VNC AUTOMOTIVE

VNC® Mobile Server SDK for iOS Embedded Reference Manual

Version 4.1.3.8315

http://automotive.realvnc.com/

Contents

Chapter 1

VNC® Mobile Server SDK for iOS Embedded

1.1 Overview

This is a RealVNC Server SDK for iOS, and can be embedded into a single iOS application, in order to allow remote control of that application only.

This is a little different from the RealVNC server SDKs which we provide on other platforms, in the following respects:

- It only provides control of a single individual application.
- This server SDK must be linked into that application at build-time.

1.2 Basic usage

This RealVNC server SDK is distributed as a statically linked library, along with corresponding header files and static libraries for the bearers. To include it within your application, you will need to:

- Add libvncserversdkios.a (and bearer libraries) to your XCode project.
- Set the project include path and linker paths to point to the correct location for the header files and library files, respectively. You can do this by choosing "Get Info" on the target.
- Add suitable code to your application to instantiate, use and destroy the RealVNC server object. The sample application is working code that shows how to make use of a VNCServer object.

1.3 Using the RealVNC server object

The RealVNC server is an Objective C object. You should instantiate, use, and deallocate this in the normal way for Objective C objects.

Here's a quick example:

```
// Create the RealVNC Server; you need to implement the protocol
// 'VNCServerDelegate', and pass a pointer to an instance here.
VNCServer* vncServer = [[[VNCServer alloc] initWithDelegate:yourDelegate] autorelease];

// Use the 'bearer wrapper' to wrap a bearer implementing the
// C API so that it can be used by the iOS Server (which requires
// bearers to use its own Objective-C API).
VNCBearerWrapper * listenBearer = [[[VNCBearerWrapper alloc] initWithBearerName:@"L" withInitializer:
      VNCBearerInitialize_L] autorelease];
[vncServer addBearer:listenBearer];

// Add your RealVNC license.
[vncServer addLicense:yourLicense withSerial:nil];

// Start listening.
[vncServer connectWithCommandString:@"vnccmd:v=1;t=L;p=5900"];
```

In the meantime, RealVNC viewers would be able to connect to your application on port 5900 of the device.

The server instance will retain the delegate you provide to it in initWithDelegate: (VNCServer). You can also disconnect at any point (using reset (VNCServer)), and reconnect with another command string:

```
[vncServer reset];
[vncServer connectWithCommandString:@"vnccmd:v=1;t=C;a=127.0.0.1;p=5500"];
```

Once you're finished with the server instance, call invalidate (VNCServer):

```
[vncServer invalidate];
```

This will then cause the server to cancel any internal threads and release its reference to the delegate you provided (so no delegate methods will be called after this point).

1.4 Thread safety

The RealVNC server SDK for iOS creates some number of internal threads as necessary to run the RealVNC server. You are responsible for calling the server SDK APIs only on the main thread, but note that all the APIs are asynchronous and will therefore not block. Similarly, the RealVNC server will call the delegate you provide on the main thread, and also expects that those methods will not block.

Don't worry about waiting for operations to complete before releasing your reference to the RealVNC server; any scheduled operations cause the server to be retained until they are complete. It's important to call invalidate (VN↩CServer) to ensure that the server cancels any internal processing and makes no further calls to the delegate.

1.5 Event Injection

By default, the server uses its own public API event injection methods, and you should almost always stick to using these. If you wish to implement any event injection yourself, you can implement one of the delegate protocols and then set the corresponding property on the server object.

1.6 Screen Capture

The server captures the screen using the public API CALayer::renderInContext on each of the visible layers of the active window. This should work correctly for most controls, however you are likely to need to implement CALayer↩ ::renderInContext for your own custom layers.

1.7 OpenGL Capture

Unfortunately, the screenshot APIs used by the provided public API screen capture method don't capture OpenGL views by default, because CAEAGLLayer does not render itself correctly when CAEAGLLayer::renderInContext is called.

The SDK does, however, attempt to overcome this limitation by using the glReadPixels function to capture the image data from OpenGL layers and render them like any other layer. This means that no special effort is required by app developers that use OpenGL.

This approach should work fine for many applications, but you may encounter problems with custom OpenGL layers, using multiple separate layers in one window, or issues with colour or shadows. If you do, contact us, and we'll try to help you resolve it.

There's also a lot of room here for performance improvement in OpenGL capture, which is something we're actively working on.

1.8 Security

Like RealVNC's other server SDKs, this server SDK provides encryption and authentication.

For encryption, use the provided key generation class to create a key, and then set this using the private key property on the server class. The useEncryption property must also be set to true; if it is set, but there is no private key, then the server will default to using no encryption.

For authentication, create the key as above, and set the authentcationMode property to the desired mode. Authentication requests, or responses, can then be received by the server delegate.

1.9 Licenses

The server needs to be provided with at least one valid license for a RealVNC connection, and you are responsible for loading the license file and passing the text contents to the addLicense:withSerial: (VNCServer) method.

1.10 Legal information

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 <NSCopying> Protocol Reference

5.2 NSObject Class Reference

5.3 <NSObject> Protocol Reference

5.4 UIAlertController(VNCExtensions) Category Reference

Properties

- void(^ **completion**)(UIAlertAction ∗)

5.4.1 Detailed Description

Definition at line 21 of file UIAlertController+VNCExtensions.h.

5.5 <VNCBearer > Protocol Reference

Objective-C Bearer API.

Instance Methods

- (NSString ∗) - name

  Get the short name of the bearer.
- (NSString ∗) - fullName

  Get the full name of the bearer.
- (NSString ∗) - description

  Get a description of the bearer.
- (NSString ∗) - version

  Get the version of the bearer.
- (id< VNCBearerConnection >) - newConnectionWithContext:

  Create a new bearer connection.

5.5.1 Detailed Description

Objective-C Bearer API.

All bearers used with the iOS Server must implement this API, and can be passed to the server using addBearer:(VNCServer).

VNCBearerWrapper bridges this API with the standard C bearer API.

Definition at line 26 of file VNCServerBearer.h.

5.5.2 Method Documentation

5.5.2.1 name()

```
- (NSString *) name
```

Get the short name of the bearer.

Returns

A pointer to a new string containing the short name.

5.5.2.2 fullName()

```
- (NSString *) fullName
```

Get the full name of the bearer.

Returns

A pointer to a new string containing the full name.

5.5.2.3 description()

```
- (NSString *) description
```

Get a description of the bearer.

Returns

A pointer to a new string containing the description.

5.5.2.4 version()

```
- (NSString *) version
```

Get the version of the bearer.

Returns

A pointer to a new string containing the version.

5.5.2.5 newConnectionWithContext:()

```
- (id<VNCBearerConnection>) newConnectionWithContext:
            (id< VNCBearerConnectionContext >) context
```

Create a new bearer connection.

Create a connection based on a connection context (which should be called to obtain any required information, such as command string fields).

Parameters

context	Context for the new connection, which is retained by the connection.

Returns

A pointer to the new connection (MUST NOT be nil). (Will be retained by this method, so callers must release/autorelease.)

5.6 <VNCBearerConnection > Protocol Reference

Objective-C Bearer Connection API.

Instance Methods

- (NSString *) - listeningInfo

  Get the connection's listening information.
- (NSString *) - localEndpoint

  Get the connection's local endpoint.
- (NSString *) - remoteEndpoint

  Get the connection's remote endpoint.
- (VNCBearerError) - error

  Get the connection's latest error.
- (VNCConnectionEventHandle) - eventHandle

  Get the connection's event handle.
- (VNCConnectionActivity) - activity

Check for connection activity.

- (NSUInteger) - read:maxLength:

  Read from the connection.
- (NSUInteger) - write:maxLength:

  Write to the connection.

5.6.1 Detailed Description

Objective-C Bearer Connection API.

This is the Objective-C API for bearer connections used by the iOS server SDK. Instances of this class are obtained from VNCBearer::newConnectionWithContext:.

Definition at line 26 of file VNCBearerConnection.h.

5.6.2 Method Documentation

5.6.2.1 listeningInfo()

```
- (NSString *) listeningInfo
```

Get the connection's listening information.

Returns

The listening information for this connection, or nil if there is nothing relevant.

5.6.2.2 localEndpoint()

```
- (NSString *) localEndpoint
```

Get the connection's local endpoint.

Returns

The local endpoint (i.e. 'our' endpoint) for this connection, or nil if there is nothing relevant.

5.6.2.3 remoteEndpoint()

```
- (NSString *) remoteEndpoint
```

Get the connection's remote endpoint.

Returns

The remote endpoint (i.e. 'their' endpoint) for this connection, or nil if there is nothing relevant.

