```
**********************************************************
    3424 Sat Jul 25 23:07:29 2015
new/usr/src/man/man1m/k5srvutil.1m
4514 k5srvutil(1m): The default keytab file is /etc/krb5/krb5.keytab
**********************************************************
   1 '\" te
   2 .\" Copyright (c) 2006, Sun Microsystems, Inc. All Rights Reserved
   3 .\" The contents of this file are subject to the terms of the Common Development
   4 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
   5 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
   6 .TH K5SRVUTIL 1M "Jul 25, 2015"
   6 .TH K5SRVUTIL 1M "Aug 29, 2006"
   7 .SH NAME
   8 k5srvutil \- host key table (keytab) manipulation utility
   9 .SH SYNOPSIS
  10 .LP
  11 .nf
  12 \fB/usr/sbin/k5srvutil\fR \fIoperation\fR [\fB-ik\fR] [\fB-f\fR \fIfilename\fR]
  13 .fi

  15 .SH DESCRIPTION
  16 .sp
  17 .LP
  18 The \fBk5srvutil\fR command allows a system manager to list or change keys
  19 currently in his keytab or to add new keys to the keytab.
  20 .sp
  21 .LP
  22 The operand \fIoperation\fR must be one of the following:
  23 .sp
  24 .ne 2
  25 .na
  26 \fB\fBlist\fR\fR
  27 .ad
  28 .RS 10n
  29 Lists the keys in a keytab, showing version number and principal name.
  30 .RE

  32 .sp
  33 .ne 2
  34 .na
  35 \fB\fBchange\fR\fR
  36 .ad
  37 .RS 10n
  38 Changes all the keys in the keytab to new randomly-generated keys, updating the
  39 keys in the Kerberos server's database to match those by using the \fBkadmin\fR
  40 protocol. If a key's version number does not match the version number stored in
  41 the Kerberos server's database, the operation fails. The old keys are retained
  42 so that existing tickets continue to work. If the \fB-i\fR flag is specified,
  43 \fBk5srvutil\fR prompts for \fByes\fR or \fBno\fR before changing each key. If
  44 the \fB-k\fR option is used, the old and new keys are displayed.
  45 .RE

  47 .sp
  48 .ne 2
  49 .na
  50 \fB\fBdelold\fR\fR
  51 .ad
  52 .RS 10n
  53 Deletes keys that are not the most recent version from the keytab. This
  54 operation should be used at some point after a change operation to remove old
  55 keys. If the \fB-i\fR flag is specified, \fBk5srvutil\fR asks the user whether
  56 the old keys associated with each principal should be removed.
  57 .RE

  59 .sp
  60 .ne 2
```



```
  61 .na
  62 \fB\fBdelete\fR\fR
  63 .ad
  64 .RS 10n
  65 Deletes particular keys in the keytab, interactively prompting for each key.
  66 .RE

  68 .sp
  69 .LP
  70 In all cases, the default keytab file is \fB/etc/krb5/krb5.keytab\fR file unless
  70 In all cases, the default keytab file is \fB/etc/krb5.keytab\fR file unless
  71 this is overridden by the \fB-f\fR option.
  72 .sp
  73 .LP
  74 \fBk5srvutil\fR uses the \fBkadmin\fR(1M) program to edit the keytab in place.
  75 However, old keys are retained, so they are available in case of failure.
  76 .SH OPTIONS
  77 .sp
  78 .LP
  79 The following options are supported:
  80 .sp
  81 .ne 2
  82 .na
  83 \fB\fB-f\fR \fIfilename\fR\fR
  84 .ad
  85 .RS 15n
  86 Specify a keytab file other than the default file, \fB/etc/krb5/krb5.keytab\fR.
  86 Specify a keytab file other than the default file, \fB/etc/krb5.keytab\fR.
  87 .RE

  89 .sp
  90 .ne 2
  91 .na
  92 \fB\fB-i\fR\fR
  93 .ad
  94 .RS 15n
  95 Prompts user before changing keys when using the \fBchange\fR or \fBdelold\fR
  96 operands.
  97 .RE

  99 .sp
 100 .ne 2
 101 .na
 102 \fB\fB-k\fR\fR
 103 .ad
 104 .RS 15n
 105 Displays old and new keys when using the \fBchange\fR operand.
 106 .RE

 108 .SH ATTRIBUTES
 109 .sp
 110 .LP
 111 See \fBattributes\fR(5) for descriptions of the following attributes:
 112 .sp

 114 .sp
 115 .TS
 116 box;
 117 c | c
 118 l | l .
