MX User's Guide

for MX 1.5.1

William M. Lavender
Illinois Institute of Technology
Chicago, IL 60616 USA

January 5, 2009

MX has been developed by the Illinois Institute of Technology and is available under the following MIT X11 style license.

Copyright 1999 Illinois Institute of Technology

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL ILLINOIS INSTITUTE OF TECHNOLOGY BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of Illinois Institute of Technology shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from Illinois Institute of Technology.

Contents

Chapter 1

Introduction

The introductory chapter for this document has not yet been written. Instead, look at the ***Introduction*** *chapter of the* ***MX for Area Detectors*** *document which covers much of the same ground, but only for area detector devices.*

Chapter 2

Scans

An MX scan is a database object that sequences through a series of measurements and then writes out the measurement data to an external object such as a file. The measurements can come from almost any type of device supported by MX, such as analog and digital I/O, motor positions, counter/timer readouts, MCA spectra, and so forth.

MX scans currently support three kinds of measurements, which differ depending on the way that the measurements are triggered or gated.

- *preset time* - In this mode, an MX timer device is used to specify the measurement duration in seconds. Typically, but not always, the timer will generate a gate pulse for the duration of the measurement.
- *preset count* - In this mode, the measurement continues until the specified MX scaler acquires a prespecified number of counts. In general, the scaler will generate a gate pulse until the specified number of counts have occurred.
- *preset pulse period* - Some multichannel scaler devices can be driven by an external pulse generator. Multichannel scaler scans that are driven by an external pulse generator will use this kind of measurement.

Several scan classes are currently available, namely, *motor scans*, *list scans*, *multichannel scaler scans*, and *XAFS scans*. These scan classes differ in terms of the details of how the scan progresses. The details of each scan type are described below.

2.1 Linear Scans

An MX *linear scan* takes a sequence of measurements that are *equally spaced* in terms of some independent variable. There are several different types of linear scans.

2.1.1 Motor scans

For *motor scans*, the independent variables are the positions of one or more motors. As an example, suppose you wanted to measure X-ray absorption as a function of X-ray energy over the range from 8000 eV to 9000 eV in 100 eV steps. For this case, you would create an MX motor scan with a start position of 8000 eV with a step size of 100 eV and a total of 11 scan steps.

Another example would be using a temperature controller to change a sample's temperature from 290 K to 310 K in 1 K steps. MX typically supports temperature controllers by treating them as if they were motor controllers, so for this case you would set up an MX motor scan with a start position of 290 K, a step size of 1 K, and with 21 measurement steps.

MX motor scans can also be *multidimensional*. Suppose that you had a large sample attached to an X-Y translation stage, and that you wanted to scan this sample over an area that was 5 cm wide by 10 cm high. If the zeros of X and Y were set at the center of the sample, then you could run a two-dimensional MX scan with the X motor scanned from -2.5 cm to +2.5 cm and the Y motor scanned from -5 cm to +5cm with whatever step size you chose.

In a multidimensional scan, the motor listed first changes position least often while the motor listed last changes position most often. Using the X-Y scan above as an example, if the X step size was 0.5 cm, then the scan would start by moving the motors to X = -2.5 cm and Y = -5 cm. The first linear slice of the scan would be performed with X fixed at -2.5 cm, while Y would step scan from -5 to +5 cm. After the first slice was completed, MX would then move the motors to X = -2.0 cm and Y = -5 cm. Then the second slice of the scan would be performed with X fixed at -2.0 cm and Y step scanning from -5 cm to +5 cm. Subsequent slices would proceed in a similar fashion until the last slice was executed with X = +2.5 cm and Y step scanning from -5 cm to +5 cm.

2.1.2 Pseudomotor scans

MX defines the concept of a *pseudomotor* as a software object that behaves like a motor object, but which are not actual motors. Pseudomotors are described in more detail in the **Motors** chapter, so we will only discuss here their interaction with scans.

