



MQGem Software

MQSCX
Extended MQSC Utility
User Guide

Version 7.5.0

26th June 2013

MQGem Software Limited

www.mqgem.com

Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

MQGEM SOFTWARE LIMITED PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

While every effort has been made to ensure the accuracy of the information contained in this document no guarantees can be made. Similarly the applicability of information in this document may well depend on the customers operating environment. If you feel that there are inaccuracies in this document please raise your concerns by sending an email to support@mqgem.com.

MQGem Software Limited may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

- IBM
- WebSphere MQ
- MVS
- z/OS

The following terms are trademarks of the Microsoft Corporation in the United States and/or other countries:

- Windows

The following terms are trademarks of the Open Group:

- Unix

First Edition, June 2013

This edition applies to Version 7.5.0 of Extended MQSC Utility and to all subsequent releases and modifications until otherwise indicated in new editions.

(c) Copyright MQGem Software Limited 2013. All rights reserved.

Table of Contents

1 INTRODUCTION.....	1
1.1 FEEDBACK.....	1
2 GETTING STARTED.....	2
2.1 INSTALLATION.....	2
2.1.1 <i>Windows</i>	2
2.1.2 <i>Unix</i>	2
2.2 LICENSING.....	2
2.3 ISSUING YOUR FIRST COMMANDS.....	2
2.4 SCREENS.....	4
2.5 COMMAND RECALL.....	5
3 EXTENDED FILTERING.....	6
3.1 WILDCARDS.....	6
3.2 FIND() FEATURE.....	6
3.3 WHERE() CLAUSE.....	8
3.3.1 <i>Abbreviations</i>	10
3.3.2 <i>Attribute presence</i>	11
4 POWER FEATURES.....	12
4.1 COMMAND RETRIEVAL.....	12
4.2 COMMAND AUTO-COMPLETION.....	12
4.2.1 <i>Auto Complete in WHERE clause</i>	13
4.3 OBJECT NAME AUTO COMPLETION.....	13
4.4 DOUBLE-CLICK COMMAND COMPLETION	14
4.5 COMMAND LISTS.....	14
4.6 REPEATING COMMANDS.....	15
4.6.1 <i>Repeating multiple commands</i>	16
4.7 KEY ASSIGNMENT.....	16
4.7.1 <i>Keyboard mapping</i>	16
4.7.2 <i>Key commands</i>	17
4.7.3 <i>Keyboard mapping example</i>	18
4.8 SYNONYMS.....	19
4.8.1 <i>A single command</i>	19
4.8.2 <i>A list of commands</i>	19
4.8.3 <i>A list of synonyms</i>	19
4.8.4 <i>Recursion</i>	19
4.9 COPY AND PASTE.....	19
4.9.1 <i>Copy</i>	19
4.9.2 <i>Paste</i>	19
4.10 UNDO/REDO	20
4.11 DISPLAY TOTALS.....	20
4.12 SEARCH.....	21
5 CHANGING APPEARANCE.....	22
5.1 PROMPT.....	22
5.1.1 <i>Conditional Insertion</i>	23
5.2 COLOURS.....	24
5.2.1 <i>Setting the various screen elements</i>	24
5.2.2 <i>Switching colours off</i>	24
5.3 COLUMNS.....	25
5.4 SEPARATORS.....	26
5.5 WINDOW RE-SIZE SUPPORT.....	28
6 CONFIGURATION FILE.....	29
7 FILE MODE.....	30
7.1 PIPING A FILE IN FROM STDIN.....	30

7.2 EXPLICITLY PASSING IN AN INPUT FILE.....	30
7.3 EXPLICITLY ASKING FOR FILE MODE.....	30
7.4 FILE MODE SEPARATORS.....	30
7.5 FILE MODE FEATURES.....	31
7.6 CONFIGURATION FILE.....	31
8 MQSCX PARAMETERS.....	32
8.1 INITIAL COMMAND.....	32
8.2 CHANNEL DEFINITION FIELDS.....	33
9 LICENSING.....	34
9.1 LICENCE TYPES.....	35
9.2 LOCATION.....	35
9.3 MULTIPLE LICENCES.....	35
9.4 CHANGING YOUR LICENCE FILE.....	35
10 COMMANDS.....	36
10.1 COMMAND HELP.....	36
10.2 =CACHE.....	36
10.3 =CLS.....	37
10.4 =COLOUR.....	37
10.4.1 Colour limitations.....	38
10.5 =CONN.....	38
10.5.1 Client Channel Definition Parameters.....	40
10.6 =DISC.....	40
10.7 END.....	40
10.8 =IMPORT.....	40
10.8.1 Nesting imports.....	41
10.9 =KEY.....	43
10.10 =KEYNAME.....	44
10.11 =PURGE.....	45
10.12 QUIT.....	45
10.13 =REPEAT.....	46
10.14 =SET.....	47
10.15 =SHOW.....	48
10.16 =SYN.....	49
11 CONNECTING TO THE WEBSHERE MQ QUEUE MANAGER.....	50
11.1 CONNECTION STATUS.....	50
11.2 SWITCHING CONNECTIONS.....	50
12 PROGRAM SETTINGS.....	51
12.1 MULTIPLE USERS.....	51
13 PERFORMANCE.....	52
13.1 PERFORMANCE TEST.....	52
14 NATIONAL LANGUAGE SUPPORT.....	53
15 SECURITY.....	54
15.1 EXAMPLE SECURITY COMMANDS FOR DISTRIBUTED PLATFORMS.....	54
16 RETURN CODE.....	55
17 PROBLEM DETERMINATION	56
17.1 GENERAL FREQUENTLY ASKED QUESTIONS.....	56
17.1.1 It says 'Please set required location of configuration file.....' ?.....	56
17.2 MQ RELATED FREQUENTLY ASKED QUESTIONS.....	56
17.2.1 What does 'Can not find WebSphere MQ on this machine' mean ?.....	56
17.2.2 Why is my connect failing ?.....	57
17.2.3 Why does MQSCX make two connections to the Queue Manager ?.....	57
17.2.4 Why do I see 'Command Server or network slow in responding' ?.....	57
17.3 KEYBOARD RELATED FREQUENTLY ASKED QUESTIONS.....	58

17.3.1 Why is there an unnatural delay when I press the <ESCAPE> key ?.....	58
17.3.2 Why do F1, F2, F3 and F4 not work but the other function keys do ?.....	58
17.3.3 Why do I get told key or key sequence is unrecognised ?.....	58
17.4 DISPLAY RELATED FREQUENTLY ASKED QUESTIONS.....	58
17.4.1 Why is there a delay for a second or two when I resize the screen ?.....	58
17.4.2 Why are Red and Blue interchanged ?.....	58
17.5 SUPPORT.....	59
APPENDIX A.=WHERE OPERATORS.....	60
APPENDIX A.1.WILDCARDS.....	61
APPENDIX A.2.OPERATOR SYNONYMS.....	61
APPENDIX B.=WHERE FUNCTIONS.....	62
APPENDIX C.KEY NAMES.....	63
APPENDIX C.1.EMULATOR KEYS.....	63
APPENDIX D.KEY ACTIONS.....	64
APPENDIX E.VALID NAMES	65

1 Introduction

The world of WebSphere MQ contains many ways to configure and monitor your WebSphere MQ system so the first question you may be asking yourself is ‘why do we need another one?’. The answer to that question is fairly straightforward. There are a number of Graphical User Interfaces (GUIs), such as MQ Explorer, but these can often seem complicated, slow and in many ways unwieldy. So a large number of administrators still use RUNMQSC to make quick simple changes to their configuration. Unfortunately RUNMQSC is perhaps a little too simplistic and some of its inherent limitations can become a little frustrating. You see RUNMQSC is great at processing commands which are read from a file. However, when used interactively it still treats the input as though it was coming from a flat file. Consequently features such as command line recall¹, undo/redo, auto completion, find and copy+paste just aren’t supported.

There are other limitations as well. Since RUNMQSC is essentially just a simplistic portal on the MQSC command server it is restricted to the features provide by the MQSC language itself.

What MQSCX does is effectively bridge the gap between RUNMQSC and a GUI. It has the quick and simple aspects of RUNMQSC with some of the ease of use of a GUI.

MQSCX can be run on a variety of platforms and can administer all versions of WebSphere MQ which are in support and which provide PCF and remote MQSC interfaces. In particular this includes the z/OS platform including Queue Sharing Group (QSG) installations.

1.1 Feedback

I am always interested in hearing user views, whether good or bad, and would love to hear your opinions, comments and suggestions. So, if you would like to make a comment either about MQSCX itself or this manual then please do contact me at support@mqgem.com.

So, without further delay let’s see an example of how we can use it.....

¹ On Windows a primitive command line retrieval is provided by the C-runtime however RUNMQSC itself has no idea this is happening.

2 Getting Started

2.1 Installation

If MQSCX has already been installed you can skip this section. By default the program will try to write the configuration file to the same directory that the program itself is in so ideally the user should have write access to this directory. If you wish you can use an environment variable, MQSCXCFG, to specify an alternate directory.

The examples in this manual assume that the program executable is in the path. The examples and screen-shots are from a Windows system but very similar screens will be seen in Unix.

2.1.1 Windows

MQSCX is provided as a simple zip file. Once you have downloaded this file you should unzip it into a location which is either in your path or can be explicitly referenced on the command line.

2.1.2 Unix

MQSCX is provided as a simple tar or gzip file. Once you have downloaded this file you should untar the file using a command such as:

```
gzip -d mqscx.tar.gz          (If file is a gz file)
tar -xvf mqscx.tar
```

Move the file to a directory which is either in your path or can be explicitly referenced on the command line.

2.2 Licensing

The next consideration is to ensure that you have a suitable licence for the program. This will depend on the type of program you have downloaded and installed.

Program Type	Licence required ?
Beta Test	No, although the program will be time limited
Trial Program	No, although the program will be time limited
Full Program	Yes

If a licence file is required then please ensure you have a licence file installed in the appropriate location (usually the same directory as the MQSCX program).

Please see section 9 *Licensing* on page 34 for more information about licence files.

2.3 Issuing your first commands

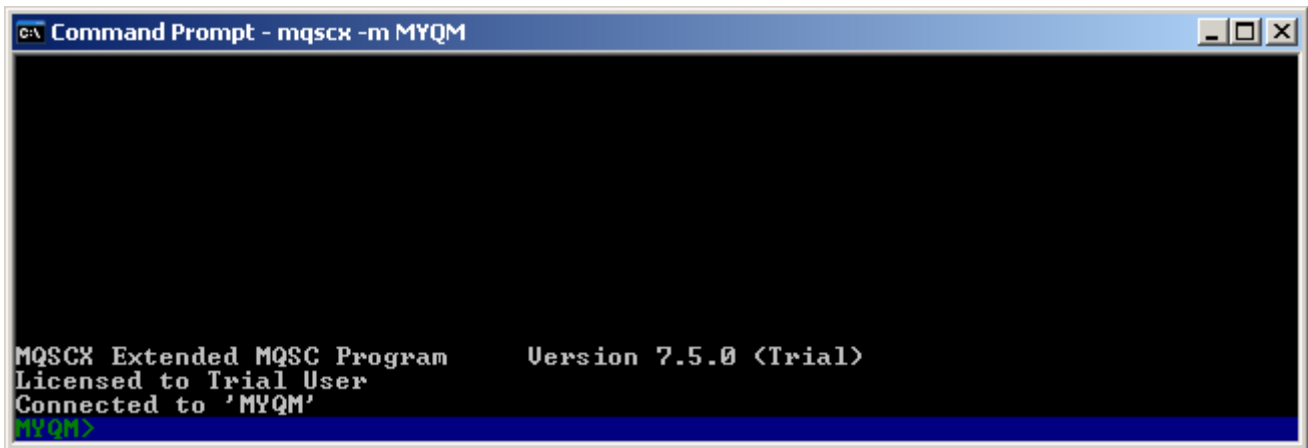
In these first examples we will assume that we have a local Queue Manager which is up and running which we are authorised to connect to. If life is more complicated than that for you then please refer to see section 11 *Connecting to the WebSphere MQ Queue Manager* on page 50 to see how to connect to your Queue Manager.

The first thing we need to do is start MQSCX and give it the name of the Queue Manager we wish to connect to.

```
C:\>mqscx -m MYQM
```

The -m parameter is not necessary if you wish to connect to the default Queue Manager.

This will run the program and display a panel something like this.....



```

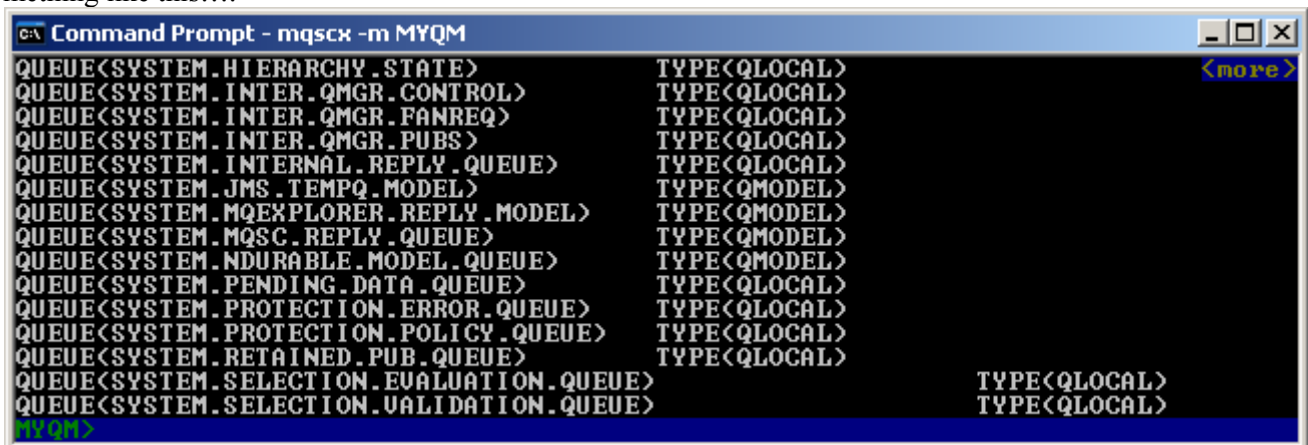
C:\ Command Prompt - mqscx -m MYQM

MQSCX Extended MQSC Program      Version 7.5.0 (Trial)
Licensed to Trial User
Connected to 'MYQM'
MYQM>

```

Now, this doesn't look hugely dissimilar to an equivalent RUNMQSC invocation but there are key differences. Firstly you will notice that the command prompt is at the bottom of the screen, it is a nice striking blue colour and the command prompt itself is preceded by the name of the Queue Manager. The prompt and colours can be changed, as we'll see later, but the location of the command prompt is always at the bottom of the screen. As commands are typed the result will be shown scrolling above the prompt but the prompt will stay in a fixed position.

So, let's see what happens when we enter a simple command. Type 'dis q(SYS*)' and press enter. We now see something like this....