 119 ATTRIBUTE TYPE  ATTRIBUTE VALUE
 120 _
 121 Interface Stability     Committed
 122 .TE

 124 .SH SEE ALSO
```

```
125 .sp
126 .LP
127 \fBktutil\fR(1), \fBkadmin\fR(1M), \fBattributes\fR(5)
```

```
**********************************************************
   20641 Sat Jul 25 23:07:30 2015
new/usr/src/man/man3nsl/rpc_clnt_create.3nsl
5730 Typos in rpc_clnt_create(3nsl) man page
**********************************************************
'\" te
.\"  Copyright 1989 AT&T
.\" Copyright (C) 2009, Sun Microsystems, Inc. All Rights Reserved
.\" The contents of this file are subject to the terms of the Common Development
.\"  See the License for the specific language governing permissions and limitat
.\" the fields enclosed by brackets "[]" replaced with your own identifying info
.TH RPC_CLNT_CREATE 3NSL "Jul 25, 2015"
.TH RPC_CLNT_CREATE 3NSL "Dec 16, 2013"
.SH NAME
rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed,
clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create,
clnt_pcreateerror, clnt_raw_create, clnt_spcreateerror, clnt_tli_create,
clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr,
clnt_door_create \- library routines for dealing with creation and manipulation
of CLIENT handles
.SH SYNOPSIS
.LP
.nf
#include <rpc/rpc.h>

\fBbool_t\fR \fBclnt_control\fR(\fBCLIENT *\fR\fIclnt\fR, \fBconst uint_t\fR \fI
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_create\fR(\fBconst char *\fR\fIhost\fR, \fBconst rpcprog_t
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst char *\fR\fInettype\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_create_timed\fR(\fBconst char *\fR\fIhost\fR, \fBconst rpc
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst char \fR*\fInettype\fR,
     \fBconst struct timeval *\fR\fItimeout\fR);
     \fIconst rpcvers_t\fR \fIversnum\fR, \fIconst\fR \fInettype\fR,
     \fIconst struct timeval *\fR\fItimetout\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_create_vers\fR (\fBconst char *\fR\fIhost\fR,
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBrpcvers_t *\fR\fIvers_outp\fR,
     \fBconst rpcvers_t\fR \fIvers_low\fR, \fBconst rpcvers_t\fR \fIvers_high\fR
     \fBconst char *\fR\fInettype\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_create_vers_timed\fR(\fBconst char *\fR\fIhost\fR,
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBrpcvers_t *\fR\fIvers_outp\fR,
     \fBconst rpcvers_t\fR \fIvers_low\fR, \fBconst rpcvers_t\fR \fIvers_high\fR
     \fBchar *\fR\fInettype\fR, \fBconst struct timeval *\fR\fItimeout\fR);
.fi

.LP
.nf
\fBvoid\fR \fBclnt_destroy\fR(\fBCLIENT *\fR\fIclnt\fR);
.fi

.LP
.nf
```



```
\fBCLIENT *\fR\fBclnt_dg_create\fR(\fBconst int\fR \fIfildes\fR,
     \fBconst struct netbuf *\fR\fIsvcaddr\fR, \fBconst rpcprog_t\fR \fIprognum\
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst uint_t\fR \fIsendsz\fR,
     \fBconst uint_t\fR \fIrecsz\fR);
.fi

.LP
.nf
\fBvoid\fR \fBclnt_pcreateerror\fR(\fBconst char *\fR\fIs\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_raw_create\fR(\fBconst rpcprog_t\fR \fIprognum\fR,
     \fBconst rpcvers_t\fR \fIversnum\fR);
.fi

.LP
.nf
\fBchar *\fR\fBclnt_spcreateerror\fR(\fBconst char *\fR\fIs\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_tli_create\fR(\fBconst int\fR \fIfildes\fR,
     \fBconst struct netconfig *\fR\fInetconf\fR, \fBconst struct netbuf *\fR\fI
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBconst rpcvers_t\fR \fIversnum\fR,
     \fBconst uint_t\fR \fIsendsz\fR, \fBconst uint_t\fR \fIrecsz\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_tp_create\fR(\fBconst char *\fR\fIhost\fR,
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBconst rpcvers_t\fR \fIversnum\fR,
     \fBconst struct netconfig *\fR\fInetconf\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_tp_create_timed\fR(\fBconst char *\fR\fIhost\fR,
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBconst rpcvers_t\fR \fIversnum\fR,
     \fBconst struct netconfig *\fR\fInetconf\fR, \fBconst struct timeval *\fR\f
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_vc_create\fR(\fBconst int\fR \fIfildes\fR,
     \fBconst struct netbuf *\fR\fIsvcaddr\fR, \fBconst rpcprog_t\fR \fIprognum\
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst uint_t\fR \fIsendsz\fR,
     \fBconst uint_t\fR \fIrecsz\fR);
.fi

.LP
.nf
\fBstruct rpc_createerr\fR \fBrpc_createerr\fR;
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_door_create\fR(\fBconst rpcprog_t\fR \fIprognum\fR,
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst uint_t\fR \fIsendsz\fR);
.fi

.SH DESCRIPTION
.sp
.LP
```

\fBRPC\fR library routines allow \fBC\fR language programs to make procedure calls on other machines across the network. First a \fBCLIENT\fR handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.
.sp
.LP
These routines are MT-Safe. In the case of multithreaded applications, the \fB-mt\fR option must be specified on the command line at compilation time. When the \fB-mt\fR option is specified, \fBrpc_createerr\fR becomes a macro that enables each thread to have its own \fBrpc_createerr\fR. See \fBthreads\fR(5).
.SS "Routines"
.sp
.LP
See \fBrpc\fR(3NSL) for the definition of the \fBCLIENT\fR data structure.
.sp
.ne 2
.na
\fB\fBclnt_control()\fR\fR
.ad
.sp .6
.RS 4n
A function macro to change or retrieve various information about a client object. \fIreq\fR indicates the type of operation, and \fIinfo\fR is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of \fIreq\fR and their argument types and what they do are:
.sp
.in +2
.nf
CLSET_TIMEOUT struct timeval * set total timeout
CLGET_TIMEOUT   struct timeval *   get total timeout
.fi
.in -2

If the timeout is set using \fBclnt_control()\fR, the timeout argument passed by \fBclnt_call()\fR is ignored in all subsequent calls. If the timeout value is set to \fB0\fR, \fBclnt_control()\fR immediately returns \fBRPC_TIMEDOUT\fR. Set the timeout parameter to \fB0\fR for batching calls.
.sp
.in +2
.nf
CLGET_SERVER_ADDR struct netbuf * get server's address
CLGET_SVC_ADDR struct netbuf * get server's address
CLGET_FD   int *   get associated file descriptor
CLSET_FD_CLOSE void   close the file descriptor when
                destroying the client handle
                (see \fBclnt_destroy()\fR)
CLSET_FD_NCLOSE void   do not close the file
                descriptor when destroying the client handle
CLGET_VERS   rpcvers_t get the RPC program's version
                number associated with the
                client handle
CLSET_VERS   rpcvers_t set the RPC program's version
                number associated with the
                client handle. This assumes
                that the RPC server for this
                new version is still listening
                at the address of the previous
                version.
CLGET_XID   uint32_t   get the XID of the previous
                remote procedure call
CLSET_XID   uint32_t   set the XID of the next
                remote procedure call
CLGET_PROG   rpcprog_t   get program number

CLSET_PROG   rpcprog_t   set program number
.fi
.in -2

The following operations are valid for connection-oriented transports only:
.sp
.in +2
.nf
CLSET_IO_MODE rpciomode_t*   set the IO mode used
                to send one-way requests. The argument for this operation
                can be either:
   - RPC_CL_BLOCKING   all sending operations block
                until the underlying transport protocol has
                accepted requests. If you specify this argument
                you cannot use flush and getting and setting buffer
                size is meaningless.