An ordinary motor scan can scan both pseudomotors and real motors. The only real difference between a pseudomotor scan and a normal motor scan is that pseudomotor scans make an extra effort to ensure that the pseudomotors are used in a sane manner. In general, this "extra effort" only makes a difference if the position of a pseudomotor depends on the position of more than one real motor.

Example: Scanning slit pseudomotors

Slit pseudomotors are a typical example of a pseudomotor that depends on more than one real motor. An X-ray slit mechanism is used to block the transmission of X-rays everywhere except between the positions of the two slit blades. Such slit blades are typically made of tungsten, since tungsten strongly attenuates X-rays. Essentially, the slit acts as a single point collimator.

FIXME: Show a picture of an X-ray slit here that matches the description below.

Figure *???* shows a pair of slit blades that define the opening for X-rays in one dimension. The metal slit blade on the left marked "L" prevents X-rays from passing through further to the left than the position marked x_{min}, while the slit blade on the right marked "R" prevents X-rays from passing through further to the right than the position marked x_{max}. In general, experimenters want to use an X-ray beam aperture that collimates the beam in both the x and y directions. This is done by adding another pair of slit blades in a horizontal orientation instead of the vertical orientation of the first two slit blades.

From a hardware point of view, the simplest way of implementing a slit mechanism is to provide two independently movable blades for the x direction and another set of two independently movable blades for the y direction. However, experimenters typically do not think in terms of the position of the edges of the individual blades. Instead, they generally think in terms of the width (or height) of the slit opening and the position of the center of the slit opening. Since this is not what the hardware provides, we use pseudomotors to provide the interface that the experimenters expect.

Normally, there will be two pseudomotors that are built out of the two real motors. For the x direction, this will be

$$x_{center} = x_{right} - x_{left}, \quad x_{width} = 0.5 * (x_{right} + x_{left})$$

Similarly for the y direction, the definitions will be

$$y_{center} = y_{top} - y_{bottom}, \quad y_{height} = 0.5 * (y_{top} + y_{bottom})$$

Suppose that we setup and run a motor scan that attempts to step scan the x_{center} pseudomotor while holding the value of x_{width} fixed. Unfortunately, motor scans can only handle this situation correctly if the motor controllers behave in a perfect fashion with infinite resolution. However, real motor controllers have neither of these features.

To make the example more concrete, suppose that we attempt to step scan the pseudomotor x_{center} from -50 mm to +50 mm with a step size of 10 mm. Furthermore, suppose that the x_{width} pseudomotor has a value of 5 mm before the start of the scan. This means that when the scan starts, MX will attempt to move the two real motors, x_{left} and x_{right}, to the starting positions of -52.5 mm and -47.5 mm respectively. As we continue the scan, the x_{left} and x_{right} pseudomotors ideally will go to the following positions:

```
x_left    x_right
-52.5     -47.5
-42.5     -37.5
-32.5     -27.5
-22.5     -17.5
-12.5      -7.5
 -2.5       2.5
  7.5      12.5
 17.5      22.5
 27.5      32.5
 37.5      42.5
 47.5      52.5
```

This is in the perfect environment.

In a real environment, stepping motors can lose steps during motion and DC servo motors can end their moves at positions slightly different than the requested position. Take a look at the move from the first step of the scan (-52.5, -47.5) to the second step of the scan (-42.5, -37.5). Suppose that x_{right} loses a few motor steps in its motion to the position $x_{right} = -37.5$ due to imperfections in the motor or its tuning, so that the motor actually ends up at the position $x_{right} = -37.75$. This means that the slit pseudomotors with the incorrect positions of $x_{center} = -40.125$ and $x_{width} = 4.75$. Errors of this sort are never desirable, but under certain circumstances one has to put up with them.

The ***real*** problem is what happens during the move from the second step scan position to the third step scan position. In general, pseudomotors such as the slit pseudomotors do not maintain an elaborate history of all the positions that they were ever at. Instead, all they know are the current positions of the real motors that they depend on.

