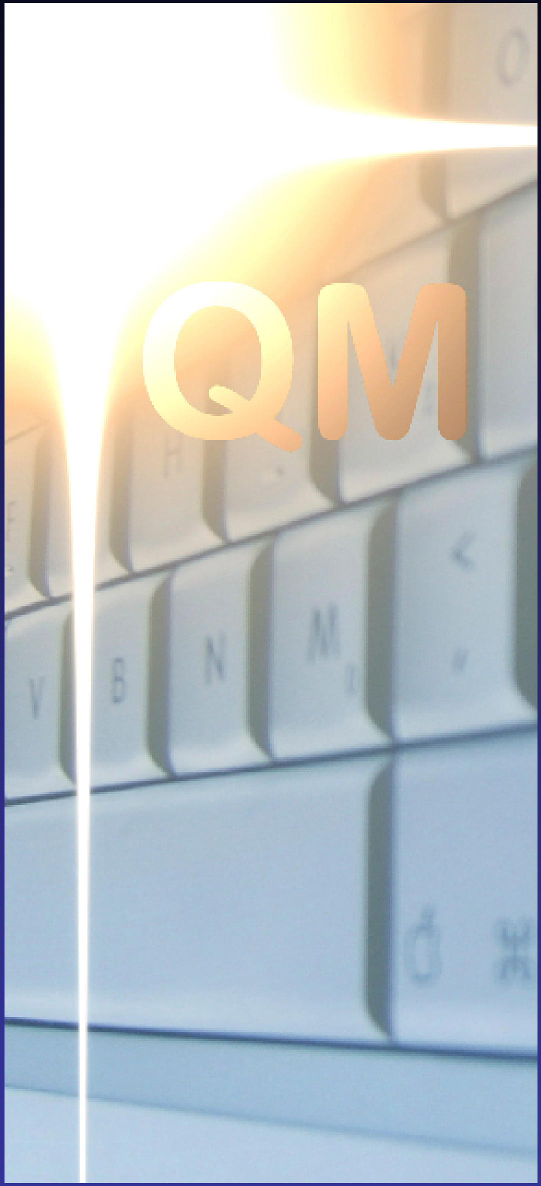


OpenQM

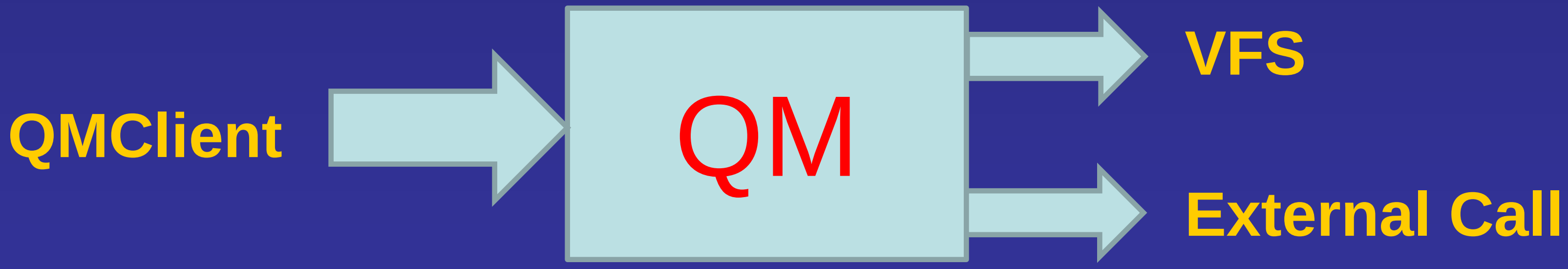
Connectivity and Integration

Martin Phillips
Ladybridge Systems Ltd

OpenQM Connectivity and Integration

QM provides many interfaces, both inward and outgoing, to connect to other software. The main ones are:

- QMClient
- The Virtual File System
- The External Call Interface



QMClient

QMClient allows external software to connect to a QM system.

It can then:

- Open files
- Read, write, delete, select, etc with full locking
- Execute commands, including interaction
- Execute programs, subroutines and objects

All access is subject to system/application level security controls.

Who Uses QMClient?

QMClient is used internally by

- DesignBais
- mv.Net (Blue Finity)
- OpenInsight (Revelation)
- eMotion (Arcoinet)
- MITS

and many others.

Where Can QMClient be Used?

QMClient is callable from

- C, C++, C#, Objective C, etc
- Visual Basic, PowerBasic, PureBasic
- Java, ASP, PHP
- QMBasic

and many others.