   - RPC_CL_NONBLOCKING   sending operations do not
                block and return as soon as requests enter the buffer.
                You can now use non-blocking I/O. The requests in the
                buffer are pending. The requests are sent to
                the server as soon as a two-way request is sent
                or a flush is done. You are responsible for flushing
                the buffer. When you choose RPC_CL_NONBLOCKING argument
                you have a choice of flush modes as specified by
                CLSET_FLUSH_MODE.
CLGET_IO_MODE rpciomode_t*   get the current IO mode
CLSET_FLUSH_MODE rpcflushmode_t*   set the flush mode.
                The flush mode can only be used in non-blocking I/O mode.
                The argument can be either of the following:
        - RPC_CL_BESTEFFORT_FLUSH: All flushes send requests
                in the buffer until the transport end-point blocks.
                If the transport connection is congested, the call
                returns directly.
        - RPC_CL_BLOCKING_FLUSH: Flush blocks until the
                underlying transport protocol accepts all pending
                requests into the queue.
CLGET_FLUSH_MODE rpcflushmode_t*   get the current flush mode.
CLFLUSH rpcflushmode_t   flush the pending requests.
                This command can only be used in non-blocking I/O mode.
                The flush policy depends on which of the following
                parameters is specified:
        - RPC_CL_DEFAULT_FLUSH, or NULL: The flush is done
           according to the current flush mode policy
                (see CLSET_FLUSH_MODE option).
        - RPC_CL_BESTEFFORT_FLUSH: The flush tries
                to send pending requests without blocking; the call
                returns directly. If the transport connection is
                congested, this call could return without the request
                being sent.
        - RPC_CL_BLOCKING_FLUSH: The flush sends all pending
                requests. This call will block until all the requests
                have been accepted by the transport layer.
CLSET_CONNMAXREC_SIZE int*   set the buffer size.
                It is not possible to dynamically
                resize the buffer if it contains data.
                The default size of the buffer is 16 kilobytes.
CLGET_CONNMAXREC_SIZE int*   get the current size of the
                buffer
CLGET_CURRENT_REC_SIZE int*   get the size of
                the pending requests stored in the buffer. Use of this
                command is only recommended when you are in non-blocking
                I/O mode. The current size of the buffer is always zero
                when the handle is in blocking mode as the buffer is not
                used in this mode.
.fi
.in -2

```
The following operations are valid for connectionless transports only:
.sp
.in +2
.nf
CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout
.fi
.in -2

The retry timeout is the time that \fBRPC\fR waits for the server to reply
before retransmitting the request.
.sp
\fBclnt_control()\fR returns \fBTRUE\fR on success and \fBFALSE\fR on failure.
.RE

.sp
.ne 2
.na
\fB\fBclnt_create()\fR\fR
.ad
.sp .6
.RS 4n
Generic client creation routine for program \fIprognum\fR and version
\fIversnum\fR. \fIhost\fR identifies the name of the remote host where the
server is located. \fInettype\fR indicates the class of transport protocol to
use. The transports are tried in left to right order in \fBNETPATH\fR variable
or in top to bottom order in the netconfig database.
.sp
\fBclnt_create()\fR tries all the transports of the \fInettype\fR class
available from the \fBNETPATH\fR environment variable and the netconfig
database, and chooses the first successful one. A default timeout is set and
can be modified using \fBclnt_control()\fR. This routine returns \fINULL\fR if
it fails. The \fBclnt_pcreateerror()\fR routine can be used to print the reason
for failure.
.sp
Note that \fBclnt_create()\fR returns a valid client handle even if the
particular version number supplied to \fBclnt_create()\fR is not registered
with the \fBrpcbind\fR service. This mismatch will be discovered by a
\fBclnt_call\fR later (see \fBrpc_clnt_calls\fR(3NSL)).
.RE

.sp
.ne 2
.na
\fB\fBclnt_create_timed()\fR\fR
.ad
.sp .6
.RS 4n
Generic client creation routine which is similar to \fBclnt_create()\fR but
which also has the additional parameter \fItimeout\fR that specifies the
maximum amount of time allowed for each transport class tried. In all other
respects, the \fBclnt_create_timed()\fR call behaves exactly like the
\fBclnt_create()\fR call.
.RE

.sp
.ne 2
.na
\fB\fBclnt_create_vers()\fR\fR
.ad
.sp .6
.RS 4n
Generic client creation routine which is similar to \fBclnt_create()\fR but
which also checks for the version availability. \fIhost\fR identifies the name
of the remote host where the server is located. \fInettype\fR indicates the
```



```
class transport protocols to be used. If the routine is successful it returns a
client handle created for the highest version between \fIvers_low\fR and
\fIvers_high\fR that is supported by the server. \fIvers_outp\fR is set to this
value. That is, after a successful return \fIvers_low\fR <= \fI*vers_outp\fR <=
\fIvers_high\fR. If no version between \fIvers_low\fR and \fIvers_high\fR is
supported by the server then the routine fails and returns \fBNULL.\fR A
default timeout is set and can be modified using \fBclnt_control()\fR. This
routine returns \fINULL\fR if it fails. The \fBclnt_pcreateerror()\fR routine
can be used to print the reason for failure.
.sp
Note: \fBclnt_create()\fR returns a valid client handle even if the particular
version number supplied to \fBclnt_create()\fR is not registered with the
\fBrpcbind\fR service. This mismatch will be discovered by a \fBclnt_call\fR
later (see \fBrpc_clnt_calls\fR(3NSL)). However, \fBclnt_create_vers()\fR does
this for you and returns a valid handle only if a version within the range
supplied is supported by the server.
.RE

.sp
.ne 2
.na
\fB\fBclnt_create_vers_timed()\fR\fR
.ad
.sp .6
.RS 4n
Generic client creation routine similar to \fBclnt_create_vers()\fR but with
the additional parameter  \fItimeout\fR, which specifies the maximum amount of
time allowed for each transport class tried.   In all other respects, the
\fBclnt_create_vers_timed()\fR call behaves exactly like the
\fBclnt_create_vers()\fR call.
