

iWARP Transport Specific Extensions for DAT 2.0

August 2006

Final Draft

Contents

1.	Requirements	3
1.1	Consumer Requirement	3
1.2	Transport Neutral Alternatives.....	3
2.	Data Structures and Types.....	5
2.1	IA specific attributes	5
2.2	Definitions (dat_iw_extensions.h).....	5
2.2.1	Socket Service Point.....	5
2.2.2	Socket ID.....	5
2.2.3	DAT_IW_SSP_PARAM.....	6
3.	APIs	7
3.1	Socket Service Point	8
3.1.1	DAT_IW_SSP_CREATE.....	8
3.1.2	DAT_IW_SSP_FREE.....	10
3.1.3	DAT_IW_SSP_QUERY.....	11
3.1.4	DAT_IW_SOCKET_CONNECT.....	12

1. Requirements

DAPL supports socket-like transport-independent connection management

- Stage 1: The active side requests a connection from Endpoint that has the parameters required for a socket for connection setup.
 1. domain
 2. type
 3. protocol
 4. IP Address
 5. port
- Stage 2: The passive side “listens” for Connection Requests on the Service Point that has parameters required for a socket to listen on and either accepts or rejects the Connection Request
- Both active and passive sides are notified of the connection establishment completion by delivering the Connection Establishment Completion event to the Connection Event Dispatchers specified by the Consumers for the Endpoints that got connected on each side.

DAPL allows a Provider to restrict the sockets that it can support. For example, a Provider will not be able to support a socket that is not currently routed through a physical interface that it controls. In certain environments the Provider may only be able to work with sockets that were explicitly created to be hardware offload eligible.

1.1 Consumer Requirement

A Socket Service Point (SSP) enables the Consumer to establish an RDMA connection by first establishing a lower layer protocol (LLP) connection using a socket and then enabling use of RDMA over the connection that the socket refers to.

The alternative connection models rely on an underlying LLP connection over iWARP, but do not expose this connection to the Consumer. There are some specialized application needs that require the LLP connection to be accessible to the user before QP creation/selection is done. These include:

- When the standard Private Data is not large enough to communicate all pre-RDMA data.
- When entry to RDMA mode is being negotiated in a protocol specific manner, especially when the connection will continue in non-RDMA mode when RDMA is not available.
- When there is a need to pass the connection to another process, especially a child process.

1.2 Transport Neutral Alternatives

SSPs allow the Consumer to establish connections using existing sockets and then request that the connection be transitioned to providing RDMA services. The only reason for doing this is the need for some socket-based service prior to enabling RDMA. This could include streaming mode exchanges or passing of a socket to another user process.

The ability to engage in streaming mode exchanges before enabling RDMA would be only useful for services that have both socket-based and RDMA-based modes. If RDMA capability can be assumed, then Send/Receive exchanges can be used as “pre-RDMA” messaging before any RMR Contexts are exchanged. There is no need for special “streaming mode” exchanges.

This model does allow a client that can operate using either conventional sockets or RDMA to fall back to the conventional socket-based service when RDMA is unavailable. The client does not have to repeat the connection establishment process. This is inherently TCP specific, because SCTP specifies RDMA capability during association establishment, and RDMA capability is a given for InfiniBand.

DAPL Extension Design and API

Socket Service Points can also be used a mechanism of passing established to another process. This would be before RDMA was enabled but potentially after some information has been exchanged in streaming mode. Generally, the Host OS will have the ability to pass a socket to another or at least a child process, but will not have the ability to pass an RDMA endpoint.

While DAT cannot support passing Local Endpoints to another process, a Connection Request can be redirected to another process using *dat_cr_handoff*.