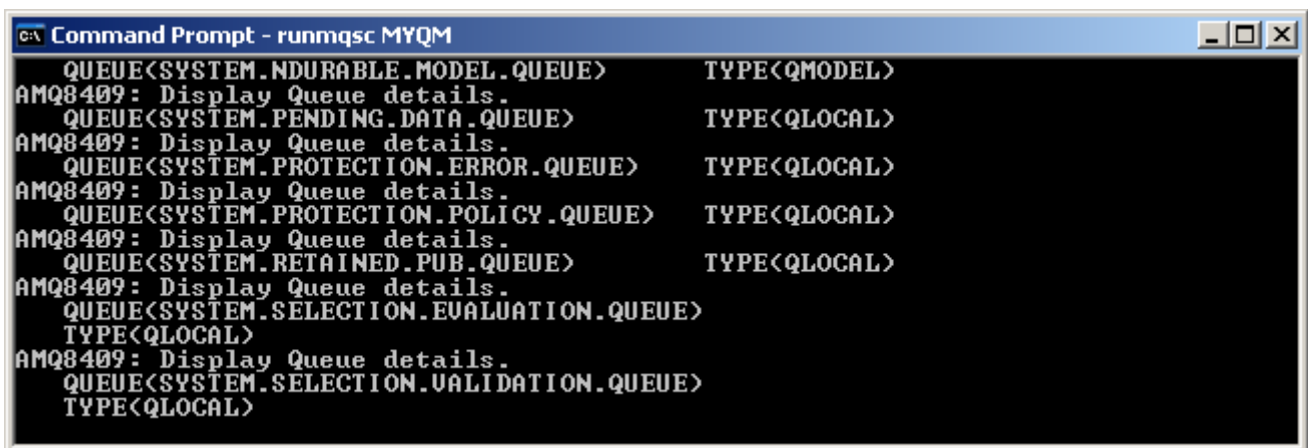
```

C:\ Command Prompt - mqscx -m MYQM

QUEUE<SYSTEM.HIERARCHY.STATE>      TYPE<QLOCAL>
QUEUE<SYSTEM.INTER.QMGR.CONTROL>   TYPE<QLOCAL>
QUEUE<SYSTEM.INTER.QMGR.FANREQ>    TYPE<QLOCAL>
QUEUE<SYSTEM.INTER.QMGR.PUBS>      TYPE<QLOCAL>
QUEUE<SYSTEM.INTERNAL.REPLY.QUEUE> TYPE<QLOCAL>
QUEUE<SYSTEM.JMS.TEMPQ.MODEL>      TYPE<QMODEL>
QUEUE<SYSTEM.MQEXPLORER.REPLY.MODEL> TYPE<QMODEL>
QUEUE<SYSTEM.MQSC.REPLY.QUEUE>     TYPE<QMODEL>
QUEUE<SYSTEM.NDURABLE.MODEL.QUEUE> TYPE<QMODEL>
QUEUE<SYSTEM.PENDING.DATA.QUEUE>   TYPE<QLOCAL>
QUEUE<SYSTEM.PROTECTION.ERROR.QUEUE> TYPE<QLOCAL>
QUEUE<SYSTEM.PROTECTION.POLICY.QUEUE> TYPE<QLOCAL>
QUEUE<SYSTEM.RETAINED.PUB.QUEUE>   TYPE<QLOCAL>
QUEUE<SYSTEM.SELECTION.EVALUATION.QUEUE> TYPE<QLOCAL>
QUEUE<SYSTEM.SELECTION.VALIDATION.QUEUE> TYPE<QLOCAL>
MYQM>

```

Again we see similarities with RUNMQSC output but the display is far from the same. For comparison let's remind ourselves what RUNMQSC would have looked like...



```

C:\ Command Prompt - runmqsc MYQM

QUEUE<SYSTEM.NDURABLE.MODEL.QUEUE>   TYPE<QMODEL>
AMQ8409: Display Queue details.
QUEUE<SYSTEM.PENDING.DATA.QUEUE>     TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<SYSTEM.PROTECTION.ERROR.QUEUE> TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<SYSTEM.PROTECTION.POLICY.QUEUE> TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<SYSTEM.RETAINED.PUB.QUEUE>     TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<SYSTEM.SELECTION.EVALUATION.QUEUE>
TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<SYSTEM.SELECTION.VALIDATION.QUEUE>
TYPE<QLOCAL>

```


You can see that the two screens look fairly different although they are both showing the same information.

So, rather like a ‘spot the difference’ puzzle, let us explain the differences.....

- **Response format**

As I’m sure you know, RUNMQSC formats the output of its responses into two columns provided the data fits. Sometimes two columns is appropriate but often it isn’t. By default MQSCX formats the output from the command server into an appropriate number of columns for the width of the output screen. In this case MQSCX is trying to fit the data into four columns however if the data doesn’t fit into a single column then it spans into the next column or columns as necessary.

If you wish you can change this behaviour or indeed ask for any other number of columns in the output. Please see section 5.3 *Columns* on page 25 for information of how to do this.

- **Display Details message**

Before each object RUNMQSC outputs a message to explain what it is about to output. These messages can often be distracting; they take up valuable screen real estate and are usually superfluous anyway. So, by default, MQSCX doesn’t always display them. You can, if you wish, reinstate them by changing a program setting – please see section 5.4 *Separators* on page 26 for more information.

- **The <more> indicator**

The <more> indicator at the top right of the screen indicates that there is more data above the displayed output. Effectively it tells you that the output can be scrolled up and down. In RUNMQSC looking at previous responses can be awkward, in MQSCX it’s as simple pressing ‘Page Up’ and ‘Page Down’, go on, try it.

- **Command prompt**

The command prompt, as we’ve mentioned before, is quite different from the simple prompt you have with RUNMQSC. Our next few exercises will see what we can do however the key difference is that you can edit this field using the left, right, delete, backspace and insert keys². Additionally, by default, pressing the ESCAPE key will empty the command line and move the cursor to the start of the command line. Unix users may wish to read the discussion about the ESCAPE key in *Appendix C.:Key Names* on page 63 if they notice a delay when using this key.

Many of the differences between RUNMQSC and MQSCX output is to make more efficient use of screen real estate. In this simple example we see that in the same size screen RUNMQSC displays 7 queue definitions where as MQSCX displays 15!

2.4 Screens

The MQSC output screen is actually just one of four display screens, although it is the most important. However, let’s introduce you to the other screens. These can be displayed using the **=show** command or, by default, pressing a function key.

Try typing in one or two of the following commands:

Command	Default Key assignment	Description
=show scrn(help)	F1 ³	Show the help screen
=show scrn(mqsc)	F2	Show the MQSC display
=show scrn(keys)	F3	Show the key assignment screen
=show scrn(sets)	F4	Show the settings screen.

We shall talk about the various commands you can enter into MQSCX later so do not be too concerned about the exact format of the commands or what you can do. All we really want to know at the moment is that there are these

² This was mainly a problem for Unix users before depending on the emulator you were using.

³ Some emulators assigned function key F1 for their own use so it may be necessary to disable the interception of F1 or reassign the **=show scrn(help)** to an alternative key.

four displays and they all give us some useful information. The majority of the time you will be using the MQSC screen but it is handy to know the others are there should you need them.

Now, if you've not already done it come back to the MQSC display and we'll continue looking at what we can do with MQSCX.

2.5 Command recall

One of the most frustrating usability features of RUNMQSC, certainly on Unix, is its lack of any command retrieval. We've probably all been there, you type in a long DEFINE CHANNEL command specifying half a dozen attributes, hit enter, only to be told that you had a syntax error and it dawns on you that you missed a quote character. You then have to type it all in again. Wouldn't life be so much simpler if you could retrieve your previous command?

Well, perhaps not surprisingly, with MQSCX you can. By default pressing 'up' and 'down' on the keyboard will cycle backwards and forwards respectively through the previous commands. So, have a play with it, type in a few simple MQSC commands and then use 'up' and 'down' to retrieve the commands, edit them and re-issue. There are a few other things you can do with retrieval, please see section *4.1 Command retrieval* on page 12 for more details.

So, that's it for the basic features, can we do anything more? Of course we can! There are plenty more exciting features for us to take a look at such as:

- Command auto-completion
- Object name auto-completion
- Keyboard assignments
- Extended filtering
- Extended editing capabilities
- and more.....

Let's take a look at some of the more powerful features of MQSCX.

3 Extended Filtering

As we all know the MQSC language has a number of ways that you can filter the displayed output. However, there are some significant restrictions. MQSCX alleviates many of these; we'll consider each in turn.

3.1 Wildcards

The native MQSC language supports wildcards in its DISPLAY commands. This is invaluable when you want a single command to return multiple entries. For example:

DIS Q(*)	Show all queues
DIS CHS(*)	Show all current channels

Native MQSC allows us to go further and specify a root value before the wildcard

DIS Q(SYS*)	Show all queues starting 'SYS'
DIS CHS(XP*)	Show all current channels starting 'XP'

Unfortunately that's where the current MQSC language stops. It will only allow a single wildcard character and it must be at the end of the name.

MQSCX extends the MQSC language and allows the user to specify '*' wildcards anywhere in the name. It also allows you to specify a '?' signifying a single character. So, with MQSCX the following commands are now possible.

DIS Q(*MVS*)	Show all queues containing the string 'MVS'
DIS Q(SYSTEM*PUB*)	Show system queues which contain the string 'PUB'
DIS Q(??)	Show any queues with only a two character name
DIS CHL(*QMX*)	Show any channels with 'QMX' in the name

Note that the way in which MQSCX achieves this feature is to modify the actual command issued to the command server. If a field contains a wildcard then the value is terminated at that first wildcard. If any responses are received the full wildcard is then applied to the returned values and only those which match the full wildcard specification are shown to the user. You can see this is operation by issuing an invalid command. Try, for example:

DIS Q(*PUB*) X

You will see that the action command issued to the command server was actually 'DIS Q(*) X'.

3.2 FIND() feature

Wildcards are a very simple way of filtering the data from the host by the object name. But suppose you want to filter by a different attribute? Well, then traditionally you would use the WHERE() clause. However, as we'll see later the WHERE() clause is quite complicated. People who have been using MQSC for years often still have to refer to the books to get their WHERE() clause correct. Primarily to get the syntax right but also to get the name of the returned attribute correct. Let's try a simple example. Suppose you know you have an application, let's say, amqsput.exe and you just wish to display it's connections. What is the command using the where clause?

Give up ?

One might imagine that this would work

dis conn(*) where(appltag lk '*amqsput*')
--

or possibly this...

```
dis conn(*) where(appltag ct 'amqsput')
```

Unfortunately what you have to enter is something like this.

```
dis conn(*) where(appltag lk 'c:\nttools\amqsput.exe*')
```

But clearly this is far more complicated that we would like. Firstly it requires you to know a weird operator LK ('like'). Secondly it is non-intuitive, the 'lk' operator matches wildcards but the wildcard must only be at the end of the value. Thirdly you need to know the exact attribute name. And lastly you even have to know the exact path to the program.

All we really want to do is filter out any responses which contain the string 'amqsput' and this is exactly what the =FIND() feature does.

Try typing in the command:

```
dis conn(*) all =find(amqsput)
```

How much simpler was that? What's more you can find any string or part of string anywhere in the response.

But you probably have one or two questions about the command right? So, let me see whether I can guess what they are:

- **Why did you specify 'all'?**

Well, the =find() will filter the responses from the command server. However, it can only work with the data returned so you need to ensure that the response contains the field you are looking for. The simplest way of doing this is to specify 'all' which says 'return all attributes. Of course if you are cleverer than me and knew that it was 'appltag' then you could have specified. 'dis conn(*) appltag =find(q1)' Of course, usually when you're issuing a command of this nature you are doing so to find out information about the object, for example its process id, so returning all the attributes may well be what you want anyway.

- **Why is it =find(...) and not just find(...)?**

This comes down to a question of future proofing. Currently the MQSC language itself has no find() attribute but it may in the future. By prefixing with a '=' I can be fairly certain that it is never going to clash with any extensions to MQSC by the WebSphere MQ product. You will see later on that I use the same principal for all the MQSCX commands.

- **Is it a case sensitive comparison?**

A good question. Pat yourself on the back if you asked this one. The answer depends on whether the value is specified in quotes or not. Without quotes, as in the example above, it is case insensitive. However, had I specified 'dis conn(*) all =find('amqsput') then it would have been a case sensitive search.

As we said, the find() feature is for doing very simple filtering based on a simple text search. If you need the filter to be a little cleverer then we need to use the =WHERE() clause below.

3.3 WHERE() clause

I have already mentioned the WHERE() clause and was not overly complimentary about it. It is complicated, has a difficult to follow syntax and has some considerable restrictions. So, why am I mentioning it again? Well, there are times unfortunately where a little complexity is unavoidable. If you want to ask more complicated questions then something a little more powerful than a simple search string is needed.

MQSCX has taken the WHERE() filter concept and removed many of those restrictions and enhanced it so that one can now issue many more powerful queries. If you followed along the =find() discussion then it won't surprise you to learn that MQSCX has implemented the WHERE() feature as =WHERE(). Again, the '=' sign ensures that it is clear who is doing the filtering. Is it the command server or is it MQSCX? Having this separation ensures that any future developments in either filter will be kept separate and they won't clash. That being said the =WHERE() filter has modelled itself very closely on the standard WHERE() clause. Clearly I could have invented a brand new syntax. However that seemed unnecessary and potentially confusing. Far better, at least for the moment, to build on the current syntax and just try to remove one or two of those little annoyances.

So, perhaps listing those restrictions might be a good place to start. If you are new to the WHERE() clause it might be worth you revisiting the MQ manuals to read a description of the syntax and come back when you have the basics. You will have seen that you can do lots of good stuff in a WHERE() clause but what might not be so obvious is what you can't do.....which is:

- You can't use the primary object name in a WHERE() clause
- You can't use any of the sub-filter attributes in a WHERE() clause
- A WHERE() clause may only contain one operation – you can't link them together with AND and OR
- You can only have a single attribute in a WHERE() clause. For example, you can't compare one attribute with another.
- Although it supports wildcard comparisons the wildcards may only be at the end of strings.
- You can only have one WHERE() clause on a single DISPLAY command
- You can't use indicators⁴ in a WHERE() clause.

Phew! Quite a list of restrictions and unfortunately most of these would be very nice to have.

But, as you may suspect, help is at hand because using the =WHERE() clause you can do all this, and more. Let's try some real world examples. Provided you know the WHERE() syntax you should have no trouble working out what the command is doing. Just to keep it interesting we'll only stick to commands which would have been illegal to issue using the standard WHERE() clause, see if you can spot why the command would have been illegal.

```
DISPLAY Q(*) =WHERE (DESCR LK '*PUB*')
```

So, this command will show you all the queues whose descriptions contain the text 'pub'. Note that we can use multiple wildcards in the operand.

```
DISPLAY Q(*) =WHERE (CURDEPTH GT 0 AND QUEUE LK '*CHANNEL*')
```

Show me all the queues which are non-empty and have 'channel' somewhere in their name. This shows that you can link multiple expressions using 'and' and 'or' operators.

⁴An indicator is an attribute, usually returned on a status display, which contains two values in list format. Usually a short term and a long term average. Examples include NETTIME on channel status or QTIME on a queue status response.

```
DISPLAY Q(*) =WHERE (CURDEPTH >= 0.80*MAXDEPTH)
```

Show me all the queues which are greater or equal to 80% of their maximum depth.
This command demonstrates a couple of points worth mentioning

- **Mathematical expressions**

The =WHERE() clause is essentially one big Boolean expression. If the expression evaluates to TRUE (non-zero) for any object then it is displayed. If it evaluates to FALSE (zero) then it is not displayed.

Within that remit you are free to enter virtually any mathematical expression you like, subject to the supported operators. If the expression evaluates to a non-zero value then the object is displayed. So, for example a filter of =WHERE(1) is perfectly valid although somewhat superfluous.

For a complete list of the =WHERE operators please see *Appendix A. =WHERE Operators* on page 60.

- **Operator synonyms**

A number of operators have synonyms. In this case the operator 'ge' could equally have been used. It is purely a matter of taste whether you have more of an SQL versus a mathematical background.

So, staying on a mathematical theme can we guess what this might do?

```
DISPLAY Q(*) =WHERE (QUEUE < 10)
```

To understand this filter we really just need to know how MQSCX handles an expression containing both strings and numbers. The answer is coercion⁵. MQSCX will automatically convert any string value to a numerical equivalent if it has an operation concerning both a string and number. The number it converts it to is its length. So, this command is saying, display any queues which have a name of less than 10 characters.

```
DISPLAY Q(*) =WHERE (QUEUE != UPPER(QUEUE))
```

Now, the first thing I should say is that this is doing a case sensitive comparison between two strings and by default string comparisons in the =WHERE() clause are case insensitive. So, before you issue this command you should make =WHERE() clauses case sensitive. You can do this with this command:

```
=set cswhere(on)
```

Don't worry too much at this stage about what this is doing just issue the =set command and then issue the DISPLAY command above. Of course depending on your system configuration you may or may not see results. What the command is doing is saying 'display only those queues whose name is not the same as the upper case version of their name'. In other words, display any queues which have lower case letters in the name. Commands like these can be useful to ensure compliance with installation standards although I will agree you won't be doing it often.

