OpenQM

Extended Character Set Support

Martin Phillips
Ladybridge Systems Ltd

Extended Character Set Support

Definition:

In QM, the term ECS relates to support for an extended range of characters by comparison to non-ECS versions.

The term ECS is also used to refer to the set of characters that can be represented.

In other contexts the term ECS is sometimes used to refer to the upper half of the 8-bit character set.

What We Are Going To See Today....

What is ECS?

How does it affect applications?

Character maps

Installation

What is ECS?

Until now, QM has worked with the 8-bit character set.

The upper half of this has various definitions to meet local language requirements.

This leads to ambiguity.

Applications may need to work with more than 256 different characters.

What is ECS?

QM adopts the Unicode Basic Multilingual Plane (BMP).

This contains the characters used by most business languages.

It does not include archaic languages such as Linear B, symbols such as musical notation, or constructed languages such as Klingon.

All of these can be handled by mapping them into the Private Use Area of the BMP.

What is ECS?

Internally, ECS data is represented as a 16 bit value, allowing 65536 different characters.

Externally, QM will translate data to an appropriate representation according to user defined rules for each data pathway.

The mark characters remain in their normal locations.

The five accented characters defined by Unicode at these locations are moved.

How Are Applications Affected?

Applications running on non-ECS systems will continue to run on ECS without change or recompilation.

Data files created on non-ECS mode systems are fully accessible to ECS mode systems.

Data files created on ECS mode systems are fully accessible to non-ECS systems unless created in ECS mode.

Character Maps

Define the type of each character (letter, digit, space, etc).

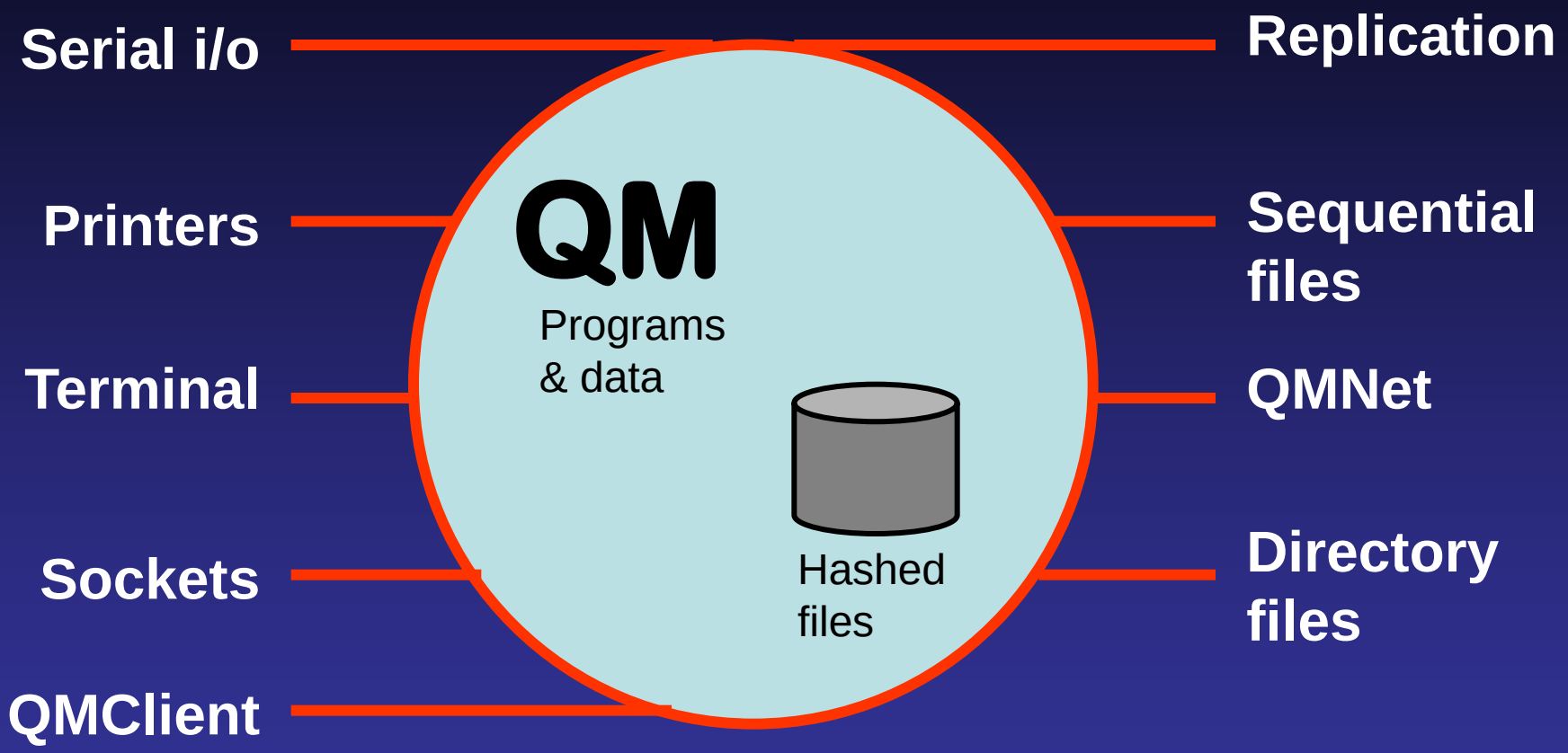
Define the upper / lower case pairings.