2.2.3 DAT_IW_SSP_PARAM

The following types support *dat_iw_ssp_query*.

```
typedef enum dat_iw_ssp_state
{
    DAT_IW_SSP_STATE_OPERATIONAL,
    DAT_IW_SSP_STATE_NON_OPERATIONAL
} DAT_IW_SSP_STATE;
typedef struct dat_iw_ssp_param
{
    DAT_IA_HANDLE ia_handle;
    DAT_IW_SOCKET socket_id;
    DAT_EVD_HANDLE evd_handle;
    DAT_EP_HANDLE ep_handle;
    DAT_IW_SSP_STATE ssp_state;
} DAT_IW_SSP_PARAM;
typedef enum dat_iw_ssp_param_mask
{
    DAT_IW_SSP_FIELD_IA_HANDLE = 0x01,
    DAT_IW_SSP_FIELD_SOCKET_ID = 0x02,
    DAT_IW_SSP_FIELD_EVD_HANDLE = 0x04,
    DAT_IW_SSP_FIELD_EP_HANDLE = 0x08,
    DAT_IW_SSP_FIELD_SSP_STATE = 0x10,
    DAT_IW_SSP_FIELD_ALL = 0x1F
} DAT_IW_SSP_PARAM_MASK;
```

3. APIs

The following function prototypes are actually implemented as pre-processor macros. The macro validates that extensions are supported and then calls the `DAT_EXTENSION_FUNC` vector in the `dat_provider` structure. The type definition for the core extension call is as follows:

```
typedef DAT_RETURN (*DAT_EXTENSION_FUNC) (  
    IN    DAT_HANDLE,           /* DAT handle           */  
    IN    DAT_EXTENDED_OP,     /* DAT extension operation */  
    IN    va_list);           /* va_list, variable arguments*/
```

Each API below details input/output arguments and completion semantics. Explicit return codes are not given but they can be assumed to be logical uses of existing DAT return codes.

A DAPL application can determine which extensions and versions are supported by a DAPL provider by making the `ep_ia_query()` call and iterating the `DAT_NAMED_ATTR` array pointed to by the `provider_specific_attr` member in `DAT_PROVIDER_ATTR`. The `DAT_NAMED_ATTR` type contains two string pointers of `name` and `value`. The table below specifies the `name/extension` relationship. In most cases, simply having the name defined implies support and the `string` value does not supply additional context.

Extension	Name Attribute
Indicates general support for extensions	<code>DAT_EXTENSION_INTERFACE</code>
Indicates version of extended API	<code>DAT_EXTENSION_VERSION</code>
Indicates support for Socket Service Points	<code>DAT_IW_ATTR_SSP</code>

3.1 Socket Service Point

This section specifies the APIs related to the Socket Service Point extension.

3.1.1 DAT_IW_SSP_CREATE

Synopsis: DAT_RETURN

```
dat_iw_ssp_create (
    IN DAT_IA_HANDLE ia_handle,
    IN DAT_IW_SOCKET socket_id,
    IN DAT_EP_HANDLE ep_handle,
    IN DAT_EVD_HANDLE evd_handle,
    IN DAT_PVOID final_sm_msg,
    IN DAT_COUNT final_sm_msg_len,
    OUT DAT_IW_SSP_HANDLE *ssp_handle
)
```

Parameters:

ia_handle: Handle for an instance of DAT IA.

socket: Connected socket ID.

ep_handle: Handle for the Endpoint associated with the SSP that is the only Endpoint that can accept a Connection Request on this Service Point. The value DAT_HANDLE_NULL requests the Provider to associate a Provider-created Endpoint with this Service Point.

evd_handle: The Event Dispatcher to which an event of Connection Request arrival is generated for.

final_sm_msg: Optional message which should be sent on the socket prior to waiting for the MPA Request frame. If the verb layer is separate from the DAT Provider, this should be submitted for transmission atomically. There must be no risk that the other side will respond before reception of the MPA Request frame is enabled.

final_sm_msg_len: Length of optional final streaming mode message.

ssp_handle: Handle to an opaque Socket Service Point.

Description:

dat_iw_ssp_create creates a Socket Service Point with the specified Endpoint that generates, at most, one Connection Request that is delivered to the specified Event Dispatcher in a Notification event.

If the socket represented by the *socket_id* is not known to the Provider, not routed through underlying IA, then the operation fails and DAT_INVALID_STATE is returned.