5.6.2.4 error()

```
- (VNCBearerError) error
```

Get the connection's latest error.

Returns

The latest error, if any (VNCBearerErrorNone for no error).

5.6.2.5 eventHandle()

```
- (VNCConnectionEventHandle) eventHandle
```

Get the connection's event handle.

Returns

The connection's event handle.

5.6.2.6 activity()

```
- (VNCConnectionActivity) activity
```

Check for connection activity.

Checks whether any activity can be performed on the connection, and processes any events on the connection.

Returns

Whether the connection is readable and/or writeable.

5.6.2.7 read:maxLength:()

```
- (NSUInteger) read:
             (uint8_t *) buffer
            maxLength:(NSUInteger) length
```

Read from the connection.

Returns 0 if no data is available or an error occurs (check VNCBearerConnection::error).

Parameters

buffer	Data buffer to read into.
length	Data buffer length.

Returns

The amount of data read.

5.6.2.8 write:maxLength:()

```
- (NSUInteger) write:
            (const uint8_t *) buffer
            maxLength:(NSUInteger) length
```

Write to the connection.

Returns 0 if no space is available or an error occurs (check VNCBearerConnection::error).

Parameters

buffer	Data buffer to write from.
length	Data buffer length.

Returns

The amount of data written.

5.7 <VNCBearerConnectionContext > Protocol Reference

Objective-C Bearer Connection Context API.

Instance Methods

- (void) - log:

  Log a message for the bearer connection.
- (NSString *) - getCommandStringField:

  Get a command string field.
- (NSString *) - getBearerConfiguration

  Get the bearer configuration.
- (void) - connectionStatusChange:

  Notify a bearer connection status change.
- (BOOL) - localFeatureCheck:

  Check if a set of features are available.

5.7.1 Detailed Description

Objective-C Bearer Connection Context API.

Objects implementing this protocol should be passed to VNCBearer::newConnectionWithContext:, and provides information to the bearer connection, as well as being notified about bearer connection events.

Definition at line 23 of file VNCBearerConnectionContext.h.

5.7.2 Method Documentation

5.7.2.1 log:()

```
- (void) log:
            (NSString *) text
```

Log a message for the bearer connection.

Parameters

text	Message to log.

5.7.2.2 getCommandStringField:()

```
- (NSString*) getCommandStringField:
            (NSString *) fieldName
```

Get a command string field.

Parameters

fieldName	The name of the command string field.

Returns

The field value, or nil if none exists.

5.7.2.3 getBearerConfiguration()

```
- (NSString*) getBearerConfiguration
```

Get the bearer configuration.

Returns

A string representing the bearer's configuration, or nil if none exists.

5.7.2.4 connectionStatusChange:()

```
- (void) connectionStatusChange:
            (VNCConnectionStatus) status
```

Notify a bearer connection status change.

Parameters

status	The new connection status.

5.7.2.5 localFeatureCheck:()

```
- (BOOL) localFeatureCheck:
            (NSArray *) features
```

Check if a set of features are available.

Parameters

features	An array of NSNumbers containing the feature values.

Returns

Whether the features are licensed.

5.8 VNCCommandString Class Reference

VNC Command String class.

Instance Methods

- (id) - initWithText:

  Initialise a VNC Command String instance by parsing a text form command.
- (BOOL) - hasParameter:

  Check if the named parameter exists in the command string.
- (NSString ∗) - getParameter:

  Get the command string parameter value.

Class Methods

- (VNCCommandString ∗) + commandStringWithText:

  Create a VNC Command String instance by parsing a text form command.

5.8.1 Detailed Description

VNC Command String class.

Parses a command string and provides access to its parameters.

Definition at line 21 of file VNCCommandString.h.

5.8.2 Method Documentation

5.8.2.1 commandStringWithText:()

```
+ (VNCCommandString∗) commandStringWithText:
            (NSString ∗) text
```

Create a VNC Command String instance by parsing a text form command.

Parameters

text	A VNC Command String in text form.

Returns

VNC Command String. Returns nil if command string is invalid.

5.8.2.2 initWithText:()

```
- (id) initWithText:
            (NSString ∗) text
```

Initialise a VNC Command String instance by parsing a text form command.

Parameters

text	A VNC Command String in text form.

Returns

VNC Command String. Returns nil if command string is invalid.

5.8.2.3 hasParameter:()

```
- (BOOL) hasParameter:
            (NSString *) parameterName
```

Check if the named parameter exists in the command string.

Parameters

parameterName	Name of the parameter.

Returns

Whether the parameter exists.

5.8.2.4 getParameter:()

```
- (NSString*) getParameter:
            (NSString *) parameterName
```

Get the command string parameter value.

Parameters

parameterName	Name of the parameter.

Returns

The parameter value.

5.9 VNCKeyGenerator Class Reference

Represents the private key generation operation, which is likely to take a significant amount of time (i.e. tens or hundreds of milliseconds).

Instance Methods

- (id) - initWithModBits:

  Initialise the key generator.

- (void) - start

  Instruct the key generator to start generating a key.
- (void) - cancel

  Instruct the key generator to stop generating a key.

Class Methods

- (BOOL) + isValidKeySize:

  Query whether a key size (in bits) can be generated by this key generator.

Properties

- id< VNCKeyGeneratorDelegate > delegate
- BOOL isComplete
- NSData * privateKey
- NSUInteger keySize

5.9.1 Detailed Description

Represents the private key generation operation, which is likely to take a significant amount of time (i.e. tens or hundreds of milliseconds).

All methods here are required to be called from the main thread (all methods are non-blocking).

Definition at line 55 of file VNCKeyGenerator.h.

5.9.2 Method Documentation

5.9.2.1 isValidKeySize:()

```
+ (BOOL) isValidKeySize:
                (size_t) keySize
```

Query whether a key size (in bits) can be generated by this key generator.

Parameters

keySize	The number of desired modulus bits in the key (e.g. 1024).

Returns

Whether the key size can be generated.

5.9.2.2 initWithModBits:()

```
- (id) initWithModBits:
             (size_t) keySize
```

Initialise the key generator.

Note that this will throw an exception if the key size is invalid; use isValidKeySize: to check if a key size is valid before attempting key generation.

Parameters

keySize	The number of desired modulus bits in the key (e.g. 1024).

Returns

Key generator.

5.9.2.3 start()

```
- (void) start
```

Instruct the key generator to start generating a key.

It will call the delegate to indicate its progress, and to indicate when the operation has completed.

Each key generator can only generate one key, so calling this more than once has no effect. This also has no effect once 'cancel' has been called.

5.9.2.4 cancel()

```
- (void) cancel
```

Instruct the key generator to stop generating a key.

This can be useful to stop the key generator calling methods on the delegate.

Once this has been called, no further keys can be generated from this key generator instance.

5.9.3 Property Documentation

5.9.3.1 delegate

```
- (id<VNCKeyGeneratorDelegate>) delegate  [read], [write], [nonatomic], [assign]
```

The key generator delegate, which will be called (on the main thread) to indicate the progress of the key generation operation. This delegate is NOT retained by the key generator.

Definition at line 63 of file VNCKeyGenerator.h.

5.9.3.2 isComplete

```
- (BOOL) isComplete  [read], [nonatomic], [assign]
```

Whether the key generation operation has completed.

Definition at line 68 of file VNCKeyGenerator.h.

5.9.3.3 privateKey

```
- (NSData*) privateKey  [read], [nonatomic], [assign]
```

The private key that has been generated, or nil if key generation is still ongoing.

Definition at line 74 of file VNCKeyGenerator.h.

5.9.3.4 keySize

```
- (NSUInteger) keySize  [read], [nonatomic], [assign]
```

The number of modulus bits in the private key (e.g. 1024).

Definition at line 79 of file VNCKeyGenerator.h.

5.10 <VNCKeyGeneratorDelegate> Protocol Reference

Delegate for keeping track of the key generation operation.

Instance Methods

- (void) - keyGenerator:atPercentage:

  Called by the key generator when progress has been made.
- (void) - keyGenerator:keyComplete:

  Called when key generation is complete. Once this is called the private key will be available as VNCKeyGenerator↩ ::privateKey.

5.10.1 Detailed Description

Delegate for keeping track of the key generation operation.

The methods here are always called on the main thread.

Definition at line 23 of file VNCKeyGenerator.h.

5.10.2 Method Documentation

5.10.2.1 keyGenerator:atPercentage:()

```
- (void) keyGenerator:
            (VNCKeyGenerator *) keyGenerator
            atPercentage:(size_t) percent   [required]
```

Called by the key generator when progress has been made.