Define the sort order.

Identifies double width characters.

Different users can use different maps resulting, for example, in reports being sorted according to local language conventions.

Data Pathways



Hashed Files

Files created in ECS mode can store records with ECS characters in their ids or data.

Configuring an existing file for ECS mode does not make it larger.

Only data that needs to be stored in ECS form is stored in that way.

Directory Files

Directory files can use an encoding to transform ECS data into some external form such as UTF-8.

Encodings can be set via the VOC file, when opening the file, or for individual read/write operations.

Sequential Files

READSEQ and WRITESEQ use encodings in the same way as directory files

READBLK and WRITEBLK are byte level operations and are unaffected by ECS.

QMNet

QMNet can read/write ECS data so long as the remote file is configured for ECS (hashed files) or has an encoding (directory files).

If either end of the connection is non-ECS, only 8-bit characters can be used.

Data Replication

Data replication works exactly as in non-ECS systems but the replication target for an ECS mode file must support ECS characters.

Terminal I/O

QMConsole sessions on Windows can use the extended character set.

Windows limits display of ECS data to certain font settings.

Unicode keyboard input allows use with all standard keyboard layouts and languages.

Code page settings become irrelevant.

Terminal I/O

Network connections to QM can operate in 8-bit mode or in UTF-8.

UTF-8 can be selected using

```
PTERM ENCODING UTF8
```

or by entering QM with the –utf8 option.

Terminal I/O - We Got It Wrong!

QM was originally developed as a Windows console mode application.

We chose to place special keys such as the cursor keys in the upper half of the 8-bit character set.

THIS WAS A BAD IDEA !!!

KEYIN() cannot distinguish between the special keys and accented European characters.

Terminal I/O - How We Fixed It

KEYIN() and KEYCODE() must continue to work exactly as before.

New functions KEYINV() and KEYCODEV() return a key value instead of a character.

They separate the special keys from other characters.

The special keys have codepoint values in the Private Use Area.

These functions are supported on ECS and non-ECS systems.

Serial Ports

These are byte level interfaces.

Encoding/decoding is the responsibility of the application.

Encodings are available via ICONV() and OCONV().

Sockets

These are also byte level interfaces.

Encoding/decoding is the responsibility of the application.

Printers

A character encoding can be specified in the SETPTR or SET.QUEUE commands.

Transliteration

The TRANSLITERATE() function converts a string possibly containing ECS characters to one formed only from 8-bit characters.

The transformation is controlled by the active character map.

Each ECS character can be assigned a one or two byte 8-bit substitute.

Müller -> Mueller

Encodings

An encoding has a name and an optional set of mode flags.

