va_list` version)

```
T_BOOL TipcMsgReadVa(msg, var_arg_list)
T_IPC_MSG msg;
va_list var_arg_list;
```

TipcMsgTraverse — traverse all fields in a message

```
T_PTR TipcMsgTraverse(msg, func, arg)
T_IPC_MSG msg;
T_IPC_MSG_TRAV_FUNC func;
T_PTR arg;
```

TipcMsgWrite — append zero or more fields to a message

```
T_BOOL TipcMsgWrite(msg, ...)
T_IPC_MSG msg;
```

TipcMsgWriteVa — append zero or more fields to a message (`va_list` version)

```
T_BOOL TipcMsgWriteVa(msg, var_arg_list)
T_IPC_MSG msg;
va_list var_arg_list;
```

Message Type Functions (TipcMt*)

The TipcMt* functions are used to work with message types, which are templates for messages.

TipcMtCreate — create a new message type

```
T_IPC_MT TipcMtCreate(name, num, grammar)
T_STR name;
T_INT4 num;
T_STR grammar;
```

TipcMtDestroy — destroy a message type

```
T_BOOL TipcMtDestroy(mt)
T_IPC_MT mt;
```

TipcMtLogAddMt — add a message type to a message file logging type

```
T_BOOL TipcMtLogAddMt(log_type, mt)
T_IPC_SRV_LOG_TYPE log_type;
T_IPC_MT mt;
```

TipcMtLogRemoveMt — remove a message type from a message file logging type

```
T_BOOL TipcMtLogRemoveMt(log_type, mt)
T_IPC_SRV_LOG_TYPE log_type;
T_IPC_MT mt;
```

TipcMtLookup — look up a message type by name

```
T_IPC_MT TipcMtLookup(name)
T_STR name;
```

TipcMtLookupByNum — look up a message type by number

```
T_IPC_MT TipcMtLookupByNum(num)
T_INT4 num;
```

TipcMtPrint— print all information about a message type in a portable, reproducible format

```
T_BOOL TipcMtPrint(mt, func)
T_IPC_MT mt;
T_OUT_FUNC func;
```

TipcMtTraverse — traverse all message types

```
T_PTR TipcMtTraverse(func, arg)
T_IPC_MT_TRAV_FUNC func;
T_PTR arg;
```

Get Functions (TipcMtGet*)

The TipcMtGet* functions are used to retrieve information about a message type.

TipcMtGetCompression — get the compression setting of a message type

```
T_BOOL TipcMtGetCompressoin(mt, compress_return)
T_IPC_MT mt;
T_BOOL *compress_return;
```

TipcMtGetDeliveryMode — determine the delivery mode of a message type

```
T_BOOL TipcMtGetDeliveryMode(mt, delivery_mode_return)
T_IPC_MT mt;
T_IPC_DELIVERY_MODE *delivery_mode_return;
```

TipcMtGetDeliveryTimeout — determine the delivery timeout property of a message type

```
T_BOOL TipcMtGetDeliveryTimeout(mt, delivery_timeout_return)
T_IPC_MT mt;
T_REAL8 *delivery_timeout_return;
```

TipcMtGetGrammar — determine the grammar of a message type

```
T_BOOL TipcMtGetGrammar(mt, grammar_return)
T_IPC_MT mt;
T_STR *grammar_return;
```

TipcMtGetHeaderStrEncode — determine the header string encode property of a message type

```
T_BOOL TipcMtGetHeaderStrEncode(mt, header_str_encode_return)
T_IPC_MT mt;
T_BOOL *header_str_encode_return;
```

TipcMtGetLbMode — determine the load balancing mode of a message type

```
T_BOOL TipcMtGetLbMode(mt, lb_mode_return)
T_IPC_MT mt;
T_IPC_LB_MODE *lb_mode_return;
```

TipcMtGetName — determine the name of a message type

```
T_BOOL TipcMtGetName(mt, name_return)
T_IPC_MT mt;
T_STR *name_return;
```

TipcMtGetNum — determine the number of a message type

```
T_BOOL TipcMtGetNum(mt, num_return)
T_IPC_MT mt;
T_INT4 *num_return;
```

TipcMtGetPriority — determine the priority of a message type

```
T_BOOL TipcMtGetPriority(mt, priority_return)
T_IPC_MT mt;
T_INT2 *priority_return;
```

TipcMtGetUserProp — determine the user-defined property of a message type

```
T_BOOL TipcMtGetUserProp(mt, user_prop_return)
T_IPC_MT mt;
T_INT4 *user_prop_return;
```

Set Functions (TipcMtSet*)

The TipcMtSet* functions are used to set the properties of a message type.

TipcMtSetCompression — set the compression setting of a message type

```
T_BOOL TipcMtSetCompression(mt, compress)
T_IPC_MT mt;
T_BOOL compress;
```

TipcMtSetDeliveryMode — set the delivery mode of a message type

```
T_BOOL TipcMtSetDeliveryMode(mt, delivery_mode)
T_IPC_MT mt;
T_IPC_DELIVERY_MODE delivery_mode;
```

TipcMtSetDeliveryTimeout — set the delivery timeout property of a message type

```
T_BOOL TipcMtSetDeliveryMode(mt, delivery_timeout)
T_IPC_MT mt;
T_REAL8 delivery_timeout;
```

TipcMtSetHeaderStrEncode — set the header string encode property of a message type

```
T_BOOL TipcMtSetHeaderStrEncode(mt, header_str_encode)
T_IPC_MT mt;
T_BOOL header_str_encode;
```

TipcMtSetLbMode — set the load balancing mode of a message type

```
T_BOOL TipcMtSetDeliveryMode(mt, lb_mode)
T_IPC_MT mt;
T_IPC_LB_MODE lb_mode;
```

TipcMtSetPriority — set the priority of a message type

```
T_BOOL TipcMtSetPriority(mt, priority)
T_IPC_MT mt;
T_INT2 priority;
```

TipcMtSetPriorityUnknown — set the priority of a message type to unknown

```
T_BOOL TipcMtSetPriorityUnknown(mt)
T_IPC_MT mt;
```

TipcMtSetUserProp — set the user-defined property of a message type

```
T_BOOL TipcMtSetUserProp(mt, user_prop)
T_IPC_MT mt;
T_INT4 user_prop;
```

Property Table Functions (TipcProperties*)

The TipcProperties* functions are used to manipulate properties tables which describe the properties of queues and queue managers. For more information on queues and queue managers, see the *TIBCO SmartMQ User's Guide*.

TipcPropertiesCreate — create a properties table

```
T_IPC_PROPERTIES TipcPropertiesCreate(void)
```

TipcPropertiesCreateMsg — create a properties table from a SmartSockets message

```
T_IPC_PROPERTIES TipcPropertiesCreateMsg(msg)
T_IPC_MSG msg;
```

TipcPropertiesDestroy — destroy a properties table

```
T_BOOL TipcPropertiesDestroy(props)
T_IPC_PROPERTIES props;
```

TipcPropertiesGet — determine the property value for the specified property name

```
T_STR TipcPropertiesGet(props, name)
T_IPC_PROPERTIES props;
T_STR name;
```

TipcPropertiesGetCount — determine the count of name-value pairs in the properties table

```
T_BOOL TipcPropertiesGetCount(props, count_return)
T_IPC_PROPERTIES props;
T_INT4 *count_return;
```

TipcPropertiesGetDefault — determine the default value for the specified property

```
T_STR TipcPropertiesGetDefault(props, name)
T_IPC_PROPERTIES props;
T_STR name;
```

TipcPropertiesMsgCreate — create a SmartSockets message from a properties table

```
T_IPC_MSG TipcPropertiesMsgCreate(props)
T_IPC_PROPERTIES props;
```

TipcPropertiesSet — set the specified name and value in the properties table

```
T_BOOL TipcPropertiesSet(props, name, value)
T_IPC_PROPERTIES props;
T_STR name;
T_STR value;
```

TipcPropertiesTraverse — traverse the properties table

```
T_STR TipcPropertiesTraverse(props, traverse_func, traverse_arg)
T_IPC_PROPERTIES props;
T_IPC_PROPERTY_TRAV_FUNC traverse_func;
T_PTR traverse_arg;
```

Global Connection Functions (TipcSrv*)

The TipcSrv* functions are used for communication between a client program (RTclient) and RTserver. These functions enable the publish-subscribe message delivery services of SmartSockets.

Global Connection Callback Functions (TipcCbSrv* and TipcSrv*Cb*)

TipcCbSrvError — callback to handle errors on the connection to RTserver

```
void TipcCbSrvError(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_ERROR_CB_DATA data;
T_CB_ARG arg;
```

TipcCbSrvProcessControl — callback to process CONTROL messages from RTserver

```
void TipcCbSrvProcessControl(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
```

TipcCbSrvProcessGmdFailure — callback on connection to RTserver to process GMD_FAILURE messages when GMD fails

```
void TipcCbSrvProcessGmdFailure(conn, data, arg)
T_IPC_CONN conn;
T_IPC_CONN_PROCESS_CB_DATA data;
T_CB_ARG arg;
```

TipcSrvCreateCbCreate — create a server create callback

```
T_CB TipcSrvCreateCbCreate(func, arg)
T_IPC_SRV_CREATE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvCreateCbLookup — look up a server create callback

```
T_CB TipcSrvCreateCbLookup(func, arg)
T_IPC_SRV_CREATE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvDefaultCbCreate — create a default callback in the connection to RTserver

```
T_CB TipcSrvDefaultCbCreate(func, arg)
T_IPC_CONN_DEFAULT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvDefaultCbLookup — look up a default callback in the connection to RTserver

```
T_CB TipcSrvDefaultCbLookup(func, arg)
T_IPC_CONN_DEFAULT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvDestroyCbCreate — create a server destroy callback

```
T_CB TipcSrvDestroyCbCreate(func, arg)
T_IPC_SRV_DESTROY_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvDestroyCbLookup — look up a server destroy callback

```
T_CB TipcSrvDestroyCbLookup(func, arg)
T_IPC_SRV_DESTROY_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvErrorCbCreate — create an error callback in the connection to RTserver

```
T_CB TipcSrvErrorCbCreate(func, arg)
T_IPC_CONN_ERROR_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvErrorCbLookup — look up an error callback in the connection to RTserver

```
T_CB TipcSrvErrorCbLookup(func, arg)
T_IPC_CONN_ERROR_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvGetCurrentConnectedLcn — get the logical connection name of the RTserver to which this RTclient is connected

```
T_BOOL TipcSrvGetCurrentConnectedLcn(lcn_return)
T_STR lcn_return;
```

TipcSrvProcessCbCreate — create a process callback in the connection to RTserver

```
T_CB TipcSrvProcessCbCreate(mt, func, arg)
T_IPC_MT mt;
T_IPC_CONN_PROCESS_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvProcessCbLookup — look up a process callback in the connection to RTserver

```
T_CB TipcSrvProcessCbLookup(mt, func, arg)
T_IPC_MT mt;
T_IPC_CONN_PROCESS_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvQueueCbCreate — create a queue callback in the connection to RTserver

```
T_CB TipcSrvQueueCbCreate(mt, func, arg)
T_IPC_MT mt;
T_IPC_CONN_QUEUE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvQueueCbLookup — look up a queue callback in the connection to RTserver

```
T_CB TipcSrvQueueCbLookup(mt, func, arg)
T_IPC_MT mt;
T_IPC_CONN_QUEUE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvReadCbCreate — create a read callback in the connection to RTserver

```
T_CB TipcSrvReadCbCreate(mt, func, arg)
T_IPC_MT mt;
T_IPC_CONN_READ_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvReadCbLookup — look up a read callback in the connection to RTserver

```
T_CB TipcSrvReadCbLookup(mt, func, arg)
T_IPC_MT mt;
T_IPC_CONN_READ_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvSubjectCbCreate — create a subject callback

```
T_CB TipcSrvSubjectCbCreate(subject, mt, func, arg)
T_STR subject;
T_IPC_MT mt;
T_IPC_SRV_SUBJECT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvSubjectCbDestroyAll— destroy all subject callbacks

```
T_BOOL TipcSrvSubjectCbDestroyAll(void)
```

TipcSrvSubjectCbLookup — lookup a subject callback

```
T_CB TipcSrvSubjectCbLookup(subject, mt, func, arg)
T_STR subject;
T_IPC_MT mt;
T_IPC_SRV_SUBJECT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvSubjectDefaultCbCreate — create a default subject callback

```
T_CB TipcSrvSubjectDefaultCbCreate(func, arg)
T_IPC_SRV_SUBJECT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvSubjectDefaultCbLookup — look up a default subject callback

```
T_CB TipcSrvSubjectDefaultCbLookup(func, arg)
T_IPC_SRV_SUBJECT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvTraverseCbCreate — create a connection names traverse callback

```
T_CB TipcSrvTraverseCbCreate(func, arg)
T_IPC_SRV_TRAVERSE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvTraverseCbLookup — look up a connection names traverse callback

```
T_CB TipcSrvTraverseCbLookup(func, arg)
T_IPC_SRV_TRAVERSE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvWriteCbCreate — create a write callback in the connection to RTserver

```
T_CB TipcSrvWriteCbCreate(mt, func, arg)
T_IPC_MT mt;
T_IPC_CONN_WRITE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvWriteCbLookup — look up a write callback in the connection to RTserver

```
T_CB TipcSrvWriteCbLookup(mt, func, arg)
T_IPC_MT mt;
T_IPC_CONN_WRITE_CB_FUNC func;
T_CB_ARG arg;
```

Subject Functions (TipcSrv*Subject*)

The TipcSrv*Subject* functions are used to work with subjects.

TipcSrvStdSubjectSetSubscribe — start or stop subscribing to all standard subjects

```
T_BOOL TipcSrvStdSubjectSetSubscribe(subscribe_status, time_status)
T_BOOL subscribe_status;
T_BOOL time_status;
```

TipcSrvSubjectGetSubscribe — determine whether an RTclient is subscribing to a subject

```
T_BOOL TipcSrvSubjectGetSubscribe(subject, subscribe_status_return)
T_STR subject;
T_BOOL *subscribe_status_return;
```

TipcSrvSubjectGetSubscribeLb — determine whether an RTclient is subscribing to a subject, including load balancing information

```
T_BOOL TipcSrvSubjectGetSubscribeLb(subject, subscribe_status_return,
lb_status_return)
T_STR subject;
T_BOOL *subscribe_status_return;
T_BOOL *lb_status_return;
```

TipcSrvSubjectGmdInit — initialize GMD accounting in RTserver for a subject to which messages will be published

```
T_BOOL TipcSrvSubjectGmdInit(subject)
T_STR subject;
```

TipcSrvSubjectLbInit — initialize load balancing accounting in RTserver for a subject to which messages will be published

```
T_BOOL TipcSrvSubjectLbInit(subject)
T_STR subject;
```

TipcSrvSubjectSetSubscribe — start or stop subscribing to a subject

```
T_BOOL TipcSrvSubjectSetSubscribe(subject, subscribe_status)
T_STR subject;
T_BOOL subscribe_status;
```

TipcSrvSubjectSetSubscribeEx — start or stop subscribing to a subject using extended options

```
T_BOOL TipcSrvSubjectSetSubscribeEx(subj_name, status, type, flags,
type_specific_struct)
T_STR subj_name;
T_BOOL status;
T_INT4 type;
T_INT4 flags;
T_PTR type_specific_struct;
```

TipcSrvSubjectSetSubscribeLb — start or stop subscribing to a subject, with or without load balancing

```
T_BOOL TipcSrvSubjectSetSubscribeLb(subject, subscribe_status,
lb_status)
T_STR subject;
T_BOOL subscribe_status;
T_BOOL lb_status;
```

TipcSrvSubjectTraverseSubscribe — traverse the subjects that the RTclient is subscribing to

```
T_PTR TipcSrvSubjectTraverseSubscribe(func, arg)
T_IPC_SRV_SUBJECT_TRAV_FUNC func;
T_PTR arg;
```

GMD Functions (TipcSrvGmd*)

The TipcSrvGmd* functions are used to work with the guaranteed message delivery services of SmartSockets when using the publish-subscribe communication mode.

TipcSrvGmdFileCreate — create a GMD area on the connection to RTserver

```
T_BOOL TipcSrvGmdFileCreate()
```

TipcSrvGmdFileDelete — delete GMD files for the connection to RTserver

```
T_BOOL TipcSrvGmdFileDelete()
```

TipcSrvGmdMsgDelete — delete a message from the GMD area after a GMD failure on the connection to RTserver

```
T_BOOL TipcSrvGmdMsgDelete(msg)
T_IPC_MSG msg;
```

TipcSrvGmdMsgServerDelete — delete a message in RTserver after a GMD failure on the connection to RTserver

```
T_BOOL TipcSrvGmdMsgServerDelete(msg)
T_IPC_MSG msg;
```

TipcSrvGmdMsgStatus — poll RTserver for GMD status of a message

