

# API Calls

The following section describes the full set of QC API calls. The API uses C interfaces, which are supplied for use in your projects, are the file QCAPI.h.

## QCInstalled

INTERFACE  
QCErr  
QCInstalled(void);

DESCRIPTION  
Test for the existence of the QC extension and availability of the API.

RESULTS  
Returns kQCNoErr (0) if the extension is loaded and no errors have occurred.  
Can possibly return kQCNotInstalled, kQCAPIMismatch. See Appendix A for result code descriptions.

## QCIsActive

INTERFACE  
Boolean  
QCIsActive(void);

DESCRIPTION  
Test to see if the current state of QC is active or not.

RESULTS  
Returns true (1) if testing is currently active on an application.  
Returns false (0) if testing is not currently active on any application heaps.

## QCActivate

INTERFACE  
QCErr  
QCActivate(THz);

DESCRIPTION  
Activate QC testing on a given heap. Passing NIL will activate testing on the current application's heap zone.

RESULTS  
Possible error results are kQCNotInstalled. If QC is already active on a particular heap, testing will cease on the current heap and begin on the specified heap.

## QCDeActivate

INTERFACE  
Boolean  
QCDeActivate(void);

DESCRIPTION  
Deactivate testing on the currently active application heap.

## QCTestingHeap

INTERFACE  
THz  
QCTestingHeap(void);

DESCRIPTION  
Get the current heap zone being tested. This call can be useful if you plan on deactivating all tests, then reactivating them later on in your code, possibly alternating heaps in between.

## QCGetState

INTERFACE  
QCStateHandle  
QCGetState(void);

DESCRIPTION  
Get the current state of QC testing for later restore. This call is useful if you wish to save the current state of QC testing, then set/clear the tests you desire for a particular section of code, then restore the state of testing to what it was before.

It is important to note that the API allocates the result and it is the responsibility of the caller to dispose of that memory with QCDisposeState().

WARNING  
The format of the data contained in QCStateHandle is undocumented and may change in the future. Any direct manipulation of the contents of QCStateHandle is discouraged and may not work in future versions of QC.

RESULTS  
If QC is not currently active, NIL will be returned.

## QCSetState

INTERFACE  
QCErr  
QCSetState(QCStateHandle);

DESCRIPTION  
Set the current state of QC testing to the state saved in QCStateHandle. Testing must be active or kQCNotActive will be returned.

It is the responsibility of the caller to dispose of the memory allocated in QCStateHandle with QCDisposeState().

WARNING  
The format of the data contained in QCStateHandle is undocumented and may change in the future. Any direct

manipulation of the contents of QCStateHandle is discouraged and may not work in future versions of QC.

## QCDisposeState

### INTERFACE

QCErr  
QCDisposeState(QCStateHandle);

### DESCRIPTION

Dispose of valid QCStateHandle information (acquired from a call to QCGetState()).

### RESULTS

If the given state information is not valid, kQCInvalidParam will be returned.

## QCInstallHandler

### INTERFACE

QCErr  
QCInstallHandler(QCCallBackUPP, long);

### DESCRIPTION

Install a routine that will be called by QC when errors are encountered. The supplied QCCallBackUPP routine is called by QC whenever an error is encountered for an active test. The 'long' value is passed to the callback routine in QCPBRecPtr->data. A good way to get back at your globals is to use this long to save/restore the current A4/A5. The specified routine is called with a pointer to a QCPBRec (see Appendix C). This routine must be resident at all times, or until the handler is removed. See Appendix C for details about the routine interface. It is important to note that this call will replace any previous handler that may have been installed.

### Multiple Error Handlers

You can install multiple error handlers with the QCInstallHandler() and remove specified handlers with the QCRemoveHandler() call. These error handlers can be installed by different applications other than the one being tested. As an example, the BadAPPL sample application installs an error handler to detect if an intentional error was detected. Another application (e.g. another process) could install an error handler to log the output. If the process that installed an error handler forgets to remove it, and then quits (ExitToShell), QC will not call that error handler and it will be removed from the list.