.RE

.sp
.ne 2
.na
\fB\fBclnt_destroy()\fR\fR
.ad
.sp .6
.RS 4n
A function macro that destroys the client's \fBRPC\fR handle. Destruction
usually involves deallocation of private data structures, including \fIclnt\fR
itself. Use of \fIclnt\fR is undefined after calling \fBclnt_destroy()\fR. If
the \fBRPC\fR library opened the associated file descriptor, or
\fBCLSET_FD_CLOSE\fR was set using \fBclnt_control()\fR, the file descriptor
will be closed.
.sp
The caller should call \fBauth_destroy(\fR\fIclnt\fR->\fBcl_auth)\fR (before
calling \fBclnt_destroy()\fR) to destroy the associated \fBAUTH\fR structure
(see \fBrpc_clnt_auth\fR(3NSL)).
.RE

.sp
.ne 2
.na
\fB\fBclnt_dg_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an \fBRPC\fR client for the remote program \fIprognum\fR
and version \fIversnum\fR; the client uses a connectionless transport. The
remote program is located at address \fIsvcaddr\fR. The parameter \fIfildes\fR
is an open and bound file descriptor. This routine will resend the call message
in intervals of 15 seconds until a response is received or until the call times
out. The total time for the call to time out is specified by \fBclnt_call()\fR
(see \fBclnt_call()\fR in \fBrpc_clnt_calls\fR(3NSL)). The retry time out and
the total time out periods can be changed using \fBclnt_control()\fR. The user
```

may set the size of the send and receive buffers with the parameters \fIsendsz\fR and \fIrecvsz\fR; values of \fB0\fR choose suitable defaults. This routine returns \fINULL\fR if it fails.
.RE

.sp
.ne 2
.na
\fB\fBclnt_pcreateerror()\fR\fR
.ad
.sp .6
.RS 4n
Print a message to standard error indicating why a client \fBRPC\fR handle could not be created. The message is prepended with the string \fIs\fR and a colon, and appended with a newline.
.RE

.sp
.ne 2
.na
\fB\fBclnt_raw_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an \fBRPC\fR client handle for the remote program \fIprognum\fR and version \fIversnum\fR. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding \fBRPC\fR server should live in the same address space; (see \fBsvc_raw_create()\fR in \fBrpc_svc_create\fR(3NSL)). This allows simulation of \fBRPC\fR and measurement of \fBRPC\fR overheads, such as round trip times, without any kernel or networking interference. This routine returns \fINULL\fR if it fails. \fBclnt_raw_create()\fR should be called after \fBsvc_raw_create()\fR.
.RE

.sp
.ne 2
.na
\fB\fBclnt_spcreateerror()\fR\fR
.ad
.sp .6
.RS 4n
Like \fBclnt_pcreateerror()\fR, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.
.sp
Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.
.RE

.sp
.ne 2
.na
\fB\fBclnt_tli_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an \fBRPC\fR client handle for the remote program \fIprognum\fR and version \fIversnum\fR. The remote program is located at address \fIsvcaddr\fR. If \fIsvcaddr\fR is \fINULL\fR and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if \fIsvcaddr\fR is \fINULL\fR, \fBRPC_UNKNOWNADDR\fR error is set. \fIfildes\fR is a file descriptor which may be open, bound and connected. If it is \fBRPC_ANYFD\fR, it opens a file descriptor on the transport specified by \fInetconf\fR. If \fIfildes\fR is \fBRPC_ANYFD\fR and \fInetconf\fR is \fINULL\fR, a \fBRPC_UNKNOWNPROTO\fR error

is set. If \fIfildes\fR is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters \fIsendsz\fR and \fIrecvsz\fR; values of \fB0\fR choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), \fBclnt_tli_create()\fR calls appropriate client creation routines. This routine returns \fINULL\fR if it fails. The \fBclnt_pcreateerror()\fR routine can be used to print the reason for failure. The remote \fBrpcbind\fR service (see \fBrpcbind\fR(1M)) is not consulted for the address of the remote service.
.RE

.sp
.ne 2
.na
\fB\fBclnt_tp_create()\fR\fR
.ad
.sp .6
.RS 4n
Like \fBclnt_create()\fR except \fBclnt_tp_create()\fR tries only one transport specified through \fInetconf\fR.
.sp
\fBclnt_tp_create()\fR creates a client handle for the program \fIprognum\fR, the version \fIversnum\fR, and for the transport specified by \fInetconf\fR. Default options are set, which can be changed using \fBclnt_control()\fR calls. The remote \fBrpcbind\fR service on the host \fIhost\fR is consulted for the address of the remote service. This routine returns \fINULL\fR if it fails. The \fBclnt_pcreateerror()\fR routine can be used to print the reason for failure.
.RE

.sp
.ne 2
.na
\fB\fBclnt_tp_create_timed()\fR\fR
.ad
.sp .6
.RS 4n
Like \fBclnt_tp_create()\fR except \fBclnt_tp_create_timed()\fR has the extra parameter \fItimeout\fR which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the \fBclnt_tp_create_timed()\fR call behaves exactly like the \fBclnt_tp_create()\fR call.
.RE

.sp
.ne 2
.na
\fB\fBclnt_vc_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an \fBRPC\fR client for the remote program \fIprognum\fR and version \fIversnum\fR; the client uses a connection-oriented transport. The remote program is located at address \fIsvcaddr\fR. The parameter \fIfildes\fR is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters \fIsendsz\fR and \fIrecvsz\fR; values of \fB0\fR choose suitable defaults. This routine returns \fINULL\fR if it fails.
.sp
The address \fIsvcaddr\fR should not be \fINULL\fR and should point to the actual address of the remote program. \fBclnt_vc_create()\fR does not consult the remote \fBrpcbind\fR service for this information.
.RE

.sp
.ne 2
.na

```
\fB\fBrpc_createerr\fR\fR
.ad
.sp .6
.RS 4n
A global variable whose value is set by any \fBRPC\fR client handle creation
routine that fails.  It is used by the routine \fBclnt_pcreateerror()\fR to
print the reason for the failure.
