

Appendix A

Types/Constants

The following types and constants are used for input and result codes by QC API calls. They are defined along with the interfaces in the file 'QC_API.h'.

```
typedef long QCError;
// QC error code type
typedef long QCType;
// QC type codes
typedef Handle QCStateHandle;
// QC save/restore state data
```

The following are possible QCResult values returned by all API calls:

```
//
API result codes
enum
```

```
kQCNoErr
```

```
= 0,
```

```
kQCInternalErr
```

```
= 9,
```

```
kQCNotInstalled,
```

```
// QC extension not installed
```

```
kQCNotActive,
```

```
// QC is not testing; must be to complete
```

```
kQCInvalidParam,
```

```
// invalid parameter passed to routine
```

```
kQCInvalidType,
```

```

// invalid QCType given to a routine
kQCPBRecMismatch,

// QCPBRec size mismatch.  QCAPI library in use

//
does not match API loaded extension needs.
kQCAPIMismatch,

// The QCAPI and loaded extension API's don't match!
kQCErrorHandlerNotFound,
// QCRemoveHandler result when given proc was not

//
installed.
;

```

Appendix B

QC Selectors

The following selector types are used in QC control and status calls to set/check individual QC tests. They are passed as QCSelector values and consistently used throughout the interface for controlling QC. They are also passed in the QCPBRec to an installed QCCallBack routine.

```

// Possible QCSelector ids
#define
qcAutoLaunch
'auto'
// auto launch this app/file

```

```

#define
qcCheckSystemHeap
'cksh'
    // check system heap
#define
qcValidateMasterPointers
'vlmp'
// validate master pointers
#define
qcValidateHandlePointers
'vlhp'
// validate handles/pointers
#define
qcDetectWriteToZero
'dtwz'
    // detect write to zero
#define
qcDerefZeroCheck
'drzc'
    // detect deref zero
#define
qcReasonableAllocation
'rall' // reasonable allocation

// checks
#define
qcCheckDisposeRelease
'dprl'
    // check DisposeHandle

// ReleaseResource
#define
qcScrambleHeap
'schp'
    // scramble heap
#define
qcPurgeHeap
'pghp'
    // purge heap
#define
qcCheckHeap
'ckhp'
    // check heap
#define
qcInvalidateFreeMemory
'infm' // trample (invalidate)
// free memory
#define
qcCheckSystemCode
'csys' // check system code
#define
qcErrorReporting
'erpt' // error reporting
#define
qcDebugBreaks
'dbrk' // debugger breaks
#define
qcBeepNotify
'beep' // beep on
// activate/deactivate

```

```

#define
qcIconNotify
'sicn'          // rotate small icon          // when active

#define
qcBlockMoveChecking
'bkmv'
    // check BlockMove calls
#define
qcBlockBoundsChecking
'bbck'
    // block bounds checking
#define
qcGrowLockChecking
'grlk'
    // grow locked block checking
#define
qcGrowNonRelocChecking
'gron'  // grow non-reloc block          // checking

#define
qcAllTestsMask 'mask'

    // all tests mask          // activate/deactivate all

```

Appendix C

QC Handlers

ou can have complete control over QC error handling by installing a routine that QC will call whenever an error is encountered. Your application then has the choice of ignoring or handling that error entirely. If your callback routine wants to handle the error entirely with no further interaction by QC, return true (1), else return false(0). See QCInstallHandler() and QCRemoveHandler() for information on installing and removing error handlers. The following list shows all the possible errors your callback routine can receive from QC.

```
struct QCPBRec
```

```
QCType
```

```
testID;
```

```
    // the selector id of the test
```

```
    // (currently not used)
```

```
QCErr
```

```
error;
```

```
// error detected (see below)
```

```
long
```

```
data;
```

```
// data/address from QCInstallHandler()
```

```
char
```

```
*errString;
```

```
// error string being reported
```

```
long
```

```
lastTestedTrapPC;
```

```
// PC of last trap QC detected error at
```

```
long
```

```

lastTrapPC;

// PC of last trap executed from
// user code
;
typedef struct QCPBRec QCPBRec;
typedef
QCPBRec *QCPBPtr;

//
PowerPC mixed mode support
enum

uppQCCallBackProcInfo = kCStackBased

| RESULT_SIZE(SIZE_CODE(sizeof(long)))

| STACK_ROUTINE_PARAMETER(1, SIZE_CODE(sizeof(QCPBPtr)))
;

// QCCallBack routine descriptors
#if GENERATINGCFM
typedef UniversalProcPtr QCCallBackUPP;
#define NewQCCallBackProc(userRoutine)

\

(QCCallBackUPP) NewRoutineDescriptor((ProcPtr)(userRoutine),
uppQCCallBackProcInfo, GetCurrentISA())
#else
typedef
ProcPtr
QCCallBackUPP;
#define
NewQCCallBackProc(userRoutine)

\

((QCCallBackUPP) (userRoutine))
#endif