If the socket becomes disconnected then the SSP transitions into non-operational state and DAT_CONNECTION_EVENT_SOCKET_DOWN event is generated on the SSP EVD.

dat_iw_ssp_create is synchronous and thread safe.

The kDAPL implementation of this operation is not UpCall safe.

Returns

DAT_SUCCESS	The operation was successful.
DAT_INSUFFICIENT_RESOURCES	The operation failed due to resource limitations.
DAT_INVALID_HANDLE	Invalid DAT handle; <i>ia_handle</i> , <i>evd_handle</i> , or <i>ep_handle</i> is invalid.
DAT_INVALID_PARAMETER	Invalid parameter; socket is invalid.
DAT_INVALID_STATE	Parameter is in an invalid state. For example, an Endpoint was not in the Idle state, or socket does not belong to the Interface Adaptor.

3.1.1.1 USAGE

The usage of a Socket Service Point is as follows:

- The DAT Consumer establishes a socket level connection. Optionally, the DAT Consumer uses socket level connection to negotiate RDMA level connection parameters.
- The DAT Consumer creates a Local Endpoint and configures it appropriately.
- The DAT Consumer creates a Socket Service Point specifying the Local Endpoint and the socket.
- The Socket Service Point does the following:
 - Collects native transport information reflecting a received Connection Request
 - Creates a Pending Connection Request
 - Creates a Connection Request Notice (event) that includes the Pending Connection Request (which includes, among others, Socket Service Point Connection Qualifier[we need to decide what to return for it or change the event definition], its Local Endpoint
 - Delivers the Connection Request Notice to the Consumer specified target (UpCall Object) *evd_handle*. The Local Endpoint is transitioned from Reserved to Passive Connection Pending state. The SSP transitions into non-operational state and socket is no longer associated with SSP. The socket become associated with the CR that was generated by the SSP.
- During the UpCall, or at some time subsequent to that, the DAT Consumer must either *accept()* or *reject()* the Pending Connection Request.
- If accepted, the original Local Endpoint is now in a *Connected* state and fully usable for this connection. The Consumer is notified that the Endpoint is in a *Connected* state by a Connection Established Event on the Endpoint *connect_evd_handle*.
- If rejected, the Local Endpoint point transitions into *Unconnected* state. The DAT Consumer can elect to destroy it or reuse it for other purposes. The Consumer should not use the socket once it is passed to the SSP.

3.1.1.2 RATIONALE

Since socket connection can fail at any time Consumer must be ready for the SSP to be created in the non-operational state or transition into *non-operational* state prior to the Consumer being able to get a Connection Request from it. Accordingly, there is no requirement for the Provider to check if the socket provided is connected or not. But the Provider must generate a `DAT_CONNECTION_EVENT_SOCKET_DOWN` event if the socket is not connected when SSP is created or at any time afterwards prior to generation of the Connection Request Arrival event.

3.1.1.3 MODEL IMPLICATIONS

When the connection is established the *socket* is under control of the Provider and is associated with the connected local EP. The Provider can keep the socket, or it may release it. If Provider is not using the Consumer provided socket then the socket used by the DAT connection must use the Consumer socket parameters for it. The socket and its ID do not go back to the Consumer control. Either disconnecting or destroying the EP ensures that socket return to OS control.

Once the SSP is in *non-operational* state it can not be used for connection establishment and cannot be associated with a socket. Consumer can destroy the SSP and create another one using the same socket if it still connected.

Neither the Provider nor RNIC can change the routing of the TCP connection for socket connection provided by Consumer.

3.1.2 DAT_IW_SSP_FREE

Synopsis: `DAT_RETURN dat_iw_ssp_free (`
 `IN DAT_IW_SSP_HANDLE ssp_handle`
`)`

Parameters:

ssp_handle: Handle for an instance of the Socket Service Point.

Description:

dat_iw_ssp_free destroys a specified instance of the Socket Service Point.

Any incoming Connection Requests on the socket of the destroyed Service Point are automatically rejected by the Provider with the return analogous to the no listening Service Point.

