UPPSALA
UNIVERSITET

UPTEC Q16 024
Examensarbete 30 hp
Maj 2016

Programming a TEM for magnetic measurements

DMscript code for acquiring EMCD data in a single scan with a q-E STEM setup

Linus Schönström

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Programming a TEM for magnetic measurements

Linus Schönström

Code written in the DigitalMicrograph® scripting language enables a new experimental design for acquiring the magnetic dichroism in EELS. Called the q-E STEM setup, it provides simultaneous acquisition of the dichroic pairs of spectra (eliminating major error sources) while preserving the real-space resolution of STEM. This gives the setup great potential for real-space maps of magnetic momenta which can be instrumental in furthering the understanding of e.g. interfacial magnetic effects. The report includes a thorough presentation of the created acquisition routine, a detailed outline of future work and a fast introduction to the DMscript language.

Handledare: Thomas Thersleff
Ämnesgranskare: Klaus Leifer
Examinator: Åsa Kassman
ISSN: 1401-5773, UPTEC Q16 024

Kan vi se var magnetismen kommer ifrån?

Linus Schönström

Magnetiska material är ett viktigt forskningsområde. Tillämpningarna är bland annat vindkraft, elbilar, hårddiskar och själva elnätet. Utvecklingspotentialen för permanentmagneter är stor, och på två fronter: de kan enligt teorin bli betydligt starkare än de är och de kan tillverkas av mer hållbara råvaror.

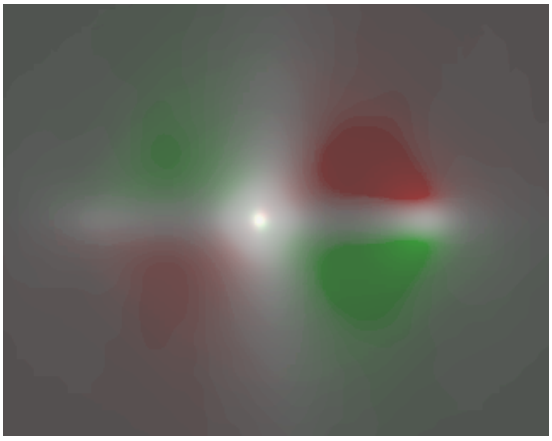
Med de analystekniker som magnetforskare har tillgång till idag är det mycket svårt att svara på var, i en kristallstruktur eller i en nanopartikel, som magnetiska effekter uppstår. Detta gör forskning på magnetiska nanopartiklar och multilagerstrukturer extra svårt.

För atomär upplösning är materialforskningens främsta verktyg TEM (transmissionselektronmikoskop). Och TEM tillhandahåller en spektroskopiteknik kallad EELS (Electron Energy Loss Spectroscopy), som genom analys av spektrum tagna från specifika områden i det reciproka rummet kan ge information om provets magnetisering — förhoppningsvis med ett TEMs atomära upplösning!

Det första experimentella beviset på att tekniken fungerar kom 2006, och tekniken gavs sitt namn EMCD (som elektronernas motsvarighet till XMCD, X-ray Magnetic Circular Dichroism)[1]. Därefter följde mycket utveckling av både teori och teknik i ett par år. Det har föreslagits flera sätt att utföra experimentet, till stor del för att det inkluderar fem dimensioner och kameran som används har ju bara två.

I examensarbetet Programming a TEM for magnetic measurements — DMscript code for acquiring EMCD data in a single scan with a q-E STEM setup visas utvecklingsarbetet med en mjukvara för att göra detta på ett nytt sätt: de två viktigaste dimensionerna projiceras på kameran samtidigt, medan elektronstrålen sveps över provet för de två spatiella dimensionerna.

[1] Trots att XMCD fått sitt namn utifrån att den tekniken baseras på cirkulär polarisation, som i EMCD är ersatt av elektronernas spridning — detta visas i den allra första artikeln, publicerad av Hébert och Schattschneider 2003.



Figur 1: Simulerad bild av var i reciproka rummet som den magnetiska effekten på EELS-spektrumet kan ses. Röda och gröna områden är påverkade med motsatt tecken, EMCD-signalen fås som skillnaden mellan dem. Notera att bara en dimension av dessa två (q_x, q_y) behövs för att urskilja de två delarna som skapar signalen. Bilden är skapad av data från Ján Rusz.

Denna metod kan kanske inte ge atomär upplösning, men den bästa upplösningen hittills för EMCD, och den eliminerar flera viktiga felkällor och svagheter i de metoder som har presenterats tidigare. Arbetet fortgår: mjukvaran är inte klar, och den är tänkt att kombineras med en specialanpassad bländare i mikroskopet.