E.g. UTF8.B

The encoding is UTF-8

The B mode flag inserts a leading byte order mark when writing data.

Users can add their own encodings as QMBasic subroutines.

The Five Problem Characters

With the M mode flag, the encoding will interchange characters 251 - 255 with the five codepoints reserved in the PUA for the accented characters.

For a file holding text that may include the accented characters, they get moved correctly.

For a file holding multivalued data and using characters 251 - 255 as the marks, this mode flag should not be used.

QMBasic Programming

QMBasic programs can include character constants that contain ECS characters.

All other program source elements must be formed from the 8-bit character set.

String variables can contain ECS data.

The ECHAR() function can be used to create ECS characters.

The SEQ() function may return values 0 to 65535.

Conversion Codes

BS	Transform ECS data to byte pairs
MXUC	Unicode variant of MX0C with four digit hexadecimal values.
X*name*	Apply named encoding.

Double Width Glyphs

Unicode defines some characters as requiring two screen positions.

The character map includes a “wide” attribute.

The IS.WIDE(*chr*) function tests this attribute.

The DISPLAY.WIDTH(*str*) function returns the number of screen positions required for the given string.

The SUBSTRDW() function extracts a substring based on display width.

Double Width Glyphs

Other display width functions...

- FOLDDW()
- FMTDW()
- INPUTDW

QMClient

The QMClient API adds a set of “wide character” functions as alternatives for all functions that return a character string or take a character string argument.

There is an inconsistency in the underlying wchar_t data type:

- Windows defines wchar_t as 16 bits
- Linux defines wchar_t as 32 bits

QMClient can also use a UTF-8 interface.

Installation

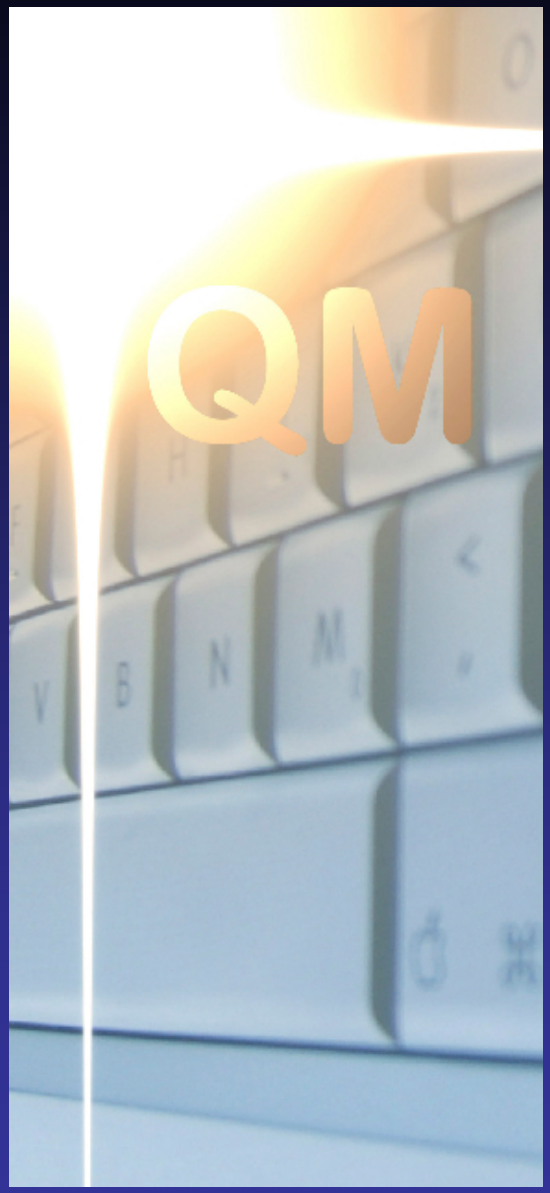
The QM installer will ask which version to install.

The base character map will be installed automatically.

Other maps can be downloaded from the OpenQM web site.

OpenQM

QUESTIONS?



OpenQM