Tablet Devices

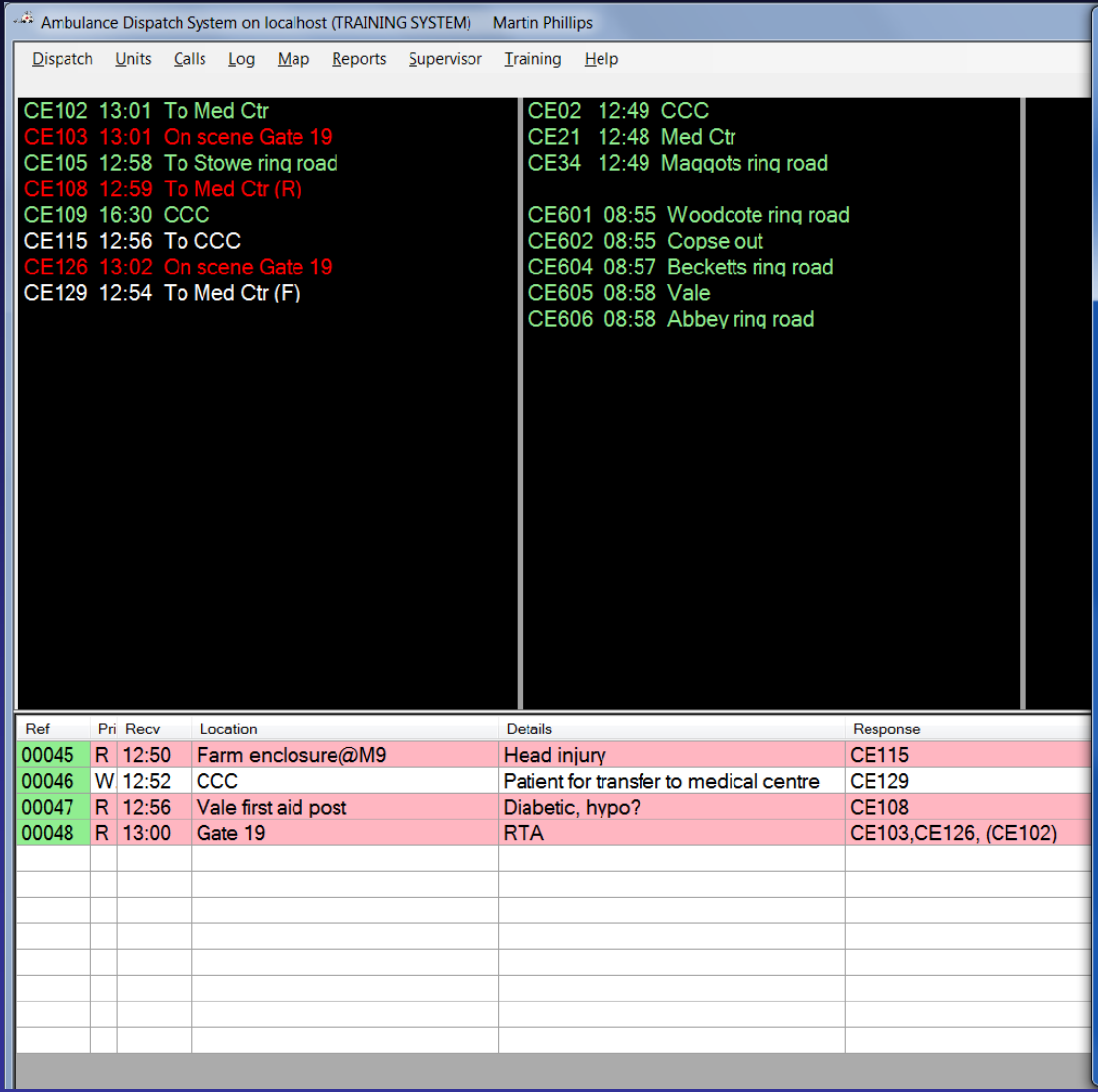
The QMClient API is available for both iOS and Android

This allows native application developers to access data on a QM server

The Android API provides the same object interface as the Windows Java API.

The iOS version exposes the QMClient functions directly for use in Objective C.

