```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++

$Id: tcp.c,v 1.17 2001/08/10 01:59:00 davidc Exp $

Copyright (c) 2001 BeComm Corporation

Filename:

    tcp.c

Abstract:

    The implementation of the TCP portion of the SocketLib API. This is
the
    Win32 (NT/95/CE) version using the standard sockets interface
exported
    by WinSock.

Owner:

    David Costanzo (davidc)

--------------------------------------------------------------------------
----*/

#include <windows.h>
#include <winsock2.h>
#include <process.h>
#include <assert.h>

#define SOS_DEBUG_ZONE "/classes/socket/tcp"
#include <sosstrings.h>
#include <sosisocket.h>
#include "socket.h"
#include "tcp.h"

SOS_SOURCE_VERSION("$Id: tcp.c,v 1.17 2001/08/10 01:59:00 davidc Exp $");

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
Constants
--------------------------------------------------------------------------
----*/
```

```
40 #define ERROR_BIND_FAILED           SOS_Error
41 #define ERROR_SOCKET_FAILED         SOS_Error
42 #define ERROR_BIND_FAILED           SOS_Error
43 #define ERROR_CONNECT_FAILED        SOS_Error
44 #define ERROR_GETSOCKNAME_FAILED    SOS_Error
45
46
47 /* size of buffer to allocate for each read */
48 static const size_t g_TcpRecvSize = 1500;
49
50 /* largest amount to call send() with */
51 static const size_t g_MaxSendSize = 2048;
52
53
54 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…++++++
55 Data Types
56 ----------------------------------------------------------------------
…----*/
57
58 /* TCP socket context to be passed back and
59    forth with caller of this library */
60 struct _SOS_ISOCKET_TCP {
61     SOCKET                      NativeSocket;
62     int                         RefCount;
63     SOS_ISOCKET_TCP_CONNECT     ConnectedCallback;
64     SOS_ISOCKET_TCP_RECEIVE     RecvCallback;
65     SOS_ISOCKET_CLOSED          ClosedCallback;
66     SOS_ISOCKET_ADDRESS         SocketAddr;
67     void*                       UserContext;
68
69     SOS_BOOLEAN                 PeerHasClosed;
70     SOS_BOOLEAN                 WeHaveClosed;
71
72     unsigned long               NativeConnectionThread;
73 };
74
75
76 struct _SOS_ISOCKET_TCP_LISTEN {
77     SOS_ISOCKET_TCP     Socket;
78     unsigned long       NativeListenThread;
79 };
80
81
```

```
83
84  typedef struct _CONNECTION_PARAM            CONNECTION_PARAM;
85
86  typedef enum _TCP_CONNECT_FUNCTION_ID {
87      TCP_CONNECT_FUNCTION_ID_TcpOnConnect,
88      TCP_CONNECT_FUNCTION_ID_TcpOnReceive,
89      TCP_CONNECT_FUNCTION_ID_TcpOnClose,
90  } TCP_CONNECT_FUNCTION_ID;
91
92  struct _CONNECTION_PARAM {
93
94      TCP_CONNECT_FUNCTION_ID  FunctionId;
95
96      union _CONNECTION_PARAMS {
97
98          struct _CONNECTION_PARAM_ON_CONNECT {
99              SOS_ISOCKET_TCP *       Socket;
100             void *                  OutUserContext;
101         } OnConnect;
102
103
104         struct _CONNECTION_PARAM_ON_RECEIVE {
105             SOS_ISOCKET_TCP *       Socket;
106             void*                   Buffer;
107             size_t                  BufferLength;
108             SOS_FREEPROC            BufferFree;
109         } OnReceive;
110
111
112         struct _CONNECTION_PARAM_ON_CLOSE {
113             SOS_ISOCKET_TCP *       Socket;
114         } OnClose;
115
116     } Params;
117 };
118
119
120 /* TcpSendWorkerProc types */
121
122 typedef struct _TCP_SEND_WORKER_PROC TCP_SEND_WORKER_PROC;
123
124 struct _TCP_SEND_WORKER_PROC {
125     SOCKET      NativeSocket;
126     void *      Buffer;
```

```
128     SOS_STATUS  OutStatus;
129 };
130
131
132 /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
…++++++
133 Internal Routines
134 ----------------------------------------------------------------------------
…----*/
135
136
137 /*++
138 Routine Name:
139
140     TcpSocketCreate
141
142 Routine Description:
143
144     This routine allocates and initializes an SOS_ISOCKET_TCP struct.
145     It does not call any sockets routines.
146     The structure is returned with a reference count of 1.
147
148 Parameters:
149
150     SOCKET NativeSocket - [in]
151         The native socket handle.
152
153     SOS_ISOCKET_TCP_CONNECT ConnectedCallback - [in]
154         The user-defined function to call when a socket connection
155         has been established.
156
157     SOS_ISOCKET_TCP_RECEIVE RecvCallback - [in]
158         The user-defined RecvCallback.
159
160     SOS_ISOCKET_CLOSED ClosedCallback - [in]
161         The user-defined ClosedCallback.
162
163     const struct sockaddr_in* RemoteSockAddr - [in]
164         The remote socket address.
165         This parameter is optional.
166
167     const SOS_ISOCKET_ADDRESS* SocketAddr - [in]
168         The socket address.
169
170     void* UserContext - [in]
```

```
171         The user context to pass into RecvCallback and ClosedCallback.
172
173 Return Value:
174
175     SOS_ISOCKET_TCP* -
176         A pointer to a newly allocated SOS_ISOCKET_TCP structure with
177         a reference count of 1.
178         NULL, if the structure could not be created.
179
180 --*/
181 static
182 SOS_ISOCKET_TCP*
183 TcpSocketCreate(
184     SOCKET                      NativeSocket,
185     SOS_ISOCKET_TCP_CONNECT     ConnectedCallback,
186     SOS_ISOCKET_TCP_RECEIVE     RecvCallback,
187     SOS_ISOCKET_CLOSED          ClosedCallback,
188     const SOS_ISOCKET_ADDRESS * SocketAddr,
189     void*                       UserContext
190 )
191 {
192     SOS_ISOCKET_TCP*  tcpSocket;
193
194     /* allocate enough size for the larger SOS_ISOCKET_TCP_LISTEN */
195     tcpSocket = malloc(sizeof(SOS_ISOCKET_TCP_LISTEN));
196     if (tcpSocket != NULL) {
197
198         tcpSocket->NativeSocket      = NativeSocket;
199         tcpSocket->ConnectedCallback = ConnectedCallback;
200         tcpSocket->ClosedCallback    = ClosedCallback;
201         tcpSocket->RecvCallback      = RecvCallback;
202         tcpSocket->SocketAddr        = *SocketAddr;
203         tcpSocket->UserContext       = UserContext;
204
205
206         tcpSocket->PeerHasClosed = SOS_False;
207         tcpSocket->WeHaveClosed  = SOS_False;
208
209         tcpSocket->RefCount       = 1;
210     }
211
212     return tcpSocket;
213 }
214
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.