```
T_BOOL TipcSrvGmdMsgStatus(msg)
T_IPC_MSG msg;
```

Miscellaneous Functions (TipcSrv*)

These functions do not fit in with any other categories. The miscellaneous functions are used for tasks such as creating a connection to RTserver, destroying a connection to RTserver, checking if an RTserver is running, checking if an RTserver is still alive, and stopping an RTserver.

TipcSrvCheck — check if data can be read from or written to connection to RTserver

```
T_BOOL TipcSrvCheck(check_mode, timeout)
T_IO_CHECK_MODE check_mode;
T_REAL8 timeout;
```

TipcSrvCreate — create the connection to RTserver

```
T_BOOL TipcSrvCreate(create_status)
T_IPC_SRV_CONN_STATUS create_status;
```

TipcSrvDestroy — destroy the connection to RTserver

```
T_BOOL TipcSrvDestroy(destroy_status)
T_IPC_SRV_CONN_STATUS destroy_status;
```

TipcSrvIsRunning — check if the RTclient can create a connection to RTserver

```
T_BOOL TipcSrvIsRunning()
```

TipcSrvKeepAlive — check if the connection to RTserver is still alive

```
T_BOOL TipcSrvKeepAlive()
```

TipcSrvLock — acquire exclusive access to the connection to RTserver

```
T_BOOL TipcSrvLock()
```

TipcSrvLogAddMt — add a message type to a message file logging type

```
T_BOOL TipcSrvLogAddMt(log_type, mt)
T_IPC_SRV_LOG_TYPE log_type;
T_IPC_MT mt;
```

TipcSrvLogRemoveMt — remove a message type from a message file logging type

```
T_BOOL TipcSrvLogRemoveMt(log_type, mt)
T_IPC_SRV_LOG_TYPE log_type;
T_IPC_MT mt;
```

TipcSrvPrint — print all information about the connection to RTserver in a portable, reproducible format

```
T_BOOL TipcSrvPrint(func)
T_OUT_FUNC func;
```

TipcSrvStop — stop RTserver and possibly more processes

```
T_BOOL TipcSrvStop(stop_type)
T_IPC_SRV_STOP_TYPE stop_type;
```

TipcSrvUnlock — release exclusive access to the connection to RTserver

```
T_BOOL TipcSrvUnlock()
```

Property Get Functions (TipcSrvGet*)

The TipcSrvGet* functions are used to retrieve the values of the properties of a connection to RTserver.

TipcSrvGetArch — determine the architecture name of the connected RTserver

```
T_BOOL TipcSrvGetArch(arch_return)
T_STR *arch_return;
```

TipcSrvGetAutoFlushSize — determine the automatic flush size of the connection to RTserver

```
T_BOOL TipcSrvGetAutoFlushSize(auto_flush_size_return)
T_INT4 *auto_flush_size_return;
```

TipcSrvGetBlockMode — determine the block mode of the connection to RTserver

```
T_BOOL TipcSrvGetBlockMode(block_mode_return)
T_BOOL *block_mode_return;
```

TipcSrvGetConnStatus — determine the status of the connection to RTserver

```
T_BOOL TipcSrvGetConnStatus(conn_status_return)
T_IPC_SRV_CONN_STATUS *conn_status_return;
```

TipcSrvGetGmdMaxSize — determine the GMD area maximum size of the connection to RTserver

```
T_BOOL TipcSrvGetGmdMaxSize(gmd_max_size_return)
T_UINT4 *gmd_max_size_return;
```

TipcSrvGetGmdNumPending — determine the number of outgoing GMD messages still pending on the connection to RTserver

```
T_BOOL TipcSrvGetGmdNumPending(gmd_num_pending_return)
T_INT4 *gmd_num_pending_return;
```

TipcSrvGetNumQueued — get the number of queued messages from the connection to RTserver

```
T_BOOL TipcSrvGetNumQueued(num_queued_return)
T_INT4 *num_queued_return;
```

TipcSrvGetSocket — determine the socket of the connection to RTserver

```
T_BOOL TipcSrvGetSocket(socket_return)
T_INT4 *socket_return;
```

TipcSrvGetTimeout — get a timeout property from the connection to RTserver

```
T_BOOL TipcSrvGetTimeout(timeout, value_return)
T_IPC_TIMEOUT timeout;
T_REAL8 *value_return;
```

TipcSrvGetXtSource — get source suitable for XtAppAddInput from the connection to RTserver

```
T_BOOL TipcSrvGetXtSource(source_return)
T_INT4 *source_return;
```

Property Set Functions (TipcSrvSet*)

The TipcSrvSet* functions are used to set the values of the properties of a connection to RTserver.

TipcSrvSetAutoFlushSize — set the automatic flush size of the connection to RTserver

```
T_BOOL TipcSrvSetAutoFlushSize(auto_flush_size)
T_INT4 auto_flush_size;
```

TipcSrvSetBlockMode — set the block mode of the connection to RTserver

```
T_BOOL TipcSrvSetBlockMode(block_mode)
T_BOOL block_mode;
```

TipcSrvSetGmdMaxSize — set the GMD area maximum size of the connection to RTserver

```
T_BOOL TipcSrvSetGmdMaxSize(gmd_max_size)
T_UINT4 gmd_max_size;
```

TipcSrvSetSocket — set the socket of the connection to RTserver

```
T_BOOL TipcSrvSetSocket(socket)
T_INT4 socket;
```

TipcSrvSetTimeout — set a timeout property of the connection to RTserver

```
T_BOOL TipcSrvSetTimeout(timeout, value)
T_IPC_TIMEOUT timeout;
T_REAL8 value;
```

TipcSrvSetUsernamePassword — set the basic credentials of the connection to RTserver

```
T_BOOL TipcSrvSetUsernamePassword(username, password)
T_STR username;
T_STR password;
```

Peer Property Get Functions (TipcSrvGet*)

These TipcSrvGet* functions are used to retrieve the values of the properties of a peer connection to RTserver.

TipcSrvGetNode — determine the node name of the connected RTserver

```
T_BOOL TipcSrvGetNode(node_return)
T_STR *node_return;
```

TipcSrvGetPid — determine the process ID of the connected RTserver

```
T_BOOL TipcSrvGetPid(pid_return)
T_INT4 *pid_return;
```

TipcSrvGetUniqueSubject — determine the unique subject of the connected RTserver

```
T_BOOL TipcSrvGetUniqueSubject(unique_subject_return)
T_STR *unique_subject_return;
```

TipcSrvGetUser — determine the user name of the connected RTserver

```
T_BOOL TipcSrvGetUser(user_return)
T_STR *user_return;
```

Reading and Processing Messages from RTserver (TipcSrv*)

These functions are used to read, process, or search for messages arriving from RTserver.

TipcSrvMainLoop — read and process messages on the connection to RTserver

```
T_BOOL TipcSrvMainLoop(timeout)
T_REAL8 timeout;
```

TipcSrvMsgNext — retrieve the next message from the connection to RTserver

```
T_IPC_MSG TipcSrvMsgNext(timeout)
T_REAL8 timeout;
```

TipcSrvMsgProcess — process a message in the connection to RTserver

```
T_BOOL TipcSrvMsgProcess(msg)
T_IPC_MSG msg;
```

TipcSrvMsgSearch — search the message queue of the connection to RTserver for a specific message

```
T_IPC_MSG TipcSrvMsgSearch(timeout, func, arg)
T_REAL8 timeout;
T_IPC_CONN_MSG_SEARCH_FUNC func;
T_PTR arg;
```

TipcSrvMsgSearchType — search the message queue of a connection to RTserver for a message with a specific type

```
T_IPC_MSG TipcSrvMsgSearchType(timeout, mt)
T_REAL8 timeout;
T_IPC_MT mt;
```

TipcSrvRead — read all available data from the connection to RTserver and queue messages in priority order

```
T_BOOL TipcSrvRead(timeout)
T_REAL8 timeout;
```

Publishing Messages (TipcSrv*)

These functions are used to publish messages using RTserver.

TipcSrvFlush — flush buffered outgoing messages on the connection to RTserver

```
T_BOOL TipcSrvFlush()
```

TipcSrvMsgInsert — insert a message into the queue of the connection to RTserver

```
T_BOOL TipcSrvMsgInsert(msg, pos)
T_IPC_MSG msg;
T_INT4 pos;
```

TipcSrvMsgSend — send a message through the connection to RTserver

```
T_BOOL TipcSrvMsgSend(msg, check_server_msg_send)
T_IPC_MSG msg;
T_BOOL check_server_msg_send;
```

TipcSrvMsgSendRpc — make a remote procedure call (RPC) with messages on the connection to RTserver

```
T_IPC_MSG TipcSrvMsgSendRpc(call_msg, timeout)
T_IPC_MSG call_msg;
T_REAL8 timeout;
```

TipcSrvMsgWrite — construct a message and send it through the connection to RTserver

```
T_BOOL TipcSrvMsgWrite(dest, mt, check_server_msg_send, ...)
T_STR dest;
T_IPC_MT mt;
T_BOOL check_server_msg_send;
```

TipcSrvMsgWriteVa — construct a message and send it through the connection to RTserver (`va_list` version)

```
T_BOOL TipcSrvMsgWriteVa(dest, mt, check_server_msg_send, var_arg_list)
T_STR dest;
T_IPC_MT mt;
T_BOOL check_server_msg_send;
va_list var_arg_list;
```

Miscellaneous IPC Functions

These inter-process communication (IPC) functions do not fit into any of the other categories:

TipcDeliveryModeToStr — convert a message delivery mode to a string

```
T_BOOL TipcDeliveryModeToStr(delivery_mode, delivery_mode_str_return)
T_IPC_DELIVERY_MODE delivery_mode;
T_STR *delivery_mode_str_return;
```

TipcFtToStr — convert a field type to a string

```
T_BOOL TipcFtToStr(type, str_return)
T_IPC_FT type;
T_STR *str_return;
```

TipcInitCommands — create SmartSockets inter-process communication (IPC) commands

```
T_BOOL TipcInitCommands()
```

TipcInitThreads — enable thread-safety on supported platforms

```
T_BOOL TipcInitThreads()
```

TipcLbModeToStr — convert a message load balancing mode to a string

```
T_BOOL TipcLbModeToStr(lb_mode, lb_mode_str_return)
T_IPC_LB_MODE lb_mode;
T_STR *lb_mode_str_return;
```

TipcStrToDeliveryMode — convert a string to a delivery mode

```
T_BOOL TipcStrToDeliveryMode(delivery_mode_str, delivery_mode_return)
T_STR delivery_mode_str;
T_IPC_DELIVERY_MODE *delivery_mode_return;
```

TipcStrToFt — convert a string to a field type

```
T_BOOL TipcStrToFt(ft_str, ft_return)
T_STR ft_str;
T_IPC_FT *ft_return;
```

TipcStrToLbMode — convert a string to a load balancing mode

```
T_BOOL TipcStrToLbMode(lb_mode_str, lb_mode_return)
T_STR lb_mode_str;
T_IPC_LB_MODE *lb_mode_return;
```

TipcSubjectGetUnique — get the unique subject of the process

```
T_STR TipcSubjectGetUnique()
```

TipcSubjectMatch — determine whether two subject names are the same

```
T_BOOL TipcSubjectMatch(subject1, subject2, match_return)
T_STR subject1;
T_STR subject2;
T_BOOL *match_return;
```

TipcSubjectValid — check if a subject name is a valid subject

```
T_BOOL TipcSubjectValid(subject)
T_STR subject;
```

Chapter 3 Summary of Advanced API

This chapter is a summary of the advanced TIBCO SmartSockets API. It is designed for both novice and experienced SmartSockets developers. If you are a novice to TIBCO SmartSockets, you can scan the entire chapter to gain an understanding of the advanced options available to you in the SmartSockets API. If you are more experienced, you can use this chapter to quickly find a function to perform a particular task or to check the arguments of a function.

For each advanced function available in the API, this chapter lists a short description of what the function does and a synopsis of its arguments and return values. For more details on these functions and examples, see the *TIBCO SmartSockets Application Programming Interface*. For a summary of the basic SmartSockets API, see Chapter 2, Summary of Basic API.

Topics

Credentials Functions (TipcSrv*Credentials)

TipcSrv*Credentials functions are used by the Security Manager to authenticate a connection to RTserver. The Security Manager is available through TIBCO Professional Services Group.

TipcSrvConnSetCredentials — set the credentials of the connection to RTserver when using multiple connections

```
T_BOOL TipcSrvConnSetCredentials(srv, auth_policy_id, auth_data,
auth_data_len)
T_IPC_SRV srv;
T_INT4 auth_policy_id;
T_PTR auth_data;
T_INT4 auth_data_len;
```

TipcSrvSetCredentials — set the credentials of the connection to RTserver

```
T_BOOL TipcSrvSetCredentials(auth_policy_id, auth_data, auth_data_len)
T_INT4 auth_policy_id;
T_PTR auth_data;
T_INT4 auth_data_len;
```

Dispatcher Functions (TipcDispatcher*)

TipcDispatcher* functions allow the managing messages from multiple RTserver connections with a single call. This is especially useful in single threaded RTclients. By adding one or more RTserver connections to a dispatcher to using TipcDispatcherSrvAdd, a single TipcDispatcherMainLoop call is equivalent to calling TipcSrvConnMainLoop on each of the RTserver connections simultaneously.

The dispatcher not only manages messages arriving on RTserver connections, but it can also manage events.

TipcDispatcherCreate — create a dispatcher

```
T_IPC_DISPATCHER TipcDispatcherCreate()
```

TipcDispatcherCreateDetached — create a dispatcher in a detached thread

```
T_IPC_DISPATCHER TipcDispatcherCreateDetached()
```

TipcDispatcherDestroy — destroy a dispatcher

```
T_BOOL TipcDispatcherDestroy(dispatcher)
T_IPC_DISPATCHER dispatcher;
```

TipcDispatcherDispatch — dispatch triggered events within *timeout* seconds and exit

```
T_BOOL TipcDispatcherDispatch(dispatcher, timeout)
T_IPC_DISPATCHER dispatcher;
T_REAL8 timeout;
```

TipcDispatcherMainLoop — dispatch events continuously for *timeout* seconds

```
T_BOOL TipcDispatcherMainLoop(dispatcher, timeout)
T_IPC_DISPATCHER dispatcher;
T_REAL8 timeout;
```

TipcDispatcherSrvAdd — add RTserver connection to a dispatcher

```
T_BOOL TipcDispatcherSrvAdd(dispatcher, srv)
T_IPC_DISPATCHER dispatcher;
T_IPC_SRV srv;
```

TipcDispatcherSrvRemove — remove RTserver connection from a dispatcher

```
T_BOOL TipcDispatcherSrvRemove(dispatcher, srv)
T_IPC_DISPATCHER dispatcher;
T_IPC_SRV srv;
```

Event Functions (TipcEvent*)

TipcEvent* functions provide an asynchronous programming model. A dispatcher can manage these types of events:

- Connection events — triggered by read or write availability on a connection. Connection events are used to bypass the RTserver when sending or receiving messages between RTclients.
- Message events — triggered when a message arrives on an RTserver connection that matches the event's criteria, such as the message's subject or type. Multiple dispatchers can manage message events, with each running in its own thread. This allows for a high degree of concurrency in processing messages.
- Socket events — triggered by read or write availability on a socket. Socket events are typically used when integrating another vendor's software into a SmartSockets main loop. The vendor's software must be socket-based.
- Timer events — triggered when a timer's time interval has elapsed. Timer events repeat but can be destroyed the first time they are invoked to simulate one-shot events.
- User events — are executed immediately without requiring a trigger. User events are useful for inter-thread communication. By running dispatchers in multiple threads, user events can be sent between the dispatchers in a thread-safe manner.

TipcEventCreate — create a user event

```
T_IPC_EVENT TipcEventCreate(dispatcher, data, event_func, event_arg)
T_IPC_DISPATCHER dispatcher;
T_PTR data;
T_IPC_EVENT_FUNC event_func;
T_PTR event_arg;
```

TipcEventCreateConn — create a connection event

```
T_IPC_EVENT TipcEventCreateConn(dispatcher, conn, check_mode,
event_func, event_arg)
T_IPC_DISPATCHER dispatcher;
T_IPC_CONN conn;
T_IO_CHECK_MODE check_mode;
T_IPC_EVENT_FUNC event_func;
T_PTR event_arg;
```

TipcEventCreateMsg — create a message event triggered by a message's subject

```
T_IPC_EVENT TipcEventCreateMsg(dispatcher, srv, subject, event_func,
event_arg)
T_IPC_DISPATCHER dispatcher;
T_IPC_SRV srv;
T_STR subject;
T_IPC_EVENT_FUNC event_func;
T_PTR event_arg;
```

TipcEventCreateMsgType — create a message event triggered by a message's type

```
T_IPC_EVENT TipcEventCreateMsgType(dispatcher, srv, mt, event_func,
event_arg)
T_IPC_DISPATCHER dispatcher;
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_EVENT_FUNC event_func;
T_PTR event_arg;
```

TipcEventCreateSocket — create a socket event