.sp
In multithreaded applications, \fBrpc_createerr\fR becomes a macro which
enables each thread to have its own \fBrpc_createerr\fR.
.RE

.sp
.ne 2
.na
\fB\fBclnt_door_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an RPC client handle over doors for the given program
\fIprognum\fR and version \fIversnum\fR. Doors is a transport mechanism that
facilitates fast data transfer between processes on the same machine. The user
may set the size of the send buffer with the parameter \fIsendsz\fR. If
\fIsendsz\fR is 0, the corresponding default buffer size is 16 Kbyte. The
\fBclnt_door_create()\fR routine returns \fINULL\fR if it fails and sets a
value for \fBrpc_createerr\fR.
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5)  for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Architecture    All
_
Interface Stability     Committed
_
MT-Level        MT-Safe
.TE

.SH SEE ALSO
.sp
.LP
\fBrpcbind\fR(1M), \fBrpc\fR(3NSL), \fBrpc_clnt_auth\fR(3NSL),
\fBrpc_clnt_calls\fR(3NSL), \fBrpc_svc_create\fR(3NSL),
\fBsvc_raw_create\fR(3NSL), \fBthreads\fR(5), \fBattributes\fR(5)
```

```
**********************************************************
    6662 Sat Jul 25 23:07:30 2015
new/usr/src/man/man3nsl/rpcbind.3nsl
5657 Typo in rpcbind(3nsl): ssvcaddr
**********************************************************
'\" te
.\" Copyright 2014 Nexenta Systems, Inc.  All Rights Reserved.
.\"  Copyright 1989 AT&T  Copyright (c) 1997, Sun Microsystems, Inc.  All Rights
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH RPCBIND 3NSL "Jul 25, 2015"
.TH RPCBIND 3NSL "Nov 24, 2014"
.SH NAME
rpcbind, rpcb_getmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set,
rpcb_unset \- library routines for RPC bind service
.SH SYNOPSIS
.LP
.nf
#include <rpc/rpc.h>



\fBstruct rpcblist *\fR\fBrpcb_getmaps\fR(\fBconst struct netconfig *\fR\fInetco
     \fBconst char *\fR\fIhost\fR);
.fi

.LP
.nf
\fBbool_t\fR  \fBrpcb_getaddr\fR(\fBconst rpcprog_t\fR  \fIprognum\fR, \fBconst
     \fBconst struct netconfig *\fR\fInetconf\fR, \fBstruct netbuf *\fR\fIsvcadd
     \fBconst struct netconfig *\fR\fInetconf\fR, \fBstruct netbuf *\fR\fIssvcad
     \fBconst char *\fR\fIhost\fR);
.fi

.LP
.nf
\fBbool_t\fR \fBrpcb_gettime\fR(\fBconst char *\fR\fIhost\fR, \fBtime_t *\fR\fIt
.fi

.LP
.nf
\fBenum clnt_stat\fR \fBrpcb_rmtcall\fR(\fBconst struct netconfig *\fR\fInetconf
     \fBconst char *\fR\fIhost\fR, \fBconst rpcprog_t\fR \fIprognum\fR,
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst rpcproc_t\fR \fIprocnum\fR,
     \fBconst xdrproc_t\fR \fIinproc\fR, \fBconst caddr_t\fR \fIin\fR,
     \fBconst xdrproc_t\fR \fIoutproc\fR \fBcaddr_t\fR \fIout\fR,
     \fBconst struct timeval\fR \fItout\fR, \fBstruct netbuf  *\fR\fIsvcaddr\fR)
.fi

.LP
.nf
\fBbool_t\fR \fBrpcb_set\fR(\fBconst rpcprog_t\fR \fIprognum\fR, \fBconst rpcver
     \fBconst struct netconfig *\fR\fInetconf\fR, \fBconst struct netbuf *\fR\fI
.fi

.LP
.nf
\fBbool_t\fR \fBrpcb_unset\fR(\fBconst rpcprog_t\fR \fIprognum\fR, \fBconst rpcv
     \fBconst struct netconfig *\fR\fInetconf\fR);
.fi

.SH DESCRIPTION
.LP
These routines allow client C programs to make procedure calls to the RPC
binder service. \fBrpcbind\fR maintains a list of mappings between programs and
```



```
their universal addresses. See \fBrpcbind\fR(1M).
.SS "Routines"
.ne 2
.na
\fB\fBrpcb_getmaps()\fR\fR
.ad
.RS 18n
An interface to the \fBrpcbind\fR service, which returns a list of the current
\fBRPC\fR program-to-address mappings on  \fIhost\fR. It uses the transport
specified through \fInetconf\fR to contact the remote \fBrpcbind\fR service on
\fIhost\fR. This routine will return \fBNULL,\fR if the remote \fBrpcbind\fR
could not be contacted.
.RE

.sp
.ne 2
.na
\fB\fBrpcb_getaddr()\fR\fR
.ad
.RS 18n
An interface to the \fBrpcbind\fR service, which finds the address of the
service on \fIhost\fR that is registered with program number \fIprognum\fR,
version \fIversnum\fR, and speaks the transport protocol associated with
\fInetconf\fR. The address found is returned in \fIsvcaddr\fR. \fIsvcaddr\fR
should be preallocated. This routine returns \fBTRUE\fR if it succeeds.  A
return value of \fBFALSE\fR means that the mapping does not exist or that the
\fBRPC\fR system failed to contact the remote \fBrpcbind\fR service. In the
latter case, the global variable \fBrpc_createerr\fR contains the \fBRPC\fR
status. See \fBrpc_clnt_create\fR(3NSL).
.RE

.sp
.ne 2
.na
\fB\fBrpcb_gettime()\fR\fR
.ad
.RS 18n
This routine returns the time on \fIhost\fR in \fItimep\fR. If \fIhost\fR is
\fINULL\fR, \fBrpcb_gettime()\fR returns the time on its own machine. This
routine returns \fBTRUE\fR if it succeeds, \fBFALSE\fR if it fails.