One of the most notable things about this command is the use of a function in the =WHERE clause. There currently aren't many functions available but for a complete list please see *Appendix B. =WHERE Functions* on page 62.

Anyway, before we continue let's go back to having case insensitive =WHERE() clauses

```
=set cswhere(off)
```

⁵Coercion, in the computing sense, is merely the process of making one data type compatible with another data type.

The next expression format we want to discuss is enumerated types. These are object attributes which have a fixed set of values, for example, suppose we wish to see the queues which are currently MQGET disabled. We would enter the command:

```
DISPLAY Q(*) =WHERE (GET = DISABLED)
```

Note that you must enter the fields this way round. It would be invalid to enter the command:

```
DISPLAY Q(*) =WHERE (DISABLED = GET)
```

This would generate a variable error since the program would look for a variable called 'disabled'.

Now let's try an example of a =WHERE() clause containing an indicator.

```
DISPLAY QSTATUS(*) =WHERE (QTIME.SHORT > 5000000)
```

This example, of course, requires that queue monitoring is on. Essentially an indicator consists of two fields, a long and a short average of the field value of time. These two fields can be accessed individually by using the suffix 'short' or 'long' to the attribute name. If no suffix is specified then the short indicator is used. Note that the short and long suffixes are something I have created to allow you to address the two values individually. WebSphere MQ does not recognise these, so you can not, for example, issue the following command:

```
DISPLAY QSTATUS(*) QTIME.SHORT
```

3.3.1 Abbreviations

Until now we've been good citizens and have always specified the full name of the attribute. However, MQSCX allows you to specify only as many characters of the name to ensure it is unique.

As an example let's take the simple command to show only those queues which have messages on them. The traditional command would be:

```
dis q(*) =where(curdepth > 0)
```

but we can use this command and save on typing

```
dis q(*) =where(cur > 0)
```

In actual fact the mathematical ones amongst you will notice that actually since the current depth of a queue can't be negative then this boolean expression is just the same as saying 'cur'. In other words, the expression is TRUE if the curdepth is non-zero. Consequently we can save even more typing and use this command:

```
dis q(*) =where(cur)
```

3.3.2 Attribute presence

Of course it is possible to use an attribute in an expression which doesn't get returned for each object. Consider the expression:

```
dis q(*) where(rname ne 'xxxxxxx')
```

Here we have a standard native MQ where clause. One might expect that this expression would show all queues since it is unlikely that you have any queues defined with an RNAME value of 'xxxxxxx'. However, the semantics of the WHERE clause is such that the expression always evaluates to FALSE if a variable is not present in the response. In other words, other queue types, such as local queues do not have a substitute value. So, in actual fact, this expression will just show us remote queue definitions.

This mechanism is quite useful since, given the WHERE expression, it is likely that the user is only interested in remote queues. If you issue the expression against your own Queue Manager you will likely see all of your remote queue definitions, unless you really do have one which points to a queue called 'xxxxxxx'!

However, as we have heard in the sections above, the =WHERE clause has been extended to include logical OR expressions so this rule has been this relaxed.

Consider the expression:

```
dis q(*) =where(rname | target)
```

If the =where() clause stuck to the original rule then this expression would never return any objects since 'rname' is an attribute of a remote queue and 'target' is an attribute of an alias queue. The two attributes are mutually exclusive since they belong to different object types. However, the rule above has been relaxed and this expression will return a list containing all remote queues which have a value for 'rname' and all alias queues with a value for 'target'.

4 Power Features

So, we've seen the basics of what MQSCX can do, now let's take a look at some other powerful features.

4.1 Command retrieval

I mentioned before how powerful command retrieval is but we didn't cover it in much detail. Essentially 'up' and 'down' will cycle backwards and forwards respectively through the previous commands. Not only that but the previous commands are maintained across program invocations so you can recall the commands you were using the last time you ran MQSCX. By default MQSCX will keep the last 50 commands you entered. If you would like to change this then you can use the `=set` command, please see section 10.14 `=set` on page 47 for a description of how to do this.

In addition there are two handy mechanisms which can help you find the command you want quicker.

- **Command prefix**

If the cursor is not in the first position on the command line then the characters before the cursor will be used as a prefix for the search. In other words, only commands starting with the given characters will be retrieved.

For example, to retrieve the previous 'ALTER' command one merely needs to type 'AL' and press 'up'.

Of course this is presupposing there is an ALTER command to find!

- **Command content**

Suppose you want the previous command which mentioned a certain string, let's say 'Q1'.

Well, to do that you merely prefix the string with a '#' character. So, for example type '#q1' and then press up.

The command retrieval feature will remove command duplicates and the command comparison is case insensitive. This means that "DIS Q(*)" and "dis q(*)" are considered the same command, this is fine since these are essentially the same command. However, "dis q('Q1')" and "dis q('q1')" are not the same command and yet will be treated as so. MQ recommends that users do not have MQ objects with same name but in different cases so it should not cause a problem but it is MQSCX behaviour to be aware of.

4.2 Command auto-completion

One of the problems with interactive MQSC is the need to remember a whole host of commands, fields and attributes. Even after years of working with the language it is still very easy to forget the exact spelling of the attributes. It would be handy if one could have the values automatically entered at the press of a button. Well, with MQSCX you can....and that button is the tab key.

Let's try it out.....let's start off with an empty command line. Now press <tab>. You should see that a command has been entered for you, "ALTER". Press a few more times. You'll see that the command changes. It is cycling through the commands you can issue. Now, try pressing <shift-tab>. Now you will see we're going through the same list backwards.

So, this is handy but it might take a long time to get to the command you want. Let's suppose it's "DISPLAY". Well, start with a blank command type 'DI' and then press <tab>. You'll see that it immediately fills in 'DISPLAY'. Pressing <tab> again will produce an error message telling you that there isn't another command starting with 'DI'. But that's OK because it was 'DISPLAY' we wanted anyway.

So, now press <space> and once again hit <tab>. It probably won't surprise you by now to see that it fills in a possible next part of the command. In this case 'AUTHINFO()⁶'. Again pressing <tab> a few times will cycle through the possible things you can 'DISPLAY'. Keep pressing <tab> until we come to 'QUEUE()'.

⁶Or ARCHIVE if you are administering a z/OS Queue Manager.

Phew! There are a lot of things you can display aren't there!. Of course you might be thinking by now that it would have been quicker just to type 'QUEUE()' itself. And, of course, you'd be right. Those of you that have been paying attention so far I'm sure will have guessed that the solution, as in the case of command retrieval, is to type the first character or two of what you are looking for. So, let's try it again. Clear the command line. A quick way to do this is just press the <esc> key. The <esc> key will clear the command line and move the cursor to the first position of the command line.

So, now let's now try typing 'di' <tab> <space>'qu'<tab>

There, that was much quicker, wasn't it ?

So, now let's complete the command. Let's say we're trying to issue the command

```
DISPLAY QUEUE(*) DESCR
```

Type '*' in the brackets. Press <end> to go to the end of the command. Type 'DES' and press <tab>.

Now, for such a trivial command this doesn't seem to buy you a great deal. However, how many of us can remember the exact spelling of all the MQ attributes? Sometimes you can only remember it contains certain characters. For example, take the heartbeat attributes. You might know there are two attributes on channels concerned with heartbeat but can't remember just what they are. So, use the trick of using the '#' character and enter the command...

```
DISPLAY CHANNEL(*) #HB_
```

Now press the <tab> key. You'll see that MQSCX cycles through the two heartbeat attributes, BATCHHB and HBINT.

So, as you can see auto completion saves you time in typing and is also a good memory aid for those of us who can't remember the exact spelling of every attribute. However, be aware that auto completion does not guarantee that the command you ended up with is valid. Nor, of course, does it guarantee that the command is the one you should be issuing so care is still needed. Generally speaking only commands and attributes appropriate for the platform you are configuring will be shown but due to the huge number of combinations this can not be guaranteed.⁷

4.2.1 Auto Complete in WHERE clause

Pressing <tab> in a WHERE() or =WHERE() clause will also suggest possible values at the point of the cursor. Note that it is still possible to construct an invalid expression this way but it can be a useful memory aid.

4.3 Object name auto completion

So far our examples have just used '*' as the object name, we haven't been targeting a particular object. But suppose that's what we wish to do, can MQSCX help here? Well, naturally it can.

Let's try it. Type in a command such as 'DISPLAY QUEUE()' ideally using the mechanisms you have used earlier. Now, type the first character of the queue name you are after in the brackets and press <tab>. You should see that the first defined queue, alphabetically, starting with the given character is now suggested in the brackets. Pressing <tab> multiple times will cycle through the defined queues.

I suspect you can guess that if you type '#' as the first character then MQSCX will show you any queues containing the characters following the '#'. This can be very useful if you can't remember the exact spelling of an object name but you know some of the characters in it.

⁷ If there are particular platforms and commands which you feel should be handled differently then please contact support and I will consider making changes to the program.

So, what is happening here? How does this work ? Well, MQSCX maintains a cache of the names of most of the MQ object types. The first time you press the <tab> character in an object field a request is sent to the server for all the object names of that type. As you tab through the values MQSCX is merely going through this retained list. This immediately begs the question of when the cache is refreshed of course. Well, the cache is refreshed for two reasons:

- **It ages out**

The object cache is maintained only for a certain amount of time before it is considered too out of date. In an ideal world the cache would be refreshed each time the user pressed <tab> but that would cause an excessive amount of background work. The amount of work required is different depending on whether the Queue Manager is local or remote. A Queue Manager is considered remote if some form of channel, be it a client or sender/receiver pair, is required to reach it. As a consequence the time outs themselves vary depending on whether it is a local or remote; being 60 and 1800 seconds respectively. You can change these values if required, please see section 10.2 *=cache* on page 36 for information on how to do this.

- **It is requested by the user**

At any time the user can request that the cache is discarded by issuing an '=cache purge' command. This command will purge all object definitions meaning that the next time the user presses <tab> in an object field a new set of definitions will be fetched from the server.

As I'm sure you know MQSC will upper case any object names which don't appear in quote (') signs. As a consequence MQSCX will automatically add quote signs around any name containing lower case letters. Similarly auto completion will remove quote signs if they are not needed.

4.4 Double-click command completion

It is quite common to issue one command to display a list of objects and then issue a more detailed command to see all the information about that object. For example issuing 'dis q(*) where(curdepth gt 0)'. Then you might want to issue a 'dis qstatus(X) all' to display details about one of those queues.

MQSCX can reduce the typing here by filling in the queue name from the previous response when you double click on the name. To see this in action issue the command 'dis q(*)'.

Now use auto-completion by typing 'di' <tab> <space> 'qs' <tab>. You should have a command line which looks like:

```
DISPLAY QSTATUS ( )
```

Now, choose a queue returned by your 'dis q(*)' command and using the mouse double click on the name. You should see the name automatically copied into the 'DISPLAY QSTATUS()' command.

Essentially any time you double click on on the screen the word you select will be copied to the command line. If there are empty brackets, '()', on the command line then the word will be copied between the brackets. If there aren't any empty brackets then it will be copied to the end of the command line.

4.5 Command Lists

Any time you can enter a single command you can actually enter a list of commands. A list of commands is created by joining a number of individual commands with semi-colons. So, for example we could enter the command:

```
dis q(Q1) ; dis q(Q2) ; dis q(Q3)
```

Pressing enter will then run each of the commands one after another and display the output.

You may feel there isn't much advantage in this since it involves just as much typing. However, there are a few advantages:

- If there are a few commands you regularly use together then using a list allows you to recall the list all together rather than recalling each command separately.
- When used together with the `=repeat` command, described later, you can easily repeat multiple commands. This is the simplest way to monitor multiple objects.
- When used with the initial command parameter it can be useful to be able to pass in multiple commands to be run when the program first initialises.

For a description of the initial command parameter please refer to section *8.1 Initial Command* on page 32.

4.6 Repeating Commands

There are times when it would be nice to have the ability to repeatedly issue a command, usually a `DISPLAY` command, and be able to just watch the output of the command. In this way a very simple monitor can be established. For example, suppose you wanted to look at how the depth of a queue is changing, or look at how many channels are running or watch the time messages spend on a queue. There is no doubt that for complicated monitoring then a sophisticated monitoring tool is the way to go but for a simple demo or as a development aid a repeating MQSC command can be quite useful.

MQSCX provides an `'=repeat'` command. For a complete definition of the command please see section *10.13 =repeat* on page 46 but for now we'll just give a simple example of its use. Let's suppose we just want to monitor the definition of a queue, say `'Q1'`.

We issue the command:

```
=repeat cmd(DISPLAY QUEUE(Q1)) delay(1)
```

This simple command requests that MQSCX repeatedly issues the `'DISPLAY QUEUE'` command waits for one second before re-issuing the command. The result of the command is something like the following:

```

c:\ Command Prompt - mqscx -m MYQM
[16:32:42] DISPLAY QUEUE(Q1)
QUEUE(Q1) TYPE(QLOCAL) ACCTQ(QMGR) ALTDAT(2013-04-18)
ALTIME(16.31.51) BOQNAME( ) BOTHRESH(0) CLUSNL( )
CLUSTER( ) CLCHNAME( ) CLWLPRTY(0) CLWLRLANK(0)
CLWLUSEQ(QMGR) CRDATE(2013-02-24) CRTIME(15.08.09) CURDEPTH(0)
CUSTOM( ) DEFBIND(OPEN) DEFPRTY(0) DEFPSIST(NO)
DEFPRESP(SYNC) DEFREADA(NO) DEFSOPT(SHARED) DEFTYPE(PREDEFINED)
DESCR( ) DISTL(NO) GET(ENABLED) HARDENBO
INITQ( ) IPPROCS(0) MAXDEPTH(5000) MAXMSGL(4194304)
MONQ(QMGR) MSGDLUSQ(PRIORITY) NOTRIGGER NPMCLASS(NORMAL)
OPPROCS(0) PROCESS( ) PUT(ENABLED) PROPCTL(COMPAT)
QDEPTHHI(80) QDEPTHLO(20) QDPHIEU(DISABLED) QDPLOEU(DISABLED)
QDPMAXEU(ENABLED) QSUCIEU(NONE) QSUCINT(999999999) RETINTUL(999999999)
SCOPE(QMGR) SHARE STATQ(QMGR) TRIGDATA( )
TRIGDPTH(1) TRIGMPRI(0) TRIGTYPE(FIRST) USAGE(NORMAL)
MYQM Repeating command 'DISPLAY QUEUE(Q1)' <press ESC to end>

```

If you do this yourself you will notice that the timestamp at the top left of the screen is 'ticking' as each command is issued. While in this mode keyboard entry is restricted. The user can not scroll around the display or enter new commands. The only key which is recognized in this mode is `<esc>` which will cancel the REPEAT command. As a consequence you really need to ensure that the screen is large enough to contain all the output data you wish to monitor.

4.6.1 Repeating multiple commands

One of the first questions many people ask is “Can I repeat multiple commands?”. Well, yes you can. There are two ways to do it.

1. **Use a command list**

Any time you enter a command you can actually enter multiple commands separated by a semi-colon (;) character. So, in the example above if we wished to display two queues we could issue:

```
=repeat cmd(DISPLAY QUEUE(Q1); DISPLAY QUEUE(Q2)) delay(1)
```

2. **Using an =import command**

An import command, which is described later, allows you run a sequence of commands read from a file. So, for example, you could issue the following command:

```
=repeat cmd(=import file(mycommands.mqs)) delay(1)
```

However, bear in mind that the output of all your commands must fit into a single screen if you wish to see the results since you can not scroll the window while in repeat mode.