Parameters

keyGenerator	The key generator instance.
percent	Percentage that has been completed.

5.10.2.2 keyGenerator:keyComplete:()

```
- (void) keyGenerator:
            (VNCKeyGenerator *) keyGenerator
            keyComplete:(NSData *) privateKey   [required]
```

Called when key generation is complete. Once this is called the private key will be available as VNCKeyGenerator↩ ::privateKey.

Parameters

keyGenerator	The key generator instance.
privateKey	The generated private key.

5.11 <VNCPublicAPIDeviceKeyResponder> Protocol Reference

A VNC Public API device key responder.

Instance Methods

- (void) - handleVNCDeviceKey:isDown:isRaw:

    Handle a VNC device key.

5.11.1 Detailed Description

A VNC Public API device key responder.

Implement this for a responder to handle VNC device key events. The RealVNC Public API device key injector will search for the first responder (see UIResponder), and call this method on that responder.

Definition at line 34 of file VNCPublicAPIDeviceKeyResponder.h.

5.11.2 Method Documentation

5.11.2.1 handleVNCDeviceKey:isDown:isRaw:()

```
- (void) handleVNCDeviceKey:
            (int) deviceKey
            isDown:(BOOL) isDown
            isRaw:(BOOL) isRaw
```

Handle a VNC device key.

VNC device keys can be either 'raw' or abstractions, the former being device key codes that are specific to the device, while the latter are values in the VNCDeviceKey enum that are applicable to most devices.

Parameters

deviceKey	The device key.
isDown	Whether the device key is being pressed down.
isRaw	Whether the device key is a raw keycode (device specific), or whether it is an abstraction (i.e. one of VNCDeviceKey).

See also

VNCDeviceKey

5.12 <VNCPublicAPIKeyResponder> Protocol Reference

A VNC Public API key responder.

Instance Methods

- (void) - handleVNCKey:isDown:

    Handle a VNC key.

5.12.1 Detailed Description

A VNC Public API key responder.

Implement this for a responder to handle VNC key events. The RealVNC Public API key injector will search for the first responder (see UIResponder), and call this method on that responder.

Definition at line 32 of file VNCPublicAPIKeyResponder.h.

5.12.2 Method Documentation

5.12.2.1 handleVNCKey:isDown:()

```
- (void) handleVNCKey:
            (uint32_t) key
           isDown:(BOOL) isDown
```

Handle a VNC key.

Parameters

key	The X11 key sym that should be handled.
isDown	Whether the key is being pressed down.

See also

- convertXKeySymToUnicode: (VNCServer)

5.13 <VNCPublicAPITouchResponder > Protocol Reference

A VNC Public API touch responder.

Instance Methods

- (CGPoint) - convertVNCTouchToLocalCoords:

  Convert screen coordinates to local coordinates.
- (id) - beginVNCTouchSequenceAtPoint:withTapCount:

  Start a VNC touch sequence.
- (void) - endVNCTouchSequenceAtPoint:withData:

  End a VNC touch sequence.
- (void) - handleVNCTouchAtPoint:withData:

  Handle a VNC touch.
- (BOOL) - shouldCancelVNCTouchSequenceWithData:

  Query whether a VNC touch sequence should be cancelled.
- (void) - cancelVNCTouchSequenceWithData:

  Cancel a VNC touch sequence.

5.13.1 Detailed Description

A VNC Public API touch responder.

Implement this to become a VNC touch handler and be able to receive touch events. Note that this handler must be returned by an instance of VNCPublicAPITouchResponderNode, that is reachable (i.e. attached to a UIWindow), to be used by the touch injector.

The RealVNC public API touch injector will search for the correct touch handler using VNCPublicAPITouch↩ResponderNode::findVNCTouchResponder, and then, if a responder is found, call:

1) VNCPublicAPITouchResponder::beginVNCTouchSequenceAtPoint: once to indicate the VNC touch sequence has started. 2) VNCPublicAPITouchResponder::handleVNCTouchAtPoint: for each VNC touch. 3) VNCPublicAP↩ITouchResponder::endVNCTouchSequenceAtPoint: once to indicate the VNC touch sequence has ended.

Definition at line 42 of file VNCPublicAPITouchResponder.h.

5.13.2 Method Documentation

5.13.2.1 convertVNCTouchToLocalCoords:()

```
- (CGPoint) convertVNCTouchToLocalCoords:
                (CGPoint) point
```

Convert screen coordinates to local coordinates.

This method is called to convert screen coordinates to 'local' coordinates, which will be passed to the other methods of VNCPublicAPITouchResponder of this object. You can use this to convert to any coordinate system that is convenient for these methods.

A typical implementation might look like:

```
const CGPoint windowCoords = (self.window != nil) ? [self.window convertPoint:point fromWindow:nil] : point
      ;
return [self convertPoint:windowCoords fromView:nil];
```

Parameters

point	A point in screen coordinates.

Returns

The equivalent point in 'local' coordinates.

5.13.2.2 beginVNCTouchSequenceAtPoint:withTapCount:()

```
- (id) beginVNCTouchSequenceAtPoint:
            (CGPoint) point
            withTapCount:(NSUInteger) tapCount
```

Start a VNC touch sequence.

This is called at the beginning of a VNC touch sequence. This method should create any necessary state for the VNC touch sequence and return it from this method.

Parameters

point	Location of the first touch in the sequence in the 'local' coordinate system (as determined by VNCPublicAPITouchResponder::convertVNCTouchToLocalCoords:). VNCPublicAPITouchResponder::handleVNCTouchAtPoint: will be called immediately after this method with the same point.
tapCount	Number of taps that caused the touch sequence (which will be at least one). This is useful, for example, to implement 'double click' behaviour (such as zooming in on a map).

Returns

A pointer to some data, that will be held by the touch injector and passed to all methods in this touch sequence.

5.13.2.3 endVNCTouchSequenceAtPoint:withData:()

```
- (void) endVNCTouchSequenceAtPoint:
            (CGPoint) point
            withData:(id) data
```

End a VNC touch sequence.

This is called to indicate the end of a VNC touch sequence. This method should destroy the state created in VNCPublicAPITouchResponder::beginVNCTouchSequenceAtPoint: for this VNC touch sequence.

Parameters

point	Location of the last touch in the sequence in the 'local' coordinate system (as determined by VNCPublicAPITouchResponder::convertVNCTouchToLocalCoords:). VNCPublicAPITouchResponder::handleVNCTouchAtPoint: will have been called immediately before this method with the same point.
data	User data pointer (that was returned by VNCPublicAPITouchResponder::beginVNCTouchSequenceAtPoint:).

5.13.2.4 handleVNCTouchAtPoint:withData:()

```
- (void) handleVNCTouchAtPoint:
```

```
            (CGPoint) point
            withData:(id) data
```

Handle a VNC touch.

This will be called for every touch in the sequence (including first and last).

Parameters

point	Location of the the touch in the 'local' coordinate system (as determined by VNCPublicAPITouchResponder::convertVNCTouchToLocalCoords:).
data	User data pointer (that was returned by VNCPublicAPITouchResponder::beginVNCTouchSequenceAtPoint:).

5.13.2.5 shouldCancelVNCTouchSequenceWithData:()

```
- (BOOL) shouldCancelVNCTouchSequenceWithData:
            (id) data   [optional]
```

Query whether a VNC touch sequence should be cancelled.

This method can be used to query whether a VNC touch sequence should be cancelled. For example, UIScrollView touch handling will call this method before cancelling touch events to determine if it is appropriate to cancel the touch sequence.

By default (i.e. when unimplemented), this method returns NO.

Parameters

data	User data pointer (that was returned by VNCPublicAPITouchResponder::beginVNCTouchSequenceAtPoint:).

Returns

Whether the touch sequence should be cancelled.

5.13.2.6 cancelVNCTouchSequenceWithData:()

```
- (void) cancelVNCTouchSequenceWithData:
            (id) data   [optional]
```

Cancel a VNC touch sequence.

This is called to indicate a VNC touch sequence should be cancelled. This method should destroy the state created in VNCPublicAPITouchResponder::beginVNCTouchSequenceAtPoint: for this VNC touch sequence.

By default (i.e. when unimplemented), this method calls VNCPublicAPITouchResponder::endVNCTouch↩ SequenceAtPoint:.

Parameters

data	User data pointer (that was returned by VNCPublicAPITouchResponder::beginVNCTouchSequenceAtPoint:).

5.14 <VNCPublicAPITouchResponderNode> Protocol Reference

A VNC Public API touch responder node.

Instance Methods

- (id< VNCPublicAPITouchResponder >) - findVNCTouchResponder:

  Find a VNC touch responder.