```
T_IPC_EVENT TipcEventCreateSocket(dispatcher, socket, check_mode,
event_func, event_arg)
T_IPC_DISPATCHER dispatcher;
T_INT4 socket;
T_IO_CHECK_MODE check_mode;
T_IPC_EVENT_FUNC event_func;
T_PTR event_arg;
```

TipcEventCreateTimer — create a timer event

```
T_IPC_EVENT TipcEventCreateTimer(dispatcher, interval, event_func,
dispatch_arg)
T_IPC_DISPATCHER dispatcher;
T_REAL8 interval;
T_IPC_EVENT_FUNC event_func;
T_PTR event_arg;
```

TipcEventDestroy — destroy an event of any kind

```
T_BOOL TipcEventDestroy(event)
T_IPC_EVENT event;
```

TipcEventGetCheckMode — get the check mode value for a connection or socket event

```
T_BOOL TipcEventGetCheckMode(event, check_mode_return)
T_IPC_EVENT event;
T_IO_CHECK_MODE *check_mode_return;
```

TipcEventGetConn — get the connection information for a connection event

```
T_BOOL TipcEventGetConn(event, conn_return)
T_IPC_EVENT event;
T_IPC_CONN *conn_return;
```

TipcEventGetData — get the data of a user event

```
T_BOOL TipcEventGetData(event, data_return)
T_IPC_EVENT event;
T_PTR *data_return;
```

TipcEventGetDispatcher — get the identifier of a dispatcher for an event of any kind

```
T_BOOL TipcEventGetDispatcher(event, dispatcher_return)
T_IPC_EVENT event;
T_IPC_DISPATCHER *dispatcher_return;
```

TipcEventGetInterval — get the interval value of a timer event

```
T_BOOL TipcEventGetInterval(event, interval_return)
T_IPC_EVENT event;
T_REAL8 *interval_return;
```

TipcEventGetSocket — get the socket information for a socket event

```
T_BOOL TipcEventGetSocket(event, socket_return)
T_IPC_EVENT event;
T_INT4 *socket_return;
```

TipcEventGetType — get an event's type (connection, message, socket, timer, or user)

```
T_BOOL TipcEventGetType(event, type_return)
T_IPC_EVENT event;
T_IPC_EVENT_TYPE *type_return;
```

TipcEventSetInterval — set the time interval for a timer event

```
T_BOOL TipcEventSetInterval(event, interval)
T_IPC_EVENT event;
T_REAL8 interval;
```

Multiple Connection Functions (TipcSrvConn*)

The TipcSrvConn* functions are used for communication between a client program (RTclient) and RTserver. These functions enable the publish-subscribe message delivery services of SmartSockets.

Callback Functions (TipcSrvConn*Cb*)

TipcSrvConnCloseCbCreate — create a server close callback

```
T_CB TipcSrvConnCloseCbCreate(srv, func, arg)
T_IPC_SRV srv;
T_IPC_SRV_DESTROY_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnCloseCbLookup — look up a server close callback

```
T_CB TipcSrvConnCloseCbLookup(srv, func, arg)
T_IPC_SRV srv;
T_IPC_SRV_DESTROY_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnDefaultCbCreate — create a default callback in the connection

```
T_CB TipcSrvConnDefaultCbCreate(srv, func, arg)
T_IPC_SRV srv;
T_IPC_CONN_DEFAULT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnDefaultCbLookup — look up a default callback in the connection

```
T_CB TipcSrvConnDefaultCbLookup(srv, func, arg)
T_IPC_SRV srv;
T_IPC_CONN_DEFAULT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnErrorCbCreate — create an error callback in the connection

```
T_CB TipcSrvConnErrorCbCreate(srv, func, arg)
T_IPC_SRV srv;
T_IPC_CONN_ERROR_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnErrorCbLookup — look up an error callback in the connection

```
T_CB TipcSrvConnErrorCbLookup(srv, func, arg)
T_IPC_SRV srv;
T_IPC_CONN_ERROR_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnGetCurrentConnectedLcn — get the logical connection name of the RTserver to which this RTclient is connected

```
T_BOOL TipcSrvConnGetCurrentConnectedLcn(srv, lcn_return)
T_IPC_SRV srv;
T_STR lcn_return;
```

TipcSrvConnOpenCbCreate — create a server open callback

```
T_CB TipcSrvConnOpenCbCreate(srv, func, arg)
T_IPC_SRV srv;
T_IPC_SRV_CREATE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnOpenCbLookup — look up a server open callback

```
T_CB TipcSrvConnOpenCbLookup(srv, func, arg)
T_IPC_SRV srv;
T_IPC_SRV_CREATE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnProcessCbCreate — create a process callback in the connection

```
T_CB TipcSrvConnProcessCbCreate(srv, mt, func, arg)
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_CONN_PROCESS_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnProcessCbLookup — look up a process callback in the connection

```
T_CB TipcSrvConnProcessCbLookup(srv, mt, func, arg)
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_CONN_PROCESS_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnQueueCbCreate — create a queue callback in the connection

```
T_CB TipcSrvConnQueueCbCreate(srv, mt, func, arg)
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_CONN_QUEUE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnQueueCbLookup — look up a queue callback in the connection

```
T_CB TipcSrvConnQueueCbLookup(srv, mt, func, arg)
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_CONN_QUEUE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnReadCbCreate — create a read callback in the connection

```
T_CB TipcSrvConnReadCbCreate(srv, mt, func, arg)
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_CONN_READ_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnReadCbLookup — look up a read callback in the connection

```
T_CB TipcSrvConnReadCbLookup(srv, mt, func, arg)
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_CONN_READ_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnSubjectCbCreate — create a subject callback

```
T_CB TipcSrvConnSubjectCbCreate(srv, subject, mt, func, arg)
T_IPC_SRV srv;
T_STR subject;
T_IPC_MT mt;
T_IPC_SRV_SUBJECT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnSubjectCbDestroyAll— destroy all subject callbacks

```
T_BOOL TipcSrvSubjectCbDestroyAll(srv)
T_IPC_SRV srv;
```

TipcSrvConnSubjectCbLookup — lookup a subject callback

```
T_CB TipcSrvConnSubjectCbLookup(srv, subject, mt, func, arg)
T_IPC_SRV srv;
T_STR subject;
T_IPC_MT mt;
T_IPC_SRV_SUBJECT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnSubjectDefaultCbCreate — create a default subject callback

```
T_CB TipcSrvConnSubjectDefaultCbCreate(srv, func, arg)
T_IPC_SRV srv;
T_IPC_SRV_SUBJECT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnSubjectDefaultCbLookup — look up a default subject callback

```
T_CB TipcSrvConnSubjectDefaultCbLookup(srv, func, arg)
T_IPC_SRV srv;
T_IPC_SRV_SUBJECT_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnWriteCbCreate — create a write callback in the connection

```
T_CB TipcSrvConnWriteCbCreate(srv, mt, func, arg)
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_CONN_WRITE_CB_FUNC func;
T_CB_ARG arg;
```

TipcSrvConnWriteCbLookup — look up a write callback in the connection

```
T_CB TipcSrvConnWriteCbLookup(srv, mt, func, arg)
T_IPC_SRV srv;
T_IPC_MT mt;
T_IPC_CONN_WRITE_CB_FUNC func;
T_CB_ARG arg;
```

Subject Functions (TipcSrvConn*Subject*)

The TipcSrvConn*Subject* functions are used to work with subjects.

TipcSrvConnStdSubjectSetSubscribe — start or stop subscribing to all standard subjects

```
T_BOOL TipcSrvConnStdSubjectSetSubscribe(srv, subscribe_status)
T_IPC_SRV srv;
T_BOOL subscribe_status;
```

TipcSrvConnSubjectGetSubscribe — determine whether an RTclient is subscribing to a subject

```
T_BOOL TipcSrvConnSubjectGetSubscribe(srv, subject,
subscribe_status_return)
T_IPC_SRV srv;
T_STR subject;
T_BOOL *subscribe_status_return;
```

TipcSrvConnSubjectGetSubscribeLb — determine whether an RTclient is subscribing to a subject, including load balancing information

```
T_BOOL TipcSrvConnSubjectGetSubscribeLb(srv, subject,
subscribe_status_return, lb_status_return)
T_IPC_SRV srv;
T_STR subject;
T_BOOL *subscribe_status_return;
T_BOOL *lb_status_return;
```

TipcSrvConnSubjectGmdInit — initialize GMD accounting in RTserver for a subject to which messages will be published

```
T_BOOL TipcSrvConnSubjectGmdInit(srv, subject)
T_IPC_SRV srv;
T_STR subject;
```

TipcSrvConnSubjectLbInit — initialize load balancing accounting in RTserver for a subject to which messages will be published

```
T_BOOL TipcSrvConnSubjectLbInit(srv, subject)
T_IPC_SRV srv;
T_STR subject;
```

TipcSrvConnSubjectSetSubscribe — start or stop subscribing to a subject

```
T_BOOL TipcSrvConnSubjectSetSubscribe(srv, subject,
subscribe_status)
T_IPC_SRV srv;
T_STR subject;
T_BOOL subscribe_status;
```

TipcSrvConnSubjectSetSubscribeEx — start or stop subscribing to a subject using extended options

```
T_BOOL TipcSrvConnSubjectSetSubscribeEx(srv, subj_name, status,
type, flags, type_specific_struct)
T_IPC_SRV srv;
T_STR subj_name;
T_BOOL status;
T_INT4 type;
T_INT4 flags;
T_PTR type_specific_struct;
```

TipcSrvConnSubjectSetSubscribeLb — start or stop subscribing to a subject, with or without load balancing

```
T_BOOL TipcSrvConnSubjectSetSubscribeLb(srv, subject,
subscribe_status, lb_status)
T_IPC_SRV srv;
T_STR subject;
T_BOOL subscribe_status;
T_BOOL lb_status;
```

TipcSrvConnSubjectTraverseSubscribe — traverse the subjects that the RTclient is subscribing to

```
T_PTR TipcSrvConnSubjectTraverseSubscribe(srv, func, arg)
T_IPC_SRV srv;
T_IPC_SRV_SUBJECT_TRAV_FUNC func;
T_PTR arg;
```

GMD Functions (TipcSrvConnGmd*)

The TipcSrvConnGmd* functions are used to work with the guaranteed message delivery services of SmartSockets when using the publish-subscribe communication mode.

TipcSrvConnGmdFileCreate — create a GMD area on the connection

```
T_BOOL TipcSrvConnGmdFileCreate(srv)
T_IPC_SRV srv;
```

TipcSrvConnGmdFileDelete — delete GMD files for the connection

```
T_BOOL TipcSrvConnGmdFileDelete(srv)
T_IPC_SRV srv;
```

TipcSrvConnGmdMsgDelete — delete a message from the GMD area after a GMD failure

```
T_BOOL TipcSrvConnGmdMsgDelete(srv, msg)
T_IPC_SRV srv;
T_IPC_MSG msg;
```

TipcSrvConnGmdMsgServerDelete — delete a message in RTserver after a GMD failure

```
T_BOOL TipcSrvConnGmdMsgServerDelete(srv, msg)
T_IPC_SRV srv;
T_IPC_MSG msg;
```

TipcSrvConnGmdMsgStatus — poll RTserver for GMD status of a message

```
T_BOOL TipcSrvConnGmdMsgStatus(srv, msg)
T_IPC_SRV srv;
T_IPC_MSG msg;
```

Miscellaneous Functions (TipcSrvConn*)

These functions do not fit with any of the other categories. The miscellaneous functions are used for tasks such as creating a connection to RTserver, destroying a connection to RTserver, checking if an RTserver is running, checking if an RTserver is still alive, and stopping an RTserver.

TipcSrvConnCheck — check if data can be read from or written to connection

```
T_BOOL TipcSrvConnCheck(srv, check_mode, timeout)
T_IPC_SRV srv;
T_IO_CHECK_MODE check_mode;
T_REAL8 timeout;
```

TipcSrvConnClose — close a physical connection to RTserver

```
T_BOOL TipcSrvConnClose(srv, conn_status)
T_IPC_SRV srv;
T_IPC_SRV_CONN_STATUS conn_status;
```

TipcSrvConnCreate — create an RTclient connection handle to an RTserver

```
T_IPC_SRV TipcSrvConnCreate(unique_subject, project, server_names,
default_subject_prefix)
T_STR unique_subject;
T_STR project;
T_STR server_names;
T_STR default_subject_prefix;
```

TipcSrvConnCreateNamed — create a named RTclient connection handle to an RTserver

```
T_IPC_SRV TipcSrvConnCreateNamed(name)
T_STR name;
```

TipcSrvConnDestroy — free resources associated with the RTclient / RTserver connection handle

```
T_BOOL TipcSrvConnDestroy(srv)
T_IPC_SRV srv;
```

TipcSrvConnKeepAlive — check if the connection to RTserver is still alive

```
T_BOOL TipcSrvConnKeepAlive(srv)
T_IPC_SRV srv;
```

TipcSrvConnLock — acquire exclusive access to the connection

```
T_BOOL TipcSrvConnLock(srv)
T_IPC_SRV srv;
```

TipcSrvConnOpen — open a connection to RTserver

```
T_BOOL TipcSrvConnOpen(srv, create_status)
T_IPC_SRV srv;
T_IPC_SRV_CONN_STATUS create_status;
```

TipcSrvConnPrint — print all information about the connection in a portable, reproducible format

```
T_BOOL TipcSrvConnPrint(srv, func)
T_IPC_SRV srv;
T_OUT_FUNC func;
```

TipcSrvConnUnlock — release exclusive access to the connection

```
T_BOOL TipcSrvConnUnlock(srv)
T_IPC_SRV srv;
```

Property Get Functions (TipcSrvConnGet*, TipcSrv*Get*, and TipcSrvConnSubjectGet*)

The TipcSrvConnGet*, TipcSrv*Get*, and TipcSrvConnSubjectGet* functions are used to retrieve the properties of a connection to an RTserver.

TipcSrvBufferGetReadSize — get the total number of bytes in a connection's read buffer to an RTserver

```
T_BOOL TipcSrvBufferGetReadSize(read_size_return)
T_INT4 *read_size_return;
```

TipcSrvBufferGetWriteSize — get the total number of bytes in a connection's write buffer to an RTserver

```
T_BOOL TipcSrvBufferGetWriteSize(write_size_return)
T_INT4 *write_size_return;
```

TipcSrvConnBufferGetReadSize — get the total number of bytes in a connection's read buffer to an RTserver

```
T_BOOL TipcSrvConnBufferGetReadSize(srv, read_size_return)
T_IPC_SRV srv;
T_INT4 *read_size_return;
```

TipcSrvConnBufferGetWriteSize — get the total number of bytes in a connection's write buffer to an RTserver

```
T_BOOL TipcSrvConnBufferGetWriteSize(conn, write_size_return)
T_IPC_SRV srv;
T_INT4 *write_size_return;
```

TipcSrvConnGetAutoFlushSize — determine the automatic flush size of the connection

```
T_BOOL TipcSrvConnGetAutoFlushSize(srv, auto_flush_size_return)
T_IPC_SRV srv;
T_INT4 *auto_flush_size_return;
```

TipcSrvConnGetConnStatus — determine the status of the connection

```
T_BOOL TipcSrvConnGetConnStatus(srv, status_return)
T_IPC_SRV srv;
T_IPC_SRV_CONN_STATUS *status_return;
```

TipcSrvConnGetGmdDirName — get the GMD area sub-directory name of the connection to RTserver

```
T_BOOL TipcSrvConnGetGmdDirName(srv, gmd_dir_name_return)
T_IPC_SRV srv;
T_STR *gmd_dir_name_return;
```

TipcSrvConnGetGmdMaxSize — determine the GMD area maximum size of the connection

```
T_BOOL TipcSrvConnGetGmdMaxSize(srv, gmd_max_size_return)
T_IPC_SRV srv;
T_UINT4 *gmd_max_size_return;
```

TipcSrvConnGetGmdNumPending — determine the number of outgoing GMD messages still pending on the connection

```
T_BOOL TipcSrvConnGetGmdNumPending(srv, gmd_num_pending_return)
T_IPC_SRV srv;
T_INT4 *gmd_num_pending_return;
```

TipcSrvConnGetNumQueued — get number of queued messages from the connection

```
T_BOOL TipcSrvConnGetNumQueued(srv, num_queued_return)
T_IPC_SRV srv;
T_INT4 *num_queued_return;
```

TipcSrvConnGetProject — get the project of the connection to RTserver

```
T_BOOL TipcSrvConnGetProject(srv, project_return)
T_IPC_SRV srv;
T_STR *project_return;
```

TipcSrvConnGetServerNamesList — get the server names of the connection to RTserver

```
T_BOOL TipcSrvConnGetServerNamesList(srv, server_names_list_return)
T_IPC_SRV srv;
T_STR_LIST *server_names_list_return;
```

TipcSrvConnGetSocket — determine the socket of the connection