\fBrpcb_gettime()\fR can be used to synchronize the time between the client and
the remote server.   This routine is particularly useful for secure RPC.
.RE

.sp
.ne 2
.na
\fB\fBrpcb_rmtcall()\fR\fR
.ad
.RS 18n
An interface to the \fBrpcbind\fR service, which instructs \fBrpcbind\fR on
\fIhost\fR to make an \fBRPC\fR call on your behalf to a procedure on that
host. The  \fBnetconfig\fR structure should correspond to a connectionless
transport. The parameter \fB*\fR\fIsvcaddr\fR will be modified to the server's
address if the procedure succeeds. See  \fBrpc_call()\fR and \fBclnt_call()\fR
in \fBrpc_clnt_calls\fR(3NSL) for the definitions of other parameters.
.sp
This procedure should normally be used for a "ping" and nothing else. This
routine allows programs to do lookup and call, all in one step.
.sp
Note: Even if the server is not running \fBrpcbind\fR does not return any error
messages to the caller. In such a case, the caller times out.
.sp
Note: \fBrpcb_rmtcall()\fR is only available for connectionless transports.
.RE
```

```
.sp
.ne 2
.na
\fB\fBrpcb_set()\fR\fR
.ad
.RS 18n
An interface to the \fBrpcbind\fR service, which establishes a mapping between
the triple [\fIprognum\fR, \fIversnum\fR, \fInetconf\fR->\fInc_netid]\fR and
\fIsvcaddr\fR on the machine's \fBrpcbind\fR service. The value of
\fInc_netid\fR must correspond to a network identifier that is defined by the
netconfig database. This routine returns \fBTRUE\fR if it succeeds, \fBFALSE\fR
otherwise. See also \fBsvc_reg()\fR in \fBrpc_svc_reg\fR(3NSL). If there
already exists such an entry with \fBrpcbind\fR, \fBrpcb_set()\fR will fail.
.RE

.sp
.ne 2
.na
\fB\fBrpcb_unset()\fR\fR
.ad
.RS 18n
An interface to the \fBrpcbind\fR service, which destroys the mapping between
the triple [\fIprognum\fR, \fIversnum\fR, \fInetconf\fR->\fInc_netid]\fR and
the address on the machine's \fBrpcbind\fR service. If  \fInetconf\fR is
\fINULL\fR, \fBrpcb_unset()\fR destroys all mapping between the triple
[\fIprognum\fR, \fIversnum\fR, \fIall-transports\fR] and the addresses on the
machine's \fBrpcbind\fR service. This routine returns \fBTRUE\fR if it
succeeds,  \fBFALSE\fR otherwise. Only the owner of the service or the
super-user can destroy the mapping. See also \fBsvc_unreg()\fR in
\fBrpc_svc_reg\fR(3NSL).
.RE

.SH ATTRIBUTES
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
MT-Level        MT-Safe
.TE

.SH SEE ALSO
.LP
\fBrpcbind\fR(1M), \fBrpcinfo\fR(1M), \fBrpc_clnt_calls\fR(3NSL),
\fBrpc_clnt_create\fR(3NSL), \fBrpc_svc_calls\fR(3NSL), \fBattributes\fR(5)
```

```
**********************************************************
    6103 Sat Jul 25 23:07:30 2015
new/usr/src/man/man9e/detach.9e
4648 detach(9e): Extra space between 'prefix' and 'detach'
**********************************************************
'\" te
.\"  Copyright (c) 2003, Sun Microsystems, Inc.  All Rights Reserved
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH DETACH 9E "Jul 25, 2015"
.TH DETACH 9E "Dec 7, 2003"
.SH NAME
detach \- detach or suspend a device
.SH SYNOPSIS
.LP
.nf
#include <sys/ddi.h>
#include <sys/sunddi.h>



\fBint prefix\fR\fBdetach\fR(\fBdev_info_t *\fR \fIdip\fR, \fBddi_detach_cmd_t\f
\fBint prefix\fR \fBdetach\fR(\fBdev_info_t *\fR \fIdip\fR, \fBddi_detach_cmd_t\
.fi

.SH INTERFACE LEVEL
.LP
Solaris DDI specific (Solaris DDI)
.SH PARAMETERS
.ne 2
.na
\fB\fIdip\fR\fR
.ad
.RS 7n
A pointer to the device's \fBdev_info\fR structure.
.RE

.sp
.ne 2
.na
\fB\fIcmd\fR\fR
.ad
.RS 7n
Type of detach; the driver should return \fBDDI_FAILURE\fR if any value other
than \fBDDI_DETACH\fR or \fBDDI_SUSPEND\fR is passed to it.
.RE

.SH DESCRIPTION
.LP
The \fBdetach()\fR function complements the \fBattach\fR(9E) routine.
.SS "DDI_DETACH"
.LP
If \fIcmd\fR is set to \fBDDI_DETACH\fR, \fBdetach()\fR is used to remove the
state associated with a given instance of a device node prior to the removal of
that instance from the system.
.sp
.LP
The \fBdetach()\fR function will be called once for each instance of the device
for which there has been a successful \fBattach()\fR, once there are no longer
any opens on the device. An attached instance of a driver can be successfully
detached only once. The \fBdetach()\fR function should clean up any per
instance data initialized in \fBattach\fR(9E) and call \fBkmem_free\fR(9F) to
free any heap allocations. For information on how to unregister interrupt
handlers, see \fBddi_add_intr\fR(9F). This should also include putting the
underlying device into a quiescent state so that it will not generate
```



```
interrupts.
.sp
.LP
Drivers that set up \fBtimeout\fR(9F) routines should ensure that they are
cancelled before returning \fBDDI_SUCCESS\fR from \fBdetach()\fR.
.sp
.LP
If \fBdetach()\fR determines a particular instance of the device cannot be
removed when requested because of some exceptional condition, \fBdetach()\fR
must return \fBDDI_FAILURE\fR, which prevents the particular device instance
from being detached. This also prevents the driver from being unloaded. A
driver instance failing the detach must ensure that no per instance data or
state is modified or freed that would compromise the system or subsequent
driver operation.