A Practical Example

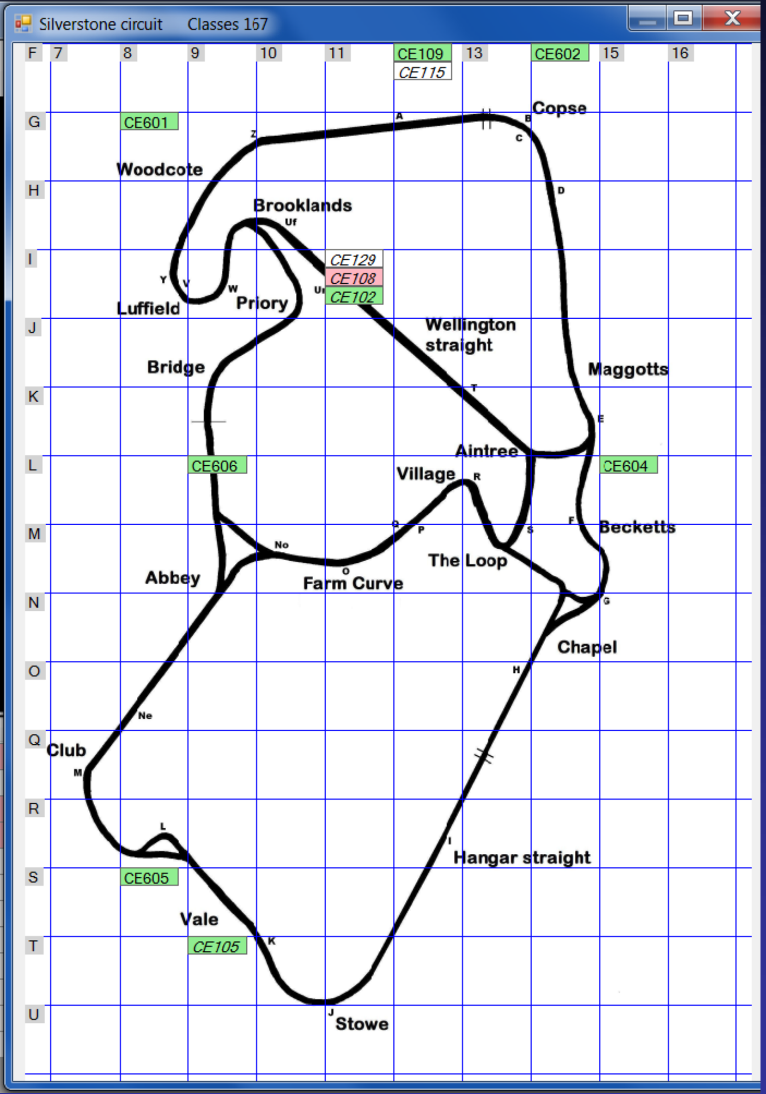


Ambulance Dispatch System on localhost (TRAINING SYSTEM) Martin Phillips

Dispatch Units Calls Log Map Reports Supervisor Training Help

```
CE102  13:01  To Med Ctr                CE02   12:49  CCC
CE103  13:01  On scene Gate 19          CE21   12:48  Med Ctr
CE105  12:58  To Stowe ring road        CE34   12:49  Maggots ring road
CE108  12:59  To Med Ctr (R)
CE109  16:30  CCC                       CE601  08:55  Woodcote ring road
CE115  12:56  To CCC                    CE602  08:55  Copse out
CE126  13:02  On scene Gate 19          CE604  08:57  Becketts ring road
CE129  12:54  To Med Ctr (F)            CE605  08:58  Vale
                                        CE606  08:58  Abbey ring road
```

Ref	Pri	Recv	Location	Details	Response
00045	R	12:50	Farm enclosure@M9	Head injury	CE115
00046	W	12:52	CCC	Patient for transfer to medical centre	CE129
00047	R	12:56	Vale first aid post	Diabetic, hypo?	CE108
00048	R	13:00	Gate 19	RTA	CE103,CE126, (CE102)



How Does QMClient Work?

QMClient makes a network connection to a QM server

The conversation between the client and server is encrypted for security

Alternatively, a pipe connection can be used to QM on the same system

A single client can open many simultaneous connections.

Character String Styles

Three styles of character string are supported:

- Standard 8-bit (SBCS)
- UTF-8 encoded (MBCS)
- Wide character (Unicode, wchar_t)

These are independent of whether the QM server is in ECS mode.