4.7 Key assignment

In a number of places in this manual I have said things along the lines of ‘press this key and this will happen’. That should indeed be true when you first start using MQSCX. However, we all have our favourite ways of doing things and, as a consequence, to a large extent, MQSCX allows you to reconfigure what happens when a key is pressed.

There are essentially two main reasons why you might want to do this, and we’ll consider those next.

4.7.1 Keyboard mapping

It is amazing how many ‘standard’ key configurations exist and, as users, we all have our preferences. Even keys that have their own functions, for example ‘page up’ and ‘page down’, may have preferable alternatives such as F7 and F8. But what of ‘repeat find’? Should it be <Ctrl-f> or <Ctrl-n> or a function key ? The fact of the matter is that whatever will feel natural to some will feel alien to others. The answer therefore seems is to allow users to set whatever mapping they choose.

By way of explanation let’s take one of the first keyboard mapping quandaries I had to deal with. What keys should represent the ‘previous command’ and ‘next command’ actions? The initial version of MQSCX had these actions assigned to <Ctrl-up> and <Ctrl-down> respectively. The keys ‘up’ and ‘down’ just moved the cursor up and down. However, users of MQSCX agreed that it was more intuitive if just the ‘up’ and ‘down’ keys cycled through the commands so the default was changed. So the <Ctrl-up> and <Ctrl-down> keys are used to navigate the cursor. Some users may find it counter intuitive that ‘left’ and ‘right’ cursor keys work as expected but you need to press the <Ctrl> button if you want to move the cursor up and down. As one user put it “I want the program to be psychic!”. Well, until we invent a way of allowing programs to determine what the user is thinking we’ll just have to use mapping.

So, suppose we decided we didn’t like these cursor defaults and wished to change them back? Let’s use this as an example to show how keys can be reassigned. What would be the commands to issue cause the alternative assignments?

Enter the following commands and observe the change in behaviour.

```
=key(up)          action(cursor up)
=key(down)         action(cursor down)
=key(ctrl-up)      action(previous command)
=key(ctrl-down)    action(next command)
```

After entering these commands you'll notice that now the cursor keys just move the cursor in the appropriate direction and in order to cycle the previous commands you need to press the 'ctrl' key. It is, of course, entirely your preference over which mapping you choose. Just for completeness though, here are the commands you need to set it back to the default values:

```
=key(up)          action(previous command)
=key(down)        action(next command)
=key(ctrl-up)     action(cursor up)
=key(ctrl-down)   action(cursor down)
```

Actually since we are just returning to the default settings it isn't actually necessary to state the action. If an empty action is given the mapping returns to the original default.

```
=key(up)          action()
=key(down)        action()
=key(ctrl-up)     action()
=key(ctrl-down)   action()
```

For a complete description of the =key() command please see section 10.9 =key on page 43.

For those of you who are not overly keen on typing these command you can use the =import command described in section 10.8 on page 40. The =import command allows you run a set of commands contained in a file. So, all need do to try these examples is copy and paste the commands into a temporary file, then use =import to see their effects.

4.7.2 Key commands

Have you ever found yourself typing the same commands into RUNMQSC over and over again. For example 'DIS Q(*)' or perhaps 'DIS CHS(*) ALL'. Wouldn't it be nice if this could be achieved in a single key press? Well, that's what key commands are all about. Just as you can assign a key to a particular action you can also assign them to a command. For example:

```
=key(ctrl-q) cmd(DISPLAY Q(*) CURDEPTH)
```

I suspect that an intelligent reader such as yourself has already figured out what this means. However, just to be clear, this command is stating that each time <Ctrl-q> is pressed that the command 'dis q(*) curdepth' should be issued. In a similar fashion many more commands can be assigned to various keys; whichever commands you find yourself issuing again and again. Needless to say, this can be a real time saver as well as avoiding typing mistakes.

Virtually any command, be it MQSC or MQSCX commands, can be assigned to a key. You could even assign an =import command which means that a whole sequence of commands can be issued by a single key press. However, I would recommend that you think twice before assigning any destructive commands such as DELETE. There are times when it's still useful for a command to be awkward to enter!

For a complete description of the =key() command please see section 10.9 =key on page 43.

4.7.3 Keyboard mapping example

During the development of this program one of the machines used was a Linux system running on a Power 64 chip. This machine was accessed using an SSH client over a VPN link. What was immediately noticeable was how different the keyboard mapping was to the other Linux machine MQSCX had been run on. The major difference was that some of the modifier keys such as 'shift' and 'ctrl' were not recognised in combinations with other keys. So, for example, 'ctrl-up' generated exactly the same keyboard code as 'shift-up'. Other keyboard combinations such as 'ctrl-home' didn't generate any keyboard codes at all!

However, it was noticeable that the 'alt' key did generate unique codes. So, in this environment the following keyboard mappings were used for MQSCX to function correctly.

```
=keyname (Shift-Down)    seq(1B5B42)
=keyname (Shift-Left)    seq(1B5B44)
=keyname (Shift-Right)   seq(1B5B43)
=keyname (Shift-Up)      seq(1B5B41)

=keyname (Alt-Home)      seq(1B1B5B317E)
=keyname (Alt-End)       seq(1B1B5B347E)
=keyname (Alt-Down)      seq(1B02)
=keyname (Alt-Up)        seq(1B03)
=keyname (Shift-Alt->)   seq(1B3E)
=keyname (Shift-Alt-<)   seq(1B3C)

=keyname (End)           seq(1B5B347E)
=keyname (Home)          seq(1B5B317E)

=key (Alt-Up)            action(Cursor Up)
=key (Alt-Down)          action(Cursor Down)
=key (Ctrl-Up)           action(none)
=key (Ctrl-Down)         action(none)
=key (Alt-Home)          action(Cursor Top)
=key (Alt-End)           action(Cursor Bottom)
=key (Shift-Alt-<)       action(Select Cursor Top)
=key (Shift-Alt->)       action(Select Cursor Bottom)
```

You can see that in this environment many of the key combinations generate escape sequences. You may find that your system generates the same or entirely different combinations. Either way, these commands give you an idea of what you can do with the `=key` and `=keyname` commands.

If you are lucky enough that your system generates the same keys then by all means enter the same commands into your own version of MQSCX. The easiest way to do this is using the `=import` command. Copy and paste these commands into a file and then use the `=import` command to run these commands from the file.

If you find that your system generates different key codes then you will have to enter different commands. The easiest way of capturing a particular key stroke is to use the special format of the `=keyname` command. Suppose the program didn't recognise the key combination 'shift-left'. All you need do is enter the command:

```
=keyname (Shift-Left) seq(#)
```

MQSCX will then wait for you to press the keyboard combination which should be associated with the name 'Shift-Left'. At any time you can look at the user assigned keys by going to the keys screen by pressing F3 (by default) or issuing the command `'=show scrn(keys)'`.

4.8 Synonyms

In a similar way to key commands you can also set up command synonyms. So, if you don't like the idea of having to remember lots of different key combinations how about giving commonly used commands nicknames. The way you do it is via the `=syn` command. A synonym can point to a number of different types of command.

4.8.1 A single command

```
=syn(dc) cmd(DISPLAY CHANNEL(*) ALL)
```

Issue this one command once and then, every time you enter a command of 'dc', the actual command issued will be 'DISPLAY CHANNEL(*) ALL'. This can save a lot of typing! Just like key commands you can assign virtually any command in this way, including both MQSC and MQSCX commands.

4.8.2 A list of commands

```
=syn(dqa) cmd(DIS Q(Q1); DIS Q(Q2))
```

As with all commands fields it can be defined as a list of commands. Each command will be run in turn one after the other.

4.8.3 A list of synonyms

Suppose we wished to run both of the synonyms above we could enter the command:

```
=syn(both) cmd(dc; dqa)
```

The list can contain a mixture of both synonyms and commands.

4.8.4 Recursion

Since a synonym can 'point' to another synonym the possibility exists to have recursive definitions. If the program detects recursion then an error message will be displayed.

For a full description of the `=syn` command please see section 10.16 `=syn` on page 49.

4.9 Copy and Paste

The MQSCX has native copy and paste capabilities rather than relying solely on the OS command shell.

UNIX only

On UNIX the clipboard buffer is restricted to the MQSCX program only. This can be useful if one wishes to copy a value from a previous command output, such as an object name or connection id, but not ideal if you wish to paste the answer into a secondary program such as a word processor. The reason for this is that UNIX lacks the notion of a global clipboard buffer inherent in the UNIX system, in a similar fashion to the Windows clipboard⁸.

4.9.1 Copy

To copy some text you first need to select the text. Move the cursor to the start of the text to be copied. Then, holding down the shift key, move the cursor to the end of the text to be copied. If you wish to select all the text in an area then you can press <Ctrl-a>. To copy the data to the clipboard press <Ctrl-c>. To cut the data to the clipboard then press <Ctrl-x>. Note that you can only cut the data from the command line since this the only input field.

4.9.2 Paste

To paste data move the cursor to the place where you want the data and then press <Ctrl-v>. Text may only be pasted into the command line since this is the only input field.

⁸ If anyone has a suggestion about how this restriction might be overcome then please feel free to contact me with any information which may be helpful.

4.10 Undo/Redo

Each change made to the command line may, if required, be undone. This includes user typing or pasting data. If changes are undone then they can, if required, be redone. By the default the keys for doing this are <Ctrl-z> and <Ctrl-y> respectively but these keys can be changed if another scheme is more suitable.

The undo/redo sequence only spans a single command. Once a command is entered the undo/redo sequence is cleared.

4.11 Display Totals

Have you ever issued a DISPLAY command and wished there was some simple way to know how many answers were returned? For example you might be trying to gauge how many channels or clients are running. Or perhaps how many connections you have to your Queue Manager. Well, MQSCX offers a very simple mechanism to find out. By default the option is switched off, since it uses an extra screen line, but to switch it on issue the command:

```
=set distotals(on)
```

Now whenever you issue a DISPLAY command you will get a summary line stating the number of responses. If you have used some sort of filter such as =FIND() or =WHERE() then you will be told both how many objects were returned to MQSCX and how many matched the filters.

So, let's try a simple example:

```

C:\> Command Prompt - mqscx -m MYQM

[09:39:17] =set distotals(on)
distotals setting changed

[09:39:25] dis q(*) =where(curdepth)
QUEUE<SYSTEM.ADMIN.QMGR.EVENT>      TYPE<QLOCAL>      CURDEPTH<1>
QUEUE<SYSTEM.AUTH.DATA.QUEUE>      TYPE<QLOCAL>      CURDEPTH<106>
QUEUE<SYSTEM.CHANNEL.SYNCQ>         TYPE<QLOCAL>      CURDEPTH<1>
QUEUE<SYSTEM.CHLAUTH.DATA.QUEUE>    TYPE<QLOCAL>      CURDEPTH<2>
QUEUE<SYSTEM.CLUSTER.REPOSITORY.QUEUE> TYPE<QLOCAL>      CURDEPTH<2>
QUEUE<SYSTEM.DURABLE.SUBSCRIBER.QUEUE> TYPE<QLOCAL>      CURDEPTH<1>
QUEUE<SYSTEM.HIERARCHY.STATE>       TYPE<QLOCAL>      CURDEPTH<2>
QUEUE<SYSTEM.RETAINED.PUB.QUEUE>    TYPE<QLOCAL>      CURDEPTH<2>
Total display responses - Received:66 Matched:8
MYQM>

```

In this example, therefore, the total line is telling us that there are 66 queue definitions and 8 of the queues have messages on them.

4.12 Search

Sometimes, after you've issued a command and seen the responses, it would be nice to be able to search the text for a particular string. MQSCX let's you do this easily using a special form of command starting with a '/' (forward slash)

For example the following command will search for the string 'q1'.

```
/q1
```

The search will be forwards through the entire output and it will be case insensitive.

There are times though where we would like to do something a little different and you can request these by adding one or more suffix characters to the command. The suffixes characters are listed below:

Character	Meaning
-	Perform the search backwards
c	Perform a case sensitive search
s	Search only a single command

So, for example, we can make the same search go backwards by adding a suffix '-' on the command.

```
/q1/-
```

Note that you need to terminate the search string with a '/' character before adding the '-' suffix.

We could make a case sensitive, backward search with a command such as this:

```
/q1/c-
```

It does not matter the order in which the '-' or 'c' characters are specified.

Once you have registered a search string you can use it over and over again using the 'find next' or 'find previous' commands. These are assigned to the key strokes <Ctrl-f> and <Ctrl-g> respectively although they can, of course, be changed. The repeat find commands will use the same options as the last actual find command. For example 'find next' will always be in the same direction as the original find command and 'find previous' will be in the reverse direction.

The starting position for the search will be the current cursor position if it lies in the output text. If it is in the command line then the starting position will either be the start or end of the data depending on the search direction.

Suppose you've just received the response to a command and want to search for a string but only in the response of that last command. You can issue a command something like:

```
/q1/s
```

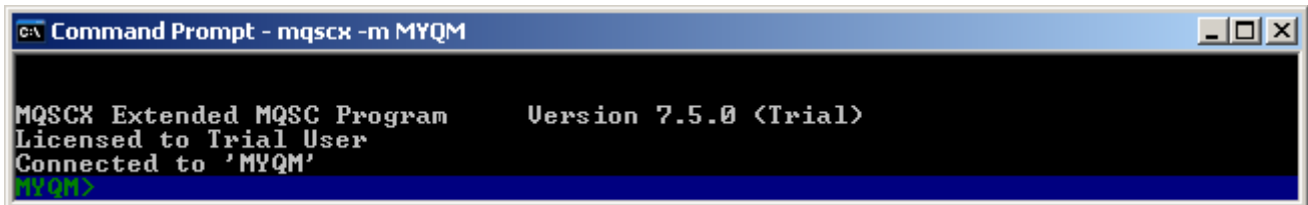
This command will start the search from the start of the last command and search both the command and all of its responses.

5 Changing appearance

One of the most immediate striking differences between RUNMQSC and MQSCX output is MQSCX's use of colour. However, there are quite a number of differences in the way the output is portrayed. Most of these can be changed in one way or another.

5.1 Prompt

By default the prompt of the MQSCX program shows the Queue Manager the program is currently administering. Normally this would also be the Queue Manager the program is connected to but not always since you can control one Queue Manager via another. When MQSCX is first started, and assuming it managed to connect to the Queue Manager, the prompt will look something like this:



This initial prompt can be changed, if required, via the `=set` command.

Take a look at the settings panel (either press F4 or enter the command `"=show scrn(sets)"`). You can see that the 'prompt' setting is set to the string `'%m>'`. Essentially whatever characters the prompt value is set to will appear as the prompt, with the exception of characters following a percentage (%) sign. These characters are substitution characters and will be replaced by the appropriate data.

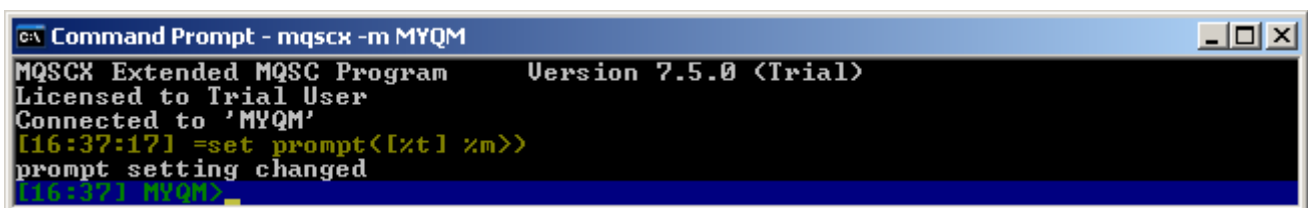
The list of substitution characters are as follows:

Character	Displayed Prompt Value
t	Time in HH:MM format
T	Time in HH:MM:SS format
m	Managed Queue Manager
M	Connected Queue Manager if different from the managed Queue Manager
%	%
c	Conditional insertion if connection is over a client (see below)
l	Conditional insertion if administration is local (see below)
r	Conditional insertion if administration is remote (see below)
C	Conditional insertion if connected to a Queue Manager
D	Conditional insertion if disconnected from the Queue Manager

So, for example, suppose wish to always see the current time in the prompt we could issue the following command:

```
=set prompt([%t] %m>)
```

the effect on the prompt is shown below:



The current time will now always be displayed in the prompt, even when no commands are entered!