```
T_BOOL TipcSrvConnGetSocket(srv, socket_return)
T_IPC_SRV srv;
T_INT4 *socket_return;
```

TipcSrvConnGetTimeout — get a timeout property from the connection

```
T_BOOL TipcSrvConnGetTimeout(srv, timeout, value_return)
T_IPC_SRV srv;
T_IPC_TIMEOUT timeout;
T_REAL8 *value_return;
```

TipcSrvConnGetXtSource — get source for XtAppAddInput from the connection

```
T_BOOL TipcSrvConnGetXtSource(srv, source_return)
T_IPC_SRV srv;
T_INT4 *source_return;
```

TipcSrvConnSubjectGetDefaultPrefix — get the default subject prefix of the connection to RTserver

```
T_BOOL TipcSrvConnSubjectGetDefaultPrefix(srv,
default_prefix_return)
T_IPC_SRV srv;
T_STR *default_prefix_return;
```

TipcSrvConnSubjectGetUnique — get the unique subject of the connection to RTserver

```
T_BOOL TipcSrvConnGetUniqueSubject(srv, unique_subject_return)
T_IPC_SRV srv;
T_STR *unique_subject_return;
```

TipcSrvConnTrafficGetBytesRecv8 — get the total number of bytes received on a connection to an RTserver

```
T_BOOL TipcSrvConnTrafficGetBytesRecv8(srv, bytes_recv_return)
T_IPC_SRV srv;
T_INT8 *bytes_recv_return;
```

TipcSrvConnTrafficGetBytesSent8 — get the total number of bytes sent on a connection to an RTserver

```
T_BOOL TipcSrvConnTrafficGetBytesSent8(srv, bytes_sent_return)
T_IPC_SRV srv;
T_INT8 *bytes_sent_return;
```

TipcSrvConnTrafficGetMsgsRecv8 — get the total number of messages received on a connection to an RTserver

```
T_BOOL TipcSrvConnTrafficGetMsgsRecv8(srv, msgs_recv_return)
T_IPC_SRV srv;
T_INT8 *msgs_recv_return;
```

TipcSrvConnTrafficGetMsgsSent8 — get the total number of messages sent on a connection to an RTserver

```
T_BOOL TipcSrvConnTrafficGetMsgsSent8(srv, msgs_sent_return)
T_IPC_SRV srv;
T_INT8 *msgs_sent_return;
```

The previous functions supersede their older counterparts. Do not use the following older functions; we have retained them only for backward source compatibility.

TipcSrvConnTrafficGetBytesRecv — get the total number of bytes received on a connection to an RTserver

```
T_BOOL TipcSrvConnTrafficGetBytesRecv(srv, bytes_recv_return)
T_IPC_SRV srv;
T_INT4 *bytes_recv_return;
```

TipcSrvConnTrafficGetBytesSent — get the total number of bytes sent on a connection to an RTserver

```
T_BOOL TipcSrvConnTrafficGetBytesSent(srv, bytes_sent_return)
T_IPC_SRV srv;
T_INT4 *bytes_sent_return;
```

TipcSrvConnTrafficGetMsgsRecv — get the total number of messages received on a connection to an RTserver

```
T_BOOL TipcSrvConnTrafficGetMsgsRecv(srv, msgs_recv_return)
T_IPC_SRV srv;
T_INT4 *msgs_recv_return;
```

TipcSrvConnTrafficGetMsgsSent — get the total number of messages sent on a connection to an RTserver

```
T_BOOL TipcSrvConnTrafficGetMsgsSent(srv, msgs_sent_return)
T_IPC_SRV srv;
T_INT4 *msgs_sent_return;
```

TipcSrvTrafficGetBytesRecv8 — get the total number of bytes received on a connection to an RTserver

```
T_BOOL TipcSrvTrafficGetBytesRecv8(bytes_recv_return)
T_INT8 *bytes_recv_return;
```

TipcSrvTrafficGetBytesSent8 — get the total number of bytes sent on a connection to an RTserver

```
T_BOOL TipcSrvTrafficGetBytesSent8(bytes_sent_return)
T_INT8 *bytes_sent_return;
```

TipcSrvTrafficGetMsgsRecv8 — get the total number of messages received on a connection to an RTserver

```
T_BOOL TipcSrvTrafficGetMsgsRecv8(msgs_recv_return)
T_INT8 *msgs_recv_return;
```

TipcSrvTrafficGetMsgsSent8 — get the total number of messages sent on a connection to an RTserver

```
T_BOOL TipcSrvTrafficGetMsgsSent8(msgs_sent_return)
T_INT8 *msgs_sent_return;
```

The previous functions supersede their older counterparts. Do not use the following older functions; we have retained them only for backward source compatibility.

TipcSrvTrafficGetBytesRecv — get the total number of bytes received on a connection to an RTserver

```
T_BOOL TipcSrvTrafficGetBytesRecv(bytes_recv_return)
T_INT4 *bytes_recv_return;
```

TipcSrvTrafficGetBytesSent — get the total number of bytes sent on a connection to an RTserver

```
T_BOOL TipcSrvTrafficGetBytesSent(bytes_sent_return)
T_INT4 *bytes_sent_return;
```

TipcSrvTrafficGetMsgsRecv — get the total number of messages received on a connection to an RTserver

```
T_BOOL TipcSrvTrafficGetMsgsRecv(msgs_recv_return)
T_INT4 *msgs_recv_return;
```

TipcSrvTrafficGetMsgsSent — get the total number of messages sent on a connection to an RTserver

```
T_BOOL TipcSrvTrafficGetMsgsSent(msgs_sent_return)
T_INT4 *msgs_sent_return;
```

Property Set Functions (TipcSrvConnSet* and TipcSrvConnSubjectSet*)

The TipcSrvConnSet* and TipcSrvConnSubjectSet* functions are used to set the values of the properties of a connection to RTserver.

TipcSrvConnSetAutoFlushSize — set the automatic flush size of the connection

```
T_BOOL TipcSrvConnSetAutoFlushSize(srv, auto_flush_size)
T_IPC_SRV srv;
T_INT4 auto_flush_size;
```

TipcSrvConnSetGmdDirName — set the GMD area sub-directory name of the connection to RTserver

```
T_BOOL TipcSrvConnSetGmdDirName(srv, gmd_dir_name)
T_IPC_SRV srv;
T_STR gmd_dir_name;
```

TipcSrvConnSetGmdMaxSize — set the GMD area maximum size of the connection

```
T_BOOL TipcSrvConnSetGmdMaxSize(srv, gmd_max_size)
T_IPC_SRV srv;
T_UINT4 gmd_max_size;
```

TipcSrvConnSetProject — set the project of the connection to RTserver

```
T_BOOL TipcSrvConnSetProject(srv, project)
T_IPC_SRV srv;
T_STR project;
```

TipcSrvConnSetServerNames — set the server names of a connection to RTserver

```
T_BOOL TipcSrvConnSetServerNames(srv, server_names)
T_IPC_SRV srv;
T_STR server_names;
```

TipcSrvConnSetServerNamesList — set the server names of the connection to RTserver in a linked list of strings

```
T_BOOL TipcSrvConnSetServerNamesList(srv, server_names_list)
T_IPC_SRV srv;
T_STR_LIST server_names_list;
```

TipcSrvConnSetTimeout — set a timeout property of the connection

```
T_BOOL TipcSrvConnSetTimeout(srv, timeout, value)
T_IPC_SRV srv;
T_IPC_TIMEOUT timeout;
T_REAL8 value;
```

TipcSrvConnSetUsernamePassword — set the basic credentials of the connection to RTserver

```
T_BOOL TipcSrvConnSetUsernamePassword(srv, username, password)
T_IPC_SRV srv;
T_STR username;
T_STR password;
```

TipcSrvConnSubjectSetDefaultPrefix — set the connection's default subject prefix

```
T_BOOL TipcSrvConnSubjectSetDefaultPrefix(srv, default_prefix)
T_IPC_SRV srv;
T_STR default_prefix;
```

TipcSrvConnSubjectSetUnique — set the unique subject of the connection to RTserver

```
T_BOOL TipcSrvConnSubjectSetUnique(srv, unique_subject)
T_IPC_SRV srv;
T_STR unique_subject;
```

Peer Property Functions (TipcSrvConnGet*)

These TipcSrvConnGet* functions are used to retrieve the values of the properties of a peer connection to RTserver.

TipcSrvConnGetArch — determine the architecture name of the connected RTserver

```
T_BOOL TipcSrvConnGetArch(srv, arch_return)
T_IPC_SRV srv;
T_STR *arch_return;
```

TipcSrvConnGetNode — determine the node name of the connected RTserver

```
T_BOOL TipcSrvConnGetNode(srv, node_return)
T_IPC_SRV srv;
T_STR *node_return;
```

TipcSrvConnGetPid — determine the process ID of the connected RTserver

```
T_BOOL TipcSrvConnGetPid(srv, pid_return)
T_IPC_SRV srv;
T_INT4 *pid_return;
```

TipcSrvConnGetUniqueSubject — get the unique subject for the peer connection

```
T_BOOL TipcSrvConnGetUniqueSubject(srv, unique_subject)
T_IPC_SRV srv;
T_STR *unique_subject;
```

TipcSrvConnGetUser — determine the user name of the connected RTserver

```
T_BOOL TipcSrvConnGetUser(srv, user_return)
T_IPC_SRV srv;
T_STR *user_return;
```

Reading and Processing Messages from RTserver (TipcSrvConn*)

These functions are used to read, process, or search for messages arriving from RTserver.

TipcSrvConnMainLoop — read and process messages on the connection

```
T_BOOL TipcSrvConnMainLoop(srv, timeout)
T_IPC_SRV srv;
T_REAL8 timeout;
```

TipcSrvConnMsgNext — retrieve the next message from the connection

```
T_IPC_MSG TipcSrvConnMsgNext(srv, timeout)
T_IPC_SRV srv;
T_REAL8 timeout;
```

TipcSrvConnMsgProcess — process a message in the connection to RTserver

```
T_BOOL TipcSrvConnMsgProcess(srv, msg)
T_IPC_SRV srv;
T_IPC_MSG msg;
```

TipcSrvConnMsgSearch — search the message queue of the connection for a specific message

```
T_IPC_MSG TipcSrvConnMsgSearch(srv, timeout, func, arg)
T_IPC_SRV srv;
T_REAL8 timeout;
T_IPC_CONN_MSG_SEARCH_FUNC func;
T_PTR arg;
```

TipcSrvConnMsgSearchType — search the message queue of a connection for a message with a specific type

```
T_IPC_MSG TipcSrvConnMsgSearchType(srv, timeout, mt)
T_IPC_SRV srv;
T_REAL8 timeout;
T_IPC_MT mt;
```

TipcSrvConnRead — read all available data from the connection and queue messages in priority order

```
T_BOOL TipcSrvConnRead(srv, timeout)
T_IPC_SRV srv;
T_REAL8 timeout;
```

Publishing Messages (TipcSrvConn*)

These functions are used to publish messages using RTserver.

TipcSrvConnFlush — flush buffered outgoing messages on the connection

```
T_BOOL TipcSrvConnFlush(srv)
T_IPC_SRV srv;
```

TipcSrvConnMsgInsert — insert a message into the queue of the connection

```
T_BOOL TipcSrvConnMsgInsert(srv, msg, pos)
T_IPC_SRV srv;
T_IPC_MSG msg;
T_INT4 pos;
```

TipcSrvConnMsgSend — send a message through the connection

```
T_BOOL TipcSrvConnMsgSend(srv, msg, check_server_msg_send)
T_IPC_SRV srv;
T_IPC_MSG msg;
T_BOOL check_server_msg_send;
```

TipcSrvConnMsgSendRpc — make a remote procedure call (RPC) with messages on the connection

```
T_IPC_MSG TipcSrvConnMsgSendRpc(srv, call_msg, timeout)
T_IPC_SRV srv;
T_IPC_MSG call_msg;
T_REAL8 timeout;
```

TipcSrvConnMsgWrite — construct a message and send it through the connection

```
T_BOOL TipcSrvConnMsgWrite(srv, dest, mt, check_server_msg_send,
...)
T_IPC_SRV srv;
T_STR dest;
T_IPC_MT mt;
T_BOOL check_server_msg_send;
```

TipcSrvConnMsgWriteVa — construct a message and send it through the connection (`va_list` version)

```
T_BOOL TipcSrvConnMsgWriteVa(srv, dest, mt, check_server_msg_send,
var_arg_list)
T_IPC_SRV srv;
T_STR dest;
T_IPC_MT mt;
T_BOOL check_server_msg_send;
va_list var_arg_list;
```

Multiple Connection Monitoring Functions (TipcSrvMon*)

There are two types of monitoring data that SmartSockets supports when monitoring your distributed application that uses multiple connections:

- data generated by SmartSockets, such as project names, RTclient names, RTserver names, subjects, message queues, message types, option values, and so on. This type of data is retrieved by the TipcSrvMon* functions.

  You can also use these functions to locate processes on the network. For example, you can find all the RTclients subscribed to subject `/system/thermal`. In general, the TipcSrvMon* functions use the RTserver to retrieve the monitoring information.

- data generated by an RTclient, referred to as extension data. Extension data is retrieved by another RTclient with the TipcSrvMonClientExtPoll function. TipcSrvMonExt* functions are used to define and update the extension data in an RTclient's MON_CLIENT_EXT_POLL_RESULT message before the data is retrieved. The RTserver is not involved in creating extension data.

In general, two types of monitoring are supported:

- Poll — synchronous monitoring, which retrieves a snapshot of information and can be set up to poll at regular intervals
- Watch — asynchronous monitoring, which notifies you when something changes

Polling Functions (TipcSrvMon*Poll)

The TipcSrvMon*Poll functions are used to do a one-time poll for information about various aspects of a SmartSockets project or to retrieve RTclient information, referred to as extension data.

TipcSrvMonClientBufferPoll — poll once for RTclient message-related buffer information

```
T_BOOL TipcSrvMonClientBufferPoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientCbPoll — poll once for RTclient callback information

```
T_BOOL TipcSrvMonClientCbPoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientCpuPoll — poll for the percentage of CPU time used by an RTclient

```
T_BOOL TipcSrvMonClientCpuPoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientExtPoll — poll for extension data of an RTclient

```
T_BOOL TipcSrvMonClientExtPoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientGeneralPoll — poll once for RTclient general information

```
T_BOOL TipcSrvMonClientGeneralPoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientInfoPoll — poll for RTclient information

```
T_BOOL TipcSrvMonClientInfoPoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientMsgTrafficPoll — poll once for RTclient message traffic information

```
T_BOOL TipcSrvMonClientMsgTrafficPoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientMsgTypeExPoll — poll once for RTclient message type information

```
T_BOOL TipcSrvMonClientMsgTypeExPoll(srv, client_name,
msg_type_name)
T_IPC_SRV srv;
T_STR client_name;
T_STR msg_type_name;
```

The previous Ex function supersedes its older counterpart for all new development. Use the following older method only when operating with clients prior to release 6.7.

TipcSrvMonClientMsgTypePoll — poll once for RTclient message type information

```
T_BOOL TipcSrvMonClientMsgTypePoll(srv, client_name,
msg_type_name)
T_IPC_SRV srv;
T_STR client_name;
T_STR msg_type_name;
```

TipcSrvMonClientNamesNumPoll — poll for the number of RTclients in the current project that match the value of the Monitor_Scope option

```
T_BOOL TipcSrvMonClientNamesNumPoll(srv)
T_IPC_SRV srv;
```

TipcSrvMonClientNamesPoll — poll once for RTclient names

```
T_BOOL TipcSrvMonClientNamesPoll(srv)
T_IPC_SRV srv;
```

TipcSrvMonClientOptionPoll — poll once for RTclient option information

```
T_BOOL TipcSrvMonClientOptionPoll(srv, client_name, option_name)
T_IPC_SRV srv;
T_STR client_name;
T_STR option_name;
```

TipcSrvMonClientSubjectExPoll — poll once for RTclient subject information

```
T_BOOL TipcSrvMonClientSubjectExPoll(srv, client_name,
subject_name)
T_IPC_SRV srv;
T_STR client_name;
T_STR subject_name;
```

The previous Ex function supersedes its older counterpart for all new development. Use the following older method only when operating with clients prior to release 6.7.