So think about what would happen during the move from the second step position to the third step position. The MX motor scan knows that the next step is supposed to move the x_{center} pseudomotor in the positive direction by 10 mm. What the motor scan does ***not*** know is the current position of the x_{center} pseudomotor. Instead, it asks the *slit_center* pseudomotor driver for the current position of x_{center} and gets back the answer $-40.125mm$. The motor scan cheerfully believes this and then commands a move of x_{center} to $-30.125mm$.

The textitslit_center driver looks at the request to move the pseudomotor by $10mm$ and interprets that as a request to move the real motors each by $10mm$. Thus, x_{left} is commanded to move to $-32.5mm$ while x_{right} is commanded to move to $-27.75mm$. Remember that on this step, the x_{right} motor should have been commanded to move to $-27.5mm$, but it was actually commanded to move to $-27.75mm$. If the same kind of error occurs in the move from the second step to the third step, then x_{right} may actually end up at $-28.0mm$. If we propagate this error, through the rest of the scan, then after the last step x_{left} is at the correct position of $47.5mm$, but x_{right} is at the significantly incorrect position of $50.0mm$ instead of $52.5mm$.

Such a result is obviously unacceptable. If the scan started with $x_{width} = 5mm$, then the scan ended with $x_{width} = 2.5mm$. If the slit blades are collimating a uniform beam, then this error will result in a 50 percent reduction in beam intensity, which cannot be tolerated.

This kind of error was the inspiration for pseudomotor scans. Pseudomotor scans and ordinary motor scans behave the same for real motors. But for certain types of pseudomotors, the pseudomotor scan records the starting position for each real motor. Then for each subsequent step of the step scan the destinations for the real motors are computed by looking at the starting positions of each real pseudomotor and then adding the value $step\_size * number\_of\_steps$ to each of those positions. Thus, when the example step scan above attempts to move the real motors to the positions for the third step in the scan, it will attempt to move x_{right} to the *correct* position of $-27.5mm$ instead of the incorrect position of $-27.75mm$.

The changes necessary to support this special pseudomotor scan behavior are currently (June 2008) only implemented for the *network_motor*, *slit_motor*, and *translation_mtr* drivers. In general, this kind of modification is only useful for pseudomotors that have internal dependencies on the relative positions of the real motors that are not obvious from the reported position of the pseudomotor. For example, this feature is totally irrelevant for a pseudomotor such as the *energy_motor* driver. An *energy_motor* pseudomotor only depends on one real motor, so there is no internal state about the relative positions of real motors that has to be preserved. On the other hand, this feature is only implementable for pseudomotors whose positions depend in a simple way on the positions of the real motors. An example of a pseudomotor driver that would have this problem is the *table_motor* driver. A *table_motor* pseudomotor reports the x, y, z, roll, pitch, or yaw positions for a table such as a diffractometer table. In general, rotation matrices must be used to compute the pseudomotor positions from the real motors positions for a diffractometer table, which means that the *table_motor* driver is not a good candidate for support by pseudomotor scans.

2.1.3 Slit Scans

Slit scans were the ancestor of pseudomotor scans. A pseudomotor scan can do everything a slit scan could do and can also handle translation pseudomotors, as well as forwarding such requests to remote MX servers. Some future version of MX may automatically replace all existing slit scans with pseudomotor scans.

2.1.4 Input Scans

Input scans are 1-dimensional scans that do not use a motor as the independent variable. Instead, the independent variable is the measurement count. This gives the experimenter a convenient way to run a scan that monitors a set of MX input devices without requiring the use of a motor. An alternate way of getting similar functionality is to run a motor scan using a motor controlled by any one of the *soft_motor*, *disabled_motor*, or *elapsed_time* motor drivers.

2.1.5 Theta-2 Theta Scans

For this type of scan, the experimenter specifies the names of two motors. Although two motors are specified, this is really a 1-dimensional scan in which the second motor is always moved to a position that is twice as large as the

position of the first motor. For example, if the user specified the names of two motors, *alpha* and *beta*, then the scan range woult be specified in terms of the positions of the motor named *alpha*. During this can, the motor *beta* would always be move to a position with a numerical value that was always twice that of the position of *alpha*. Obviously, for this kind of scan to do the right thing, the zeros of the *alpha* and *beta* motors must be set to appropriate values such that *alpha* and *beta* are lined up with each other when they are both at positions with a numerical value of 0.

