Generalised Socket Addresses for Unix Squeak 3.9–11

Ian Piumarta

2007–06–08

This document describes several new SocketPlugin primitives that allow IPv6 (and arbitrary future other) address formats to be supported with minimal changes to the current API.

The new API replaces and obsoletes all primitives that accept or answer internet host addresses or service port numbers. Unless otherwise noted, primitives that do not accept or answer a host address or service port number are not affected and remain current.

1 Socket creation

The netType parameter to the function sqSocketCreateNetTypeSocketTypeRecvBytesSendBytesSemaIDReadSemaIDWriteSemaID should now be passed one of the address family values shown below. For backward compatibility passing zero (family unspecified) is equivalent to requesting INET4.

2 Internet (socket) address lookup

The old API assumes 32-bit INET4 (host) addresses and numbered service (port) addresses throughought. The new API supports arbitray host address sizes and both numbered and symbolic host and service names.

The old API treated the host and service parts of an internet address seperately. The new API keeps them together as a single conceptual *socket address*.

Although not specified here, a socket address is likely to be stored in a variable ByteArray object within the image. Socket addresses are opaquely typed and timestamped (i.e., socket addresses do not survive beyond their network session).

void **sqResolverGetAddressInfoHostSizeServiceSizeFlagsFamilyTypeProtocol**
(char *hostName, sqInt nameSize, char *servName, sqInt servSize, sqInt flags, sqInt family, sqInt type, sqInt protocol)

Converts hostName and/or servName into one or more socket addresses.

The flags argument should contain zero or more of the following bits:

SQ_SOCKET_NUMERIC The hostName is a numeric IPv4 or IPv6 address. No name resolution will be attempted.

SQ_SOCKET_PASSIVE Controls how the IP address portion of the returned address(es) should be treated when hostName is empty. If this bit is set then an empty hostName corresponds to the 'wildcard' address (INADDR_ANY, IN6ADDR_ANY), making the address(es) suitable for a listening server. If this bit is not set then an empty hostName corresponds to the 'loopback' address, suitable for a TCP connection (or UDP send) to a local service.

The family argument should be set to one of the follwing values:

SQ_SOCKET_FAMILY_UNSPECIFIED The lookup should find as many addresses as it can for the given host name or number, irrespective of their address families.

SQ_SOCKET_FAMILY_LOCAL The lookup should only find local (sometimes called 'Unix') domain host addresses. In this case, the host name should be empty and the service name should be a path to a node in the filesystem associated with a named socket.

SQ_SOCKET_FAMILY_INET4 The lookup should only find IPv4 host addresses.

SQ_SOCKET_FAMILY_INET6 The lookup should only find IPv6 host addresses.

Other address families (ISO/OSI, ISDN, etc.) can be supported trivially by extending the mapping from the Squeak enumeration onto the operating system's address family encoding.

The type argument should be set to one of the follwing values:

SQ_SOCKET_TYPE_UNSPECIFIED The lookup should find as many socket types as it can for the given service name or number.

SQ_SOCKET_TYPE_STREAM The lookup should only find stream-oriented socket types for the service.

SQ_SOCKET_TYPE_DGRAM The lookup should only find datagram-oriented socket types for the service.

Other connection types (RAW, RDM, SEQPACKET, etc.) can be supported trivially by extending the mapping from the Squeak enumeration onto the operating system's socket type encoding.

The protocol argument should be set to one of the follwing values:

SQ_SOCKET_PROTOCOL_UNSPEC The lookup should find as many communication protocols as it can for the given service name or number.

SQ_SOCKET_PROTOCOL_TCP The lookup should only find addresses using the TCP protocol for the given service.

SQ_SOCKET_PROTOCOL_UDP The lookup should only find addresses using the UDP protocol for the given service.

Other protocols (ICMP, IGMP, RDP, RAW, etc.) can be supported trivially by extending the mapping from the Squeak enumeration onto the operating system's protocol encoding.

sqResolverGetAddressInfo signals the resolver semaphore when the lookup is complete. The following primitives retrieve the result(s) of the lookup.

sqInt **sqResolverGetAddressInfoSize**(void)

Answers the size in bytes of the current socket address. The image should allocate a variable byte object of at least this size to hold the address. The contents are opaque. The primitive returns -1 if there is no current address.

void **sqResolverGetAddressInfoResultSize**(char *addr, sqInt addrSize)

Copies the current socket address to the addrSize bytes (determined by calling sqResolverGetAddressInfoSize) of memory starting at addr. The primitive fails if addrSize is too small or if there is no current socket address to copy into addr.

sqInt **sqResolverGetAddressInfoFamily**(void)

Answers the address family (see family argument above) of the current address. The primitive fails if there is no current address.