The behavior of the Connection Requests in progress is undefined and left to an implementation. But it must be consistent. This means that either a Connection Requested Event was generated for the Event Dispatcher associated with the Service Point, including the creation of the Connection Request instance, or the Connection Request is rejected by the Provider without any local notification.

This operation shall have no effect on previously generated Connection Request Event and Connection Request.

For the Socket Service Point, the Consumer-provided Endpoint reverts to the Consumer control. Consumers shall be aware that due to a race condition, this Socket Service Point might have generated a Connection Request Event and passed the associated Endpoint to a Consumer in it.

If the socket is still associated with the SSP when the call is made then the socket reverts back to its previous owner. If the Consumer had provided the socket in the *dat_iw_ssp_create* then socket is back under Consumer control. If SSP is in non-operational state then there no socket associated with the SSP.

The destruction of SSP has no effect on the connectivity of its socket. It is illegal to use the destroyed handle in any subsequent operation

dat_iw_ssp_free is synchronous and non-thread safe.

The KDAPL implementation of this operation is not UpCall safe.

RETURNS

DAT_SUCCESS	The operation was successful.
DAT_INVALID_HANDLE	Invalid DAT handle; <i>ssp_handle</i> is invalid.

3.1.2.1 USAGE

3.1.2.2 RATIONALE

3.1.2.3 MODEL IMPLICATIONS

If Provider detects the use of deleted object handle it should return *DAT_INVALID_HANDLE*. Provider should avoid assigning the used handle as long as possible. Once reassigned the handle is no longer belongs to a destroyed object.

3.1.3 DAT_IW_SSP_QUERY

Synopsis: DAT_RETURN

```
dat_iw_ssp_query (  
    IN DAT_IW_SSP_HANDLE ssp_handle,  
    IN DAT_IW_SSP_PARAM_MASK ssp_param_mask,  
    OUT DAT_IW_SSP_PARAM *ssp_param  
)
```

Parameters:

ssp_handle: Handle for an instance of Socket Service Point.

ssp_param_mask: Mask for SSP parameters.

ssp_param: Pointer to a Consumer-allocated structure that the Provider fills for Consumer-requested parameters.

Description:

DAPL Extension Design and API

dat_iw_ssp_query provides to the Consumer parameters of the Socket Service Point. The Consumer passes in a pointer to the Consumer allocated structures for SSP parameters that the Provider fills.

ssp_param_mask allows Consumers to specify which parameters to query. The Provider returns values for *ssp_param_mask* requested parameters. The Provider can return values for any other parameters.

Dat_iw_ssp_query is synchronous and thread safe.

The KDAPL implementation of this operation is not UpCall safe.

Returns:

<i>DAT_SUCCESS</i>	<i>The operation was successful.</i>
<i>DAT_INVALID_HANDLE</i>	<i>Invalid DAT handle; ssp_handle is invalid.</i>
<i>DAT_INVALID_PARAMETER</i>	<i>Invalid parameter; ssp_param_mask is invalid.</i>

3.1.3.1 USAGE

3.1.3.2 RATIONALE

3.1.3.3 MODEL IMPLICATIONS

3.1.4 DAT_IW_SOCKET_CONNECT

Synopsis: DAT_RETURN

```
dat_iw_socket_connect (  
    IN DAT_EP_HANDLE ep_handle,  
    IN DAT_IW_SOCKET socket_id,  
    IN DAT_TIMEOUT timeout,  
    IN DAT_COUNT private_data_size,  
    IN const DAT_PVOID private_data  
)
```

Parameters:

ep_handle: Handle for an instance of an Endpoint.

socket_id: Socket connected to remote peer.

timeout: Duration of time, in microseconds, that a Consumer waits for Connection establishment. The value of DAT_TIMEOUT_INFINITE represents no timeout, indefinite wait. Values must be positive.

private_data_size: Size of the private_data. Must be non-negative.

private_data: Pointer to the private data that should be provided to the remote Consumer as part of the Connection Request. If *private_data_size* is zero, then *private_data* can be NULL.