An alternate way of getting similar functionality is to use the *theta_2theta* pseudomotor driver. This driver has the advantage that it can be used seamlessly as part of an ordinary motor scan. However, one advantage of the scan is that the user can specify the names of the two motors to use without having to change the MX motor database.

2.2 List Scans

If the scan you want to perform has motor positions that are not evenly spaced or measurement times that are not all the same, then you may want to use an MX *list scan*. An MX list scan does not compute the needed motor positions itself. Instead, it fetches the requested measurement time and motor positions from some external source. At present, the only supported external source is a user-specified file, but it would be easy to add support for reading the positions from somewhere else such as an in-memory array.

2.2.1 File List Scans

For a file list scan, the user is expected to provide the name of the timing source, the names of the scan motors, and the name of a file containing the requested timing information and the motor positions. List scans support all of the standard measurement types, namely, *preset time*, *preset count*, and *preset pulse period* (for pulse generators). The file should contain columns of numbers with the first column corresponding to the measurement interval and the rest of the columns corresponding to the different motors.

As an example, suppose we want to scan the motors *energy*, *x*, and *y* along some path and the measurement times at the different scan positions are not all the same. Then, we can specify the time and the positions using an external file such as the following:

```
0.5   8900  4.3  3.1
0.6   8908  4.7  3.2
1.4   8940  5.1  2.9
0.7   8932  5.8  2.4
```

The list scan, when run, would then perform the following measurements:

- A 0.5 second measurement at *energy* = 8900 eV, *x* = 4.3 mm, and *y* = 3.1 mm.
- A 0.6 second measurement at *energy* = 8908 eV, *x* = 4.7 mm, and *y* = 3.2 mm.
- A 1.4 second measurement at *energy* = 8940 eV, *x* = 5.1 mm, and *y* = 2.9 mm.
- A 0.7 second measurement at *energy* = 8932 eV, *x* = 5.8 mm, and *y* = 2.4 mm.

In general, it is better that motors not change direction during a scan because of backlash, but the list scan does not require this as you can see above.

2.3 Quick Scans

A disadvantage of linear and list scans is that between each step of the measurement, the motors have to accelerate up to speed and then decelerate back down to zero before taking the next data point. The acceleration and deceleration can actually end up taking a significant amount of time, so some effort has been put into finding a way around this. One solution is to acquire data without stopping the motors. For example, suppose you command a rotary stage to move at 1 degree/second and then tell your measurement equipment to acquire 0.5 second measurements. Then all of your measurements will be averaged over 0.5 degree regions. In general, it is best for the dead time between measurements to be as short as possible (ideally zero).

Quick scans are also known as fast scans or slew scans in other control systems. In retrospect, the name *slew scan* would have been more appropriate, since many "quick" scans actually run the motor at a very slow speed.

At present, there are two kinds of quick scans available:

- Multichannel scaler (MCS) quick scans - These scans use an MX multichannel scaler device to acquire measurements over equally spaced intervals.
- APS insertion device quick scans - This kind of scan is only useable at the Advanced Photon Source. It attempts to synchronize the motion of an X-ray monochromator with a continuous change to the X-ray energy of a wiggler or undulator device. In practice, the APS insertion device scans have been little used, since the minimum speed of the undulator motors is too high for many experiments.

For the rest of this discussion, we will focus on multichannel scaler (MCS) scans, since most quick scans are of that type.

A multichannel scaler is a device that contains multiple scaler channels and which takes a sequence of measurements without stopping. Ideally, the sequence timing is handled entirely by the multichannel scaler itself, so that one can be sure of getting deterministic timing for the measurement sequence. A commonly used multichannel scaler is the Struck SIS-3801 which contains 32 input scaler channels. It can be commanded to take a series of measurements using either an internal clock or a sequence of external triggers.