Contents

Table of abbreviations

TEM	Transmission Electron Microscopy
STEM	Scanning TEM
EFTEM	Energy-Filtered TEM
EFDIF	Energy-Filtered Diffraction
EELS	Electron Energy-Loss Spectroscopy
XMCD	X-ray Magnetic Circular Dichroism
EMCD	See section 1.3.1
GIF	Gatan Imaging Filter
CCD	Charge-Coupled Device
GMS	Gatan Microscopy Suite®, software suite
DM	DigitalMicrograph®, main executable of GMS
SI	Spectrum Imaging, a component of GMS
GUI	Graphical User Interface
CLI	Command-Line Interface

1 Introduction

1.1 Magnetic information in the TEM

In research on materials, many of the questions asked find their answers on the atomic scale. The transmission electron microscope (TEM) has become ubiquitous in this research. It is a powerful tool, providing information on many of the micro- and nano-scale properties and processes that gives rise to the material properties we use. But one important property is missing: magnetism.

Magnetic materials are extremely important for society. They play several crucial roles in our all-encompassing use of electricity, and some of the rare earth elements used have problematic supply chains. Research on magnetic materials is sometimes hindered by the fact that most techniques for characterising magnetism are bulk measurements, and no readily available technique has the nano-scale resolution to show for example what happens at an interface.

Given this situation, it is clear that enabling the TEM to give information on magnetic properties and processes — if at all possible — is set to be an important step.

1.2 This project: enabling the single pass STEM EMCD

In this project, an acquisition routine that utilizes a new experimental setup called q-E STEM is developed. The goal of this setup is to utilize the high spatial resolution and the multiple signal acquisition possibilities of scanning TEM (STEM) while still acquiring related pairs of spectra simultaneously, eliminating artifacts from non-magnetic effects on the white-line ratio.

The work presented in this report is mainly initial code work, which proved more involved than anticipated and still needs to be expanded upon. A first set of data was acquired but has not yet been analyzed.

1.3 Background

1.3.1 Initial work on EMCD

In 2003, an article by Hébert and Schattschneider proposed that an experiment analogous to X-ray magnetic circular dichroism (XMCD) should be possible with electron energy-loss spectroscopy (EELS) in the TEM. They arrived at this conclusion from a mathematical observation that equations analogous to those of XMCD can be formed for electron scattering. The left- and righthand circular polarization of X-rays used in XMCD is substituted by specific values of the momentum transfer $\hbar\vec{q}$, that is, specific scattering conditions (C. Hébert and P. Schattschneider 2003).

This was backed up with an experimental demonstration of the effect three years later (P. Schattschneider et al. 2006). This letter also establishes the name EMCD for this technique, which has later been written out in many, slightly different, ways. It should be noted that while this technique has many similarities with XMCD, the probe is no longer circularly polarized[1]. The name is best taken as derived from analogy with XMCD, not as a meaningful abbreviation[2].

In 2007 two articles, published at the same time, demonstrated that the sum rules for EMCD work out to (among other things) a relation between the acquired spectra and the ratio between orbital and spin magnetic moments — again analogous to XMCD (Calmels et al. 2007; Rusz et al. 2007). This is a very important step, as the quantitative measurement of m_l/m_s is the desired application.

In 2008 a major differentiator from XMCD, other than requiring a TEM rather than a synchrotron, was published: a spatial resolution on the scale of 2 nanometers (Peter Schattschneider, Michael Stöger-Pollach, et al. 2008). This was achieved with a scanning TEM (STEM) setup, acquiring one spectrum for each real-space point, and two passes — one for the + and −

[1] One could argue that EMCD measures outgoing electron beams that are *roughly* circularly polarized, or an analogy thereof, but the original electron beam is disordered.

[2] A more descriptive name would be "magnetic information from angle-resolved EELS", which could be abbreviated as MIFARE. Or simply "magnetic EELS" (not to be confused with electric eels).

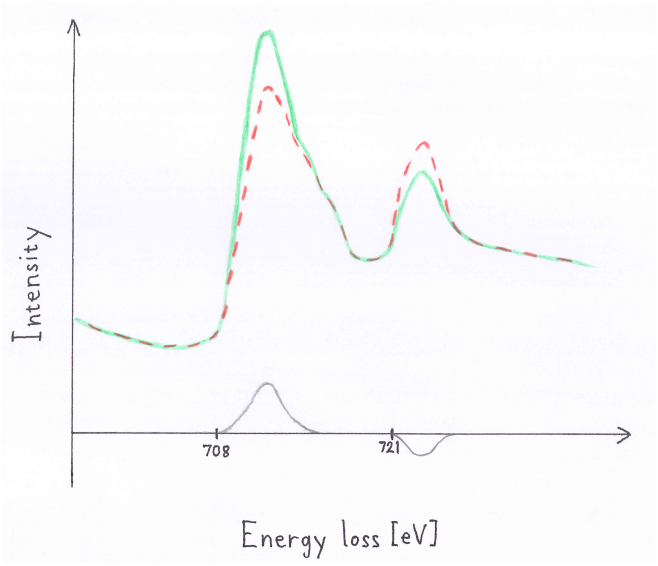


Figure 1: The spectra of EMCD: one + (green), one − (red, dashed) and the difference between them — the EMCD signal (pencil). Note that the signal is very low in intensity and this *illustration* shows none of the normally high noise. The L3 peak starts at 708 eV and L2 starts at 721 eV with roughly half the intensity, in iron; the effect is present in other magnetic transition metals as well.

spectra respectively. This setup enables the second part of the desired application: mapping, with high spatial resolution. But the low signal to noise ratio and the error sources of serial + and − acquisition means a reliable quantitative result is very hard to achieve.