5.14.1 Detailed Description

A VNC Public API touch responder node.

Implement this to find and return a VNC touch responder - many UI elements implement this and return themselves as the handler.

Definition at line 150 of file VNCPublicAPITouchResponder.h.

5.14.2 Method Documentation

5.14.2.1 findVNCTouchResponder:()

```
- (id<VNCPublicAPITouchResponder>) findVNCTouchResponder:
            (CGPoint) point
```

Find a VNC touch responder.

This is called by upper tree elements (or RealVNC Public API Touch Injector directly), to find a VNC touch responder for a VNC touch sequence.

If this method returns nil, the RealVNC touch injector will continue to search for touch responders in views/windows behind this node; to prevent this, return self in this method and provide empty implementations of the methods of VNCPublicAPITouchResponder, such as:

```
-(id) beginVNCTouchSequenceAtPoint:(CGPoint)point withTapCount:(NSUInteger)tapCount {
    return nil;
}

-(void) endVNCTouchSequenceAtPoint:(CGPoint)point withData:(id)data { }

-(void) handleVNCTouchAtPoint:(CGPoint)point withData:(id)data { }

-(id<VNCPublicAPITouchResponder>) findVNCTouchResponder:(CGPoint)point {
    return self;
}
```

Parameters

point	Location of the first VNC touch in screen coordinates.

Returns

A pointer to the responder, if one is found, or nil otherwise.

5.15 <VNCRPCaptureDelegate > Protocol Reference

Instance Methods

- (BOOL) - hasScreenChanged
- (CMSampleBufferRef) - captureScreen
- (void) - **captureFinished**

5.15.1 Detailed Description

Protocol to be implemented by classes which can capture screenshots.

Definition at line 17 of file VNCRPCaptureDelegate.h.

5.15.2 Method Documentation

5.15.2.1 hasScreenChanged()

```
- (BOOL) hasScreenChanged    [required]
```

Asks whether the screen has changed, so that the server knows whether it's worthwhile to call 'captureScreen'.

Returns

Whether the screen has changed.

5.15.2.2 captureScreen()

```
- (CMSampleBufferRef) captureScreen    [required]
```

Capture the screen to an image. The delegate implementation should return the most recent CMSampleBuffer received from the OS.

Returns

The buffer, which should be autoreleased (must NOT be nil).

5.16 VNCServer Class Reference

VNC Server class.

Instance Methods

- (id) - initWithDelegate:

  Initialise a VNC Server instance.
- (id) - **initWithDelegate:withRPCapture:**
- (void) - invalidate

  Invalidate the VNC Server instance.
- (VNCServerError) - addBearer:

  Add a bearer to the server.
- (VNCServerError) - connect:

  Connect using a command string object.
- (VNCServerError) - connectWithCommandString:

  Connect using a command string.
- (void) - reset

  Reset the current connection, if any.
- (VNCServerError) - addLicense:withSerial:

  Add a license.
- (VNCServerError) - addLicenseFeature:withKey:

  Add a licensing feature to the server.
- (VNCServerError) - localFeatureCheck:withResult:

  Perform a local feature for a set of feature IDs.
- (VNCServerError) - provideLicenseFeature:

  Mark the server as providing the specified license feature.
- (VNCServerError) - scheduleRemoteFeatureCheck:withFeatureCheckId:

  Schedules a remote feature check to be made against VNC viewers in all future sessions.
- (VNCServerError) - acceptAuthentication:

  Accept or reject authentication credentials (i.e. username and/or password) that have been received from a VNC viewer.
- (VNCServerError) - acceptConnection:

  Accept or reject an established connection from a VNC viewer.
- (VNCServerError) - acceptRemoteKey:

  Accept or reject an RSA key from a VNC viewer.
- (VNCServerError) - provideUsername:withPassword:

  Provide username and/or password to VNC viewer during reverse authentication.
- (uint32_t) - convertXKeySymToUnicode:

  Translate an X key symbol into a Unicode character.

Class Methods

- (NSString ∗) + buildVersion

  Gets the SDK build version.

Properties

- VNCServerState state

  The server's current state.
- BOOL isActive

  Whether the server is currently in any state other than VNCStateDisconnected.
- NSString ∗ localEndpoint

  The server's local address, or nil if there is none.
- NSString ∗ remoteEndpoint

  The server's remote address, or nil if there is none.
- NSString ∗ listeningInfo

  The server's listening info, or nil if there is none.
- BOOL useEncryption

  Whether encryption is used.
- NSData ∗ privateKey

  VNC Server private key.
- VNCAuthMode authenticationMode

  Authentication mode.
- BOOL clipboardSendOnConnect

  Whether to send the clipboard when a connection is first established.
- BOOL clipboardEnabled

  Whether to send clipboard updates during an active connection.

5.16.1 Detailed Description

VNC Server class.

Note that all APIs MUST be called on the main thread.

Definition at line 77 of file VNCServer.h.

5.16.2 Method Documentation

5.16.2.1 buildVersion()

```
+ (NSString*) buildVersion
```

Gets the SDK build version.

Returns

SDK build version.

5.16.2.2 initWithDelegate:()

```
- (id) initWithDelegate:
            (id< VNCServerDelegate >) delegate
```

Initialise a VNC Server instance.

Parameters

delegate	The server's delegate, which will be retained by the server. Methods of delegates are always called on the main thread.

Returns

VNC Server.

5.16.2.3 invalidate()

```
- (void) invalidate
```

Invalidate the VNC Server instance.

This will cancel any active connections/threads of the server. This MUST be called before the server is released to ensure that these threads are stopped and the server releases its reference to the delegate.

5.16.2.4 addBearer:()

```
- (VNCServerError) addBearer:
            (id< VNCBearer >) bearer
```

Add a bearer to the server.

Parameters

bearer	A pointer to the bearer.

Returns

VNCServerError indicating operation result.

5.16.2.5 connect:()

```
- (VNCServerError) connect:
            (VNCCommandString *) commandString
```

Connect using a command string object.

This will start a new VNC server connection using the bearer and parameters specified in the command string.

Parameters

commandString	The command string object for the connection.

Returns

VNCServerError indicating operation result.

5.16.2.6 connectWithCommandString:()

```
- (VNCServerError) connectWithCommandString:
            (NSString *) commandString
```

Connect using a command string.

This will start a new VNC server connection using the bearer and parameters specified in the command string.

Parameters

commandString	The command string for the connection.

Returns

VNCServerError indicating operation result.

5.16.2.7 addLicense:withSerial:()

```
- (VNCServerError) addLicense:
            (NSString *) licenseString
            withSerial:(out NSData **) serialData
```

Add a license.

Parameters

licenseString	The license text.
serialData	Out parameter returning the serial data of the license. This will be set to point to the license's serial if the license is added successfully (and won't be set otherwise); pass nil for this parameter if the serial data isn't needed.

Returns

VNCServerError indicating operation result.

5.16.2.8 addLicenseFeature:withKey:()

```
- (VNCServerError) addLicenseFeature:
            (NSUInteger) featureId
            withKey:(NSData *) featureKey
```

Add a licensing feature to the server.

Features must be added be added to the server before they can be used in remote feature checks.

Parameters

featureId	The id of the feature to be added.
featureKey	A 16 byte secret key for the feature.

Returns

VNCServerError indicating operation result.

5.16.2.9 localFeatureCheck:withResult:()

```
- (VNCServerError) localFeatureCheck:
            (NSArray *) featureIds
            withResult:(BOOL *) result
```

Perform a local feature for a set of feature IDs.

Parameters

featureIds	An array of NSNumber (holding NSUInteger) feature IDs.
result	Whether at least one of these features is licensed.

Returns

VNCServerError indicating operation result.

5.16.2.10 provideLicenseFeature:()

```
- (VNCServerError) provideLicenseFeature:
            (NSUInteger) featureId
```

Mark the server as providing the specified license feature.

Parameters

feature↩ Id	The ID of the feature.

Returns

VNCServerError indicating operation result.

5.16.2.11 scheduleRemoteFeatureCheck:withFeatureCheckId:()

```
- (VNCServerError) scheduleRemoteFeatureCheck:
              (NSArray *) featureIds
              withFeatureCheckId:(NSUInteger *) featureCheckId
```

Schedules a remote feature check to be made against VNC viewers in all future sessions.

A VNC viewer must prove that it has the correct feature key for at least one of the features specified here.

Parameters

featureIds	Features to be checked; viewer must have the feature key for at least one of these features to pass the remote feature check.
feature↩ CheckId	Out parameter returning an ID associated with the remote feature check; this will be passed to the relevant callbacks.