5.1.1 Conditional Insertion

Conditional insertion is a way of adding text to a prompt but only if a 'condition' is true. The general idea is that all text between the tags is included but only if the condition is true. So, for example:

```
=set prompt(%cClient %c% >)
```

In this case the text between the two %c tags is only displayed in the prompt if the connection is made over an MQ client. This setting would show the following prompt if MQSCX was connect over client connection.

However it would show the more normal prompt if it was a local connection.

In a similar way we could indicate that the administration of the Queue Manager is being made via an intermediate Queue Manager with a command such as this:.

```
=set prompt(%m%r(via %M) %r>)
```

This would show this type of prompt if MQSCX was administering Queue Manager MYQM via an intermediary Queue Manager NTPGC1.

However, again, it would show the normal prompt if not going via an intermediate Queue Manager.

5.2 Colours

One of the most noticeable differences between the RUNMQSC program and MQSCX is the use of colour to distinguish different output. Colours can be very useful to help the eye make sense of the display screen, without colour a screen full of data can very quickly become tiring to look at. Of course not all of us see colours the same way and each of us has their own preferences so with this in mind the MQSCX program allows the user to modify the colours shown.

5.2.1 Setting the various screen elements

The various screen elements can be set to a range of colours independently. Each element has a foreground and background colour setting. There is also the concept of a 'default' value. The default value allows you to, for example, set a number of elements to have the same background colour without having to set them individually.

Suppose we wanted to set the prompt background colour to red. The command would be:

```
=colour bgprompt(red)
```

if we now wanted it's foreground colour to be white we would enter:

```
=colour fgprompt(white)
```

As you can see the colours are set using the 'colour' command and a prefix of 'bg' and 'fg' before the element name.

We could, if we wished, make both changes in a single command:

```
=colour fgprompt(white) bgprompt(red)
```

A description of the elements and the colours you can use is listed in the =colour command description please see section 10.4 =colour on page 37.

5.2.2 Switching colours off

Some users may find colours distracting and actually prefer all text to be output in the same colours. This can be achieved in a single command:

```
=colour colours(off)
```

The screen will immediately revert to displaying all the data using the 'default' foreground and background for each element. Of course when you get tired of the monochrome display you can always go back to the pretty display using:

```
=colour colours(on)
```

5.3 Columns

MQSCX adopts a more flexible approach to columns than the rather rigid two column format used by RUNMQSC. By default MQSCX will display as many columns of data as will reasonably fit across the width of a given screen. As a consequence MQSCX makes much better use of the available screen real estate.

So, for example, if we display a queue definition we see output like this:

```

C:\ Command Prompt - mqscx -m MYQM
Connected to 'MYQM'
[16:45:12] dis q(Q1)
QUEUE(Q1) TYPE(QLOCAL) ACCTQ(QMGR) ALTDATE(2013-04-18)
ALTIME(16.44.20) BOQNAME( ) BOTHRESH(0) CLUSNL( )
CLUSTER( ) CLCHNAME( ) CLWLPRTY(0) CLWLRANK(0)
CLWLUSEQ(QMGR) CRDATE(2013-02-24) CRTIME(15.08.09) CURDEPTH(0)
CUSTOM( ) DEFBIND(OPEN) DEFPRTY(0) DEFPSIST(NO)
DEFPRESP(SYNC) DEFREADA(NO) DEFSOFT(SHARED) DEFTYPE(PREDEFINED)
DESCR(This is a fairly long description) DISTL(NO)
GET(ENABLED) HARDENBO INITQ( ) IPPROCS(0)
MAXDEPTH(5000) MAXMSGL(4194304) MONQ(QMGR) MSGDLUSQ(PRIORITY)
NOTRIGGER NPMCLASS(NORMAL) OPPROCS(0) PROCESS( )
PUT(ENABLED) PROPCTL(COMPAT) QDEPTHHI(80) QDEPTHLO(20)
QDPHIEU(DISABLED) QDPLOEU(DISABLED) QDPMAXEU(ENABLED) QSUCIEU(NONE)
QSUCINT(999999999) RETINTUL(999999999) SCOPE(QMGR) SHARE
STATQ(QMGR) TRIGDATA( ) TRIGDPH(1) TRIGMPRI(0)
TRIGTYPE(FIRST) USAGE(NORMAL)
MYQM>

```

In general you will notice that the output is ordered into four columns. However, if an attribute is too large to fit in a column the output is adjusted accordingly. In this example the description field (DESCR) is longer than the column so the remaining attributes are adjusted accordingly.

By default MQSCX will adjust the number of columns according to the screen width. However, if you prefer you can explicitly set the number of columns. Suppose we wanted three columns, we would enter the following command:

```
=set columns(3)
```

The setting will apply to all previous output so you will immediately see the effect on the screen. To go back to automatic setting of columns issue the command:

```
=set columns(auto)
```

When in 'auto' mode the number of columns is set by the 'ideal' width of each column. This can also be adjusted. By default it has the value of 20 characters, however, experiment with other values:

```
=set colwidth(30)
```

You will notice that this command will probably reduce the number of columns displayed. However, this can produce a neater display if you have many long attributes. Setting a really small column width is likely to produce a rather messy looking display unless all the attributes are of similar length.

5.4 Separators

The standard RUNMQSC 'DISPLAY' output displays a 'detail' message between each object. Like this:

```

C:\ Command Prompt - runmqsc MYQM
QUEUE<Q3>                                TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<Q4>                                TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<Q5>                                TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<Q6>                                TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<Q7>                                TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<Q8>                                TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<Q9>                                TYPE<QLOCAL>
AMQ8409: Display Queue details.
QUEUE<QFULL>                             TYPE<QLOCAL>

```

These detail messages can be distracting and clearly they take up a significant amount of the output real estate. MQSCX allows the user to replace these messages or remove them all together.

For example, the same command issued by MQSCX will show:

```

C:\ Command Prompt - mqscx -m MYQM
MQSCX Extended MQSC Program      Version 7.5.0 (Trial)
Licensed to Trial User
Connected to 'MYQM'
[16:50:44] dis q(Q*)
QUEUE<Q1>                                TYPE<QLOCAL>
QUEUE<Q2>                                TYPE<QLOCAL>
QUEUE<Q3>                                TYPE<QLOCAL>
QUEUE<Q4>                                TYPE<QLOCAL>
QUEUE<Q5>                                TYPE<QLOCAL>
QUEUE<Q6>                                TYPE<QLOCAL>
QUEUE<Q7>                                TYPE<QLOCAL>
QUEUE<Q8>                                TYPE<QLOCAL>
QUEUE<Q9>                                TYPE<QLOCAL>
QUEUE<QFULL>                             TYPE<QLOCAL>
MYQM>

```

You can clearly see that more objects are shown. Of course there are times when having some form of separator is useful. For example, suppose we decided we wanted a little more information about the queues and changed the command to 'dis q(Q*) curdepth descr ipprocs ipprocs'.

The output we see from MQSCX is now:

```

C:\ Command Prompt - mqscx -m MYQM
QUEUE<Q6>                                TYPE<QLOCAL>    CURDEPTH<0>    DESCR<  >
IPPROCS<0>                                OPPROCS<0>
QUEUE<Q7>                                TYPE<QLOCAL>    CURDEPTH<0>    DESCR<  >
IPPROCS<0>                                OPPROCS<0>
QUEUE<Q8>                                TYPE<QLOCAL>    CURDEPTH<0>    DESCR<  >
IPPROCS<0>                                OPPROCS<0>
QUEUE<Q9>                                TYPE<QLOCAL>    CURDEPTH<0>    DESCR<  >
IPPROCS<0>                                OPPROCS<0>
QUEUE<QFULL>                             TYPE<QLOCAL>    CURDEPTH<0>    DESCR<  >
IPPROCS<0>                                OPPROCS<0>
MYQM>

```

You can see that we now have some form of separator between the object definitions. Why ? Well, since one or more of the object definitions spans more than one line the screen could look confusing without some form of separator. MQSCX will, by default, automatically insert a separator if it thinks it helps. The setting which controls this behaviour is 'autosep'. By default 'autosep' is on but you can switch it off with the command:

```
=set autosep(off)
```

Issue the command and scroll up (if necessary) to look at the previous 'dis q(*)' command. You will see now that we have separators between each of the objects. Since you have switched auto-separators off they are always displayed.

So, the next question you may be asking is “why do we see a solid line rather than the familiar MQ message?”. Well again the answer is simply clarity. The MQ message can be distracting to the eye and a single line is less intrusive. However, different people will prefer different things so, not surprisingly, you can change which separator gets displayed. For example, suppose you issue the following command:

```
=set separator(msg)
```

You will see that the display we had previously will immediately change to:

```

Command Prompt - mqscx -m MYQM
IPPROCS<0>      OPPOCS<0>
AMQ8409: Display Queue details.
QUEUE<Q7>      TYPE<QLOCAL>      CURDEPTH<0>      DESCR< >
IPPROCS<0>      OPPOCS<0>
AMQ8409: Display Queue details.
QUEUE<Q8>      TYPE<QLOCAL>      CURDEPTH<0>      DESCR< >
IPPROCS<0>      OPPOCS<0>
AMQ8409: Display Queue details.
QUEUE<Q9>      TYPE<QLOCAL>      CURDEPTH<0>      DESCR< >
IPPROCS<0>      OPPOCS<0>
AMQ8409: Display Queue details.
QUEUE<QFULL>   TYPE<QLOCAL>      CURDEPTH<0>      DESCR< >
IPPROCS<0>      OPPOCS<0>
[18:10:13] =set separator(msg)
separator setting changed
MYQM>
  
```

You can see that we have our MQ message back.

The separator setting can be set to a number of values, these are:

Value	Meaning
msg	The original MQSC detail message is shown
none	No separator is shown
blank	A blank line
dotted	A dotted line
solid	A solid line (subject to command font)

You will notice that changing the separator setting will immediately affect the existing display, it is not necessary to re-issue the DISPLAY command after changing the setting.

The last thing you may notice about the separator line is that it is displayed in a different colour from the MQ objects. Again this is to make the display less intrusive. You can change the colour used using a command like the following:

```
=colour fgmsgsgs(white) bgmsgsgs(cyan)
```


Feel free to adjust the colours according or go back to the default with the command:

```
=colour fgmsgs (green) bgmsgs (default)
```

For more information about the colour command and the other elements of the screen you can change please see section *10.4 =colour* on page 37.

5.5 Window re-size support

MQSCX will automatically detect the size of the command window and set the display accordingly.

Unix

On Unix the display will be adjust accordingly as the command window is re-sized.

Windows

Unfortunately on Windows MQSCX does not get notifications of screen size changes so if the window is resized for any reason it is necessary to manually cause MQSCX to resize it's screen⁹. The way this is done is to issue the 'Refresh' key action which by default is set to the key combination <Ctrl-r>.

⁹ I would be interested to hear from any Developers on the correct way to detect command window resize events.

6 Configuration file

When MQSCX runs it will look for a file containing the user preference settings. This file is called the configuration file. The file name is always named *mqscx.cfg* and by default it should be in the same directory as the MQSC program.

If you wish to change the directory where the configuration file is read and written to then set the MQSCXCFG environment variable to the appropriate value. For example:

On Windows

```
set MQSCXCFG=c:\tools
```

On Unix

```
export MQSCXCFG=/usr/tools
```

The configuration file format is a simple keyword/value pair. If necessary the file can be edited by hand but this is not recommended.

The configuration file will be updated when the program ends, unless you are running in File Mode described later.

If MQSCX can not write to the configuration file for some reason then an error message will be displayed. MQSCX does try to ensure that a failure during the file write still maintains a complete and valid configuration file. However, it is recommended that you keep a backup of your configuration file just in case you lose the configuration.

7 File Mode

Up until now we have been talking about using MQSCX in interactive mode where the output is being written to a console window and input has been taken from a user typing at a keyboard. Of course there are times when you wish to process a set of commands from a command file and have the output also written to a file. This is referred to as file mode.

There are essentially three ways MQSCX can be run in file mode:

7.1 Piping a file in from stdin

```
c:\> mqscx < mycommand.mqs
```

Since you have piped a file in to the program MQSCX will switch to file mode since it knows that key user input is not possible.

7.2 Explicitly passing in an input file

```
c:\> mqscx -i mycommand.mqs
```

Here we explicitly pass a file to process and it has the same effect as piping in a file. Since input is from a file the program does not expect key input.

7.3 Explicitly asking for file mode

```
c:\> mqscx -f
```

The -f parameter requests that MQSCX runs in file mode but clearly doesn't supply an input command file. In this case MQSCX will read commands in from stdin but in file mode rather than key mode. This means that many of the MQSCX features are switched off. For example, the different display panels are not available nor are colours. Essentially the MQSCX program looks and behaves very similarly to the RUNMQSC program.

7.4 File mode separators

By default the separator used in file mode is the same as configured in interactive mode if the output is to the console window. However, if the output is to a file then the default will always resort to 'msg'. You can use the `=set` separator command to change it if you wish.

In file mode the concept of 'autosep' does not exist so separators are always shown.

7.5 File mode features

Most MQSCX features are not available in file mode since they wouldn't make sense. For example, copy+paste, undo/redo, auto complete are all somewhat meaningless when the command is read from a file. However, some features do still apply when running in file mode. These are:

- **=conn**
- **=disc**
- **=find()**
- **=import**
- **=set**

Although not all settings make sense in file mode. Note also that any changes will not be saved.

The settings which may be changed are:

- **authmsgs**
- **columns**

Of course when writing to a file there is no real concept of screen width however, for the sake of columns, the value must be known. By default it has a value of 80 but can be changed using the -w parameter.

- **colwidth**
- **cswhere**
- **distotals**
- **formatting**
- **separators**

- **=show**

Although only a subset of the show command.

- **licence**
- **machine**

- **=syn()**
- **=where()**

7.6 Configuration file

User preferences and settings are not saved when running in file mode.

8 MQSCX Parameters

The MQSCX program does not **require** any parameters if you are connecting to the default Queue Manager however in all other cases you will need to pass at least one parameter. The parameters are passed as command flags in the standard fashion.

The list of parameters are:

Flag	Parameter (if any)	File Mode Only?	Description
-c	<value>	Yes	The number of columns to display Default value: auto
-C	<Command(s)>	No	One or more commands which should be issued immediately see below for more information.
-d	<Channel Definition>	No	Channel Definition fields e.g. CHANNEL(MYCHANNEL)
-e		Yes	No command echo
-f		Yes	Run in file mode
-l			Connect as a client
-i	<file>	Yes	MQSC input file
-m	<Queue Manager>		Queue Manager to administer
-q	<Queue name>		Command Queue name Default value:SYSTEM.ADMIN.COMMAND.QUEUE
-r	<Queue name>		Reply Queue name Default value:SYSTEM.DEFAULT.MODEL.QUEUE
-s		Yes	Stop on first command failure.
-v	<Queue Manager>		Via Queue Manager The Queue Manager to connect to in cases where you don't administer the Queue Manager directly.
-w	<value>	Yes	Display output width Default value: 80 A value of 0 indicates an effective infinite screen width. This causes each response to be output on a single line.
?		N/A	Display simple command usage help text

8.1 Initial Command

The -C parameter allows the user to pass one or more commands to the program which will be run as soon as the program starts. On most command processors it is necessary to enclose the command in quotes. For example:

```
MQSCX -m MYQM -C "STOP CHL(X)"
```

To run multiple commands you should separate them with a semi-colon (;) character. For example:

```
MQSCX -m MYQM -C "STOP CHL(X) ; STOP CHL(Y)"
```

If the program is being run in file mode and no input file is provided then the after running the initial commands the program will end.