Description:

dat_iw_socket_connect requests that a connection be established between the *local Endpoint* and a *remote Endpoint* over the specified socket connection. This operation is used by the active/client side Consumer of the Connection establishment model. The remote Endpoint is identified by socket connection.

As part of the successful completion of this operation, the local Endpoint is bound to a Port Qualifier of the local IA. The Port Qualifier is passed to the remote side of the requested connection and is available to the remote Consumer in the Connection Request of the `DAT_CONNECTION_REQUEST_EVENT`.

The Consumer-provided *private_data* is passed to the remote side and is provided to the remote Consumer in the Connection Request. Consumers can encapsulate any local Endpoint attributes that remote Consumers need to know as part of an upper-level protocol. Consumers can also use the socket communication prior to using the socket for this connect call. Providers can also provide a Provider on the remote side any local Endpoint attributes and Transport-specific information needed for Connection establishment by the Transport.

Upon successful completion of this operation, the local Endpoint is transferred into `DAT_EP_STATE_ACTIVE_CONNECTION_PENDING` and the socket ownership is transferred from Consumer to the Endpoint. If connection establishment is not successful then socket ownership is returned to the Consumer.

The `DAT_CONNECTION_EVENT_NON_PEER_REJECTED` event is returned if the connection cannot be established for all reasons of not establishing the connection, except timeout, remote host not reachable, and remote peer reject. In this case, the local Endpoint is automatically transitioned into `DAT_EP_STATE_DISCONNECTED` state.

The acceptance of the requested connection by the remote Consumer is reported to the local Consumer through a `DAT_CONNECTION_EVENT_ESTABLISHED` event on the *connect_evd_handle* of the local Endpoint and the local Endpoint is automatically transitioned into a `DAT_EP_STATE_CONNECTED` state.

The rejection of the connection by the remote Consumer is reported to the local Consumer through a `DAT_CONNECTION_EVENT_PEER_REJECTED` event on the *connect_evd_handle* of the local Endpoint and the local Endpoint is automatically transitioned into a `DAT_EP_STATE_DISCONNECTED` state and the socket become disassociated from the EP and is returned to the Consumer control.

When the Provider cannot reach the remote host or the remote host does not respond within the Consumer requested Timeout, a `DAT_CONNECTION_EVENT_UNREACHABLE` event is generated on the *connect_evd_handle* of the Endpoint. The Endpoint transitions into a `DAT_EP_STATE_DISCONNECTED` state and the socket become disassociated from the EP and is returned to the Consumer control.

The local Endpoint is automatically transitioned into a `DAT_EP_STATE_CONNECTED` state when a Connection Request accepted by the remote Consumer and the Provider completes the Transport-specific Connection establishment. The local Consumer is notified of the established connection through a `DAT_CONNECTION_EVENT_ESTABLISHED` event on the *connect_evd_handle* of the local Endpoint.

When the timeout expired prior to completion of the Connection establishment, the local Endpoint is automatically transitioned into a `DAT_EP_STATE_DISCONNECTED` state and the local Consumer through a `DAT_CONNECTION_EVENT_TIMED_OUT` event on the *connect_evd_handle* of the local Endpoint. The timeout of 0 is invalid since connection cannot be established instantaneously. If `timeout=0` is specified the `DAT_INVALID_PARAMETER` is returned and the socket become disassociated from the EP and is returned to the Consumer control.

DAPL Extension Design and API

If the local Endpoint does not have a Protection Zone defined or one of its EVDs is not defined then the operation fails with `DAT_INVALID_STATE` return.

If the socket represented by the `socket_id` is not known to the Provider, not routed through underlying IA then the operation fails and `DAT_INVALID_STATE` is returned.

If the Endpoint parameters conflict with the socket parameters the operation fails with `DAT_INVALID_PARAMETER`. If the Endpoint Communicator is `DAT_COMM_DEFAULT` then the Endpoint inherits socket's domain, type and protocol parameters. Analogously, the Endpoint inherits socket's addressing information including IP Address, and port.

`dat_iw_socket_connect` is synchronous. Its thread safety is Provider dependent.