Returns

VNCServerError indicating operation result.

See also

VNCServerDelegate::onRemoteFeatureCheckSucceeded:withFeature:
VNCServerDelegate::onRemoteFeatureCheckFailed:

5.16.2.12 acceptAuthentication:()

```
- (VNCServerError) acceptAuthentication:
              (BOOL) accept
```

Accept or reject authentication credentials (i.e. username and/or password) that have been received from a VNC viewer.

The server must be in state VNCStateAuth (or this will return VNCServerErrorState).

Parameters

accept	Whether to accept the authentication credentials.

Returns

VNCServerError indicating operation result.

See also

VNCServerDelegate::onAuthUsername:withPassword:

5.16.2.13 acceptConnection:()

```
- (VNCServerError) acceptConnection:
              (BOOL) accept
```

Accept or reject an established connection from a VNC viewer.

The server must be in state VNCStateAccepting (or this will return VNCServerErrorState).

Parameters

accept	Whether to accept the connection.

Returns

VNCServerError indicating operation result.

See also

VNCServerDelegate::onConnectedAtLocalEndpoint:toRemoteEndpoint:

5.16.2.14 acceptRemoteKey:()

```
- (VNCServerError) acceptRemoteKey:
              (BOOL) accept
```

Accept or reject an RSA key from a VNC viewer.

The server must be in state VNCStateAcceptRemoteKey (or this will return VNCServerErrorState).

Parameters

accept	Whether to accept the RSA key.

Returns

VNCServerError indicating operation result.

See also

VNCServerDelegate::onRemoteKey:withSignature:

5.16.2.15 provideUsername:withPassword:()

```
- (VNCServerError) provideUsername:
              (NSString *) username
              withPassword:(NSString *) password
```

Provide username and/or password to VNC viewer during reverse authentication.

The server must be in state VNCStateReverseAuth (or this will return VNCServerErrorState).

Parameters

username	Username to be supplied to VNC viewer (pass nil if the username wasn't requested).
password	Password to be supplied to VNC viewer (pass nil if the password wasn't requested).

Returns

VNCServerError indicating operation result.

See also

VNCServerDelegate::onAuthNeedUsername:needPassword:

5.16.2.16 convertXKeySymToUnicode:()

```
- (uint32_t) convertXKeySymToUnicode:
              (uint32_t) keysym
```

Translate an X key symbol into a Unicode character.

This takes account of the type of the VNC Viewer to which the VNC Server is connected (or uses a default conversion if not connected).

Parameters

keysym	An X11 key symbol.

Returns

The Unicode character corresponding to the X key symbol, or VNC_INVALID_XKEYSYM_UNICODE if there is none.

5.16.3 Property Documentation

5.16.3.1 state

```
- (VNCServerState) state  [read], [nonatomic], [assign]
```

The server's current state.

Initial state is VNCStateDisconnected.

Definition at line 91 of file VNCServer.h.

5.16.3.2 useEncryption

```
- (BOOL) useEncryption  [read], [write], [nonatomic], [assign]
```

Whether encryption is used.

Default is NO.

Definition at line 119 of file VNCServer.h.

5.16.3.3 privateKey

```
- (NSData*) privateKey  [read], [write], [nonatomic], [copy]
```

VNC Server private key.

Use the VNCKeyGenerator class to generate this. This is required for both authentication and encryption.

Default is nil.

Definition at line 129 of file VNCServer.h.

5.16.3.4 authenticationMode

```
- (VNCAuthMode) authenticationMode  [read], [write], [nonatomic], [assign]
```

Authentication mode.

See above for a list and description of values that can be provided for this property.

Default is VNC_AUTHMODE_NONE.

Definition at line 139 of file VNCServer.h.

5.16.3.5 clipboardSendOnConnect

```
- (BOOL) clipboardSendOnConnect  [read], [write], [nonatomic], [assign]
```

Whether to send the clipboard when a connection is first established.

Default is NO.

Definition at line 147 of file VNCServer.h.

5.16.3.6 clipboardEnabled

```
- (BOOL) clipboardEnabled  [read], [write], [nonatomic], [assign]
```

Whether to send clipboard updates during an active connection.

Default is YES.

Definition at line 155 of file VNCServer.h.

5.17 <VNCServerDelegate > Protocol Reference

VNC Server Delegate.

Instance Methods

- (void) - onAuthUsername:withPassword:

  Handle a (normal) authentication event, by either accepting or rejecting the authentication information provided by the viewer.
- (void) - onAuthNeedUsername:needPassword:

  Handle a (reverse) authentication request, by providing the credentials to the viewer that it has requested.
- (void) - onConnecting

  Handle the server moving to a connecting state.
- (void) - onConnectedAtLocalEndpoint:toRemoteEndpoint:

  Handle the server moving to a connected state.
- (void) - onDisconnected

  Handle the server moving to a disconnected state.
- (void) - onListeningWithInfo:

  Handle the server moving to a listening state.
- (void) - onRemoteFeatureCheckSucceeded:withFeature:

  Handle a remote feature check success.
- (BOOL) - onRemoteFeatureCheckFailed:

  Handle a remote feature check failure.
- (void) - onRemoteKey:withSignature:

  Handle a remote key event, by either accepting or rejecting the encryption key.
- (void) - onRunning

  Handle the server moving to a running state.
- (void) - onServerError:

  Handle a VNC Server error.
- (void) - onServerLog:withLevel:

  Handle a VNC Server log message.

5.17.1 Detailed Description

VNC Server Delegate.

Protocol to be implemented by classes which can handle VNC Server events. This should be passed to initWith←Delegate: (VNCServer) (which will retain this instance, hence it must implement NSObject).

Definition at line 26 of file VNCServerDelegate.h.

5.17.2 Method Documentation

5.17.2.1 onAuthUsername:withPassword:()

```
- (void) onAuthUsername:
            (NSString *) username
            withPassword:(NSString *) password   [required]
```

Handle a (normal) authentication event, by either accepting or rejecting the authentication information provided by the viewer.

Parameters

username	The username provided by the viewer.
password	The password provided by the viewer.

See also

- acceptAuthentication: (VNCServer)

5.17.2.2 onAuthNeedUsername:needPassword:()

```
- (void) onAuthNeedUsername:
            (BOOL) needUsername
            needPassword:(BOOL) needPassword   [required]
```

Handle a (reverse) authentication request, by providing the credentials to the viewer that it has requested.

Parameters

needUsername	Whether the viewer requests a username.
needPassword	Whether the viewer requests a password.

See also

- provideUsername:withPassword: (VNCServer)

5.17.2.3 onConnectedAtLocalEndpoint:toRemoteEndpoint:()

```
- (void) onConnectedAtLocalEndpoint:
            (NSString *) localEndpoint
            toRemoteEndpoint:(NSString *) remoteEndpoint   [required]
```

Handle the server moving to a connected state.

No data will be transferred until acceptConnection: (VNCServer) is called, to either accept or reject this connection.

Parameters

localEndpoint	A string describing the local endpoint, or nil if there is no relevant information.
remoteEndpoint	A string describing the remote endpoint, or nil if there is no relevant information.

See also

- acceptConnection: (VNCServer)

5.17.2.4 onDisconnected()

```
- (void) onDisconnected   [required]
```

Handle the server moving to a disconnected state.

When a connection terminates, the server SDK will call exactly one of VNCServerDelegate::onDisconnected (to nofify a graceful disconnection) and VNCServer::onServerError: (to notify a non-graceful disconnection.)

5.17.2.5 onListeningWithInfo:()

```
- (void) onListeningWithInfo:
            (NSString *) listeningInfo   [required]
```

Handle the server moving to a listening state.

Parameters

listeningInfo	A string describing the local endpoint, or nil if there is no relevant information.

5.17.2.6 onRemoteFeatureCheckSucceeded:withFeature:()

```
- (void) onRemoteFeatureCheckSucceeded:
            (NSUInteger) featureCheckId
            withFeature:(NSUInteger) featureId   [required]
```

Handle a remote feature check success.

Parameters

feature↩ CheckId	The id of the remote feature check.
featureId	The id of the feature that the viewer is licensed for.

See also

- scheduleRemoteFeatureCheck:withFeatureCheckId: (VNCServer)

5.17.2.7 onRemoteFeatureCheckFailed:()

```
- (BOOL) onRemoteFeatureCheckFailed:
            (NSUInteger) featureCheckId   [required]
```

Handle a remote feature check failure.

Unlike other delegate methods, this delegate is required to return a BOOL value indicating whether the remote feature check failure should be considered critical (i.e. trigger a disconnection).

Parameters

feature↩ CheckId	The id of the remote feature check.