8.2 Channel Definition Fields

If you wish to make a client connection to the Queue Manager the -d parameter can be used to pass in many of the channel definition parameters explicitly. This means that you don't need either the MQSERVER environment variable or a Client Channel Definition Table (CCDT).

An example of passing in the definition this way is:

```
MQSCX -m MYQM -d "CHANNEL(MYCHANNEL) CONNAME(1.2.3.4(1415))"
```

Values are not upper-cased and should therefore be specified in the case that is required.

The channel fields that can be specified are:

Field	Description	Default (if any)
CHANNEL	Channel Name	SYSTEM.ADMIN.SVRCONN
CONNAME	Connection Name	localhost
RCVDATA	Receive Exit User Data	
RCVEXIT	Receive Exit ¹⁰	
SCYDATA	Security Exit User Data	
SCYEXIT	Security Exit	
SENDDATA	Send Exit User Data	
SENDEXIT	Send Exit ¹⁰	
SHARECNV	Share Conversation	10
SSLCIPH	SSL/TLS Cipher Specification	

Specifying any of the channel definition fields on the command line implies that the connection should be made over client bindings. It is therefore not necessary to also specify the -l flag.

¹⁰Only one exit may be specified; chaining is not supported

9 Licensing

Unless you are running a trial or beta version of MQSCX you will need a valid licence file to run the program. A licence file is a simple text file which contains the definitions of the purchased MQSCX licences. This file should only be used for the intended purpose. Licence files should not be transferred or given to third parties.

The licence file is always called **mqgem.lic** and is a simple, human readable file which looks a little like this:

```
* MQGEM Software Licence - created Mon Apr 01 17:39:50 2013
product   = MQSCX
version   = 7.5.0
issue      = 010413
email      = john.smith@example.com
licensee   = John Smith
contact    = John Smith
machine    = JOHNSMACHINE
userid     = JSMITH
expire     = 310413
authcode   = 4C2A-37DB-54CB-3786
```

Some of these fields are optional so the licence file which you have may not contain all the fields. However, let's briefly discuss them all.

- **product**
This field identifies the name of the product for which this is a licence. In this case it must be 'MQSCX'.
- **version**
This field identifies the version of the program that is licensed. A licence will also authorise all previous versions of the program.
- **issue**
The date the licence was issued.
- **email**
The email address of the licensee. This is usually the email address that the licence was initially emailed to.
- **licensee**
The name of the person or organisation who owns the licence. This name will be shown in the MQSCX program when it runs.
- **contact**
The name of the person who is associated with the purchase of this licence.
- **machine**
This field is optional depending on the type of licence you have bought.
If present the licence is only valid on a machine of that name.
You can see the current machine name by showing the help screen in the MQSCX program by pressing F1 or issuing the command **=show scrn(help)** .
- **userid**
This field is optional depending on the type of licence you have bought.
If present the licence is only valid when run by this user id.
You can see the current user id by showing the help screen in the MQSCX program by pressing F1 or issuing the command **=show scrn(help)** .

- **expire**
This field is optional depending on the type of licence you have bought.
If present it shows the date, in DDMMYY format, of when the licence will expire.
- **authcode**
This is the authentication code which ensures that the previous fields are valid and have not been tampered with. The authentication code must be present and must match the previous fields for the licence to be valid.
You should never change the authentication code.

9.1 Licence Types

There are four types of licences which allow different levels of flexibility about who can run the program. Essentially this is controlled by the presence, or not, of the userid and machine fields.

Type	Fields Set	Description
Emerald	userid, machine	This licence must be used by a single userid on a single machine.
Ruby	machine	This licence can be used by any userid on a single machine.
Sapphire	userid	This licence can be used by a single userid on any machine.
Diamond	neither	This licence can be used by any userid on any machine.

9.2 Location

The licence file can be in two different locations:

1. In the same directory as the MQSCX configuration file. This will be the same as the program location unless the MQSCXCFG environment variable is in use.
2. At a directory pointed to by a MQGEML environment variable.

If a valid licence is found in either file then the licence check will pass.

9.3 Multiple licences

If you have multiple licences then they can be concatenated into a single **mqgem.lic** file. This can be done using simple OS commands such as **copy** or by using your favourite editor.

Alternatively, if you want to keep user configurations totally separate you can use the MQGEML environment variable to point each user to their own separate licence file.

9.4 Changing your licence file

I said before that the authentication code must match the previous fields in the licence, and this is true. However, that doesn't mean that the previous fields are totally unchangeable. You can modify them slightly if you wish in the following manner:

- You can change the case of any of the values.
- You can add or remove white space such as blanks

10 Commands

Commands are entered on the command line just like MQSC commands. All MQSCX commands start with an '=' to distinguish them from MQSC commands. You may enter an MQSCX command from any panel although the output will only ever be displayed in the MQSC screen. So, if you want to see the output or are uncertain about the correctness of the command you are entering ensure that you are on the MQSC screen.

10.1 Command Help

Before we talk about the commands themselves it is worth mentioning that MQSCX comes with some simple command help text. For example to see a little of commands you can use enter the following:

```
=?
```

This will then display the MQSCX commands. Now, suppose we are interested in the =set command we could type:

```
=set ?
```

To get a list of the parameters which can be passed to the =set command. We can go one further. Suppose we are curious about the 'cmdhist' parameter. Try typing:

```
=set cmdhist(?)
```

This will give us a brief description of the parameter and its values.

Of course, this command help text is not meant as a replacement for this User Guide but it can be a useful memory jogger if you can't remember the command or the spelling of its parameters.

Auto-completion also applies to the MQSCX commands so if you are uncertain of what needs to be typed next then hitting the <tab> key is often a good reminder.

So anyway.....let's see what commands we can use.....

10.2 =cache

The cache is the retrieved set of object definitions which is used when auto completing the object names in the various MQSC commands. This command allows the user to modify the cache behaviour.

The syntax of the =cache command is as follows:

```
=cache [maxagel (<age> )  
       [maxager (<age> )  
       [purge]
```

where the parameters are:

- **maxagel**
Sets the number of seconds the cache data is considered valid when administering a local Queue Manager before a new set of definitions is retrieved.
The default value is 60 seconds.
- **maxager**
Sets the number of seconds the cache data is considered valid when administering a remote Queue Manager

before a new set of definitions is retrieved. A Queue Manager is considered remote is connected through a client channel or administered via an intermediary Queue Manager.
The default value is 1800 seconds.

- **purge**
When this is specified the current contents of the cache is deleted.

10.3 =cls

This command allows the user to clear the current screen. The previous data is still available by scrolling or moving the cursor.

The syntax of the =cls command is as follows:

```
=cls
```

10.4 =colour

This command is used to control the use of colour in the display. You can set the colour of the various screen elements or, indeed, switch off the use of colours.

The syntax of the =colour command is as follows:

```
=colour [bright(on | off)]  
        [colours(on | off)]  
        [fg<element>(<colour>)]  
        [bg<element>(<colour>)]
```

where the parameters are:

- **bright(on | off)**
If you find the screen colours are too bright or too dull you can use this setting to adjust them. Set the value to whichever is most comfortable for your monitor and ambient light.
The default setting depends on the platform. It is off for Windows and on for Unix.
Note that not all terminals support the concept of 'bright'.
- **colours(on | off)**
Switches the use of colours on or off. When set off all output will be in the colours of the 'default' element colour setting.
- **fg<element>(<colour>)**
- **bg<element>(<colour>)**
The screen elements are:

Element name	Description
command	The command line at the bottom of the screen
default	The default colour.
error	The colour of error messages
input	The colour of command input lines
msgs	The colour of supplementary messages and the separators
output	The colour of command output lines
prompt	The colour of the prompt
scroll	The colour of the scroll indicators

Each of these elements can have their foreground and background colours changed independently.

For example you can change the prompt colours using the command:

```
=colour fgprompt(white) bgprompt(green)
```

The list of colours you can choose from are:

- default
Note that the 'default' colour is not a default value set by the program but the default colour setting you can set using the fgdefault and bgdefault values. The concept of a default setting is useful since some users prefer command prompts to have a black background and others prefer a white background. By setting a screen element to 'default' you are saying that it should be displayed in the 'default' colour, whatever that is set to.
- black
- blue
- red
- green
- cyan
- magenta
- yellow
- white

If you set the foreground and background colours of a particular element to the same value then the foreground will be set to white unless the background is also white in which case it will be set to black. This ensures that the text is not entirely invisible although, of course, it doesn't guarantee it won't be hard to read!

10.4.1 Colour limitations

The actual colours displayed will depend on a number of factors such as the implementation of the colour library, the emulator used and the monitor. On many systems 'yellow' particularly does not display as very yellow, especially with brightness switched off. Similarly white, when used as a background, is more of a murky grey colour¹¹.

10.5 =conn

This command will connect to a Queue Manager. MQSCX can only be connected to a single Queue Manager at any one time. If this command is issued and MQSCX is already connected to a Queue Manager then it will be disconnected automatically before the new connection attempt is made.

The syntax of the =conn command is as follows:

```
=conn [channel(<Channel name>]  
      [client]  
      [cmdq(<Queue name>)]  
      [conname(<Connection Name>)]  
      [qm(<Queue Manager name>)]  
      [rcvdata(<Receive Exit User Data>)]  
      [rcvexit(<Receive Exit>)]  
      [replyq(<Queue name>)]  
      [scydata(<Security Exit User Data>)]  
      [scyexit(<Security Exit>)]  
      [senddata(<Send Exit User Data>)]  
      [sendexit(<Send Exit>)]  
      [sharecnv(<Share Conversation>)]  
      [sslciph(<SSL/TLS Cipher Specification>)]  
      [via(<Queue Manager name>)]
```

¹¹ Improving the colour selection and clarity is a candidate area for improvement in the next version of MQSCX. I would be interested to hear any user views.

where the parameters are:

- **channel(<Channel Name>)**
The client channel definition channel name.
Default: SYSTEM.ADMIN.SVRCONN
- **client**
Indicates that the connection should be made using a client connection.
- **cmdq(<Queue name>)**
The name of the queue to which commands are directed.

If not specified, SYSTEM.ADMIN.COMMAND.QUEUE will be used.
- **conname(<Connection Name>)**
The client channel connection name.
Default: localhost
- **qm(<Queue Manager name>)**
The name of the Queue Manager to administer. If not specified then the default Queue Manager will be assumed.
- **rcvdata(<Receive Exit user Data>)**
The client channel definition receive exit user data.
- **rcvexit(<Receive Exit>)¹²**
The client channel definition receive exit.
- **replyq(<Queue name>)**
The name of the local or model queue which should be used for reply messages.

It is recommended that you use a model queue for your reply queue. Local queues can not be shared amongst multiple MQSCX users, each user will need a different local queue. If you are using a model queue the dynamic prefix which is used is MQSCX.*. The actual reply queue used, therefore, will always start with the characters 'MQSCX.*'. This can be useful for tracking and security profile purposes. The model queue should be defined with DEFTYPE(TEMPDYN) so these reply queues are discarded when the MQSCX program ends. If not specified, SYSTEM.DEFAULT.MODEL.QUEUE will be used.
- **scydata(<Security Exit user Data>)**
The client channel definition security exit user data.
- **scyexit(<Security Exit>)**
The client channel definition security exit.
- **senddata(<Send Exit user Data>)**
The client channel definition send exit user data.
- **sendexit(<Send Exit>)¹²**
The client channel definition send exit.
- **sharecnv(<Share Conversation>)**
The client channel definition share conversation setting.
Default: 10

¹²Only a single exit may be specified; exit chaining is not supported.

- **sslcipph(<SSL/TLS Cipher Specification>)**
The client channel definition SSL/TLS Cipher Specification.
- **via(<Queue Manager name>)**
The name of the Queue Manager to connect to. This parameter is only necessary if the Queue Manager to connect to is different from the one to be administered.

10.5.1 Client Channel Definition Parameters

A number of the parameters allow you to specify the fields of a client channel definition. This can obviate the need for a Client Channel Definition Table (CDDT). If any of the client channel parameters are specified then client binding is assumed and the 'client' parameter is necessary. It is only necessary to specify the parameters which are required, any unspecified parameters will take the default value, if any.

No upper-casing of the parameters is performed so they should be specified in the case that is required.

10.6 =disc

The =disc command will disconnect from the currently connected Queue Manager. This can be useful if you wish to connect and administer a different Queue Manager without ending the MQSCX program.

The syntax of the =disc command is as follows:

```
=disc
```

10.7 end

Entering a command of 'end' will end the program. If running in interactive mode the current settings will be saved to the configuration file. The end command is synonymous with 'quit'.

The syntax of the 'end' command is as follows:

```
end
```

10.8 =import

The =import command allows the user to run a sequence of commands from a file. The commands can be MQSC commands or MQSCX commands. There are many possible uses of this command:

- Just a simple list of MQSC commands to run in sequence
- You could have a sequence of commands in a command file which set up your favourite colours or column widths.
- You could set up a test environment by connecting to multiple Queue Managers and issuing commands to each one.

The syntax of the =import command is as follows:

```
=import file(<filename>)
      [cmdhist]
      [quiet]
      [sof]
```

where the parameters are:

- **file(<filename>)**
The name of the file containing the commands.
The path of the file can either be an explicit name or relative. If it is a relative path then the actual path used will be based on the file name of the file containing the =import command, if any. Otherwise it will be relative to the current directory.
- **cmdhist**
If specified the commands read and processed from the command file are added to the command history.
By default the processed commands are not added to the command history.
Command history only applies to interactive mode so this parameter is ignored in file mode.
- **quiet**
If specified and in filemode the input commands will not be copied to the output. If MQSCX is running in interactive mode then only the local commands (e.g.=colour) are suppressed. In interactive mode MQSC commands are always displayed.
- **sof**
Stop on first failure. The import process will stop if any of the commands fail.

10.8.1 Nesting imports

An import command file can itself also contain =import() commands. This can be very convenient for setting up a more complicated environment especially when you consider that the command file can contain =conn and =disc commands.

Let's say we have a simple example Suppose we have two Queue Managers QM1 and QM2 we could define two configuration files such as:

```
=conn QM(QM1)
DEFINE QLOCAL(MYQUEUE1)
DEFINE QLOCAL(MYQUEUE2)
DEFINE QLOCAL(QM2) USAGE(XMITQ)
DEFINE CHANNEL(QM1.QM2) CHLTYPE(SDR) CONNAME('localhost(1415)') XMITQ(QM2)
DEFINE CHANNEL(QM2.QM1) CHLTYPE(RCVR)
```

File: QM1.mqs

```
=conn QM(QM2)
DEFINE QLOCAL(MYQUEUE1)
DEFINE QLOCAL(MYQUEUE2)
DEFINE QLOCAL(QM1) USAGE(XMITQ)
DEFINE CHANNEL(QM2.QM1) CHLTYPE(SDR) CONNAME('localhost(1516)') XMITQ(QM1)
DEFINE CHANNEL(QM1.QM2) CHLTYPE(RCVR)
```

File: QM2.mqs

```
=import(QM1mqs)
=import(QM2mqs)
=disc
```

File: setup.mqs

Although you can nest import files you can not recurse. In other words you can not effectively have a loop of files which all reference each other. This would cause an infinite loop and is therefore detected and prevented by MQSCX.

When using nested `=import` files it is recommended you can use relative, rather than explicit, paths in the file names. In the example above no path is given to the QM1.mqs and QM2.mqs files. This implies the files are in the same directory as the setup.mqs file. Using relative paths makes copying and using `=import` file much easier since a set of import files, which reference each other, can be copied from one directory tree to another without requiring any of the `=import` files to be changed.

10.9 =key

The =key command allows the user to assign either an action or a command to a keystroke. Note that either a command or an action may be specified but not both.

To remove a user key assignment enter the command with either an empty action or empty cmd keyword. When deleted the key action will revert to the default behaviour if one exists.

If you wish to remove a default action associated with a key and have the key do nothing then specify an action of 'none'. Keys of this type will not appear in the keys screen.