Connecting to a QM Server from C

```
if (QMConnect(server, -1, username,
               password, account))
  {
   …processing…
  }
```

Reading a Record

```
fno = QMOpen(“MYFILE”);
if (fno)
 {
  rec = QMRead(fno, id, &ErrNo);
  if (ErrNo == 0)
   {
    …processing…
    QMFree(rec);
   }
```

Processing Multivalues

```
s = QMExtract(rec, 1, 0, 0);
n = QMDcount(s, VM);
for(i = 1; i <= n; i++)
 {
  p = QMExtract(s, 1, i, 0);
  printf(“%s\n”, p);
  QMFree(p);
 }
QMFree(s);
```

Calling a Subroutine (1)

```
char code[20] = “”;
QMCall(“TAXCODE”, 2, client, code);
…processing…
```

Calling a Subroutine (2)

```
char * code = NULL;
QMCallx(“TAXCODE”, 2, client, code);
code = QMGetArg(2);
…processing…
QMFree(code);
```

Using a Class Module

```
obj = QMCreateObject(“MYCLASS”, 0);

QMSet(obj, “name”, 1, “value”);

p = QMGet(obj, “name”, 0);
…processing…
QMFree(p);
```

Security Issues

Applications normally implement security rules inside the server software without the user having access to the command prompt

Use of QMClient could potentially weaken security by allowing a malicious user to write a client application that accesses data normally hidden by the application

This is not acceptable.

QMCLIENT Configuration Parameter

Value 0	No restrictions
Value 1	Cannot open files or execute commands
Value 2	Additionally, can only execute programs, subroutines and classes compiled with $QMCLIENT option.

QMCLIENT Configuration Parameter

The initial value of this parameter is taken from the QM configuration data

An application can increase the parameter value to impose tighter restrictions, typically from the LOGIN paragraph on connecting to the server

An application cannot decrease the parameter value.

Why Use QMClient from QMBasic?

Using the QMClient class module from a QMBasic program allows an application to execute commands, programs, subroutines and classes on a remote server

We use this in our business systems to coordinate actions in servers based in the UK and US.

The Virtual File System (VFS)

The VFS allows an application to access external data sources / targets as though they are QM files

Two variants:

- Internal VFS – Uses a QMBasic class module
- External VFS – Uses a C program.

How Does the VFS Work?

The VFS captures file system actions:

- Open / close file
- Record and file level locking actions
- Read / write / delete
- Indexing operations
- Clear file
- Select list operations
- FILEINFO()

For each operation, it emulates the normal file system action.

VFS VOC References

The F-type VOC entry to define a VFS file is

```
F
VFS:handlername:filename
VFS:handlername:dictname
```

For an external VFS handler the *handlername* is prefixed by EXT

The *filename* element is passed into the VFS handler and can be used in any way the application designer wishes.

Virtual File System Use

The Virtual file system can be used to interact with any other data store for which suitable interfaces are provided

The examples that follow allow access to files in a UniVerse system.

The Internal VFS

A skeleton class module is in the BP file of the QMSYS account

This contains extensively commented template versions of each function or subroutine in the class module.

The Internal VFS

Example – READ / READL / READU

```
* V$READ
* Called by QMBasic operations that read a record.
* The record id is passed in via the ID argument.
* The FLAGS argument is bit significant and is formed from the following
* additive values:
*   2 The LOCKED clause is present.
*   4 Obtain a shared (READL) lock.     } At most one of these flags will be
*   8 Obtain an update (READU) lock.   } set. A simple READ sets neither.
* Other bits may be set and should be ignored.
* The record data should be returned via the STR argument.

   public function v$read(id, flags, str)
      str = '...your data...'
      return 0
   end
```

The Internal VFS

Example – READ / READL / READU

```
public function v$read(id, flags, str)
      str = '...your data...'
      return 0
   end
```

The Internal VFS - Example Use

Using file format information published by Rocket Software, it is not difficult to build a QM VFS handler that can read UniVerse files

Because this has no access to UniVerse’s group locking mechanism, only read access is practical and the file must not be being updated while the VFS is used.

The External VFS

A skeleton C program can be downloaded from the openqm.com web site

Again, this contains template code for the application developer to expand

Use of macros makes coding largely independent of whether the server is in ECS mode or not.