An MCS quick scan typically looks like this:

- Move the motor to the start position of the scan.
- Start the multichannel scaler.
- Command the motor to move to the end position of the scan.
- Wait for the measurement sequence to complete.
- Read out the measured data points from the multichannel scaler.

2.3.1 Recording Motor Positions

In general, the multichannel scaler is completely capable of handling the acquisition of periodic scaler measurements. However it is also necessary to correlate these measurements with the corresponding motor positions. The strategy we have chosen in MX is to convert motor position measurements into something that can be recorded by a multichannel scaler.

In MX, we have defined a device class we call a "multichannel encoder (MCE)" whose job is to record the incremental position change of the motor for each measurement by the multichannel scaler. If you move your motors to the start position, start the multichannel scaler, and then command the motors to move, you then have a complete record of the motor position by using the incremental position changes as an offset to the original start position.

Typically, we construct multichannel encoders by arranging for the motor controller to generate an output signal that depends on the motor position. One way is to use a quadrature encoder signal output, if available. Another way (for motor controllers that support slaving) is to slave a second axis to the moving axis and then programming the slave axis to generate step and direction output signals. Either way, the next step is to add a bit of electronics that converts the quadrature signal or the step and direction signal into a pair of pulse trains typically called "clockwise" and "counter-clockwise". As you can guess, clockwise pulses appear when the motor is moving in one direction, while counter-clockwise pulses appear when the motor is moving in the other direction. These clockwise and counter-clockwise pulse trains are designed to be easily recordable in a pair of multichannel scaler channels. If you then start the multichannel scaler before you start the motor, you can compute the incremental position shift of the motor for each MCS measurement by simply taking the difference between the number of clockwise and counter-clockwise pulses for each measurement.

Some motor controllers, such as the Delta Tau PMAC, allow you to slave any motor axis to any other motor axis. For such controllers, if you dedicate one motor axis to generating the output for the multichannel encoder, you can then quick scan all of the other axes in the controller by arranging to make the MCE axis be a slave to the scan motor axis before starting the scan.

2.3.2 Quick Scanning Pseudomotors

... TBD ...

2.4 XAFS Scans

XAFS scans are a specialized kind of motor scan used to examine the behavior of the absorption of X-rays by a material near one of the absorption edges of the material. They are generally only of interest to experimenters doing X-ray absorption or X-ray fluorescence experiments.

An X-ray absorption edge is a fairly sharp discontinuity in the absorption of X-rays by the material. Absorption edges are a consequence of quantum mechanics and are due to the fact that electrons in an atom are arranged in shells. For a given shell of electrons, if an X-ray comes into the atom with slightly more energy than the absorption edge energy, then an electron can be ejected from the atom. If the X-ray has slightly less energy than the absorption edge energy, then no electron will be ejected. Each atomic element has its own characteristic absorption edge energies and one way to measure the atomic composition of a material is to look for the presence of absorption edges. The absorption spectrum shows some amount of oscillation just above the edge which is collectively called X-ray absorption fine structure (XAFS).

2.4.1 XAFS Scan Regions

In MX, XAFS scans consist of a series of regions that are contiguous with each other. If we define E_0 as the absorption edge energy, then regions that are below that energy, or which overlap that energy are specified in terms of the difference between the X-ray energy and the absorption edge energy. This quantity is typically called $E - E_0$ and is represented in the MX database by a pseudomotor called `e_minus_e0`. Energies above the absorption energies are typically specified in terms of the quantum mechanical wavenumber of the ejected electron. The wavenumber k is defined by the equation

$$E = E_0 + \frac{\hbar^2 k^2}{2m_e}$$

where E is the energy of the incoming photon, E_0 is the absorption edge energy, $\hbar$ is Planck's constant h divided by 2π, and m_e is the mass of the electron. The k value is represented in the MX database by a pseudomotor that is also called `k`.