The syntax of the =key command is as follows:

```
=key(keyname) cmd(<command>) | action(<key action>)
                [clear]
                [noenter]
                [nolog]
```

where the parameters are:

- **keyname**
The name of the key to which the command applies. For example, <Ctrl-x>. The key names are case insensitive. Some keys can not be assigned to alternate actions, for example 'left', to avoid confusion.

The key name specified may be:
 - a) A program defined key name
For a complete list of the key names please see *Appendix C. Key Names* on page 63.
 - b) A user defined key name
A value specified on a previous =keyname command
- **cmd(<command>)**
Specifies the MQSC or MQSCX command which should be issued. For example "DISPLAY QUEUE(*)". An empty cmd value signifies that the key assignment should be deleted.
- **action(<key action>)**
Specified the action which should be performed. For example, "Cursor Up".
An empty action value signifies that the key assignment should be deleted.
An action of 'none' indicates that this key should be removed from the key assignment table.

For a complete list of the key actions please see *Appendix D. Key Actions* on page 64
- **clear**
This parameter is only applicable when creating a key command.
When specified the current contents of the command line are cleared before the command is issued.
- **name(<key name>)**
This parameter allows the user to name the key so that a user name is displayed on the keys panel. This is most useful when the program does not know a key or key combination.
For a complete list of the key names please see *Appendix C. Key Names* on page 63.

User defined names must conform to the name rules.

For the valid name rules please refer to *Appendix E. Valid Names* on page 65.

- **noenter**

This parameter is only applicable when creating a key command.

When specified the command is copied to the command line but the command is not issued. For example, suppose you frequently issue the command like **dis chs(*) =where(conname lk 'xxx')** however each time you do the IP address you specify in the =WHERE clauses changes. In this case it can be useful to define a key such as **=key(ctrl-i) cmd(dis chs(*) =where(conname lk '')) noenter clear**

Now, every time you press <Ctrl-i> the command will be shown to you ready for you to change the IP address mask and issue the command.

- **nolog**

This parameter is only applicable when creating a key command.

When specified the command is not logged to the output stream. This only applies to MQSCX commands. MQSC commands will always be logged.

10.10 =keyname

The =keyname command allows the user to create or delete a new keyname. A keyname is a name given to a keyboard keystroke. Due to the nature of platform differences, OS versions and emulators it is not feasible for the MQSCX program to come with predefined keys defined for every key stroke the user may enter. You will tend to find that on Windows most keys will already be defined however on Unix it is far more likely that some key combinations are recognised. It all depends on the OS and emulator you are using.

The keyname command allows us to 'fill the gaps'. Any key strokes which are received by the program which aren't recognised can be given a name. This key name can then be used in a =key command and the key stroke can therefore be assign to an action or command.

To remove a keyname enter the command with an empty sequence parameter. When a user keyname is deleted the keyname will revert to the default behaviour if one exists.

The syntax of the =keyname command is as follows:

```
=keyname (keyname) seq (<key sequence>)
```

where the parameters are:

- **keyname**

The name of the key to which the command applies. For example, <MyKey>. The key names are case insensitive.

The key name specified may be:

- a) A program defined key name

This is useful to override the default key assignment of the key.

For a complete list of the key names please see *Appendix C. Key Names* on page 63.

- b) A user defined key name¹³.

A value specified in a name() parameter on a previous =key command

For the valid name rules please refer to *Appendix E. Valid Names* on page 65.

¹³ If you find that commonplace key combinations you use do not have a key name then please feedback your requirement. It is possible that some key names are common enough that they could be added.

- **seq(<key sequence>)**

Specifies the sequence of key strokes, one or more, which are received by the program when this key is pressed. The value can be:

1. **A hexadecimal string of the keystroke**

In the form : 0xHH [(0xH)] - for example 0x87 or 0x1B (0x2)

If a key is pressed which isn't recognised this hex string will be displayed on the screen.

2. **An escape sequence**

On some Unixes, depending on the emulator, a key press may actually cause multiple key strokes to be sent to the program one after the other. These sequences always start with the <Escape> key hence the name.

If a key is pressed which generates an escape sequence which isn't recognised the program will display the sequence on the screen.

3. **'#'**

The special value of # tells the program to assign the next key stroke issued by the user. The program will then wait for a key to be pressed. This is useful to avoid having to remember the key sequence being generated. Pressing <Escape> at this point will cancel the =keyname command.

10.11 =purge

The =purge command allows the user to purge the current output. Note that all the output will be deleted and can not be restored. If you only want to clear the current screen without losing the output history then use the =cls command.

The syntax of the =purge command is as follows:

```
=purge
```

10.12 quit

Entering a command of 'quit' will end the program. If running in interactive mode the current settings will be saved to the configuration file. The quit command is synonymous with 'end'.

The syntax of the 'quit' command is as follows:

```
quit
```

10.13 **=repeat**

The repeat command allows you to issue a command repeatedly, every few seconds, and display the results. This command is useful for development and demonstration purposes. It is not proposed that one uses this command to monitor the WebSphere MQ product.

The syntax of the =repeat command is as follows:

```
=repeat [clear]
        [cmd(<command>)]
        [delay(<delay time>)]
        [keep]
        [history]
```

where the parameters are:

- **clear**
Specifies that the screen should be cleared before the commands are issued.
- **cmd(<command>)**
Specifies the command to be issued.
- **delay(<delay time>)**
Specifies the number of seconds delay between issuing each command.
The default value is 10 seconds.
- **keep**
Specifies that the output should be retained once the command sequence is complete.
- **history**
By default the =repeat command will display the output of the commands overlaid over each other. However, if you specify the 'history' option then the output will scroll showing a history of all the command outputs.

For example the following command will repeatedly show a queue definition and allow the user to monitor it's depth and usage.

```
=repeat cmd(dis q(q1)) clear delay(1)
```

10.14 =set

The =set command is used to set various user preferences. The syntax of the =set command is as follows:

```
=set [authmsgs(on | off)]
    [autosep(on | off)]
    [cmdhist(<history>)]
    [columns(<columns> | auto)]
    [colwidth(<width>)]
    [cswhere(on | off)]
    [distotals(on | off)]
    [formatting(on | off)]
    [history(<history>)]
    [mcursor(on | off)]
    [prompt(<prompt>)]
    [separator(<type>)]
    [sound(on | off)]
    [timestamp(on | off)]
    [wrap(on | off)]
```

where the parameters are:

- **authmsgs(on | off)**
Specifies whether MQSC failed authorisation messages, such as AMQ8135, should be shown or not on a DISPLAY command.
- **autosep(on | off)**
Controls whether separator lines should be dependant on the data context. If 'off' the separator will be shown according to the *separator* setting. If *autosep* is 'on' a separator will only be shown if one or more MQ objects spans multiple lines and multiple objects are returned.
- **cmdhist(<history>)**
Specifies the size of the command history.
The default value is 50.
- **columns(<columns> | auto)**
Specified how many columns should be displayed.
The default value is auto.
- **colwidth(<width>)**
Specifies the width of the MQSC output columns.
This value is only used when the columns value is set to 'auto'. The default value is 20.
- **cswhere(on | off)**
Specifies whether the =where() clause should be case sensitive.
By default the where clause is not case sensitive.
- **distotals(on | off)**
Specified whether a total line should be displayed at the end of a DISPLAY command output. This total line will state how many items were returned and how many items matched any local filters.
- **formatting(on | off)**
Specifies whether MQSCX should format the MQSC responses. If set to 'off' then the MQSC responses are shown just as they were received from the command server additional formatting.

- **cmdhist(<history>)**
Specifies the size of the output history.
The default value is 20000 lines.
- **mcursor(on | off)**
Specifies whether a mouse click on the command window should move the cursor.
- **prompt(<prompt>)**
Specifies the format of the command prompt.
For a full description of the format please see section 5.1 *Prompt* on page 22.
- **separator(<type>)**
Specifies the type of separator which should be used when displaying object output.
For a full description of separators please see section 5.4 *Separators* on page 26.
- **sound(on | off)**
Specifies whether any errors issues by the program should be accompanied by a 'beep' sound.
- **timestamp(on | off)**
Specifies whether each command in the output display should be displayed with a timestamp.
- **wrap(on | off)**
Controls the level of formatting. If set to 'on' then the MQSC responses will not be displayed in columns and will wrap around the end of the screen.

10.15 =show

The =show command is used to control what is displayed to the user.

Note that not all options are available in file mode. The syntax of the =show command is as follows:

```
=show [licence]
      [machine]
      [scrn(help | keys | mqsc | next | sets)]
      [scroll(on | off)]
```

where the parameters are:

- **licence**
Brief information about the licence is displayed. This information is also shown on the help screen.
- **machine**
Brief information about the machine and any connected Queue Manager is displayed. This can be useful to determine the running user and machine name needed for licensing. This information is also shown on the help screen.
- **scrn(help | keys | mqsc | next | sets)**
Sets which screen should be displayed

Value	Meaning
help	Show the help screen
keys	Show the key assignment screen
mqsc	Show the MQSC display
next	Show the 'next' screen. This command cycles round the screens.
sets	Show the settings screen.

- **scroll(on | off)**
Sets whether the scroll indicators should be shown or not.

10.16 =syn

The =syn command is used to set and remove command synonyms. It allows the user to give commonly used commands a shorter nickname.

For example the command:

```
=syn(dc) cmd(DISPLAY CHANNEL(*) ALL)
```

This command will assign a synonym such that any time the command 'dc' is entered the actual command issued is 'DISPLAY CHANNEL(*) ALL'.

To remove the synonym enter the command with an empty cmd value. In this example it would be :

```
=syn(dc) cmd()
```

The current list of synonyms is displayed on the keys screen. For a description of the screens please see 2.4 Screens on page 4.

The syntax of the =syn command is as follows:

```
=syn(<synonym>) cmd(<command>)
```

where the parameters are:

- **syn(<synonym>)**
The name of the synonym to be used.
This value should be 10 characters or less and must be a valid name. The synonym name is case insensitive.

For the valid name rules please refer to Appendix E.Valid Names on page 65.

- **cmd(<command>))**
Sets the command to be used. The value can be:

Description	Example
An MQSC Command	=syn(dc) cmd(DIS CHL(X))
An MQSCX Command	=syn(help) cmd(=show scrn(help))
A list of commands	=syn(dqa) cmd(DIS Q(Q1); DIS Q(Q2); DIS Q(Q3))
A list of synonyms	=syn(dall) cmd(dc ; dqas)

Since a synonym can 'point' to one or more other synonyms there is a possibility of a recursive definition. If an attempt is made to run a recursive synonym then an error message will be displayed.

11 Connecting to the WebSphere MQ Queue Manager

MQSCX supports either connecting directly to the Queue Manager being administered or going via an intermediary Queue Manager. If you are using the 'via' method then clearly channels need to be set up to exchange messages between the two Queue Managers in the normal way.

To connect to the Queue Manager MQSCX supports both local and client bindings. The type of connection is controlled by

- parameter '-l' passed to the program
- the parameter 'client' used on the `=conn` command which is described in section 10.5 on page 38.

If connecting as a client then normal MQ rules apply. The client channel configuration can be either

- Client Channel Definition Table (CCDT)
- MQSERVER variable
- Active Directory (Windows)
- PreConn exit

It is not currently possible to directly configure MQSCX with the channel definition to use.

11.1 Connection Status

If an error occurs with the a connection, such as the Queue Manager ending or a network failure, then MQSCX will display the failure in a status bar at the top of the screen. MQSCX will then disconnect from the Queue Manager. Once the Queue Manager is available again then the user can issue an `=conn` command to connect back to the Queue Manager to continue administration.

11.2 Switching connections

MQSCX allows the user to disconnect and subsequently reconnect to either the same or different Queue Manager without ending the program. The way that this is done is by using the `=conn` and `=disc` commands.

This ability to connect to different Queue Managers without requiring the program to end enables the configuration of multiple Queue Managers from a single script with can be very useful. In addition the use of the `=import` command can make this scenario very easy to configure. Please see section 10.8.1 *Nesting imports* on page 41 for an example.

12 Program Settings

MQSCX has many settings which can be tuned by the user. The current values of these are displayed on the settings panel which is display by the command '=show scrn(sets)' or by default pressing F4.

The user settings can be changed by issuing one of the following commands:

- =set on page 47.
- =cache on page 36.
- =colour on page 37.

User settings are written to the configuration file except if MQSCX is running in file mode. In this way user settings are restored when the program is restarted.

12.1 Multiple Users

If there is more than one user of the MQSCX program then it is possible for them to each have their own program settings. This is achieved by giving each user their own configuration file.

There are two ways to achieve this:

1. Give each user a separate copy of the MQSCX program
By default MQSCX will look for a configuration file in the same directory as the program itself.
2. Use the MSCXCFG environment variable
The MQSCX environment variable can be set to point to the directory where the configuration file should be stored. Please see section 6 file on page 29 for more information.

13 Performance

The purpose of MQSCX is not to provide a faster MQSC program and generally speaking the performance of MQSCX is comparable with RUNMQSC. However, there is one area where MQSCX has a considerable advantage and that is in writing to the console. This is particularly noticeable on Windows where writing to the console is particularly slow.

The difference lies in the way that the two programs write to the screen. RUNMQSC essentially treats the screen just like a file. It writes each response line to the screen so the user sees a scrolling list of replies. MQSCX on the other hand uses direct screen access. This means that it only needs to write to the screen when it believes it has something to say. So, when issuing a command with a lot of replies it doesn't need to update the screen for every reply but just at the end¹⁴.

Of course results will vary wildly depending on the OS version, command being issued and the machine the command is running on but to give you an idea here's what I measured on my own machine.

13.1 Performance Test

Command Issued	: 'dis q(*) all' with 5,000 defined alias queues
OS	: Windows 7 SP1
Machine	: Intel i5 3.2 GHz 8GB Ram
RUNMQSC	: 67 seconds
MQSCX	: 1.5 seconds

¹⁴ In actual fact if it is taking a long time to gather all the responses to a command MQSCX will update the screen after the first second to show avoid the user just looking at a blank screen.

14 National Language Support

MQSCX is currently only available in English. If there is sufficient demand for alternative languages then this may be considered in the future.

15 Security

MQSCX is an MQI application, it is written in 'C' and uses the standard MQI to interface with WebSphere MQ. It does not use any private interfaces into WebSphere MQ. As such it is subject to exactly the same security mechanisms as any other MQ program.

Consequently MQSCX does not allow a user to do any more than they could do using other mechanisms, such as writing an MQI program themselves. In particular, MQSCX is not a setuid program and therefore does not require the user to be a member of the 'mqm' group to run it.

MQSCX always communicates, through queues, to the command server, even when locally connected to the Queue Manager. Therefore, for successful operation:

- The command server must be running
- The user must have the authority to connect and inquire the connected Queue Manager.
- The user must have the authority to display the administered Queue Manager.
- The user must have MQPUT authority to the SYSTEM.ADMIN.COMMAND.QUEUE
- The user must have MQGET authority to the reply queue which is being used.
If the reply queue is a model queue, which it is by default, then the reply queue will be created with a dynamic queue name of MQSCX.*. This can be useful in identifying queue usage and also creating security profiles specific to MQSCX if required.

On z/OS you will need MQPUT access to the generic profile MQSCX.*

- The user must have the authority run the particular command in question.
If MQSCX reports that 'The Command Server is slow in responding' and this message never clears then the most likely cause is that the request has been rejected by the command server for some reason. Look on the Dead Letter Queue and see whether there are messages put by the Command Server complaining of a security failure.