Returns

Whether to refuse the connection as a result of this failure.

See also

- scheduleRemoteFeatureCheck:withFeatureCheckId: (VNCServer)

5.17.2.8 onRemoteKey:withSignature:()

```
- (void) onRemoteKey:
            (NSData *) remoteKeyData
            withSignature:(NSData *) remoteKeySignature   [required]
```

Handle a remote key event, by either accepting or rejecting the encryption key.

Parameters

remoteKeyData	Key data provided by the viewer.
remoteKeySignature	Key signature provided by the viewer.

See also

- acceptRemoteKey: (VNCServer)

5.17.2.9 onRunning()

```
- (void) onRunning    [required]
```

Handle the server moving to a running state.

The 'running' state is where the server and viewer are connected, the connection has been accepted, any remote keys have been accepted, authentication has completed successfully and no critical remote feature checks have failed.

At this point the server SDK will begin responding to events from the viewer, as well as sending the screen framebuffer.

5.17.2.10 onServerError:()

```
- (void) onServerError:
            (VNCServerError) error    [required]
```

Handle a VNC Server error.

When a connection terminates, the server SDK will call exactly one of VNCServerDelegate::onDisconnected (to nofify a graceful disconnection) and VNCServer::onServerError: (to notify a non-graceful disconnection.)

Parameters

error	The VNC Server error.

See also

VNCServerError

5.17.2.11 onServerLog:withLevel:()

```
- (void) onServerLog:
            (NSString *) message
            withLevel:(VNCServerLogLevel) level    [required]
```

Handle a VNC Server log message.

Note that the server SDK will not log other than through this method, so it's important that implementations do not discard log messages given to this method in a situation where debugging might be required.

A simple implementation might be:

```
-(void) onServerLog:(NSString*)message withLevel:(VNCServerLogLevel)level {
    NSLog(@"RealVNC iOS Server Log (level %d): %@.", level, message);
}
```

Parameters

message	The log message.
level	The log level.

See also

VNCServerLogLevel

Chapter 6

File Documentation

6.1 VNCBearerConnection.h File Reference

VNC Bearer Connection API.

Data Structures

- protocol <VNCBearerConnection >

  Objective-C Bearer Connection API.

6.1.1 Detailed Description

VNC Bearer Connection API.

Copyright RealVNC Ltd. 2011-2018. All rights reserved.

6.2 VNCBearerConnectionContext.h File Reference

VNC Bearer Connection Context API.

Data Structures

- protocol <VNCBearerConnectionContext >

  Objective-C Bearer Connection Context API.

6.2.1 Detailed Description

VNC Bearer Connection Context API.

Copyright RealVNC Ltd. 2013-2018. All rights reserved.

6.3 VNCCommandString.h File Reference

VNC Command String.

Data Structures

- class VNCCommandString

  VNC Command String class.

6.3.1 Detailed Description

VNC Command String.

Copyright RealVNC Ltd. 2014-2018. All rights reserved.

6.4 VNCDeviceKey.h File Reference

VNC Device Keys.

Enumerations

- enum VNCDeviceKey {
  VNCDeviceKeyLeftSoftKey = 0, VNCDeviceKeyRightSoftKey = 1, VNCDeviceKeySend = 2, VNCDeviceKey↩
  End = 3,
  VNCDeviceKeyVolumeUp = '+', VNCDeviceKeyVolumeDown = '-', VNCDeviceKeyKeypad0 = '0', VNC↩
  DeviceKeyKeypad1 = '1',
  VNCDeviceKeyKeypad2 = '2', VNCDeviceKeyKeypad3 = '3', VNCDeviceKeyKeypad4 = '4', VNCDeviceKey↩
  Keypad5 = '5',
  VNCDeviceKeyKeypad6 = '6', VNCDeviceKeyKeypad7 = '7', VNCDeviceKeyKeypad8 = '8', VNCDeviceKey↩
  Keypad9 = '9',
  VNCDeviceKeyKeypadStar = '∗', VNCDeviceKeyKeypadPoundSign = '#', VNCDeviceKeyLeft = 'h', VNC↩
  DeviceKeyRight = 'l',
  VNCDeviceKeyUp = 'k', VNCDeviceKeyDown = 'j', VNCDeviceKeySelect = 'S', VNCDeviceKeyDismiss = 'D',
  VNCDeviceKeyBack = 'B', VNCDeviceKeyHome = 'H', VNCDeviceKeyMenu = 'M', VNCDeviceKeyUnlock =
  'U',
  VNCDeviceKeyEdit = 'E', VNCDeviceKeyPower = 'P' }

  Abstracted button codes for use with VNCPublicAPIDeviceKeyResponder.

6.4.1 Detailed Description

VNC Device Keys.

6.4.2 Enumeration Type Documentation

6.4.2.1 VNCDeviceKey

```
enum VNCDeviceKey
```

Abstracted button codes for use with VNCPublicAPIDeviceKeyResponder.

These abstractions represent buttons that are present on the majority of mobile devices, and, where they are present, are necessary to perform the majority of remote support tasks. Buttons that are present only on a minority of devices, or which have functions that can be carried out by other means, are deliberately not included.

The number keys in this enumeration represent number buttons on a telephone keypad. These are distinct from the X key symbols representing numbers on the top row of a US keyboard (e.g. XK_1) and those representing numbers on a PC keyboard's numeric keypad (e.g. XK_KP_1).

See also

- handleVNCDeviceKey:isDown:isRaw: (VNCPublicAPIDeviceKeyResponder-p)

Enumerator

VNCDeviceKeyLeftSoftKey	A cellphone handset's left soft key.
VNCDeviceKeyRightSoftKey	A cellphone handset's right soft key.
VNCDeviceKeySend	A cellphone handset's send (make voice call) button.
VNCDeviceKeyEnd	A cellphone handset's end (hangup) button.
VNCDeviceKeyVolumeUp	A cellphone handset's volume up button.
VNCDeviceKeyVolumeDown	A cellphone handset's volume down button.
VNCDeviceKeyKeypad0	The 0 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypad1	The 1 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypad2	The 2 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypad3	The 3 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypad4	The 4 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypad5	The 5 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypad6	The 6 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypad7	The 7 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypad8	The 8 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypad9	The 9 key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypadStar	The star key on a cellphone handset's phone keypad.
VNCDeviceKeyKeypadPoundSign	The pound sign (hash) key on a cellphone handset's phone keypad.
VNCDeviceKeyLeft	The left directional button, or a roll of the trackball to the left.
VNCDeviceKeyRight	The right directional button, or a roll of the trackball to the right.
VNCDeviceKeyUp	The up directional button, or a roll of the trackball upwards.
VNCDeviceKeyDown	The down directional button, or a roll of the trackball downwards.
VNCDeviceKeySelect	An abstraction for a device button that selects an item from a menu.
VNCDeviceKeyDismiss	An abstraction for a device button that cancels an action or navigates backwards.
VNCDeviceKeyBack	An abstraction for a device button that navigates backwards.
VNCDeviceKeyHome	An abstraction for a device button that returns the device to its homescreen.
VNCDeviceKeyMenu	An abstraction for a device button that opens the device's main menu.
VNCDeviceKeyUnlock	An abstraction for a device button that unlocks a locked phone keypad.
VNCDeviceKeyEdit	An abstraction for a device button that changes the text input mode.
VNCDeviceKeyPower	A cellphone handset's power button.

Definition at line 28 of file VNCDeviceKey.h.

6.5 VNCExport.h File Reference

VNC Symbol Exporting.

Macros

- #define VNCEXPORT

  Specify whether to export Server SDK symbols.

6.5.1 Detailed Description

VNC Symbol Exporting.

Copyright RealVNC Ltd. 2013-2018. All rights reserved.

6.5.2 Macro Definition Documentation

6.5.2.1 VNCEXPORT

```
#define VNCEXPORT
```

Specify whether to export Server SDK symbols.

By default, don't specify any symbol visibility for Server SDK symbols; this only has an effect when building a library that contains the SDK.

For example, to specify hidden visibility for all Server SDK symbols (before including any Server SDK header files):

```
#define VNCEXPORT __attribute__((visibility("hidden")))
```

Definition at line 28 of file VNCExport.h.

6.6 VNCKeyGenerator.h File Reference

VNC Private Key Generator.

Data Structures

- protocol <VNCKeyGeneratorDelegate>

  Delegate for keeping track of the key generation operation.
- class VNCKeyGenerator

  Represents the private key generation operation, which is likely to take a significant amount of time (i.e. tens or hundreds of milliseconds).

6.6.1 Detailed Description

VNC Private Key Generator.