.sp
.LP
The system guarantees that the function will only be called for a particular
\fBdev_info\fR node after (and not concurrently with) a successful
\fBattach\fR(9E) of that device. The system also guarantees that \fBdetach()\fR
will only be called when there are no outstanding \fBopen\fR(9E) calls on the
device.
.SS "DDI_SUSPEND"
.LP
The \fBDDI_SUSPEND\fR \fIcmd\fR is issued when the entire system is being
suspended and power removed from it or when the system must be made quiescent.
It will be issued only to devices which have a \fBreg\fR property or which
export a \fBpm-hardware-state\fR property with the value needs-suspend-resume.
.sp
.LP
If \fIcmd\fR is set to \fBDDI_SUSPEND\fR, \fBdetach()\fR is used to suspend all
activity of a device before power is (possibly) removed from the device. The
steps associated with suspension must include putting the underlying device
into a quiescent state so that it will not generate interrupts or modify or
access memory. Once quiescence has been obtained, \fBdetach()\fR can be called
with outstanding \fBopen\fR(9E) requests. It must save the hardware state of
the device to memory and block incoming or existing requests until
\fBattach()\fR is called with \fBDDI_RESUME\fR.
.sp
.LP
If the device is used to store file systems, then after \fBDDI_SUSPEND\fR is
issued, the device should still honor \fBdump\fR(9E) requests as this entry
point may be used by suspend-resume operation (see \fBcpr\fR(7)) to save state
file. It must do this, however, without disturbing the saved hardware state of
the device.
.sp
.LP
If the device driver uses automatic device Power Management interfaces (driver
exports \fBpm-components\fR(9P) property), it might need to call
\fBpm_raise_power\fR(9F) if the current power level is lower than required to
complete the \fBdump\fR(9E) request.
.sp
.LP
Before returning successfully from a call to \fBdetach()\fR with a command of
\fBDDI_SUSPEND\fR, the driver must cancel any outstanding timeouts and make any
driver threads quiescent.
.sp
.LP
If \fBDDI_FAILURE\fR is returned for the \fBDDI_SUSPEND\fR \fIcmd\fR, either
the operation to suspend the system or to make it quiescent will be aborted.
.SH RETURN VALUES
.ne 2
.na
\fB\fBDDI_SUCCESS\fR\fR
.ad
.RS 15n
For \fBDDI_DETACH\fR, the state associated with the given device was
```

```
126 successfully removed. For \fBDDI_SUSPEND\fR, the driver was successfully
127 suspended.
128 .RE

130 .sp
131 .ne 2
132 .na
133 \fB\fBDDI_FAILURE\fR\fR
134 .ad
135 .RS 15n
136 The operation failed or the request was not understood. The associated state is
137 unchanged.
138 .RE

140 .SH CONTEXT
141 .LP
142 This function is called from user context only.
143 .SH ATTRIBUTES
144 .LP
145 See \fBattributes\fR(5) for descriptions of the following attributes:
146 .sp

148 .sp
149 .TS
150 box;
151 c | c
152 l | l .
153 ATTRIBUTE TYPE  ATTRIBUTE VALUE
154 _
155 Interface Stability     Committed
156 .TE

158 .SH SEE ALSO
159 .LP
160 \fBcpr\fR(7), \fBpm\fR(7D), \fBpm\fR(9P), \fBpm-components\fR(9P),
161 \fBattach\fR(9E), \fBdump\fR(9E), \fBopen\fR(9E), \fBpower\fR(9E),
162 \fBddi_add_intr\fR(9F), \fBddi_dev_is_needed\fR(9F), \fBddi_map_regs\fR(9F),
163 \fBkmem_free\fR(9F), \fBpm_raise_power\fR(9F), \fBtimeout\fR(9F)
164 .sp
165 .LP
166 \fIWriting Device Drivers\fR
```

```
**********************************************************
    6934 Sat Jul 25 23:07:30 2015
new/usr/src/man/man9f/taskq.9f
5036 taskq(9f): Typos in the man page
**********************************************************
'\" te
.\" Copyright (c) 2005, Sun Microsystems, Inc. All Rights Reserved.
.\" Copyright 1989 AT&T
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH TASKQ 9F "Jul 25, 2015"
.TH TASKQ 9F "Mar 1, 2005"
.SH NAME
taskq, ddi_taskq_create, ddi_taskq_destroy, ddi_taskq_dispatch, ddi_taskq_wait,
ddi_taskq_suspend, taskq_suspended, ddi_taskq_resume \- Kernel task queue
operations
.SH SYNOPSIS
.LP
.nf
#include <sys/sunddi.h>

\fBddi_taskq_t *\fR\fBddi_taskq_create\fR(\fBdev_info_t *\fR\fIdip\fR, \fBconst
     \fBint\fR \fInthreads\fR, \fBpri_t\fR  \fIpri\fR, \fBuint_t\fR  \fIcflags\f
.fi

.LP
.nf
\fBvoid\fR \fBddi_taskq_destroy\fR(\fBddi_taskq_t *\fR\fItq\fR);
.fi

.LP
.nf
\fBint\fR \fBddi_taskq_dispatch\fR(\fBddi_taskq_t *\fR\fItq\fR, \fBvoid (*\fR \f
     \fBvoid *\fR\fIarg\fR, \fBuint_t\fR \fIdflags\fR);
.fi

.LP
.nf
\fBvoid\fR \fBddi_taskq_wait\fR(\fBddi_taskq_t *\fR\fItq\fR);
.fi

.LP
.nf
\fBvoid\fR \fBddi_taskq_suspend\fR(\fBddi_taskq_t *\fR\fItq\fR);
.fi

.LP
.nf
\fBboolean_t\fR \fBddi_taskq_suspended\fR(\fBddi_taskq_t *\fR\fItq\fR);
.fi

.LP
.nf
\fBvoid\fR \fBddi_taskq_resume\fR(\fBddi_taskq_t *\fR\fItq\fR);
.fi

.SH INTERFACE LEVEL
.sp
.LP
Solaris DDI specific (Solaris DDI)
.SH PARAMETERS
.sp
.ne 2
.na
\fB\fIdip\fR\fR
```



```
.ad
.RS 12n
Pointer to the device's dev_info structure. May be NULL for kernel
modules that do not have an associated dev_info structure.