15.1 Example security commands for Distributed Platforms

Suppose you have a user, in the group MQSCXGroup, who wishes to use MQSCX. Below are listed the minimum authorities the user requires in order to use MQSCX.

```
SET AUTHREC OBJTYPE(QMGR) +
  GROUP(MQSCXGroup) AUTHADD(dsp,connect,inq)

SET AUTHREC OBJTYPE(QUEUE) GROUP(MQSCXGroup) +
  PROFILE(SYSTEM.ADMIN.COMMAND.QUEUE) AUTHADD(put)

SET AUTHREC OBJTYPE(QUEUE) GROUP(MQSCXGroup) +
  PROFILE(SYSTEM.DEFAULT.MODEL.QUEUE) AUTHADD(all)

SET AUTHREC OBJTYPE(QUEUE) GROUP(MQSCXGroup) +
  PROFILE(SYSTEM.DEFAULT.MODEL.QUEUE) AUTHADD(put,get,dsp)
```

16 Return Code

MQSCX will return an integer value to indicate success or failure of the program. The following table lists the possible values.

Please note that the list of return code is subject to change. If you are writing code to examine the reason code then please use a range comparison in addition to an exact comparison.

Success Reason Codes (n < 64)	Meaning
0	One or more commands processed successfully, no failures
Warning Reason Codes (64 ≤ n < 128)	
64	No failures but no messages processed
65	One or more commands processed, one or more MQSC commands failed
Error Reason Codes (128 ≤ n < 256)	
128	Could not connect to Queue Manager
129	An MQI call failed
130	Out of memory error
131	No MQSC commands were successful.
132	Parameter error
133	Input file error
255	Internal error

17 Problem Determination

If you have a problem when using MQSCX please spend a little time reading the Frequently Asked Questions (FAQ) to see whether your situation is covered in there. It is usually faster to solve your problem yourself. Remember that software fails for a reason and most often it is environmental. This means that most of the time there is some configuration or setting or installed code on your machine which is causing the failure.

Look for error messages produced by MQSCX, by WebSphere MQ and your OS console to see whether there is a message explaining the failure to you. Remember particularly that WebSphere MQ has a number of places it writes error message. The client error log plus error logs for each of the Queue Managers.

If your situation is not covered in the FAQ then please do raise a question with Support, details of how to do this are given below.

17.1 General frequently asked questions

17.1.1 It says 'Please set required location of configuration file.....' ?

Normally the configuration file would live in the same directory as the MQSCX program itself and that is the default location the program used. However, on some platforms, for example AIX, it is not possible for the program to query the directory it is running from¹⁵. On the platforms therefore it is necessary for the user to use the environment variable MQSCXCFG to explicitly give the location where they would like the configuration file stored.

17.2 MQ Related frequently asked questions

17.2.1 What does 'Can not find WebSphere MQ on this machine' mean ?

MQSCX dynamically loads WebSphere MQ. It does this so that it can dynamically load either the local or client bindings as required. However, that does mean that although the program may load and run it is not able to then subsequently load the required MQ libraries at run time. This can be for a number of reasons:

- You don't actually have WebSphere MQ installed on the machine. This may seem obvious but you wouldn't be the first person who has tried to run an MQ program without first installing WebSphere MQ. MQSCX requires either the MQ server or the MQ client or both are installed.
- You are trying to use the 'wrong' binding. Make sure that if you have the server installed, then you are trying to do a local connect. If you have the client installed that you are doing a client connect. Check the '-l' parameter or 'client' parameter on the =conn command.
- Check your library path (LIBPATH on Unix and PATH on Windows). It may be necessary to change your library path to include the relevant MQ libraries.

For example on Unix, export LIBPATH=/usr/mqm/lib64

Make sure you specify the right directory and point to the 32bit or 64bit libraries depending on the version of MQSCX you are using. For example, if you are running a 32bit MQSCX then you must point it to the 32bit version of the MQ libraries. It is not possible to mix and match, that is not supported by the OS.

If all else fails you can see what library MQSCX is trying to load by issuing the following command sequence:

```
export MQACCESS_DEBUG=yes
mqscx -f -Dg
```

¹⁵This is quite surprising. There are many discussion on the web about this 'omission'.

The environment variable tells the program to output what files are being dynamically loaded or any error message returned by the system. Since these errors are written to stdout it is necessary to run the MQSCX program in file mode, hence the -f parameter. Finally we run the MQSCX program in debug mode so that it prints out whether it is a 32bit or 64bit program after the initial message, for example:

```
MQSCX Extended MQSC Program - Version 7.5.0
Build Date: Jun  1 2013 64Bit BigEndian
```

17.2.2 Why is my connect failing ?

MQSCX will output the reason code which broadly explains the failure. Unfortunately many of the reason codes can cover many situations which can make diagnosis more tricky. The key thing to remember is that MQ will write out errors to its error logs. So, if you have a failure that you weren't expecting looking at the MQ error logs is a good place to start. Remember that there is a different error log for client connections than for server connections. However, it is probably best to check both just to be sure. Look in the following files.

- *<MQ Installation Directory>/errors/AMQERR01.LOG*
- *<MQ Installation Directory>/Qmgrs/<QMNAME>/errors/AMQERR01.LOG*

Common causes for a connection failure include:

- Misspelling or forgetting to set the Queue Manager name
- Connecting using server rather than client bindings (or vice versa)
- Not setting up the necessary client connection security
- Not setting up the correct multi-install environment, using setmqenv,

17.2.3 Why does MQSCX make two connections to the Queue Manager ?

Making two MQ connections is a common approach taken by many MQ applications. Essentially one connection is used for putting messages to the Queue Manager and the other connection is used for consuming messages. This allows the application to remain responsive at all times.

If you are connecting over a client connection then setting SHARECNV to a value of 2 or more will ensure that only a single socket is used for the two connections and therefore overhead is kept to a minimum.

17.2.4 Why do I see 'Command Server or network slow in responding' ?

The answer to this question will probably depend on how often it happens.

- **It has never worked, this is the first time it has been set up**
The most likely cause here is that you don't have the appropriate authority at the command server and the command messages are being written to the Dead Letter Queue. Examine the Dead Letter Queue for any message indicating a processing failure.

Another common reason is that the channels to the target Queue Manager are either not started or have been incorrectly configured. Again there could be messages on the Dead Letter Queues indicating a routing failure. Remember to check all Queue Managers involved and channels in both directions.

- **It usually works but I see it occasionally**
If you only occasionally see this message then it could be that you channels are either not started or are taking some considerable time to start.
- **It has worked but now I see this message all the time**
The most likely reason here is that one or other of the channels to or from the Queue Manager is down and is not restarting. Check that your channels are configured for triggering.

17.3 Keyboard related frequently asked questions

17.3.1 Why is there an unnatural delay when I press the <ESCAPE> key ?

This is one of those vagaries of Linux which is hard to justify in this day and age. Some Unixes still make extensive use of escape sequences to indicate special characters from the keyboard. To distinguish between these escape sequences and just the user pressing the escape button the curses library will wait some period of time after the escape character to see whether any more characters are forthcoming.

You can modify this wait time using the ESCDELAY environment variable. In my experience setting a fairly low value produces the desired effect. For example:

```
export ESCDELAY=10
```

Of course you may find your environment needs a higher value so set this variable accordingly.

17.3.2 Why do F1, F2, F3 and F4 not work but the other function keys do ?

This can happen when using an emulator which doesn't send the key sequences which the curses library expects. Try changing the terminal emulator to 'xterm-R6' which often gives the correct result.

17.3.3 Why do I get told key or key sequence is unrecognised ?

MQSCX uses the curses library for keyboard input. Although curses has a fairly extensive knowledge of all the different Unix emulators and terminal types it is not feasible for it to know them all. Consequently it concentrates on the most common key combinations. The less common combinations are still given to the MQSCX program however they are not decoded by curses but are instead passed in raw form. These key codes are just a sequence of one or more bytes and therefore don't have an inherent name. For example there is no way for MQSCX to implicitly know what the escape sequence '0x1B1B5B347E' means. However, you, the user, know what it means because you pressed the key to generate it. Consequently you can issue an appropriate =keyname command and give this key code a sequence an explicit name.

17.4 Display related frequently asked questions

17.4.1 Why is there a delay for a second or two when I resize the screen ?

You may also see the screen clear to a colour during this period. Some of the old versions of the curses library did not support terminal resizing. This means that when the terminal is resized curses itself doesn't know about it. On these systems the way resizing is done is for MQSCX itself to catch the resize signal and then inform curses. Unfortunately though there is no simple mechanism to wake the curses keyboard function. The solution to the problem is to upgrade your version of the curses library if possible.

17.4.2 Why are Red and Blue interchanged ?

This is a bug in early versions of curses but was fixed in Version 4.1. If you can, install a later version of curses.

17.5 Support

I am sorry you are having problems and need support. Please feel free to email me the details of the problem and I will do my best to help you. Remember that the more complete you make the description of the problem the better chance I have of solving it. The kind of information you should include in your email is:

- Your full name
- The exact version, including build date, of the MQSCX you are using.
- The email address and issue date from within your licence file (if you have one)
- The OS platform and version you are using
- A complete description of the problem and how to recreate it. Please include as much detail as possible such as frequency of occurrence. For example, does the problem happen every time or just occasionally. Are there any error messages produced by MQSCX or WebSphere MQ at the time of the problem ?

Once you have gathered this information please email it to support@mqgem.com. I shall reply as soon as possible. Note that priority will be given to customers based on the severity of the problem and the type of licence held.

Appendix A. =WHERE Operators

Listed below is a complete list of the operators which can be used in an =WHERE expression.

Operator	Meaning	Synonyms
Standard WHERE clause operators		
EQ	Equals	=
NE	Not Equals	<> !=
GE	Greater Than or equals	>=
LE	Less Than or equals	<=
GT	Greater Than	>
LT	Less Than	<
LK	Like – wildcard comparison. (see wildcard note below)	==
NL	Not Like – wildcard comparison. (see wildcard note below)	
CT	Contains	
EX	Does not contain	
CTG	Contains generic (see wildcard note below)	
EXG	Does not contain generic (see wildcard note below)	
Additional Operators		
=	Equals	EQ
==	Wildcard comparison	LIKE
!=	Not equals	NE <>
<>	Not equals	NE !=
OR	Logical OR	
	Logical OR	OR
	Bitwise OR	
AND	Logical AND	&
&	Logical AND	AND
&&	Bitwise AND	
>	Greater Than	GT
>=	Greater Than or Equals	GE
<	Less Than	LT
<=	Less Than or Equals	LE
-	Minus	
+	Plus	
*	Multiply	
/	Divide	
%	Modulus	
NOT	Logical NOT	!
!	Logical Not	NOT

Appendix A.1. Wildcards

Wherever wildcards are allowed the restriction of only having a single wildcard at the end of the string has been lifted. Any wildcarded string can contain as many wildcards as you like and in any position. The character '*' is used to refer to zero or more characters. In addition the character '?' can be used to signify exactly one character.

So the expression below is perfectly valid:

```
=where(descr == '*TEST?? Machine*')
```

Appendix A.2. Operator Synonyms

A number of the operators have synonyms. This is, to a large extent a matter of taste but it can also save you typing. Consider a simple command asking to see all queues with more than 10 messages. The standard WHERE clause would be:

```
display queue(*) where(curdepth gt 10)
```

Note that we have to type spaces around the operator 'gt' to prevent the command processor thinking that there might be an attribute called 'curdepthgt'! If you use symbol rather than letter based operators then there is no such problem.

Using the =WHERE clause we could use exactly the same expression however we can also use this command:

```
display queue(*) =where(curdepth>10)
```

Now, personally I find this easier to understand and it is also less typing since I don't need add spaces around all my operators. Of course if we're trying to save typing we could even do this:

```
dis q(*) =where(cur>10)
```

As I said, it's personal taste. Use which ever style you feel is more natural.

Appendix B. =WHERE Functions

The =WHERE clause also supports a few functions which can be useful in certain circumstances.

Function	Meaning	Number of Parameters
lower()	Return the lower case version of the string	1
max()	Return the maximum of the parameters	any
min()	Return the minimum of the parameters	any
sqrt()	Return the square root of the parameter	1
upper()	Return the upper case version of the string	1

Appendix C. Key Names

Below is the full list of key names known to the program. MQSCX uses the curses library and is therefore limited to the keys which that library exposes as known combinations. So, although pressing a certain key on the keyboard may produce a unique key code it may not have a predefined name nor is it guaranteed to be the same on each terminal emulator. MQSCX allows the user to assign their own names to keys or over-ride the standard names using the `=keyname` command.

Key Name	Description (if not obvious)
Enter	Non-assignable
Insert	Non-assignable
Delete	Non-assignable
Backspace	Non-assignable
Escape	(On Unix) The CURSES library on Unix inserts a delay when processing an 'escape' key - this is to distinguish it from other function keys. In my experience the delay is excessively long. You can usually tell the curses library to use a different delay using the ESCDELAY environment variable. Before starting MQSCX ensure that you have set this variable something like: <pre>export ESCDELAY=10</pre>
PageDown	
PageUp	
Up	
Down	
Left	Non-assignable
Right	Non-assignable
Ctrl-up	
Ctrl-down	
Ctrl-left	(Windows only)
Ctrl-right	(Windows only)
Home	
End	
Tab	
Shift-Tab	
Num-A1 <====> Num-C3	(Windows only) The 9 keys of the numeric keypad Num-A1 being top left key, Num-C3 being bottom right key
Shift-left	
Shift-right	
Shift-Home	
Shift-End	
Shift-Ctrl-Home	
Shift-Ctrl-End	
Alt-left	(Windows only)
Alt-right	(Windows only)
Ctrl-a <====> Ctrl-z	
Alt-0 <====> Alt-9	(Windows only)
Alt-a <====> Alt-z	(Windows only)
F1 <====> F24	Function Keys

Appendix C.1. Emulator keys

In some cases you may find that a key already has a special meaning in the terminal emulator that you are using. For example, on some Unix emulators function key F1 is caught and used to display emulator help. In these circumstances you should either disable or change the assign key in your emulator or use the `=key` command in MQSCX to associate a different key with the required action.

Appendix D. Key Actions

Below is the full list of actions which can be used with the =key command.

Key Action	Description
Enter Command	Enter the command on the command line
Toggle Insert	Switch between insert and over-write mode
Delete Character	Delete character under the cursor
Backspace	Delete character immediately before the cursor
Clear command line	Clears the command line
Next Page	Scrolls forward one page
Previous Page	Scrolls backwards one page
Previous Command	Shows previous command in command history
Next Command	Shows next command in command history
Next Value	Shows the next command value
Previous Value	Shows the previous command value
Undo last change	Undoes the last change
Redo last change	Redoes the last change
Cursor up	Moves cursor up
Cursor down	Moves cursor down
Cursor left	Moves cursor left
Cursor right	Moves cursor right
Cursor Home	Moves cursor to start of line
Cursor End	Moves cursor to end of line
Cursor Top	Moves cursor to top of display
Cursor Bottom	Moves cursor to bottom of display
Select Cursor up	Moves cursor up and selects intervening text
Select Cursor down	Moves cursor down and selects intervening text
Select Cursor left	Moves cursor left and selects intervening text
Select Cursor right	Moves cursor right and selects intervening text
Select Cursor Home	Moves cursor to start of line and selects intervening text
Select Cursor End	Moves cursor to end of line and selects intervening text
Select Cursor Top	Moves cursor to top of display and selects intervening text
Select Cursor Bottom	Moves cursor to bottom of display and selects intervening text
Select All	Select all text If cursor is in command line then selects all command line, otherwise it selects all text
Copy	Copy current selection to clipboard
Cut	Copy current selection to clipboard and remove text
Paste	Paste current selection at cursor position. Cursor must be on command line
Refresh	Refresh screen information including screen size
Find Next	Find the next search string forwards through the output
Find Previous	Find the previous search string backwards through the output
Show Help	Show the help panel
Show MQSC	Show the MQSC panel
Show Keys	Show the Key assignment panel
Show Settings	Show the current user settings panel

Appendix E. Valid Names

A names field must conform to the following rules.

- Names are case insensitive
- Names must not start with a numeric field
- Names may only valid characters, where valid characters are any of:
 - Alphanumeric characters
 - Underscore (_)
 - Minus sign (-)
 - Period (.)
 - Angle brackets (<) and (>)
 - Square brackets ([])

End of document