Copyright RealVNC Ltd. 2012-2018. All rights reserved.

6.7 VNCPublicAPIDeviceKeyResponder.h File Reference

VNC Public API Device Key Responder API.

Data Structures

- protocol <VNCPublicAPIDeviceKeyResponder>

  A VNC Public API device key responder.

6.7.1 Detailed Description

VNC Public API Device Key Responder API.

The protocol documented here makes it possible to implement public API device key injection for custom controls.

This behaviour can be added to existing control classes using Objective-C categories. For example:

```
@implementation YourCustomControlClass (VNCPublicAPIDeviceKeyResponderCategory)
// Your implementation...
@end
```

See also

VNCDeviceKey

6.8 VNCPublicAPIKeyResponder.h File Reference

VNC Public API Key Responder API.

Data Structures

- protocol <VNCPublicAPIKeyResponder>

  A VNC Public API key responder.

6.8.1 Detailed Description

VNC Public API Key Responder API.

The protocol documented here makes it possible to implement public API key injection for custom controls.

This behaviour can be added to existing control classes using Objective-C categories. For example:

```
@implementation YourCustomControlClass (VNCPublicAPIKeyResponderCategory)
// Your implementation...
@end
```

6.9 VNCPublicAPITouchResponder.h File Reference

VNC Public API Touch Responder API.

Data Structures

- protocol <VNCPublicAPITouchResponder >

  A VNC Public API touch responder.
- protocol <VNCPublicAPITouchResponderNode>

  A VNC Public API touch responder node.

6.9.1 Detailed Description

VNC Public API Touch Responder API.

The protocols documented here make it possible to implement public API touch injection for custom controls.

This behaviour can be added to existing control classes using Objective-C categories. For example:

```
@implementation YourCustomControlClass (VNCPublicAPITouchResponderCategory)
// Your implementation...
@end
```

6.10 VNCRPCaptureDelegate.h File Reference

Interface for screen capture.

Data Structures

- protocol <VNCRPCaptureDelegate >

6.10.1 Detailed Description

Interface for screen capture.

Copyright RealVNC Ltd. 2018. All rights reserved.

6.11 VNCServer.h File Reference

VNC Server.

Data Structures

- class VNCServer

  VNC Server class.

Enumerations

- enum VNCAuthMode { VNC_AUTHMODE_NONE, VNC_AUTHMODE_REV, VNC_AUTHMODE_PASS, V↩ NC_AUTHMODE_USER_PASS }

  Authentication mode values.

Variables

- VNCEXPORT const uint32_t VNC_INVALID_XKEYSYM_UNICODE

  Constant value that indicates invalid unicode value for XKeySym.

6.11.1 Detailed Description

VNC Server.

Copyright RealVNC Ltd. 2011-2018. All rights reserved.

6.11.2 Enumeration Type Documentation

6.11.2.1 VNCAuthMode

```
enum VNCAuthMode
```

Authentication mode values.

Set these as the VNCServer::authenticationMode property.

Enumerator

VNC_AUTHMODE_NONE	No authentication.
VNC_AUTHMODE_REV	Viewer can choose to request a username and/or a password from the server.
VNC_AUTHMODE_PASS	Viewer must send a password to the server.
VNC_AUTHMODE_USER_PASS	Viewer must send a username and a password to the server.

Definition at line 40 of file VNCServer.h.

6.11.3 Variable Documentation

6.11.3.1 VNC_INVALID_XKEYSYM_UNICODE

```
VNCEXPORT const uint32_t VNC_INVALID_XKEYSYM_UNICODE
```

Constant value that indicates invalid unicode value for XKeySym.

This is returned by convertXKeySymToUnicode: (VNCServer) if there is no unicode value corresponding to the XKeySym provided.

6.12 VNCServerBearer.h File Reference

VNC Bearer API.

Data Structures

- protocol <VNCBearer >

  Objective-C Bearer API.

6.12.1 Detailed Description

VNC Bearer API.

Copyright RealVNC Ltd. 2011-2018. All rights reserved.

6.13 VNCServerDelegate.h File Reference

VNC Server Delegate.

Data Structures

- protocol <VNCServerDelegate >

  VNC Server Delegate.

6.13.1 Detailed Description

VNC Server Delegate.

Copyright RealVNC Ltd. 2013-2018. All rights reserved.

6.14 VNCServerError.h File Reference

VNC Server Error Codes.

Enumerations

- enum VNCServerError {
  VNCServerErrorNone = 0, VNCServerErrorResources = 1, VNCServerErrorState = 2, VNCServerError↩
  PermissionDenied = 3,
  VNCServerErrorNetworkUnreachable = 20, VNCServerErrorHostUnreachable = 21, VNCServerError↩
  ConnectionRefused = 22, VNCServerErrorDNSFailure = 23,
  VNCServerErrorAddressInUse = 24, VNCServerErrorBadPort = 25, VNCServerErrorDisconnected = 26,
  VNCServerErrorConnectionTimedOut = 27,
  VNCServerErrorBearerAuthenticationFailed = 28, VNCServerErrorUSBNotConnected = 30, VNCServer↩
  ErrorUnderlyingLibraryNotFound = 31, VNCServerErrorBearerConfigurationNotProvided = 32,
  VNCServerErrorBearerConfigurationInvalid = 33, VNCServerErrorBearerLoadFailed = 34, VNCServer↩
  ErrorProtocolMismatch = 40, VNCServerErrorLoginRejected = 41,
  VNCServerErrorNotLicensedForViewer = 42, VNCServerErrorConnectionClosed = 43, VNCServerError↩
  InvalidCommandString = 44, VNCServerErrorUnsupportedAuth = 45,
  VNCServerErrorKeyTooBig = 46, VNCServerErrorBadCrypt = 47, VNCServerErrorNoEncodings = 48, VN↩
  CServerErrorBadPixelformat = 49,
  VNCServerErrorBearerNotFound = 50, VNCServerErrorSignatureRejected = 51, VNCServerError↩
  InsufficientBufferSpace = 52, VNCServerErrorLicenseNotValid = 53,
  VNCServerErrorFeatureNotLicensed = 54, VNCServerErrorInvalidParameter = 60, VNCServerErrorKey↩
  Generation = 63, VNCServerErrorUnableToStartService = 64,
  VNCServerErrorAlreadyExists = 65, VNCServerErrorTooManyExtensions = 66, VNCServerErrorReset = 67,
  VNCServerErrorDataRelayProtocolError = 80,
  VNCServerErrorUnknownDataRelaySessionId = 81, VNCServerErrorBadChallenge = 82, VNCServerError↩
  DataRelayChannelTimeout = 83, VNCServerErrorUserRefusedConnection = 100,
  VNCServerErrorCommandFetchFailed = 101, VNCServerErrorFailed = 102, VNCServerErrorNot↩
  Implemented = 103, VNCServerErrorCommandSuperseded = 106,
  VNCServerErrorEnvironment = 107, VNCServerErrorCaptureFrameBufferNotImplemented = 120 }

  Error codes that may be notified via the server delegate.

Functions

- VNCEXPORT NSString ∗ VNCConvertServerErrorToString (VNCServerError error)

  Utility function to produce a human-readable string from a server error value. Note that strings produced are not localised.

6.14.1 Detailed Description

VNC Server Error Codes.

Copyright RealVNC Ltd. 2013-2018. All rights reserved.

6.14.2 Enumeration Type Documentation

6.14.2.1 VNCServerError

```
enum VNCServerError
```

Error codes that may be notified via the server delegate.

Any error indicates that the VNC session has ended.

Enumerator

VNCServerErrorNone	No error.
VNCServerErrorResources	Insufficient system resources. When the device has insufficient resources to satisfy a request then this error will be reported. For instance if the device does not have enough free memory available then an API may fail with this error code.
VNCServerErrorState	An invalid API call was made. Some API calls are only valid when the server is in a particular state. For instance it is illegal to ask the server to connect while it is already connected. This error will be reported in such circumstances.
VNCServerErrorPermissionDenied	Insufficient device permissions.
VNCServerErrorNetworkUnreachable	General network error.
VNCServerErrorHostUnreachable	IP address could not be contacted.
VNCServerErrorConnectionRefused	Port could not be contacted.
VNCServerErrorDNSFailure	Domain name could not be resolved.
VNCServerErrorAddressInUse	Address/Port is in use. This error occurs when the server is told to listen on a port which is already being used by another application.
VNCServerErrorBadPort	Invalid port number. Valid TCP port numbers range from 1 to 65535 inclusive.
VNCServerErrorDisconnected	No network connection, The connection to the server was lost.
VNCServerErrorConnectionTimedOut	A general network time-out occured.
VNCServerErrorBearerAuthenticationFailed	The bearer failed to establish a connection due to an authentication failure. Note that this error indicates a transport-level failure, rather than an application-level failure. That is, the failure originates from within a pluggable bearer implementation, rather than with the Viewer SDK.