## QCRemoveHandler

### INTERFACE

QCErr  
QCRemoveHandler(QCCallBackUPP);

### DESCRIPTION

Remove (de-install) the specified error handler. This will be the same QCCallBackUPP previously installed with QCInstallHandler().

### RESULTS

If the specified error handler is not installed, kQCErrorHandlerNotFound will be returned. If successful, kQCNoErr will be returned.

## QCGetErrorText

## INTERFACE

QCErr

QCGetErrorText(QCResult, StringPtr);

## DESCRIPTION

Given a valid QC error value (see Appendix D), get the text string that QC would report/log for that error. This can be useful if you have installed an error handler but want access to the same error string QC would report for that error. QC does not have to be active for this call.

## RESULTS

Possible errors could be kQCInvalidParam or kQCNotInstalled.

## QCGetTestState

## INTERFACE

Boolean

QCGetTestState(QCSelector);

## DESCRIPTION

Test whether a given test is currently active or not. If the test is active true (1) is returned. If the test is inactive or any error was encountered, false (0) is returned.

## QCSetTestState

## INTERFACE

QCErr

QCSetTestState(QCSelector, Boolean);

## DESCRIPTION

Set the test specified by QCSelector to an on (true) or off (false) state.

## RESULTS

Possible errors could be kQCInvalidParam or kQCNotActive.

## QCGetTestOptions

## INTERFACE

QCErr

QCGetTestOptions(QCTestOptionsPtr);

## DESCRIPTION

Get the test options for the test specified in the 'testID' parameter if the QCOptionsPtr. Depending on which test you are setting specific options for, different parameters of the QCOptionsPtr will be returned.

See Appendix E for further information. Version 1.0 of QC only has test options for the 'AutoLaunch' and 'Reasonable Allocation' tests.

## RESULTS

Possible errors could be kQCInvalidType or kQCNotInstalled.

## QCSetTestOptions

## INTERFACE

QCErr

QCSetTestOptions(QCTestOptionsPtr);

## DESCRIPTION

Set the test options for the desired test, as specified in the 'testID' parameter if the QCTestOptionsPtr. Depending on which test you are setting specific options for, different parameters must be filled in. Please note that this call does not change the specified test's active state (on/off), just the options associated with it.

See Appendix E for further information. Version 1.0 of QC only has test options for the 'AutoLaunch' and 'Reasonable Allocation' tests.

## RESULTS

Possible errors could be kQCInvalidType or kQCNotInstalled.

## QCHeapCheckNow

### INTERFACE

QCErr

QCHeapCheckNow(void);

### DESCRIPTION

Perform a heap check on the currently active heap, whether qcCheckHeap (see Appendix B) is active or not.

Depending on the state of the test qcCheckSystemHeap this call will perform a block bounds check on the system heap as well.

### RESULTS

Possible error could be kQCNotInstalled.

## QCScrambleHeapNow

### INTERFACE

QCErr

QCScrambleHeapNow(void);

### DESCRIPTION

Perform a heap scramble on the currently active heap, whether qcHeapScramble (see Appendix B) is active or not.

Depending on the state of the test qcCheckSystemHeap this call will perform a heap scramble on the system heap as well.

### RESULTS

Possible error could be kQCNotInstalled.

## QCBlockBoundsCheckNow

### INTERFACE

QCErr

QCBlockBoundsCheckNow(void);

### DESCRIPTION

Perform a block bounds check on the currently active heap, whether qcBlockBoundsChecking (see Appendix B) is active or not. Depending on the state of the test qcCheckSystemHeap this call will perform a block bounds check on the system heap as well.

## RESULTS

Possible error could be kQCNotInstalled.

## QCVerifyHandle

### INTERFACE

QCErr

QCVerifyHandle(Handle);

### DESCRIPTION

Given a handle, validate that it is a valid handle within the currently tested heap zone or the system heap zone. This test will be performed even if the qcValidateHandlePointer test is not active.

## QCVerifyPointer

### INTERFACE

QCErr

QCVerifyPointer(Ptr);

### DESCRIPTION

Given a pointer, validate that it is a valid pointer within the currently tested heap zone or the system heap zone. This test will be performed even if the qcValidateHandlePointer test is not active.