TipcSrvMonClientSubjectPoll — poll once for RTclient subject information

```
T_BOOL TipcSrvMonClientSubjectPoll(srv, client_name, subject_name)
T_IPC_SRV srv;
T_STR client_name;
T_STR subject_name;
```

TipcSrvMonClientSubscribeNumPoll — poll for the number of subjects subscribed to by an RTclient

```
T_BOOL TipcSrvMonClientSubscribeNumPoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientSubscribePoll — poll once for subjects to which an RTclient subscribes

```
T_BOOL TipcSrvMonClientSubscribePoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientTimePoll — poll once for RTclient time information

```
T_BOOL TipcSrvMonClientTimePoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonClientVersionPoll — poll once for RTclient version information

```
T_BOOL TipcSrvMonClientVersionPoll(srv, client_name)
T_IPC_SRV srv;
T_STR client_name;
```

TipcSrvMonProjectNamesPoll — poll once for project names

```
T_BOOL TipcSrvMonProjectNamesPoll(srv)
T_IPC_SRV srv;
```

TipcSrvMonServerBufferPoll — poll once for RTserver message-related buffer information

```
T_BOOL TipcSrvMonServerBufferPoll(srv, server_name,
connected_process_name)
T_IPC_SRV srv;
T_STR server_name;
T_STR connected_process_name;
```

TipcSrvMonServerConnPoll — poll once for RTserver connections

```
T_BOOL TipcSrvMonServerConnPoll(srv)
T_IPC_SRV srv;
```

TipcSrvMonServerCpuPoll — poll for the percentage of CPU time used by an RTserver

```
T_BOOL TipcSrvMonServerCpuPoll(srv, server_name)
T_IPC_SRV srv;
T_STR server_name;
```

TipcSrvMonServerGeneralPoll — poll once for RTserver general information

```
T_BOOL TipcSrvMonServerGeneralPoll(srv, server_name)
T_IPC_SRV srv;
T_STR server_name;
```

TipcSrvMonServerMsgTrafficExPoll — poll once for RTserver message traffic information

```
T_BOOL TipcSrvMonServerMsgTrafficExPoll(srv, server_name,
connected_process_name)
T_IPC_SRV srv;
T_STR server_name;
T_STR connected_process_name;
```

The previous Ex function supersedes its older counterpart for all new development. Use the following older method only when operating with servers prior to release 6.7.

TipcSrvMonServerMsgTrafficPoll — poll once for RTserver message traffic information

```
T_BOOL TipcSrvMonServerMsgTrafficPoll(srv, server_name,
connected_process_name)
T_IPC_SRV srv;
T_STR server_name;
T_STR connected_process_name;
```

TipcSrvMonServerNamesPoll — poll once for RTserver names

```
T_BOOL TipcSrvMonServerNamesPoll(srv)
T_IPC_SRV srv;
```

TipcSrvMonServerOptionPoll — poll once for RTserver option information

```
T_BOOL TipcSrvMonServerOptionPoll(srv, server_name, option_name)
T_IPC_SRV srv;
T_STR server_name;
T_STR option_name;
```

TipcSrvMonServerRoutePoll — poll once for RTserver route information

```
T_BOOL TipcSrvMonServerRoutePoll(srv, server_name, dest_server_name)
T_IPC_SRV srv;
T_STR server_name;
T_STR dest_server_name;
```

TipcSrvMonServerStartTimePoll — poll once for RTserver start time and elapsed time

```
T_BOOL TipcSrvMonServerStartTimePoll(srv, server_name)
T_IPC_SRV srv;
T_STR server_name;
```

TipcSrvMonServerTimePoll — poll once for RTserver time information

```
T_BOOL TipcSrvMonServerTimePoll(srv, server_name)
T_IPC_SRV srv;
T_STR server_name;
```

TipcSrvMonServerVersionPoll — poll once for RTserver version information

```
T_BOOL TipcSrvMonServerVersionPoll(srv, server_name)
T_IPC_SRV srv;
T_STR server_name;
```

TipcSrvMonSubjectNamesPoll — poll once for subject names

```
T_BOOL TipcSrvMonSubjectNamesPoll(srv)
T_IPC_SRV srv;
```

TipcSrvMonSubjectSubscribePoll — poll once for RTclients that are subscribing to a subject

```
T_BOOL TipcSrvMonSubjectSubscribePoll(srv, subject_name)
T_IPC_SRV srv;
T_STR subject_name;
```

Watch Functions (TipcSrvMon*Watch)

The TipcSrvMon*Watch functions are used to set up asynchronous notification when specified information in a SmartSockets project changes.

TipcSrvMonClientBufferGetWatch — determine whether this RTclient is watching message-related buffer information in an RTclient

```
T_BOOL TipcSrvMonClientBufferGetWatch(srv, client_name,
watch_status_return)
T_IPC_SRV srv;
T_STR client_name;
T_BOOL *watch_status_return;
```

TipcSrvMonClientBufferSetWatch — start or stop watching RTclient message-related buffer information

```
T_BOOL TipcSrvMonClientBufferSetWatch(srv, client_name,
watch_status)
T_IPC_SRV srv;
T_STR client_name;
T_BOOL watch_status;
```

TipcSrvMonClientCongestionGetWatch — determine if the RTclient is watching for congestion in the RTserver write buffer of a connected process

```
T_BOOL TipcSrvMonClientCongestionGetWatch(srv, client_name,
watch_status_return)
T_IPC_SRV srv;
T_STR client_name;
T_BOOL *watch_status_return;
```

TipcSrvMonClientCongestionSetWatch — start or stop watching for congestion in an RTclient's write buffer

```
T_BOOL TipcSrvMonClientCongestionSetWatch(srv, client_name,
high_water, low_water, watch_status)
T_IPC_SRV srv;
T_STR client_name;
T_INT4 high_water;
T_INT4 low_water;
T_BOOL watch_status;
```

TipcSrvMonClientMsgRecvGetWatch — determine whether this RTclient is watching received messages in an RTclient

```
T_BOOL TipcSrvMonClientMsgRecvGetWatch(srv, client_name,
msg_type_name, watch_status_return)
T_IPC_SRV srv;
T_STR client_name;
T_STR msg_type_name;
T_BOOL *watch_status_return;
```

TipcSrvMonClientMsgRecvSetWatch — start or stop watching RTclient received messages

```
T_BOOL TipcSrvMonClientMsgRecvSetWatch(srv, client_name,
msg_type_name, watch_status)
T_IPC_SRV srv;
T_STR client_name;
T_STR msg_type_name;
T_BOOL watch_status;
```

TipcSrvMonClientMsgSendGetWatch — determine whether this RTclient is watching sent messages in an RTclient

```
T_BOOL TipcSrvMonClientMsgSendGetWatch(srv, client_name,
msg_type_name, watch_status_return)
T_IPC_SRV srv;
T_STR client_name;
T_STR msg_type_name;
T_BOOL *watch_status_return;
```

TipcSrvMonClientMsgSendSetWatch — start or stop watching RTclient sent messages

```
T_BOOL TipcSrvMonClientMsgSendSetWatch(srv, client_name,
msg_type_name, watch_status)
T_IPC_SRV srv;
T_STR client_name;
T_STR msg_type_name;
T_BOOL watch_status;
```

TipcSrvMonClientNamesGetWatch — determine whether this RTclient is watching RTclient names

```
T_BOOL TipcSrvMonClientNamesGetWatch(srv, watch_status_return)
T_IPC_SRV srv;
T_BOOL *watch_status_return;
```

TipcSrvMonClientNamesSetWatch — start or stop watching RTclient names

```
T_BOOL TipcSrvMonClientNamesSetWatch(srv, watch_status)
T_IPC_SRV srv;
T_BOOL watch_status;
```

TipcSrvMonClientSubscribeGetWatch — determine whether this RTclient is watching the subjects that an RTclient is subscribing to

```
T_BOOL TipcSrvMonClientSubscribeGetWatch(srv, client_name,
watch_status_return)
T_IPC_SRV srv;
T_STR client_name;
T_BOOL *watch_status_return;
```

TipcSrvMonClientSubscribeSetWatch — start or stop watching the subjects that an RTclient is subscribing to

```
T_BOOL TipcSrvMonClientSubscribeSetWatch(srv, client_name,
watch_status)
T_IPC_SRV srv;
T_STR client_name;
T_BOOL watch_status;
```

TipcSrvMonClientTimeGetWatch — determine whether this RTclient is watching time information in an RTclient

```
T_BOOL TipcSrvMonClientTimeGetWatch(srv, client_name,
watch_status_return)
T_IPC_SRV srv;
T_STR client_name;
T_BOOL *watch_status_return;
```

TipcSrvMonClientTimeSetWatch — start or stop watching RTclient time information

```
T_BOOL TipcSrvMonClientTimeSetWatch(srv, client_name, watch_status)
T_IPC_SRV srv;
T_STR client_name;
T_BOOL watch_status;
```

TipcSrvMonPrintWatch — print all monitoring categories being watched

```
T_BOOL TipcSrvMonPrintWatch(srv, func)
T_IPC_SRV srv;
T_OUT_FUNC func;
```

TipcSrvMonProjectNamesGetWatch — determine whether this RTclient is watching project names

```
T_BOOL TipcSrvMonProjectNamesGetWatch(srv, watch_status_return)
T_IPC_SRV srv;
T_BOOL *watch_status_return;
```

TipcSrvMonProjectNamesSetWatch — start or stop watching project names

```
T_BOOL TipcSrvMonProjectNamesSetWatch(srv, watch_status)
T_IPC_SRV srv;
T_BOOL watch_status;
```

TipcSrvMonServerCongestionGetWatch — determine if the server is watching a process's write buffer for congestion

```
T_BOOL TipcSrvMonServerCongestionGetWatch(srv, server_name,
connected_process_name, watch_status_return)
T_IPC_SRV srv;
T_STR server_name;
T_STR connected_process_name;
T_BOOL *watch_status_return;
```

TipcSrvMonServerCongestionSetWatch — start or stop watching for congestion in the RTserver write buffer of a connected process

```
T_BOOL TipcSrvMonServerCongestionSetWatch(srv, server_name,
connected_process_name, high_water, low_water, watch_status)
T_IPC_SRV srv;
T_STR server_name;
T_STR connected_process_name;
T_INT4 high_water;
T_INT4 low_water;
T_BOOL watch_status;
```

TipcSrvMonServerConnGetWatch — determine whether this RTclient is watching RTserver connections

```
T_BOOL TipcSrvMonServerConnGetWatch(srv, watch_status_return)
T_IPC_SRV srv;
T_BOOL *watch_status_return;
```

TipcSrvMonServerConnSetWatch — start or stop watching RTserver connections

```
T_BOOL TipcSrvMonServerConnSetWatch(srv, watch_status)
T_IPC_SRV srv;
T_BOOL watch_status;
```

TipcSrvMonServerMaxClientLicensesGetWatch — determine if the RTserver is watching for the server cloud to exceed the licensed number of RTclient connections

```
T_BOOL TipcSrvMonServerMaxClientLicensesGetWatch(srv,
server_name, watch_status_return)
T_IPC_SRV srv;
T_STR server_name;
T_BOOL *watch_status_return;
```

TipcSrvMonServerMaxClientLicensesSetWatch — start or stop watching for a server to exceed the licensed number of RTclient connections

```
T_BOOL TipcSrvMonServerMaxClientLicensesSetWatch(srv,
server_name, watch_status)
T_IPC_SRV srv;
T_STR server_name;
T_BOOL watch_status;
```

TipcSrvMonServerNamesGetWatch — determine whether this RTclient is watching RTserver names

```
T_BOOL TipcSrvMonServerNamesGetWatch(srv, watch_status_return)
T_IPC_SRV srv;
T_BOOL *watch_status_return;
```

TipcSrvMonServerNamesSetWatch — start or stop watching RTserver names

```
T_BOOL TipcSrvMonServerNamesSetWatch(srv, watch_status)
T_IPC_SRV srv;
T_BOOL watch_status;
```

TipcSrvMonSubjectNamesGetWatch — determine whether this RTclient is watching subject names

```
T_BOOL TipcSrvMonSubjectNamesGetWatch(srv, watch_status_return)
T_IPC_SRV srv;
T_BOOL *watch_status_return;
```

TipcSrvMonSubjectNamesSetWatch — start or stop watching subject names

```
T_BOOL TipcSrvMonSubjectNamesSetWatch(srv, watch_status)
T_IPC_SRV srv;
T_BOOL watch_status;
```

TipcSrvMonSubjectSubscribeGetWatch — determine whether this RTclient is watching the RTclients that are subscribing to a subject

```
T_BOOL TipcSrvMonSubjectSubscribeGetWatch(srv, subject_name,
watch_status_return)
T_IPC_SRV srv;
T_STR subject_name;
T_BOOL *watch_status_return;
```

TipcSrvMonSubjectSubscribeSetWatch — start or stop watching the RTclients that are subscribing to a subject

```
T_BOOL TipcSrvMonSubjectSubscribeSetWatch(srv, subject_name,
watch_status)
T_IPC_SRV srv;
T_STR subject_name;
T_BOOL watch_status;
```

Extension Data Functions (TipcSrvMonExt*)

The TipcSrvMonExt* functions are used to delete, define, or update a field in an RTclient's MON_CLIENT_EXT_POLL_RESULT message. This message is returned to another RTclient that polls for RTclient information by issuing TipcSrvMonClientExtPoll.

TipcSrvMonExtDelete — remove a field from an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtDelete(srv, field_name)
T_IPC_SRV srv;
T_STR field_name;
```

TipcSrvMonExtSetBinary — add or update a field of type BINARY in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetBinary(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_PTR data;
T_INT4 size;
```

TipcSrvMonExtSetBool — add or update a field of type BOOL in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetBool(srv, field_name, data)
T_IPC_SRV srv;
T_BOOL field_name;
T_PTR data;
```

TipcSrvMonExtSetBoolArray — add or update a field containing an array of type BOOL in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetBoolArray(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_BOOL *data;
T_INT4 size;
```

TipcSrvMonExtSetInt2 — add or update a field of type INT2 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetInt2(srv, field_name, data)
T_IPC_SRV srv;
T_STR field_name;
T_INT2 data;
```

TipcSrvMonExtSetInt2Array — add or update a field containing an array of type INT2 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetInt2Array(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_INT2 *data;
T_INT4 size
```

TipcSrvMonExtSetInt4 — add or update a field of type INT4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetInt4(srv, field_name, data)
T_IPC_SRV srv;
T_STR field_name;
T_INT4 data;
```

TipcSrvMonExtSetInt4Array — add or update a field containing an array of type INT4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetInt4Array(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_INT4 *data;
T_INT4 size;
```

TipcSrvMonExtSetInt8 — add or update a field of type INT8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetInt8(srv, field_name, data)
T_IPC_SRV srv;
T_STR field_name;
T_INT8 data;
```

TipcSrvMonExtSetInt8Array — add or update a field containing an array of type INT8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetInt8Array(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_INT8 *data;
T_INT4 size;
```

TipcSrvMonExtSetReal4 — add or update a field of type REAL4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetReal4(srv, field_name, data)
T_IPC_SRV srv;
T_STR field_name;
T_REAL4 data;
```

TipcSrvMonExtSetReal4Array — add or update a field containing an array of type REAL4 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetReal4Array(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_REAL4 *data;
T_INT4 size;
```

TipcSrvMonExtSetReal8 — add or update a field of type REAL8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetReal8(srv, field_name, data)
T_IPC_SRV srv;
T_STR field_name;
T_REAL8 data;
```

TipcSrvMonExtSetReal8Array — add or update a field containing an array of type REAL8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetReal8Array(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_REAL8 *data;
T_INT4 size;
```

TipcSrvMonExtSetReal16 — add or update a field of type REAL16 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetReal16(srv, field_name, data)
T_IPC_SRV srv;
T_STR field_name;
T_REAL16 data;
```

TipcSrvMonExtSetReal16Array — add or update a field containing an array of type REAL16 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetReal16Array(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_REAL16 *data;
T_INT4 size;
```

TipcSrvMonExtSetStr — add or update a field of type STR in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetStr(srv, field_name, data)
T_IPC_SRV srv;
T_STR field_name;
T_STR data;
```

TipcSrvMonExtSetStArray — add or update a field containing an array of type STR in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetStrArray(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_STR *data;
T_INT4 size;
```

TipcSrvMonExtSetUtf8 — add or update a field of type UTF8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetUtf8(srv, field_name, data)
T_IPC_SRV srv;
T_STR field_name;
T_UTF8 data;
```

TipcSrvMonExtSetUtf8Array — add or update a field containing an array of type UTF8 in an RTclient's MON_CLIENT_EXT_POLL_RESULT message

```
T_BOOL TipcSrvMonExtSetUtf8Array(srv, field_name, data, size)
T_IPC_SRV srv;
T_STR field_name;
T_UTF8 *data;
T_INT4 size;
```

Miscellaneous Functions

These functions do not fit with any of the other established categories.

TipcSrvMonGetIdentStr — retrieve the monitoring identification string of this process

```
T_BOOL TipcSrvMonGetIdentStr(srv, ident_str_return)
T_IPC_SRV srv;
T_STR *ident_str_return;
```

TipcSrvMonSetIdentStr — set the monitoring identification string of this process

```
T_BOOL TipcSrvMonSetIdentStr(srv, ident_str)
T_IPC_SRV srv;
T_STR ident_str;
```

Chapter 4 Utility Functions and Macros

This chapter is a summary of the interfaces for the TIBCO SmartSockets utilities. The utility functions (Tut*) and Macros (T_*) are generally not associated with messaging or communication. In general, these functions can be used in a variety of applications, including those involving TIBCO SmartSockets.