Enumerator

VNCServerErrorUSBNotConnected	USB Not Connected. There is nothing connected via USB or the device is unable to communicate via USB.
VNCServerErrorUnderlyingLibraryNotFound	Underlying Library Not Found. Failed to load a library for some particular functionality, for example OEM software for driving a particular type of communications.
VNCServerErrorBearerConfigurationNotProvided	A static configuration required by the bearer has not been provided.
VNCServerErrorBearerConfigurationInvalid	A static configuration provided for the bearer is invalid.
VNCServerErrorBearerLoadFailed	A bearer could not be loaded.
VNCServerErrorProtocolMismatch	Protocol incompatible with that of VNC Mobile Viewer. This error can occur if VNC Mobile Server is attempting to connect to VNC Viewer from a RealVNC edition other than VNC Mobile Solution, to a VNC-compatible application from a third party, or to something other than a VNC viewer (eg a HTTP server).
VNCServerErrorLoginRejected	User rejected authentication credentials.
VNCServerErrorNotLicensedForViewer	License incompatible with that of VNC Mobile Viewer.
VNCServerErrorConnectionClosed	VNC Mobile Viewer terminated the remote control session.
VNCServerErrorInvalidCommandString	Invalid command string.
VNCServerErrorUnsupportedAuth	Invalid authentication type. The connection is encrypted but VNC Mobile Server did not provide RSA keys. Alternatively, VNC Mobile Viewer specified an unsupported authentication type.
VNCServerErrorKeyTooBig	The RSA key is too large.
VNCServerErrorBadCrypt	RFB protocol or AES checksum is corrupt, or VNC Mobile Viewer did not have a matching private key.
VNCServerErrorNoEncodings	VNC Mobile Viewer specified an unsupported encoding.
VNCServerErrorBadPixelformat	VNC Mobile Viewer specified an unsupported pixel color depth.
VNCServerErrorBearerNotFound	Transport mechanism specified in command string missing or corrupt.
VNCServerErrorSignatureRejected	VNC Mobile Viewer signature specified in command string not the same as that of the actual VNC Mobile Viewer that connects.
VNCServerErrorInsufficientBufferSpace	The requested operation could not be completed due to insufficient buffer space.
VNCServerErrorLicenseNotValid	The requested operation could not be completed due to the provided license not being valid.
VNCServerErrorFeatureNotLicensed	The requested operation could not be completed due to the feature not being licensed.
VNCServerErrorInvalidParameter	An invalid parameter was passed to an API call. This can occur when registering a custom extension with an invalid name, or sending an extension message with an invalid length.
VNCServerErrorKeyGeneration	The RSA key generation algorithm failed.
VNCServerErrorUnableToStartService	The underlying VNC Mobile Server service could not be started.

Enumerator

VNCServerErrorAlreadyExists	A custom extension with the same name has already been registered.
VNCServerErrorTooManyExtensions	The maximum number of custom extensions (8) have already been registered.
VNCServerErrorReset	The server was reset by an external action
VNCServerErrorDataRelayProtocolError	VNC Data Relay received an invalid message from the server.
VNCServerErrorUnknownDataRelaySessionId	Either the command string contained an invalid VNC Data Relay session ID, or the communication channel to which it refers is no longer reserved.
VNCServerErrorBadChallenge	VNC Data Relay could not authenticate the server.
VNCServerErrorDataRelayChannelTimeout	VNC Mobile Viewer did not connect to the other end of the reserved VNC Data Relay communication channel in time.
VNCServerErrorUserRefusedConnection	Device user rejected prompt authorizing remote control.
VNCServerErrorCommandFetchFailed	HTTP or HTTPS request to command string web service failed.
VNCServerErrorFailed	General error.
VNCServerErrorNotImplemented	Feature not implement
VNCServerErrorCommandSuperseded	A command string for a different remote control session is received before the device user accepts the prompt authorizing the original session.
VNCServerErrorEnvironment	The application environment is unsupported.
VNCServerErrorCaptureFrameBufferNotImplemented	Screen capture is not implemented in this platform.

Definition at line 21 of file VNCServerError.h.

6.14.3 Function Documentation

6.14.3.1 VNCConvertServerErrorToString()

```
VNCEXPORT NSString* VNCConvertServerErrorToString (
            VNCServerError error )
```

Utility function to produce a human-readable string from a server error value. Note that strings produced are not localised.

Parameters

error	The error enum value.

Returns

A human-readable string representing the error.

6.15 VNCServerLogLevel.h File Reference

VNC Server Log Levels.

Enumerations

- enum VNCServerLogLevel {
  VNCSERVER_LOG_CRITICAL = 0, VNCSERVER_LOG_WARNING = 1000, VNCSERVER_LOG_NOTICE = 2000, VNCSERVER_LOG_INFO = 3000,
  VNCSERVER_LOG_DEBUG = 4000 }

  VNC Server Log Levels.

6.15.1 Detailed Description

VNC Server Log Levels.

Copyright RealVNC Ltd. 2013-2018. All rights reserved.

6.15.2 Enumeration Type Documentation

6.15.2.1 VNCServerLogLevel

```
enum VNCServerLogLevel
```

VNC Server Log Levels.

Currently the server SDK will only pass log levels to VNCServerDelegate::onServerLog:withLevel: that are one of these log levels; the mapped values are spaced to allow adding intermediate log levels in future.

See also

VNCServerDelegate::onServerLog:withLevel:

Enumerator

VNCSERVER_LOG_CRITICAL	CRITICAL log level. Highest/most important log level. Indicates critical failures.
VNCSERVER_LOG_WARNING	WARNING log level. Indicates significant problems.
VNCSERVER_LOG_NOTICE	NOTICE log level. Indicates important information.
VNCSERVER_LOG_INFO	INFO log level. Indicates potentially interesting information.
VNCSERVER_LOG_DEBUG	DEBUG log level. Indicates logging for debugging purposes.

Definition at line 24 of file VNCServerLogLevel.h.

6.16 VNCServerState.h File Reference

VNC Server States.

Enumerations

- enum VNCServerState {
  VNCStateDisconnected = 0, VNCStateSetup = 1, VNCStateAwaitingKey = 2, VNCStateGeneratingKey = 3,
  VNCStateListening = 4, VNCStateConnecting = 5, VNCStateConnectingRelay = 6, VNCStateAccepting = 10,
  VNCStateHandshaking = 11, VNCStateAcceptRemoteKey = 12, VNCStateAuth = 13, VNCStateReverseAuth
  = 14,
  VNCStateRunning = 15, VNCStateDisconnecting = 16, VNCStateInvalid = 255 }

  States provided to the server delegate.

Functions

- VNCEXPORT NSString ∗ VNCConvertServerStateToString (VNCServerState state)

  Utility function to produce a human-readable string from a server state value. Note that strings produced are not localised.

6.16.1 Detailed Description

VNC Server States.

Copyright RealVNC Ltd. 2013-2018. All rights reserved.

6.16.2 Enumeration Type Documentation

6.16.2.1 VNCServerState

```
enum VNCServerState
```

States provided to the server delegate.

The sequence of notifications varies according to the connection type.

Enumerator

VNCStateDisconnected	Server is idle.
VNCStateSetup	Server is setting the parameters for the RFB session.
VNCStateAwaitingKey	Server is waiting for an encryption key to be set.
VNCStateGeneratingKey	Server is generating an encryption key.
VNCStateListening	Server is listening for an incoming connection.
VNCStateConnecting	Server is initiating an outbound connection.
VNCStateConnectingRelay	Server is performing a data relay handshake.
VNCStateAccepting	Server is waiting for a connection to be accepted.
VNCStateHandshaking	Server is processing the RFB handshaking phase.
VNCStateAcceptRemoteKey	Server is waiting for a remote key to be accepted.
VNCStateAuth	Server is waiting for viewer credentials to be authenticated by the application.
VNCStateReverseAuth	Server is waiting for a reverse authentication password from the application.

Definition at line 21 of file VNCServerState.h.

6.16.3 Function Documentation

6.16.3.1 VNCConvertServerStateToString()

```
VNCEXPORT NSString* VNCConvertServerStateToString (
             VNCServerState state )
```

Utility function to produce a human-readable string from a server state value. Note that strings produced are not localised.

Parameters

state	The state enum value.

Returns

A human-readable string representing the state.

Index