A typical XAFS scan will contain a few scan regions before the absorption edge and a few after the edge. Here is an example scan:

- `e_minus_e0` region 1
  Scan from $E - E_0$ = -1500 eV to -100 eV in 200 eV steps with a 0.1 second measurement time.
- `e_minus_e0` region 2
  Scan from $E - E_0$ = -100 eV to -20 eV in 5 eV steps with a 2 second measurement time.
- `e_minus_e0` region 3
  Scan from $E - E_0$ = -20 eV to 15 eV in 0.3 eV steps with a 4 second measurement time.
- `e_minus_e0` region 4
  Scan from $E - E_0$ = 15 eV to 30 eV in 0.5 eV steps with a 6 second measurement time.
- `k` region 1
  Scan from $E - E_0$ = 30 eV to k = 8 Å^{-1} in 0.07 Å^{-1} steps with a 8 second measurement time.
- `k` region 2
  Scan from k = 8 Å^{-1} to k = 12 Å^{-1} in 0.07 Å^{-1} steps with a 10 second measurement time.

The choice of regions depends on the type of experiment being performed and is typically chosen to make most effective use of the (typically limited) available X-ray time.

2.4.2 XAFS Scan-related Database Records

XAFS scans expect the presence of a number of MX database records with specific names and types.

Required Records

- `edge_energy` - This is an MX database variable that is expected to contain the value of the X-ray absorption edge energy expressed in electron volts (eV).
- `e_minus_e0` - This is a pseudomotor record that is expected to control the offset between the current energy of the X-ray beam and the X-ray absorption edge energy. It is normally implemented using a *delta_motor* pseudomotor.
- `k` - This is a pseudomotor record of type *xafs_wavenumber* which expresses the wavenumber of the ejected electron in inverse Angstroms. The driver for this pseudomotor **assumes** the presence of the database variable `edge_energy` without actually mentioning the variable in its database record.

Here is an example database that demonstrates the use of these records using an *epics_motor* record for the θ axis.

```
edge_energy variable inline double "" "" 1 1 8980.0
d_spacing   variable inline double "" "" 1 1 3.13555
theta       device motor epics_motor "" "" 0 0 -100000 100000 0 -1 -1 0.001 0 deg wml:m1
energy      device motor energy_motor "" "" 0 0 0 1e+08 0 -1 -1 1 0 eV theta d_spacing 0.0
e_minus_e0  device motor delta_motor "" "" 0 0 -1e+08 1e+08 0 -1 -1 1 0 eV energy edge_ene
k           device motor xafs_wavenumber "" "" 0 0 0 1e+08 0 -1 -1 1 0 A-1 energy
```

The `theta`, `energy`, and `d_spacing` records are not required to have the names shown above, but those names are conventional.

Note: Some future version of MX may make the required names configurable. However, this is only likely to happen if some future beamline has an experiment that needs to simultaneously use multiple edge energies.

Required for *XAFS* Format Datafiles

If you save the data from your XAFS scan using MRCAT's *xafs* datafile format, then you need several more records in your MX database which are used to generate the *xafs* format header for the data files. Here is an example database of that:

```
beamline_name variable inline string "" "" 1 20 "APS Sector 10-ID"
ring_energy variable inline double "" "" 1 1 7
#
amplifier_list variable inline string "" "" 2 3 17 keithley1 keithley2 keithley3
keithley1      device amplifier soft_amplifier "" "" 10000000 0 0
keithley2      device amplifier soft_amplifier "" "" 10000000 0 0
keithley3      device amplifier soft_amplifier "" "" 10000000 0 0
#
xafs_header1   variable inline string "" "" 1 81 "Header line 1"
xafs_header2   variable inline string "" "" 1 81 "Header line 2"
xafs_header3   variable inline string "" "" 1 81 "Header line 3"
```

The XAFS header file variables are described in the datafile format section of this document that follows.

2.5 Scan Datafile Formats

2.5.1 *Text* Format

2.5.2 *SFF* Format

2.5.3 *XAFS* Format