This chapter is designed for both novice and experienced SmartSockets developers. If you are a novice to SmartSockets you can read the entire chapter to gain an understanding of all that is available to you in the SmartSockets utilities. For more experienced users, this chapter can be used to quickly find a utility to perform a particular task or to check its arguments.

For each utility or macro, this chapter lists a short description of what it does and a synopsis of its arguments and return values. For more details, see the *TIBCO SmartSockets Utilities.*

Topics

Macros

T_ASSERT — program verification macro

```
void T_ASSERT(expression)
int expression;
```

T_CALLOC — convenience macro to allocate and clear memory

```
void T_CALLOC(ptr, size, type)
type ptr;
int size;
```

T_ENTRY — required in the definition and prototype of callback and thread functions

```
void T_ENTRY(thread or callback function)
void thread or callback_function;
```

T_FREE — return allocated memory to the heap

```
void T_FREE(ptr)
T_PTR ptr;
```

T_MALLOC — convenience macro to allocate memory

```
void T_MALLOC(ptr, size, type)
type ptr;
int size;
```

T_REALLOC — convenience macro to reallocate memory

```
void T_REALLOC(ptr, size, type)
type ptr;
int size;
```

T_STRDUP — duplicate a string

```
void T_STRDUP(dest, src)
T_STR dest;
T_STR src;
```

Functions to Work with Buffers (TutBuf*)

The TutBuf* functions are used to perform operations with buffers.

TutBufAllocAligned — allocate space in a buffer object

```
T_PTR TutBufAllocAligned(buf, size, align)
T_BUF buf;
T_INT4 size;
T_INT4 align;
```

TutBufAppendBuf — append the contents of one buffer into another buffer

```
T_PTR TutBufAppendBuf(dest, src, size)
T_BUF dest;
T_BUF src;
T_INT4 size;
```

TutBufAppendRange — append the contents of a range of one buffer into another buffer

```
T_PTR TutBufAppendRange(dest, src, begin, end)
T_BUF dest;
T_BUF src;
T_INT4 begin;
T_INT4 end;
```

TutBufCloneRange — create a new buffer containing a portion of an existing buffer

```
T_BUF TutBufCloneRange(src, begin, end)
T_BUF src;
T_INT4 begin;
T_INT4 end;
```

TutBufCreate — create a new buffer

```
T_BUF TutBufCreate(size)
T_INT4 size;
```

TutBufCreateStatic — wrap a buffer around an existing block of data

```
T_BUF TutBufCreateStatic(ptr, size)
T_PTR ptr;
T_INT4 size;
```

TutBufDeleteRange — delete a block of data from a buffer

```
T_BUF TutBufDeleteRange(buf, begin, end)
T_BUF buf;
T_INT4 begin;
T_INT4 end;
```

TutBufDestroy — destroy a buffer

```
T_BOOL TutBufDestroy(buf)
T_BUF buf;
```

TutBufGetSize — retrieve the number of bytes written to the buffer

```
T_BOOL TutBufGetSize(buf, size_return)
T_BUF buf;
T_INT4 *size_return;
```

TutBufGrow — grow the allocated size of the buffer

```
T_BOOL TutBufGrow(buf, new_size)
T_BUF buf;
T_INT4 new_size;
```

TutBufNextAligned — allocate space in a buffer object

```
T_PTR TutBufNextAligned(buf, size, align)
T_BUF buf;
T_INT4 size;
T_INT4 align;
```

TutBufReset — resets the buffer contents

```
T_BOOL TutBufReset(buf, ptr)
T_BUF buf;
T_PTR ptr;
```

TutBufRewind — rewinds the current position in the buffer

```
T_BOOL TutBufRewind(buf, ptr)
T_BUF buf;
T_PTR ptr;
```

TutBufSetSize — sets the valid number of bytes in the buffer

```
T_BOOL TutBufSetSize(buf, size)
T_BUF buf;
T_INT4 size;
```

Functions to Work with Callbacks (TutCb*)

The TutCb* functions are used to operate on callbacks, destroying them (TutCbDestroy), looking up information about them (TutCbGet*), and setting their properties (TutCbSet*). Callbacks are created using a specific callback creation routine of the form (Tipc*CbCreate).

TutCbDestroy — destroy a callback and removes it from the list

```
T_BOOL TutCbDestroy(cb)
T_CB cb;
```

TutCbGetArgument — determine the callback argument of the specified callback

```
T_BOOL TutCbGetArgument(cb, arg_return)
T_CB cb;
T_CB_ARG *arg_return;
```

TutCbGetFunction — determine the callback function of the specified callback

```
T_BOOL TutCbGetFunction(cb, func_return)
T_CB cb;
T_CB_FUNC *func_return;
```

TutCbGetPriority — determine the current priority of a callback

```
T_BOOL TutCbGetPriority(cb, pri_return)
T_CB cb;
T_CB_PRIORITY *pri_return;
```

TutCbSetArgument — set the callback argument used by the specified callback

```
T_BOOL TutCbSetArgument(cb, arg)
T_CB cb;
T_CB_ARG arg;
```

TutCbSetFunction — set the callback function to be called

```
T_BOOL TutCbSetFunction(cb, func)
T_CB cb;
T_CB_FUNC func;
```

TutCbSetPriority — set the priority of a callback

```
T_BOOL TutCbSetPriority(cb, pri)
T_CB cb;
T_CB_PRIORITY pri;
```

Command Functions (TutCommand*)

These functions are used to execute commands.

TutCommandParseFile — submit a file to the SmartSockets process command interface

```
T_BOOL TutCommandParseFile(file_name)
T_STR file_name;
```

TutCommandParseStr — submit a non-interactive string to the SmartSockets process command interface

```
void TutCommandParseStr(command_string)
T_STR command_string;
```

TutCommandParseTypedStr — submit a string to SmartSockets command interface

```
void TutCommandParseTypedStr(command_string)
T_STR command_string;
```

Error Functions

These functions are used to work with the global SmartSockets error code.

TutErrNumGet — determine the value of a global SmartSockets error number

```
T_INT4 TutErrNumGet()
```

TutErrNumGetC — determine the value of a C error number

```
T_INT4 TutErrNumGetC()
```

TutErrNumGetOs — determine the value of an operating system error number

```
T_INT4 TutErrNumGetOs()
```

TutErrNumGetSocket — determine the value of a socket error number

```
T_INT4 TutErrNumGetSocket()
```

TutErrNumSet — set the value of a global SmartSockets error number

```
void TutErrNumSet(err_num)
T_INT4 err_num;
```

TutErrNumToStr — convert a SmartSockets error number into a error string

```
T_STR TutErrNumToStr(err_num)
T_INT4 err_num;
```

TutErrStrCreate — add a new string to the error string table

```
T_BOOL TutErrStrCreate(err_num, err_str)
T_INT4 err_num;
T_STR err_str;
```

TutErrStrGet — retrieve the global SmartSockets error number as a descriptive string

```
T_STR TutErrStrGet()
```

TutPerror — report UNIX error messages to the process output window

```
void TutPerror(msg)
T_STR msg;
```

TutWarning — unconditional printf-style warning output to a SmartSockets process output window

```
void TutWarning(format_str, ...)
T_STR format_str;
```

File Functions

These functions are used to work with files.

TutFileTraverse — traverse open files

```
T_PTR TutFileTraverse(trav_func, trav_arg)
T_FILE_TRAV_FUNC trav_func;
T_PTR trav_arg;
```

TutFOpen — open a file in a platform independent way

```
FILE *TutFOpen(file_name, file_mode)
T_STR file_name;
T_STR file_mode;
```

TutSetFileCloseFunc — set a user defined function to be used for file closing

```
T_BOOL TutSetFileCloseFunc(close_func)
T_FILE_CLOSE_FUNC close_func;
```

TutSetFileOpenFunc — set the function to use when opening a file

```
T_BOOL TutSetFileOpenFunc(open_func)
T_FILE_OPEN_FUNC open_func;
```

TutSetFileReadFunc — set the function to use when reading from a file

```
T_BOOL TutSetFileReadFunc(read_func)
T_FILE_READ_FUNC read_func;
```

Get Platform and Software Info Functions (TutGet*)

These functions are used to retrieve various information about the platform the process is running and what version of SmartSockets is being used.

TutGetIntFormat — determine the integer number format of this computer

```
T_INT_FORMAT TutGetIntFormat()
```

TutGetNodeName — determine the name of the current node

```
T_STR TutGetNodeName(name_return)
T_STR name_return;
```

TutGetRealFormat — determine the real number format of this computer

```
T_REAL_FORMAT TutGetRealFormat()
```

TutGetSocketDir — get name of directory where local socket files are written

```
T_STR TutGetSocketDir()
```

TutGetVersionName — get SmartSockets software version name

```
T_STR TutGetVersionName()
```

TutGetVersionNumber — get SmartSockets software version number

```
T_INT4 TutGetVersionNumber()
```

Hash Table Functions (TutHash*)

The TutHash* functions are used to work with hash tables.

TutHashBucketGetValue — determine the value in a hash bucket

```
T_PTR TutHashBucketGetValue(bucket)
T_HASH_BUCKET bucket;
```

TutHashBucketSetValue — set the value in a hash bucket

```
T_PTR TutHashBucketSetValue(bucket, new_value)
T_HASH_BUCKET bucket;
T_PTR new_value;
```

TutHashCreate — create a hash table

```
T_HASH_TABLE TutHashCreate(init_size, hash_func, test_func)
T_INT4 init_size;
T_HASH_CALC_FUNC hash_func;
T_HASH_TEST_FUNC test_func;
```

TutHashCreateInt — create a hash table with integer keys

```
T_HASH_TABLE TutHashCreateInt(init_size)
T_INT4 init_size;
```

TutHashCreatePtr — create a hash table with pointer keys

```
T_HASH_TABLE TutHashCreatePtr(init_size)
T_INT4 init_size;
```

TutHashCreateStr — create a hash table with case-sensitive string keys

```
T_HASH_TABLE TutHashCreateStr(init_size)
T_INT4 init_size;
```

TutHashCreateStri — create a hash table with string keys that are not case sensitive

```
T_HASH_TABLE TutHashCreateStri(init_size)
T_INT4 init_size;
```

TutHashDelete — delete a hash table entry by key

```
T_PTR TutHashDelete(table, key)
T_HASH_TABLE table;
T_PTR key;
```

TutHashDestroy — deallocate the memory used by a hash table and destroy the table

```
void TutHashDestroy(table, destroy_func, destroy_arg)
T_HASH_TABLE table;
T_HASH_DESTROY_FUNC destroy_func;
T_PTR destroy_arg;
```

TutHashDump — dump the information about a hash table

```
void TutHashDump(table, output_func)
T_HASH_TABLE table;
T_OUT_FUNC output_func;
```

TutHashGetSize — return the number of entries in the hash table

```
T_INT4 TutHashGetSize(table)
T_HASH_TABLE table;
```

TutHashInsert — insert an entry into the hash table

```
void TutHashInsert(table, key, value)
T_HASH_TABLE table;
T_PTR key;
T_PTR value;
```

TutHashLookup — find an entry in the hash table

```
T_PTR TutHashLookup(table, key)
T_HASH_TABLE table;
T_PTR key;
```

TutHashLookupBucket — look up a bucket in a hash table

```
T_HASH_BUCKET TutHashLookupBucket(table, key)
T_HASH_TABLE table;
T_PTR key;
```

TutHashReplace — replace a value stored in the hash table

```
T_PTR TutHashReplace(table, key, new_value)
T_HASH_TABLE table;
T_PTR key;
T_PTR new_value;
```

TutHashSort — return an array of sorted entries from a hash table

```
T_PTR TutHashSort(table, array_size_return, cmp_func, select_func)
T_HASH_TABLE table;
T_INT4 *array_size_return;
T_HASH_SORT_CMP_FUNC cmp_func;
T_HASH_SORT_SELECT_FUNC select_func;
```

TutHashTraverse — traverse all entries in a hash table

```
T_PTR TutHashTraverse(table, traverse_func, traverse_arg)
T_HASH_TABLE table;
T_HASH_TRAV_FUNC traverse_func;
T_PTR traverse_arg;
```

Memory Allocation

These functions are used to allocate and free memory.

TutCalloc — allocate and clear memory

```
T_PTR TutCalloc(size)
T_UINT4 size;
```

TutFree — free the previously allocated memory block

```
void TutFree(ptr)
T_PTR ptr;
```

TutMalloc — allocate a block of memory

```
T_PTR TutMalloc(size)
T_UINT4 size;
```

TutRealloc — reallocate memory

```
T_PTR TutRealloc(ptr, size)
T_PTR ptr;
T_UINT4 size;
```

Option Functions (TutOption*)

The TutOption* functions are used to work with options. These functions allow you to create new options, destroy existing options, retrieve the values of options, and set the values of options.

Option Change Callback Functions (TutOptionChangeCb*)

The functions are used to create and look up callback functions which are executed when the value of an option changes.

TutOptionChangeCbCreate — create a callback that is called when the option value changes

```
T_CB TutOptionChangeCbCreate(option, change_func, arg)
T_OPTION option;
T_OPTION_CHANGE_CB_FUNC change_func;
T_CB_ARG arg;
```

TutOptionChangeCbLookup — look up the option change callback

```
T_CB TutOptionChangeCbLookup(option, change_func, arg)
T_OPTION option;
T_OPTION_CHANGE_CB_FUNC change_func;
T_CB_ARG arg;
```

Creating and Destroying

TutOptionCreate — create a new option

```
T_OPTION TutOptionCreate(name, type)
T_STR name;
T_OPT_TYPE type;
```

TutOptionDestroy — destroy an existing option

```
T_BOOL TutOptionDestroy(option)
T_OPTION option;
```

Retrieving Values or Properties (TutGet*)

TutOptionGetBool — determine the value of a boolean option

```
T_BOOL TutOptionGetBool(option, bool_val_return)
T_OPTION option;
T_BOOL *bool_val_return;
```

TutOptionGetEnum — determine the value of an enumerated option

```
T_BOOL TutOptionGetEnum(option, enum_val_return)
T_OPTION option;
T_STR *enum_val_return;
```

TutOptionGetEnumList — determine the value of an enumerated list option

```
T_BOOL TutOptionGetEnumList(option, enum_list_val_return)
T_OPTION option;
T_STR_LIST *enum_list_val_return;
```

TutOptionGetKnown — retrieve the known state for an option

```
T_BOOL TutOptionGetKnown(option, known_flag)
T_OPTION option;
T_BOOL *known_flag;
```

TutOptionGetName — return the name of the option

```
T_STR TutOptionGetName(option)
T_OPTION option;
```

TutOptionGetNum — determine the value of a numeric option

```
T_BOOL TutOptionGetNum(option, num_val_return)
T_OPTION option;
T_REAL8 *num_val_return;
```

TutOptionGetReadOnly — retrieve read-only property of option

```
T_BOOL TutOptionGetReadOnly(option, read_only_flag)
T_OPTION option;
T_BOOL *read_only_flag;
```

TutOptionGetRequired — retrieve required property of option

```
T_BOOL TutOptionGetRequired(option, required_flag)
T_OPTION option;
T_BOOL *required_flag;
```

TutOptionGetStr — determine the value of a string option

```
T_BOOL TutOptionGetStr(option, str_val_return)
T_OPTION option;
T_STR *str_val_return;
```

TutOptionGetStrList — determine the value of a string list option

```
T_BOOL TutOptionGetStrList(option, str_list_val_return)
T_OPTION option;
T_STR_LIST *str_list_val_return;
```

TutOptionGetType — return the type of the option

```
T_BOOL TutOptionGetType(option, opt_type)
T_OPTION option;
T_OPT_TYPE *opt_type;
```

TutOptionGetVerifyFunc — retrieve the verify function

```
T_BOOL TutOptionGetVerifyFunc(option, verify_func_return)
T_OPTION option;
T_OPTION_VERIFY_FUNC *verify_func_return;
```

Miscellaneous Functions

TutOptionLegValAdd — add a legal value to an enumerated option

```
T_BOOL TutOptionLegValAdd(option, val)
T_OPTION option;
T_STR val;
```

TutOptionLookup — look up an option by name

```
T_OPTION TutOptionLookup(name)
T_STR name;
```

TutOptionNamedLookup — look up a named option by name

```
T_OPTION TutOptionNamedLookup(name, option_name)
T_STR name;
T_SRV option_name;
```

Setting the Values or Properties (TutOptionSet*)

TutOptionSetBool — set the value of a boolean option

```
T_BOOL TutOptionSetBool(option, bool_val)
T_OPTION option;
T_BOOL bool_val;
```

TutOptionSetEnum — set the value of an enumerated option

```
T_BOOL TutOptionSetEnum(option, enum_val)
T_OPTION option;
T_STR enum_val;
```

TutOptionSetEnumList — set the value of an enumerated list option

```
T_BOOL TutOptionSetEnumList(option, enum_list_val)
T_OPTION option;
T_STR_LIST enum_list_val;
```

TutOptionSetNum — set the value of a numeric option

```
T_BOOL TutOptionSetNum(option, num_val)
T_OPTION option;
T_REAL8 num_val;
```

TutOptionSetReadOnly — set whether or not an option is read only

```
T_BOOL TutOptionSetReadOnly(option, read_only)
T_OPTION option;
T_BOOL read_only;
```

TutOptionSetRequired — set whether or not an option is required

```
T_BOOL TutOptionSetRequired(option, required)
T_OPTION option;
T_BOL required;
```

TutOptionSetStr — set the value of a string option

```
T_BOOL TutOptionSetStr(option, str_val)
T_OPTION option;
T_STR str_val;
```

TutOptionSetStrList — set the value of a string list option

```
T_BOOL TutOptionSetStrList(option, str_list_val)
T_OPTION option;
T_STR_LIST str_list_val;
```

TutOptionSetUnknown — set the value of the option to UNKNOWN

```
T_BOOL TutOptionSetUnknown(option)
T_OPTION option;
```

TutOptionSetVerifyFunc — create a function to verify that the option's value is acceptable

```
T_BOOL TutOptionSetVerifyFunc(option, verify_func, arg)
T_OPTION option;
T_OPTION_VERIFY_FUNC verify_func;
T_PTR arg;
```

Pointer Array Functions (TutPtrAry*)

The TutPtrAry* are used to work with pointer arrays.