The External VFS

Example – READ / READL / READU

```
/* v_read()  -  Read a record
   The flags argument is bit significant and is formed from the following
   additive values:
      2  The LOCKED clause is present (e.g. READU)
      4  Obtain a shared (READL) lock     } At most one of these flags will
      8  Obtain an update (READU) lock  } be set. A READ sets neither.
   Other bits may be set and should be ignored.
   The record data should be returned via the rec argument.              */

static int v_read(FILEINFO * fptr, QMString * id, int flags, QMString ** rec)
{
 *rec = AllocQMString(11, qmstr("Hello world"));

 return 0;
}
```

The External VFS

Example – READ / READL / READU

```
static int v_read(FILEINFO * fptr,
    QMString * id, int flags, QMString ** rec)
{
 *rec = AllocQMString(11, qmstr("Hello world"));

 return 0;
}
```

The External VFS - Example Use

Using the InterCall API that allows client software to access UniVerse files, it is easy to write a QM VFS handler that accesses UniVerse files

Because InterCall performs group locking, full read and write access is possible, even while the file is in use by UniVerse.

The External VFS

READ / READL / READU from UniVerse

```
static int v_read(FILEINFO * fptr, QMString * id, int flags, QMString ** rec)
{
 long int lock_type, id_len, status, code;
 long int rec_len = id->len;
 long int max_len = MAX_DATA_LEN,

 if (flags & 8) lock_type = IK_READU;
 else if (flags & 4) lock_type = IK_READL;
 else lock_type = IK_READ;

 *rec = AllocQMString(MAX_DATA_LEN, NULL);
 ic_read(&(fptr->u2_id), &lock_type, id->data, &id_len, (*rec)->data, &max_len,
         &rec_len, &status, &code);
 (*rec)->len = rec_len;
 if (status) return ER_LCK;  /* Record is locked by another user */
 return (code == 0)?0:ER_RNF;
}
```

The External VFS

The external VFS handler to access UniVerse files via InterCall is available for download on the openqm.com web site

Some functions such as index scanning are not available because they are not supported by InterCall

Similar handlers could be developed to connect to other external data stores.

The External VFS - Dictionaries

Because dictionary records that contain compiled code (e.g. I-types) are not compatible with UniVerse, a local copy of the dictionary must be used.

The External Call Interface

Sometimes an application needs to call subroutines written in other languages

There is a serious danger that doing this from within a QM process could result in system instability such as file corruptions if the user written code misbehaves

The External Call Interface uses a separate child process to maintain integrity of the managed code environment of QM.

The External Call Interface Handler

The application developer writes a separate C program than includes an interface library supplied with QM

This program will receive requests from the parent QM process and should then perform the required operation and return any result back to QM

The overheads of this child / server architecture are low and preserve system integrity.

The External Call Interface Handler

The child process uses a simple request processing loop:

```
while(GetCallRequest(function_name))
 {
  err = 0;    /* Default returned status value */
  if (!strcmp(function_name, "STAT"))
   {
    if (!stat(GetString(1), &statbuf)) ReturnInteger(0, statbuf.st_size);
    else { ReturnInteger(0, -1); err = errno; }
   }
  else      /* Function name not recognised */
   {
    err = -ER_FUNCNAME;
   }

  CallCompleted(err);   /* Send updated arguments and result */
 }
```

The External Call Interface Handler

The request loop:

```
while(GetCallRequest(function_name))
 {
  err = 0;    /* Default returned status value */
   ... process request ...
  CallCompleted(err);
 }
```

The External Call Interface Handler

Accessing input arguments

GetString(1)	Fetches argument 1 as pointer to an 8-bit string

Other functions are:

GetInteger(1)	Fetches argument 1 as a 32 bit signed integer
GetFloat(1)	Fetches argument 1 as a double
GetStringW(1)	Fetches argument 1 as pointer to a wchar_t string
StringLength(1)	Returns the number of characters in argument string

Type conversion is performed automatically as needed

Strings are null terminated but can contain char(0).

The External Call Interface Handler

Updating argument values

ReturnString(1, str, len)	Returns 8-bit string
ReturnInteger(1, value)	Returns 32 bit signed integer
ReturnFloat(1, value)	Returns double
ReturnStringW(1, str, len)	Returns wchar_t string

Argument 0 is the returned value of the QMBasic function.