The KDAPL implementation UpCall safety of this operation is not guaranteed. **Returns:**

<code>DAT_SUCCESS</code>	The operation was successful
<code>DAT_INSUFFICIENT_RESOURCES</code>	The operation failed due to resource limitations.
<code>DAT_INVALID_PARAMETER</code>	Invalid parameter.
<code>DAT_INVALID_HANDLE</code>	Invalid DAT handle; <code>ep_handle</code> is invalid
<code>DAT_INVALID_STATE</code>	Parameter is in an invalid state. For example, endpoint was not in <code>DAT_EP_STATE_UNCONNECTED</code> state, or socket is not known to RNIC

3.1.4.1 USAGE

It is up to the Consumer to negotiate outstanding RDMA Read incoming and outgoing with a remote peer. The outstanding RDMA Read outgoing attribute should be smaller than the remote Endpoint outstanding RDMA Read incoming attribute. If this is not the case, Connection establishment might fail.

DAT API does not define a protocol on how remote peers exchange Endpoint attributes. The exchange of outstanding RDMA Read incoming and outgoing attributes of EPs is left to the Consumer ULP. The Consumer can use Private Data for it or use the connected socket prior to using it in this call.

If the Consumer does not care about posting RDMA Read operations or remote RDMA Read operations on the connection, it can set the two outstanding RDMA Read attribute values to 0.

3.1.4.2 If the Consumer does not set the two outstanding RDMA Read attributes of the Endpoint, the Provider is free to pick up any value for default. The Provider is allowed to change these default values during connection setup.**RATIONALE**

Since socket connection can fail at any time Consumer must be ready for it. Because of it there is not requirement for the Provider to check if the socket provided is connected or not as part of the successful return of the operation.

3.1.4.3 MODEL IMPLICATIONS

Note to Provider: For iWARP the connection request must be mapped into MPA request frame over the socket connection.

Note to Provider: The remote side not responding to the request within the Consumer-specified *Timeout* is mapped to the *DAT_CONNECTION_EVENT_TIMEOUT_EXPIRED*. In contrast, if the remote side, but not the remote Consumer, is responding, either a *reject* or a Message Receipt Acknowledgement is mapped to *DAT_CONNECTION_EVENT_NON_PEER_REJECT*.

The Provider is not allowed to fail connection establishment because of insufficient resources to support the Provider-chosen outstanding RDMA Read default attributes for the Endpoint.

DAT Providers are required not to change any Consumer-specified Endpoint attributes. If the Consumer does not specify outstanding RDMA Read incoming or outgoing attributes, Providers can change them. It is recommended that the Provider set the outstanding RDMA Read attributes to 0 if the Consumer has not specified them, to ensure that a connection establishment does fail due to insufficient local or remote resources to satisfy local or remote Provider-chosen values for the outstanding RDMA Read incoming and outgoing for the Endpoint.

NOTE to Consumer: Consumer should not use the socket for any streaming data upon successful return of this operation. Consumer can stream data over the socket connection if it gets the socket back. For example, if the timeout expired.

If Consumer specified more private data than local Provider supports the operations fails synchronously with *DAT_INVALID_PARAMETER*. If local Provider support the amount of private data but remote Provider cannot the remote Provider will pass the truncated private data to the Consumer and set the *truncate_flag* in the Connection Request Arrival event.

For iWARP/TCP Providers that support IETF MPA both the size of the private data and the private data shall be mapped into MPA Request frame.

Neither the Provider nor RNIC can change the routing of the TCP connection for socket connection provided by Consumer.

For iWARP/SCTP Providers the size of the private data and the private data shall be mapped into a DDP Session Initiate message.

We should also note that the Provider should release the socket back to the system promptly after the connection is broken. It may do so earlier if the socket resource is not tied to the TCP connection itself.

When a socket is converted for use by an EP the existing TCP options should be preserved, except when they are contrary to normal RDMA operations. For example, iWARP implementations should disable Nagle.

But options such as *KEEPALIVE* should be used as is whenever reasonably possible.