```

Appendix D

QC Error Codes

he following are error codes that QC encounters and reports to an installed error handler routine in the 'errorID' field of the given QCParamPtr.

```
enum
// API error detection result codes (given to DebugCallback
// routine (if installed)

kErrorBase = 300,
    // base index of all errors

kBadBlockLenErr,
    // invalid physical block length

kBadRelHandErr,
    // Offset from zone start to
    // masterPtr invalid

kLostMasterPtrErr,
    // master pointer is not in the heap

kBadMasterPtrErr,
    // MasterPtr does not point to data

kBadNonRelocErr,
    // Bad Non-reloc block: heap addr
    // must follow physSize.

kBadBlockTypeErr,
    // invalid block type (not reloc,
    // non-reloc, or free)

kBadLastBlockErr,
    // invalid physical block length
    // (last block)

kWriteToZeroErr,
    // write to location zero detected

kNilHandleErr,
    // nil handle error

kHandleInFreeErr,
    // handle is in a free block

kBadHandleErr,
    // handle is bad - not at start of
    // relocatable block

kBadPtrErr,
    // Ptr does not point to
    // non-relocatable block

kNilPointerErr,
```

```

        // nil pointer detected

kUnused,

kHeapStartEndErr,
    // bkLim of heap has heap end before

        // it starts

kFreeByteHeapErr,
    // no. of free bytes exceeds heap

        // size

kGrowZoneMismatchErr,        // grow zone function mismatch!

kUnreasonableNewHandleErr,
    // new handle size is questionable

kUnreasonableNewPtrErr,

kUnreasonableSetHandleSizeErr,

kUnreasonableReallocHandleErr,

kUnreasonableSetPtrSizeErr,

kReleaseHandleErr,
    // called release resource on a

        // handle

kDisposResourceErr,
    // Trying to perform a DisposHandle

        // on a resource

kBlockMoveDestFree,
    // Destination ptr for BlockMove is

        // in a free block

kBlockMoveDestMultiple,
    // Destination ptr for BlockMove

        // spans multiple blocks

kBlockMoveHeadOverwrite,
    // Blockmove will overwrite a block

        // header

kBlockMovePadOverwrite,
    // Blockmove will overwrite padding

        // in a block

kUnreasonableBlockMoveSize,
    // Blockmove attempting to move an

```



```
                                // unreasonable size

kMemErrDetected,
    // a MemErr value has been detected

                                // after a call

kEmptyHandleErr
    // Empty handle used by routine that

                                // needs data

kPtrBoundsErr,
    // Write past end of non-relocatable

                                // block detected

kHandleBoundsErr,
    // Write past end of relocatable

                                // block detected

kFreeMemOverwrite,
    // Write has been made to a free

                                // block

kGrowLock,
    // App is growing a locked block

kGrowPtr,
    // App is growing a non-relocatable

                                // block

kBlockMoveNilSrc,

// source pointer is NIL!

kBlockMoveNilDest,
```

```

// destination pointer is NIL!

kErrorLimit
    // max error id place holder
;

```

Appendix E

QC Test Options

The following struct is used to get and set individual test options with the QCGetTestOptions() and QCSetTestOptions() calls. Please note that use of these calls does not change the state (on/off) of the desired test. For example, you can get and set AutoLaunch options without turning on or off the AutoLaunch test. That would have to be done with QCGetTestState() and QCSetTestState().

```

// Specific test option structs for use in QCGetTestOptions()/QCSetTestOptions
struct QCAutoLaunchRec

```

```

short
which;          // 1 = _InitGraf; 2 = _Get1Resource;

```

```

short
id;             // id for useGet1Resource

```

```

OSType
type;           // type for useGet1Resource
;

```

```

typedef struct QCAutoLaunchRec QCAutoLaunchRec;

```

```

// Reasonable Allocation data contained within each LaunchRec
struct QCReasonAllocRec

```

```

short
which;          // 1 = use app heap; 2 = use specified

```

```

long
size;
;
typedef struct QCReasonAllocRec QCReasonAllocRec;

```

```

struct QCBlockBoundsRec

```

```

short

tagSize;        // block bounds size to tag blocks with
;
typedef
struct QCBlockBoundsRec QCBlockBoundsRec;

```

```

union QCOptionData

```

```

QCAutoLaunchRec

```

```
autoLaunch;
```

```
// AutoLaunch info
```

```
QCReasonAllocRec  
reasonableAlloc; // Reasonable Allocation info
```

```
QCBlockBoundsRec  
blockBounds;
```

```
// BlockBounds info
```

```
;  
typedef  
union QCOptionData QCOptionData;
```

```
struct QCTestOptions
```

```
QCType  
testID; // the test this data belongs to
```

```
QCOptionData  
optionData;  
;  
typedef struct QCTestOptions QCTestOptions;  
typedef QCTestOptions *QCTestOptionsPtr;
```