The External Call Interface Handler

Example - Calling the stat() function

```
if (!strcmp(function_name, "STAT"))
   {
    if (!stat(GetString(1), &statbuf))
       ReturnInteger(0, statbuf.st_size);
    else { ReturnInteger(0, -1); err = errno; }
   }
```

Using External Functions in QMBasic

An external function is defined in QMBasic with:

```
DEFFUN name(arg1, arg2) EXTERNAL
```

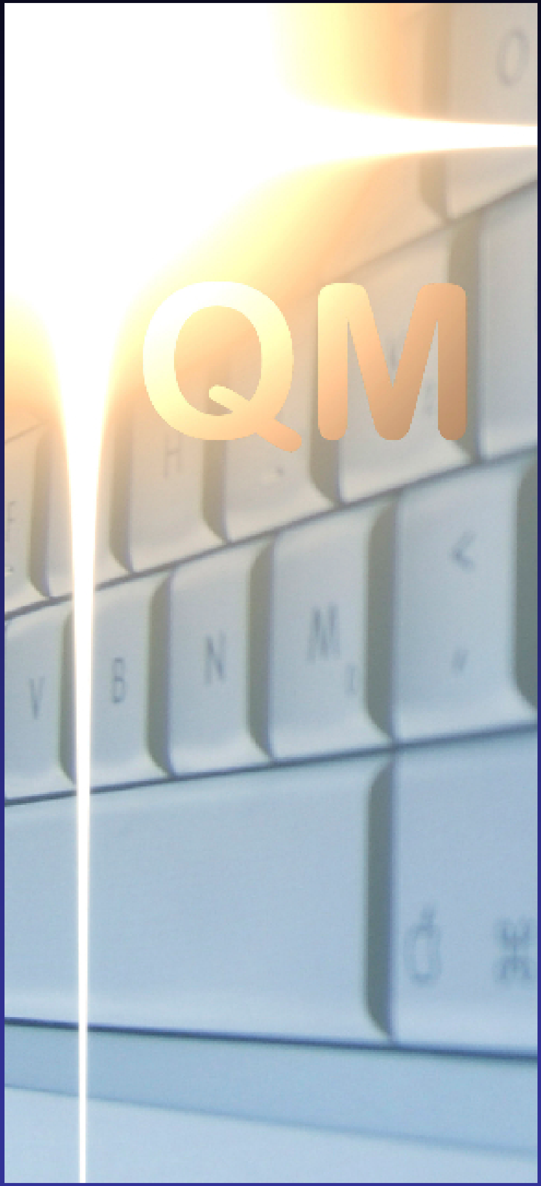
Arguments may be prefixed with

IN:	Argument is input only
OUT:	Argument is output only

Use of these qualifiers can reduce data passed between the child and parent processes.

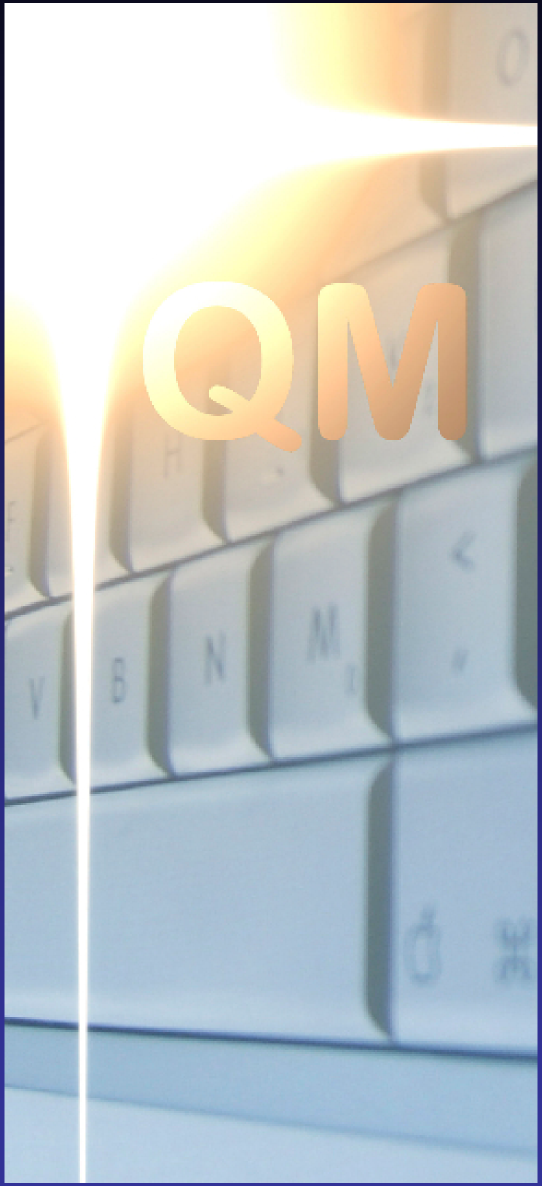
And Finally….

We are keen to receive suggestions for other interfaces that will improve QM.



OpenQM

QUESTIONS?



OpenQM