TutPtrAryAppend — append an item to the pointer array

```
T_BOOL TutPtrAryAppend(ptr_array, pointer, index)
T_PTR_ARY ptr_array;
T_PTR pointer;
T_INT4 *index;
```

TutPtrAryChangeCbCreate — create a callback that is called when a pointer array changes

```
T_CB TutPtrAryChangeCbCreate(ptr_ary, change_func, arg)
T_PTR_ARY ptr_ary;
T_PTR_ARY_CHANGE_CB_FUNC change_func;
T_CB_ARG arg;
```

TutPtrAryChangeCbLookup — look up a pointer array change callback

```
T_CB TutPtrAryChangeCbLookup(ptr_ary, change_func, arg)
T_PTR_ARY ptr_ary;
T_PTR_ARY_CHANGE_CB_FUNC change_func;
T_CB_ARG arg;
```

TutPtrAryClear — clear the pointer array

```
T_BOOL TutPtrAryClear(ptr_ary, user_func, user_arg)
T_PTR_ARY ptr_ary;
T_PTR_ARY_USER_FUNC user_func;
T_PTR user_arg;
```

TutPtrAryCreate — create a pointer array

```
T_PTR_ARY TutPtrAryCreate(size)
T_INT4 size;
```

TutPtrAryDestroy — destroy the pointer array

```
T_BOOL TutPtrAryDestroy(ptr_ary, user_func, user_arg)
T_PTR_ARY ptr_ary;
T_PTR_ARY_USER_FUNC user_func;
T_PTR user_arg;
```

TutPtrAryGetItemAtIndex — retrieve the item stored at the given index in the array

```
T_BOOL TutPtrAryGetItemAtIndex(ptr_ary, index, element)
T_PTR_ARY ptr_ary;
T_INT4 index;
T_PTR *element;
```

TutPtrAryGetItemCount — determine the number of items in the pointer array

```
T_BOOL TutPtrAryGetItemCount(ptr_ary, count)
T_PTR_ARY ptr_ary;
T_INT4 *count;
```

TutPtrAryGetItemIndex — determine the array index of the specified item

```
T_BOOL TutPtrAryGetItemIndex(ptr_ary, item, index)
T_PTR_ARY ptr_ary;
T_PTR item;
T_INT4 *index;
```

TutPtrAryInsertAfter — insert a new item in the array after the specified item

```
T_BOOL TutPtrAryInsertAfter(ptr_ary, index_item, new_item)
T_PTR_ARY ptr_ary;
T_PTR index_item;
T_PTR new_item;
```

TutPtrAryInsertBefore — insert a new item in the array before the specified item

```
T_BOOL TutPtrAryInsertBefore(ptr_ary, index_item, new_item)
T_PTR_ARY ptr_ary;
T_PTR index_item;
T_PTR new_item;
```

TutPtrAryInsertItemAtIndex — insert item in the array at the specified index

```
T_BOOL TutPtrAryInsertItemAtIndex(ptr_ary, new_item, index)
T_PTR_ARY ptr_ary;
T_PTR new_item;
T_INT4 index
```

TutPtrAryItemExists — verify that an item exists in the array

```
T_BOOL TutPtrAryItemExists(ptr_ary, item)
T_PTR_ARY ptr_ary;
T_PTR item;
```

TutPtrAryRemove — remove an item from a pointer array

```
T_BOOL TutPtrAryRemove(ptr_ary, item)
T_PTR_ARY ptr_ary;
T_PTR item;
```

TutPtrAryRemoveAll — remove all instances of an item from an array

```
T_BOOL TutPtrAryRemoveAll(ptr_ary, item, counter)
T_PTR_ARY ptr_ary;
T_PTR item;
T_INT4 *counter;
```

TutPtrAryReplaceItemAtIndex — replace item at the specified index with a new one

```
T_BOOL TutPtrAryReplaceItemAtIndex(ptr_ary, new_item, index,
old_item)
T_PTR_ARY ptr_ary;
T_PTR new_item;
T_INt4 index;
T_PTR *old_item;
```

TutPtrArySort — sort the pointer array

```
T_BOOL TutPtrArySort(ptr_ary, compare_func)
T_PTR_ARY ptr_ary;
T_PTR_ARY_CMP_FUNC compare_func;
```

TutPtrArySwapItems — exchange the two specified elements in the array

```
T_BOOL TutPtrArySwapItems(ptr_ary, index1, index2)
T_PTR_ARY ptr_ary;
T_INT4 index1;
T_INT4 index2;
```

Socket Functions (TutSocket*)

The TutSocket* functions are used to work with sockets on a low level. These routines can be useful if you are trying to mix SmartSockets connections and sockets.

TutSocketAccept — accept socket connection from client

```
T_INT4 TutSocketAccept(server_fd)
T_INT4 server_fd;
```

TutSocketCheck — check whether data can be read from or written to socket

```
T_BOOL TutSocketCheck(socket_fd, check_mode, timeout)
T_INT4 socket_fd;
T_IO_CHECK_MODE check_mode;
T_REAL8 timeout;
```

TutSocketCreateClientLocal — create the client side of a local socket

```
T_INT4 TutSocketCreateClientLocal(file_name)
T_STR file_name;
```

TutSocketCreateClientTcp — create the client side of a TCP/IP socket

```
T_INT4 TutSocketCreateClientTcp(node_name, port_num)
T_STR node_name;
T_INT4 port_num;
```

TutSocketCreateServerLocal — create the server side of a local socket

```
T_INT4 TutSocketCreateServerLocal(file_name)
T_STR file_name;
```

TutSocketCreateServerTcp — create the server side of a TCP/IP socket

```
T_INT4 TutSocketCreateServerTcp(port_num)
T_INT4 port_num;
```

TutSocketGetBlockMode — determine the block mode of a socket

```
T_BOOL TutSocketGetBlockMode(socket_fd, block_mode_return)
T_INT4 socket_fd;
T_BOOL *block_mode_return;
```

TutSocketRecvAll — read multiple times from a socket until all data is received

```
T_INT4 TutSocketRecvAll(fd, buf, size, flags)
T_INT4 fd;
T_PTR buf;
T_INT4 size;
T_INT4 flags;
```

TutSocketRecvTimeout — read data from a socket with a timeout

```
T_INT4 TutSocketRecvTimeout(socket_fd, buf, size, flags, timeout,
timeout_reached_return)
T_INT4 socket_fd;
T_PTR buf;
T_INT4 size;
T_INT4 flags;
T_REAL8 timeout;
T_BOOL *timeout_reached_return;
```

TutSocketSetBlockMode — set the block mode of a socket

```
T_BOOL TutSocketSetBlockMode(socket_fd, block_mode)
T_INT4 socket_fd;
T_BOOL block_mode;
```

String Functions (Tut*Str*)

These functions are used to convert and duplicate strings.

TutRealToStr — convert a real number to a string using the option Real_Number_Format

```
T_STR TutRealToStr(num)
T_REAL8 num;
```

TutStrDup — duplicate a string

```
T_STR TutStrDup(str)
T_STR str;
```

TutStrLwr — convert string to lower case

```
T_STR TutStrLwr(dest, src)
T_STR dest;
T_STR src;
```

TutStrUpr — convert string to upper case

```
T_STR TutStrUpr(dest, src)
T_STR dest;
T_STR src;
```

Thread-Related Functions

These functions provide a portable thread API that is loosely based on the POSIX standard 1003.1 (more commonly called Pthreads). This API is supported on all operating systems with kernel-level threads.

Thread Functions

The TutThread* functions manipulate threads.

TutThreadCreate — create a thread

```
T_THREAD TutThreadCreate(thread_func, thread_arg, ...)
T_THREAD_FUNC thread_func;
T_PTR thread_arg;
```

TutThreadCreateVa — create a thread (`va_list` version)

```
T_THREAD TutThreadCreateVa(thread_func, thread_arg, var_arg_list)
T_THREAD_FUNC thread_func;
T_PTR thread_arg;
va_list var_arg_list;
```

TutThreadDetach — allow a thread to run to completion without requiring the operating system to remember its exit code

```
T_THREAD TutThreadDetach(thread_func, thread_arg, var_arg_list)
T_THREAD_FUNC thread_func;
T_PTR thread_arg;
va_list var_arg_list;
```

TutThreadEqual — compare two thread handles for equality

```
T_BOOL TutThreadEqual(thread_1, thread_2)
T_THREAD thread_1;
T_THREAD thread_2;
```

TutThreadExit — exit from a thread

```
void TutThreadExit(exit_code)
T_PTR exit_code;
```

TutThreadSelf — return the `T_THREAD` of the current thread

```
T_THREAD TutThreadSelf()
```

TutThreadWait — wait for another thread to exit

```
T_BOOL TutThreadWait(thread, exit_return)
T_THREAD thread;
T_PTR *exit_return;
```

Mutex Functions

The TutMutex* functions manipulate mutual exclusion synchronization (mutex) objects.

TutMutexCreate — create a mutex object

```
T_MUTEX TutMutexCreate(initial_state)
T_BOOL initial_state;
```

TutMutexCreateFast — create a high-performance mutex object

```
T_MUTEX TutMutexCreateFast(initial_state)
T_BOOL initial_state;
```

TutMutexDestroy — destroy a mutex object

```
T_BOOL TutMutexDestroy(mutex)
T_MUTEX mutex;
```

TutMutexLock — lock a mutex object

```
T_BOOL TutMutexLock(mutex, timeout)
T_MUTEX mutex;
T_REAL8 timeout;
```

TutMutexLockFast — lock a high-performance mutex object

```
T_BOOL TutMutexLockFast(mutex)
T_MUTEX mutex;
```

TutMutexUnlock — unlock a mutex object

```
T_BOOL TutMutexUnlock(mutex)
T_MUTEX mutex;
```

Condition Variable Functions

The TutCond* functions manipulate condition variable synchronization objects.

TutCondCreate — create a condition variable

```
T_COND TutCondCreate()
```

TutCondDestroy — destroy a condition variable

```
T_BOOL TutCondDestroy(cond)
T_COND cond;
```

TutCondWait — wait to be awakened by a condition variable

```
T_BOOL TutCondWait(cond, mutex, timeout)
T_COND cond;
T_MUTEX mutex;
T_REAL8 timeout;
```

TutCondWakeAll — wake all threads waiting for a condition variable

```
T_BOOL TutCondWakeAll(cond)
T_COND cond;
```

Read/Write Mutex Functions

The TutRwMutex* functions manipulate read and write mutex synchronization objects.

TutRwMutexCreate — create a multiple-reader / single-writer mutex

```
T_RW_MUTEX TutRwMutexCreate(initial_state, read_quota)
T_BOOL initial_state;
T_INT4 read_quota;
```

TutRwMutexDestroy — destroy a multiple-reader / single-writer mutex

```
T_BOOL TutRwMutexDestroy(rw_mu)
```

TutRwMutexGetReadQuota — determine the current simultaneous reader quota of a read or write mutex

```
T_BOOL TutRwMutexGetReadQuota(rw_mutex, read_quota_return)
T_RW_MUTEX rw_mutex;
T_INT4 *read_quota_return;
```

TutRwMutexReadLock — read-lock a read or write mutex

```
T_BOOL TutRwMutexReadLock(rw_mutex)
T_RW_MUTEX rw_mutex;
```

TutRwMutexReadLocked — check to see if a read or write mutex is read-locked by the current thread

```
T_BOOL TutRwMutexReadLocked(rw_mutex)
T_RW_MUTEX rw_mutex;
```

TutRwMutexSetReadQuota — set the simultaneous reader quota of a read or write mutex

```
T_BOOL TutRwMutexSetReadQuota(rw_mutex, read_quota)
T_RW_MUTEX rw_mutex;
T_INT4 read_quota;
```

TutRwMutexUnlock — release a lock on a read or write mutex

```
T_BOOL TutRwMutexUnlock(rw_mutex)
T_RW_MUTEX rw_mutex;
```

TutRwMutexUpgradeLock — upgrade a read-lock on a read or write mutex to a write-lock

```
T_BOOL TutRwMutexUpgradeLock(rw_mutex, timeout)
T_RW_MUTEX rw_mutex;
T_REAL8 timeout;
```

TutRwMutexWriteLock — write-lock a read or write mutex

```
T_BOOL TutRwMutexWriteLock(rw_mutex)
T_RW_MUTEX rw_mutex;
```

TutRwMutexWriteLocked — check to see if a read or write mutex is write-locked by the current thread

```
T_BOOL TutRwMutexWriteLocked(rw_mutex)
T_RW_MUTEX rw_mutex;
```

Thread-Specific Data Functions

The TutTsd* functions manipulate thread-specific data.

TutTsdGetValue — determine the value of a thread-specific data key

```
T_BOOL TutTsdGetValue(key, value_return)
T_TSD_KEY key;
T_PTR *value_return;
```

TutTsdKeyCreate — create a thread-specific data key

```
T_BOOL TutTsdKeyCreate(key_return, destructor_func)
T_TSD_KEY *key_return;
T_TSD_KEY_DESTRUCTOR_FUNC destructor_func;
```

TutTsdKeyDestroy — destroy a thread-specific data key

```
T_BOOL TutTsdKeyDestroy(key)
T_TSD_KEY key;
```

TutTsdSetValue — set the value of a thread-specific data key

```
T_BOOL TutTsdSetValue(key, value)
T_TSD_KEY key;
T_PTR value;
```

Time Functions

These functions are used to work with time values. Functions are included to retrieve various kinds of time, set various kinds of time, set up callbacks which are executed when time changes and more.

TutGetCurrentTime — determine the current SmartSockets data time

```
T_REAL8 TutGetCurrentTime()
```

TutGetCurrentTimeStruct — determine the current SmartSockets data time as a time struct

```
T_BOOL TutGetCurrentTimeStruct(time_return)
T_TIME_STRUCT *time_return;
```

TutGetWallTime — determine the current wall clock (computer) time

```
T_REAL8 TutGetWallTime()
```

TutGetWallTimeStruct — determine the current wall clock (computer) time as a time struct

```
T_BOOL TutGetWallTimeStruct(time_return)
T_TIME_STRUCT *time_return;
```

TutSetCurrentTime — set the current data time for the process

```
T_BOOL TutSetCurrentTime(time_val)
T_REAL8 time_val;
```

TutSetCurrentTimeStruct — set the current data time for the process, using the most precise method

```
T_BOOL TutSetCurrentTimeStruct(timep)
T_TIME timep;
```

TutTimeChangeCbCreate — create a time change callback

```
T_CB TutTimeChangeCbCreate(func, arg)
T_TIME_CHANGE_CB_FUNC func;
T_PTR arg;
```

TutTimeChangeCbLookup — look up a time change callback

```
T_CB TutTimeChangeCbLookup(func, arg)
T_TIME_CHANGE_CB_FUNC func;
T_PTR arg;
```

TutTimeCvtCreate — create a time converter

```
T_TIME_CVT TutTimeCvtCreate(name, num_to_str_func, str_to_num_func,
destroy_func)
T_STR name;
T_TIME_CVT_NUM_TO_STR_FUNC num_to_str_func;
T_TIME_CVT_STR_TO_NUM_FUNC str_to_num_func;
T_TIME_CVT_DESTROY_FUNC destroy_func;
```

TutTimeCvtDestroy — destroy a time converter

```
T_BOOL TutTimeCvtDestroy(time_cvtr)
T_TIME_CVT time_cvtr;
```

TutTimeCvtLookup — look up a time converter by name

```
T_TIME_CVT TutTimeCvtLookup(name)
T_STR name;
```

TutTimeCvtNumToStr — convert a numeric time value to a string

```
T_BOOL TutTimeCvtNumToStr(time_cvtr, time_num, time_str_return)
T_TIME_CVT time_cvtr;
T_REAL8 time_num;
T_STR *time_str_return;
```

TutTimeCvtStrToNum — convert a string time value to a number

```
T_BOOL TutTimeCvtStrToNum(time_cvtr, time_str, time_num_return)
T_TIME_CVT time_cvtr;
T_STR time_str;
T_REAL8 *time_num_return;
```

TutTimeCvtTraverse — traverse all time converters

```
T_PTR TutTimeCvtTraverse(traverse_func, traverse_arg)
T_TIME_CVT_TRAV_FUNC traverse_func;
T_PTR traverse_arg;
```

TutTimeNumToStr — convert a time number to string using the Time_Format option

```
T_STR TutTimeNumToStr(time_num)
T_REAL8 time_num;
```

TutTimeStrToNum — convert a string time to a number using the Time_Format option

```
T_BOOL TutTimeStrToNum(time_str, time_num_return)
T_STR time_str;
T_REAL8 *time_num_return;
```

XML Functions

These functions are used to clone, create, and destroy XML objects, as well as to set and return XML strings.