.RE

.sp
.ne 2
.na
\fB\fIname\fR\fR
.ad
.RS 12n
Descriptive string. Only alphanumeric characters can be used           in name
and spaces are not allowed. The name should be unique.
.RE

.sp
.ne 2
.na
\fB\fInthreads\fR\fR
.ad
.RS 12n
Number of threads servicing the task queue. Note that the request ordering is
guaranteed (tasks are processed in the order scheduled) if the \fBtaskq\fR is
created with a single servicing thread.
.RE

.sp
.ne 2
.na
\fB\fIpri\fR\fR
.ad
.RS 12n
Priority of threads servicing the task queue. Drivers and modules should
specify TASKQ_DEFAULTPRI.
.RE

.sp
.ne 2
.na
\fB\fIcflags\fR\fR
.ad
.RS 12n
Should pass 0 as flags.
.RE

.sp
.ne 2
.na
\fB\fIfunc\fR\fR
.ad
.RS 12n
Callback function to call.
.RE

.sp
.ne 2
.na
\fB\fIarg\fR\fR
.ad
.RS 12n
Argument to the callback function.
.RE

.sp
.ne 2
```

```
.na
\fB\fIdflags\fR\fR
.ad
.RS 12n
Possible \fIdflags\fR are:
.sp
.ne 2
.na
\fBDDI_SLEEP\fR
.ad
.RS 15n
Allow sleeping (blocking) until memory is available.
.RE

.sp
.ne 2
.na
\fBDDI_NOSLEEP\fR
.ad
.RS 15n
Return DDI_FAILURE immediately if memory is not  available.
.RE

.RE

.sp
.ne 2
.na
\fB\fItq\fR\fR
.ad
.RS 12n
Pointer to a task queue (ddi_taskq_t *).
.RE

.sp
.ne 2
.na
\fB\fItp\fR\fR
.ad
.RS 12n
Pointer to a thread structure.
.RE

.SH DESCRIPTION
.sp
.LP
A kernel task queue is a mechanism for general-purpose asynchronous task
scheduling that enables tasks to be performed at a later time by another
thread. There are several reasons why you may utilize asynchronous task
scheduling:
.RS +4
.TP
1.
You have a task that isn't time-critical, but a current code path that is.
.RE
.RS +4
.TP
2.
You have a task that may require grabbing locks that a thread already holds.
.RE
.RS +4
.TP
3.
You have a task that needs to block (for example, to wait for memory), but you
have a thread that cannot block in its current context.
```



```
.RE
.RS +4
.TP
4.
You have a code path that can't complete because of a specific condition,
but also can't sleep or fail. In this case, the task is immediately queued and
then is executed after the condition disappears.
.RE
.RS +4
.TP
5.
A task queue is just a simple way to launch multiple tasks in parallel.
.RE
.sp
.LP
A task queue consists of a list of tasks, together with one or more threads to
service the list. If a task queue has a single service thread, all tasks are
guaranteed to execute in the order they were dispatched. Otherwise they can be
executed in any order. Note that since tasks are placed on a list, execution of
one task should not depend on the execution of another task or a deadlock
may occur.
one task and should not depend on the execution of another task or a deadlock
may occur. A \fBtaskq\fR created with a single servicing thread guarantees that
all the tasks are serviced in the order in which they are scheduled.
.sp
.LP
The \fBddi_taskq_create()\fR function creates a task queue instance.
.sp
.LP
The \fBddi_taskq_dispatch()\fR function places \fBtaskq\fR on the list for
later execution. The \fIdflag\fR argument specifies whether it is allowed sleep
waiting  for memory. DDI_SLEEP dispatches can sleep and are guaranteed to
succeed. DDI_NOSLEEP dispatches are guaranteed not to sleep but may fail
(return \fBDDI_FAILURE\fR) if resources are not available.
.sp
.LP
The \fBddi_taskq_destroy()\fR function waits for any scheduled tasks to
complete, then destroys the \fBtaskq\fR. The caller should guarantee that no
new tasks are scheduled for the closing \fBtaskq\fR.
.sp
.LP
The \fBddi_taskq_wait()\fR function waits for all previously scheduled tasks to
complete. Note that this function does not stop any new task dispatches.
.sp
.LP
The \fBddi_taskq_suspend()\fR function suspends all task execution until
\fBddi_taskq_resume()\fR is called. Although \fBddi_taskq_suspend()\fR attempts
to suspend pending tasks, there are no guarantees that they will be suspended.
The only guarantee is that all tasks dispatched after \fBddi_taskq_suspend()\fR
will not be executed. Because it will trigger a deadlock, the
\fBddi_taskq_suspend()\fR function should never be called by a task executing
on a \fBtaskq\fR.
.sp
.LP
The \fBddi_taskq_suspended()\fR function returns \fBB_TRUE\fR if \fBtaskq\fR is
suspended, and \fBB_FALSE\fR otherwise. It is intended to ASSERT that the task
queue is suspended.
.sp
.LP
The \fBddi_taskq_resume()\fR function resumes task queue execution.
.SH RETURN VALUES
.sp
.LP
The \fBddi_taskq_create()\fR function creates an opaque handle that is used for
all other \fBtaskq\fR operations. It returns a \fBtaskq\fR pointer on success
and NULL on failure.
```

```
.sp
.LP
The \fBddi_taskq_dispatch()\fR function returns \fBDDI_FAILURE\fR if it can’t
dispatch a task and returns \fBDDI_SUCCESS\fR if dispatch succeeded.
.sp
.LP
The \fBddi_taskq_suspended()\fR function returns \fBB_TRUE\fR if \fBtaskq\fR is
suspended. Otherwise \fBB_FALSE\fR is returned.
.SH CONTEXT
.sp
.LP
All functions may be called from the user or kernel contexts.
.sp
.LP
Addtionally, the \fBddi_taskq_dispatch\fR function may be called from the
interrupt context only if the DDI_NOSLEEP flag is set.
```