sqInt **sqResolverGetAddressInfoType**(void)

Answers the socket type (see type argument above) of the current address. The primitive fails if there is no current address.

sqInt **sqResolverGetAddressInfoProtocol**(void)

Answers the protocol (see protocol argument above) of the current address. The primitive fails if there is no current address.

sqInt **sqResolverGetAddressInfoNext**(void)

If the current socket address is not the last in the set of addresses satisfying the most recent address lookup, the next address becomes the current address and the primitive answers true. If there is no current address or if the current address is the last in the set then the primitive answers false and the current address becomes void.

sqInt **sqResolverHostNameSize**(void)

Answers the size of the name of the host.

void **sqResolverHostNameResultSize**(char *name, sqInt nameSize)

Copies the name of the host into name which must point to at least nameSize bytes of memory. The primitive fails if nameSize is too small to contain the entire host name.

3 Internet (socket) address access and manipulation

The new API extends the old API to provide read (and convenience write) access to the opaque socket addresses that encapsulate host and service addresses.

void **sqResolverGetNameInfoSizeFlags**(char *addr, sqInt addrSize, sqInt flags)

Begins a lookup of the host and/or service name stored in the socket address at addr of size addrSize.

The flags argument should contain zero or more of the following bits:

SQ_SOCKET_NUMERIC The host and/or service numbers should be returned in numeric form (instead of name form).

This primitive signals the resolver semaphore when the lookup is complete. The primitive fails if addr or addrSize is invalid.

sqInt **sqResolverGetNameInfoHostSize**(void)

Answers the size of the host name (or number) returned from the last call to sqResolverGetNameInfo. The primitive fails if there was no previous call.

void **sqResolverGetNameInfoHostResultSize**(char *name, sqInt nameSize)

Copies the the host name (or number) into name which contains at least nameSize bytes. The primitive fails if nameSize is too small or if there was no previous call.

sqInt **sqResolverGetNameInfoServiceSize**(void)

Answers the size of the service name (or number) returned from the last call to sqResolverGetNameInfo. The primitive fails if there was no previous call.

void **sqResolverGetNameInfoServiceResultSize**(char *name, sqInt nameSize)

Copies the the service name (or number) into name which contains at least nameSize bytes. The primitive fails if nameSize is too small or if there was no previous call.

sqInt **sqResolverAddressSizeGetPort**(char *addr, sqInt addrSize)

Convenience primitive that answers the port number stored in addr. Fails if addr or addrSize is invalid.

void **sqResolverAddressSizeSetPort**(char *addr, sqInt addrSize, sqInt port)

Convenience primitive that changes the port number stored in addr to be port. Fails if addr or addrSize is invalid.

4 Socket communications

The new API provides replacements for the communications primitives that accepted or answered raw host and/or service addresses. Such addresses are now always transferred to/from the image as opaque socket addresses.

void **sqSocketBindToAddressSize**(SocketPtr s, char *addr, sqInt addrSize)

Binds the socket s to the socket address addr of size addrSize. The primitive fails if addr or addrSize is invalid or if the underlying call to `bind()` fails.

void **sqSocketListenBacklog**(SocketPtr s, sqInt backlog)

Starts listening on s for incoming connections with the given backlog size. The primitive fails if backlog is greater than 1 and s is not stream-oriented, or if the underlying call to `listen()` fails.

void **sqSocketConnectToAddressSize**(SocketPtr s, char *addr, sqInt addrSize)

Connects the socket s to the given socket address. The primitive fails if addr or addrSize is invalid.

sqInt **sqSocketLocalAddressSize**(SocketPtr s)

Answers the size of the local socket address for the connection associated with socket s.

void **sqSocketLocalAddressResultSize**(SocketPtr s, char *addr, int addrSize)

Copies the local address of the connection associated with s into addr which contains at least addrSize bytes. The primitive fails if addrSize is too small.

sqInt **sqSocketRemoteAddressSize**(SocketPtr s)

Answers the size of the remote (peer) socket address for the connection associated with socket s.

void **sqSocketRemoteAddressResultSize**(SocketPtr s, char *addr, int addrSize)

Copies the remote (peer) address of the connection associated with s into addr which contains at least addrSize bytes. The primitive fails if addrSize is too small.

sqInt **sqSocketSendUDPToSizeDataBufCount**(SocketPtr s, char *addr, sqInt addrSize, char *buf, sqInt bufSize)

Sends bufSize bytes of data from buf to the UDP socket whose address is stored in addr of size addrSize bytes.

sqInt **sqSocketReceiveUDPDataBufCount**(SocketPtr s, char *buf, sqInt bufSize)

Receives at most bufSize bytes into buf from the socket s. The address of the most recent sender is available via the sqSocketGetRemoteAddress interface.