TutXmlClone — make a copy of an XML object

```
T_XML TutXmlClone(xml)
T_XML xml;
```

TutXmlCreate — create a new XML object

```
T_XML T_ENTRY T_EXPORT TutXmlCreate(xml_str)
T_STR xml_str
```

TutXmlCreateStatic — create an XML object from a static buffer

```
T_XML TutXmlCreateStatic(xml_str)
T_XML xml_str;
```

TutXmlDestroy — destroy an XML object

```
T_BOOL TutXmlDestroy(xml)
T_XML xml;
```

TutXmlGetStr — return the XML string associated with the XML object

```
T_BOOL TutXmlGetStr(xml, str_return)
T_XML xml;
T_STR *str_return;
```

TutXmlSetStr — set the XML string

```
T_BOOL TutXmlSetStr(xml, str)
T_XML xml;
T_STR str;
```

Miscellaneous Functions

The miscellaneous functions are those that do not fit nicely into any other categories.

TutExit — call exit handlers and terminate process

```
void TutExit(status)
T_INT4 status;
```

TutGetenv — get a SmartSockets environment variable

```
T_BOOL TutGetenv(name, value_return)
T_STR name;
T_STR *value_return;
```

TutGetOutFlushFunc — find the function used for TutOutFlush output

```
T_OUT_VA_FLUSH_FUNC TutGetOutFlushFunc()
```

TutGetOutputFunc — find the function used for TutOut output

```
T_OUT_VA_FUNC TutGetOutputFunc()
```

TutOut — unconditional printf-style output to a SmartSockets process output window

```
void TutOut(format_str, ...)
T_STR format_str;
```

TutOutFlush — unconditional printf-style output to `stdout` and flush

```
void TutOutFlush(format_str, ...)
T_STR format_str;
```

TutPutenv — set a SmartSockets environment variable

```
T_BOOL TutPtrArySwapItems(name_and_value)
T_STR name_and_value;
```

TutSetOutFlushFunc — set the function to use for TutOutFlush output

```
void TutSetOutFlushFunc(flush_func)
T_OUT_VA_FLUSH_FUNC flush_func;
```

TutSetOutputFunc — set the function to use for TutOut output

```
void TutSetOutputFunc(output_func)
T_OUT_VA_FUNC output_func;
```

TutSleep — suspend execution for specified interval

```
T_BOOL TutSleep(interval)
T_REAL8 interval;
```

TutSystem — execute a shell command

```
T_INT4 TutSystem(command)
T_STR command;
```

TutWinSetOutputListBox — set Windows output function

```
void TutWinSetOutputListBox(list_box_hwnd, max_num_lines)
HWND list_box_hwnd;
T_INT4 max_num_lines;
```

Chapter 5 Error Codes

TIBCO SmartSockets provides some simple error handling for its C API functions. The error handling system provides additional information about why a function call failed. This is most often found in functions that return a TRUE or FALSE status.

For function calls that support error handling, the global SmartSockets error number is set to a value that provides additional information about why the function failed. This global error number can be retrieved with TutErrNumGet and set with TutErrNumSet. In multithreaded programs, the global error number is stored in a value for each thread.

To simplify logging or printing notification of errors, a string describing the error can be retrieved with TutErrStrGet or TutErrNumToStr.

The Diagnostics section of each function in the *TIBCO SmartSockets Application Programming Interface* describes how error handling is supported. The error number should only be used immediately after a failure value is returned by a function that supports error handling. At any other time, the value of the error number is meaningless.

Topics

Error Codes

Table 2 shows all the error codes and numbers that the SmartSockets IPC and Utilities API functions use.

Table 2 SmartSockets Errors

Number	Error Code	Description
1	T_ERR_C	A C error occurred. Call the function TutErrNumGetC to obtain the related C-specific error value.
2	T_ERR_OS	An operating system error occurred. Call the function TutErrNumGetOs to obtain the related error value specific to the operating system. On OpenVMS, if the error also has a C equivalent, call the function TutErrNumGetC to obtain the related C-specific error value. (TutErrNumGetC returns zero if there is no equivalent C error.)
3	T_ERR_NULL_PTR	Indicates that a null pointer was passed as a function argument in a case where NULL is not allowed.
4	T_ERR_VAL_TOO_SMALL	Indicates a range error where a value was too small to be valid.
5	T_ERR_VAL_TOO_LARGE	Indicates a range error where a value was too large to be valid.
6	T_ERR_ALREADY_EXISTS	The operation, usually creation or insertion, did not succeed because the *object* already exists.
7	T_ERR_VAL_INVALID	Indicates some kind of range error where a value was invalid. There is some overlap between T_ERR_VAL_INVALID versus T_ERR_VAL_TOO_SMALL and T_ERR_VAL_TOO_LARGE.
8	T_ERR_DOESNT_EXIST	The operation (usually looking up an *object*) did not occur because *object* does not exist.

Number	Error Code	Description
9	T_ERR_TIMEOUT_REACHED	The operation did not complete within a certain user-specified amount of time.
10	T_ERR_EOF	Indicates an end of file condition.
11	T_ERR_VAL_SAME	The operation, usually setting a value, did not occur because the new value was the same as the current value.
12	T_ERR_VAL_UNKNOWN	The operation, usually getting a value, did not occur because the current value was unknown.
13	T_ERR_VAL_REQUIRED	The operation did not occur because a certain value was required to be set.
14	T_ERR_TYPE_INVALID	The operation did not occur because of an invalid type.
15	T_ERR_TYPE_MISMATCH	The operation did not occur because of a type mismatch between two operands.
16	T_ERR_INTERRUPTED	The operation did not occur because it was interrupted by a signal.
17	T_ERR_VERIFY_FAILED	The operation, usually setting a value, did not occur because verification of the new value failed. There is some overlap between T_ERR_VERIFY_FAILED versus T_ERR_VAL_INVALID.
18	T_ERR_END_OF_TRAVERSAL	Used to distinguish between a failed traversal and one that traversed all entries.
19	T_ERR_EMPTY	Used to distinguish between a failed traversal and one that traversed no entries.
20	T_ERR_VAL_MISMATCH	The operation did not occur because of a value mismatch between two operands.
21	T_ERR_TYPE_UNKNOWN	The operation did not occur because the type of the operand was unknown.

Number	Error Code	Description
24	T_ERR_FILE_CORRUPT	The operation did not occur because a file was corrupt.
25	T_ERR_WOULD_BLOCK	The operation did not occur because the socket's block mode is set to FALSE and the operation would have blocked the process from continuing.
28	T_ERR_READ_ONLY	The operation did not occur because the object was read-only and could not be modified.
29	T_ERR_SOCKET	A socket error occurred. Call the function TutErrNumGetSocket to obtain the related socket-specific error value.
31	T_ERR_UNSUPPORTED	The operation did not occur because the function is unsupported in the current context.
500	T_ERR_CONN_INIT_FAILED	Indicates that a connection was not created because of an incompatibility or error during the one-time connection initialization procedure. The most common reason for this error number is mixing incompatible versions of SmartSockets on your network.
502	T_ERR_SRV_NOT_CONNECTED	The operation did not occur because a connection to RTserver was not created or does not exist. Check the settings of the Server_* options.
503	T_ERR_SRV_ALREADY_CONNECTED	The operation did not occur because a connection to RTserver already exists.
504	T_ERR_SRV_ACCESS_DENIED	A connection to RTserver was not created because another RTclient or RTserver has the same value for the option Unique_Subject or RTserver reports that the maximum number of client connections was exceeded.

Number	Error Code	Description
505	T_ERR_MSG_CORRUPT	The operation did not occur because the message was not in the proper format.
507	T_ERR_MSG_INVALID	The operation did not occur because the message was not a valid message. Messages have an internal fingerprint which ensures that only valid messages are allowed.
508	T_ERR_MSG_EOM	Indicates an end of message condition. The operation did not occur because the *message* does not have any more fields (the current field is at the end of the message).
509	T_ERR_FT_INVALID	The operation did not occur because the specified field type was not valid.
510	T_ERR_FT_MISMATCH	The operation did not occur because the current field is not of the proper type.
512	T_ERR_IN_PROGRESS	The operation did not occur because a similar operation is already in progress. This can happen when trying to read from a connection in a connection read callback.
514	T_ERR_FAILURE_DETECTED	A network failure was detected on a connection. The connection should be destroyed and then a new connection created if necessary.
515	T_ERR_LICENSE_DENIED	The operation did not occur because RTlm denied access to a license needed for the operation to complete.
517	T_ERR_IN_USE	The operation did not occur because some other *object* was already in use and the operation requires that the *object* not exist.

Number	Error Code	Description
518	T_ERR_GMD_SENDER_TIMEOUT	GMD failed because the message was not acknowledged within time to meet the connection delivery timeout property. This error number is used only in GMD_FAILURE messages, not in the global SmartSockets error number accessible with TutErrNumGet. This error number is applicable for both TipcConn* GMD and TipcSrv* GMD.
519	T_ERR_GMD_RECEIVER_TIMEOUT	GMD failed because a receiving RTclient (with the option Server_Disconnect_Mode set to `warm`) disconnected from RTserver but did not reconnect within time to meet the RTserver option Client_Reconnect_Timeout. This error number is used only in GMD_FAILURE messages, not in the global SmartSockets error number accessible with TutErrNumGet. This error number is applicable only for TipcSrv* GMD.
520	T_ERR_GMD_RECEIVER_EXIT	GMD failed because a receiving RTclient (with the option Server_Disconnect_Mode set to `gmd_failure`) disconnected from RTserver. This error number is used only in GMD_FAILURE messages, not in the global SmartSockets error number accessible with TutErrNumGet. This error number is applicable only for TipcSrv* GMD.
521	T_ERR_GMD_NO_RECEIVERS	GMD failed because there were no receiving RTclients for the message and the RTserver option Zero_Recv_Gmd_Failure was set to `TRUE`. This error number is used only in GMD_FAILURE messages, not in the global SmartSockets error number accessible with TutErrNumGet. This error number is applicable only for TipcSrv* GMD.

Number	Error Code	Description
522	T_ERR_PROTOCOL_MISMATCH	The operation did not occur because the feature is not supported in all the SmartSockets processes that are running. The most common reason for this error number is mixing incompatible versions of SmartSockets on your network.
525	T_ERR_SRV_CTLMSG_REJECTED	The operation did not occur because the control message was rejected.
4006	T_ERR_MAX_CLIENTS_EXCEEDED	The operation did not occur because the maximum allowed number of clients was exceeded.
4007	T_ERR_CONN_REQUEST_REJECTED	The operation did not occur because the connection request was rejected.
4009	T_ERR_DISCONNECT_MODE_INVALID	The operation did not occur because the disconnect mode was invalid.
4010	T_ERR_AUTHENTICATION_FAILED	The operation did not occur because authentication failed.
4011	T_ERR_CLIENT_NOT_LOCAL	The operation did not occur because the client is not local.
4012	T_ERR_MAX_CLIENT_LICENSES_EXCEEDED	The operation did not occur because the maximum allowed number of client licenses was exceeded.
4013	T_ERR_CLIENT_CREATE_REJECTED	The operation did not occur because the client creation was rejected.
4014	T_ERR_CLIENT_NAME_NOT_UNIQUE	The operation did not occur because the client name was not unique.
4015	T_ERR_CLIENT_UNIQUENESS_CHECK_IN_PROGRESS	A client uniqueness check is in progress.
4016	T_ERR_AUTHORIZATION_DENIED	The operation did not occur because authorization was denied.
4017	T_ERR_SUBJECT_INVALID	The operation did not occur because the subject was invalid.

Number	Error Code	Description
4018	T_ERR_SUBJECT_ALREADY_SUBSCRIBED	The operation did not occur because *subject* was already subscribed to.
4019	T_ERR_SUBJECT_NOT_SUBSCRIBED	The operation did not occur because *subject* was not subscribed to.
4020	T_ERR_DUP_CLIENT_MID	The operation did not occur because a duplicate member ID was used.

Chapter 6 Contacting Technical Support

For problems or questions, contact TIBCO Product Support. Go to the TIBCO website at `http://www.tibco.com` for the latest information on how to contact TIBCO Product Support. If you do not have access to the Web, you can also contact Technical Support through e-mail or telephone:

- support@tibco.com
- 800-883-8050 extension 4 (US only)
- 650-965-8050 extension 4

Standard technical support is available Monday through Friday between 5:00 am - 5:00 pm Pacific Standard Time. There are other support options, including 24x7 (24 hours a day, every day) and premium (immediate response instead of the standard 4 hour response time). The TIBCO website has complete information on all technical support options. Contact TIBCO Software about purchasing their Premium Support service.

Before contacting Product Support, be sure you have your Customer ID and the version and revision numbers of SmartSockets and other TIBCO products that you are using. Be prepared to send debug files, stack trace files, and screen shots of output as attachments to Technical Support to aid us in isolating and identifying the problem.

To find out your Customer ID and other license information, you can use the SmartSockets `rtlic` shell script. At a SmartSockets command prompt, enter:

```
rtlic
```

You'll see your license information displayed. For more information on using `rtlic`, see the *TIBCO SmartSockets Utilities* reference.

What to have

Technical Support needs additional information about your problem in order to help you. Give them the full error message you received, your logging information, and stack traces if you had a core dump. Also tell them about your cloud topology, and the exact version(s) of SmartSockets and any associated products, including any hotfixes, you are using.

If you have a core dump, do not send it unless specifically requested. Instead, rename it so that it will not be accidentally overwritten and send a stack trace instead. If Technical Support wants your core file, they will ask for it. You should also generate and save a checksum, using 'md5sum' if possible, so that the integrity of your core file can be confirmed if you are later asked for it.

In order to ensure that none of the data related to your problem is overwritten, it is strongly recommended that you create an archive (for example, using 'tar') containing the core file, any logs, and a copy of your application and its source code (if this is a problem with a client and not RTserver). This is especially important with development applications as rebuilding the application will make your core file infinitely less useful.

Stack Traces

Stack traces help find where a problem started when a process results in a core dump. A stack trace takes a core file in as an argument, and gives you a list of functions that called each other, all the way back to where the error occurred.

A stack trace looks like this:

```
0 TipcSomethingOrOther(stuff in here)
1 TipcSomethingElse(stuff in here)
2 TipcSomethingMore(stuff in here)
3 TipcFirstFunction(stuff here)
```

This tells you that a call was made to `TipcFirstFunctions`, which made a call to `TipcSomethingMore`, which called `TipcSomethingElse`, which called `TipcSomethingOrOther`, which failed. The stuff between the parenthesis is more information. If you compiled your program with the -g option, the information between the parenthesis is more helpful. For example, you can find the line number for each function and other things.

This is how to generate a stack trace:

1. Run your native command line debugger against the executable the created the core dump:

   debugger executable `core`

   For example, if you are using the DBX debugger and the core dump came from the RTserver, use this command:

   ```
   dbx $RTHOME/bin/$RTARCH/rtserver core
   ```

2. When the debugger prompt appears, enter `where`

   For example:

   ```
   dbx> where
   ```

Cut and paste or copy the resulting stack trace into an email to Technical Support. If possible, configure your mail client so it does not insert additional line breaks. No additional formatting is needed.

Core Files

If you are asked to send a core file, first make sure you have an *uncompressed* copy of it in a safe place. Then, generate a checksum of the file, using the 'md5sum' utility if possible. Next, compress it using `bzip2` if available (usually available on Linux installations), otherwise using gzip (pre-compiled binaries of gzip can be obtained from www.gzip.org). Finally, generate a checksum of the compressed file as well. This saves time by allowing Technical Support to immediately detect and diagnose transmission errors. Technical Support will provide details on how to send the core file.

Index

A

C

D

E

F

G

H

I

M

O

P

Q

R

S

T

V

W