1.3.2 Different experimental setups

EMCD (and XMCD) is, essentially, the analysis of a magnetic effect on the white-line ratio. The spectroscopic term white-line ratio refers to the relative intensities of the L3 and L2 edges in the spectrum. Ideally, two spectra are acquired so that the magnetic effect has changed sign, whereby it can be isolated as the difference between them. This difference is called the EMCD signal. The two spectra will be labeled the + and − spectra in this report. See figure 1.

In EMCD, the + and − spectra are acquired from different parts of reciprocal space, whereby a total of five dimensions are part of the experiment:

1. Energy E, the dimension of a spectrum. The feature of interest is the white-line ratio, which could be reduced to a scalar, but due to the large background in EELS it needs to be measured in full and reduced in post-processing.

2. Reciprocal space (q_x, q_y) — a two-dimensional space in which the feature of interest for this experiment is best described by (Lidbaum, Rusz, Liebig, et al. 2009). A simple view of the regions where the signal can be found is shown by figure 2. From parts of these regions the + and − spectra are created.

3. Real space (x, y), in which a good resolution and good correlation to other measurements (such as normal imaging) is desired.

There are several different experimental setups, most of which are presented by (Rubino et al. 2012). These make different compromises about which of these dimensions to reduce, which to scan, and which to acquire in parallel.

Reducing a large part or all of real space into one signal is a popular and effective approach to get a good signal-to-noise ratio, but the sacrifice of real-space resolution is of course to not utilize the inherent strength of the TEM. When using scanning TEM (STEM) this tradeoff can be balanced in post-processing (as real space is scanned, data that is spread in real space can be summed as desired).

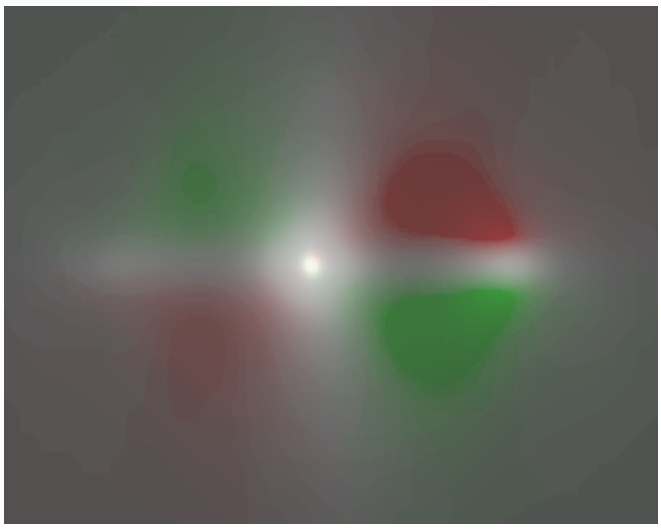


Figure 2: Simulated view of where in reciprocal space the magnetic influence on the EELS spectrum can be seen. Red and green is magnetically affected signal, white is signal without magnetic effect in it and the background grey means no signal at all. The right diffraction spot is overly diffuse. Note the asymmetry: this means a signal remains when the q_x dimension is collapsed. For other setups, a symmetrical case can also be used. Figure created from data provided by Ján Rusz.

One major concern is that when data for the + and − spectra are acquired in a serial manner non-magnetic changes in the white-line ratio can corrupt the EMCD signal. The most important error sources would be beam damage, oxidation and sample drift. Sample drift could mean the paired-up spectra are from slightly different regions of the sample, which could mean vastly different chemistry, but can to some extent be corrected for. Beam damage and oxidation cannot be corrected for. All this presents obstacles for obtaining quantitative values of m_l/m_s.

1.3.3 Towards the goal: quantitative real-space maps

Applying the sum rules to experimental data to obtain quantitative values of m_l/m_s was done by (Lidbaum, Rusz, Liebig, et al. 2009). This experiment used a setup where an energy-filtered diffraction pattern (q_x, q_y) is projected on the camera (EFDIF), scanning over E. An important step for the development of the technique: the data reveals the optimal parts of reciprocal space to acquire.

In an attempt to obtain real-space maps of m_l/m_s another energy-scanning setup was used, where (x, y) was projected on the camera as in energy-filtered TEM (EFTEM) imaging (Lidbaum, Rusz, Rubino, et al. 2010). This implies separate acquisition of + and − which is quite likely the culprit in the result that the obtained EMCD signal maps could not be translated into quantitative values of m_l/m_s, while the setup mentioned in the above paragraph were again shown to give quantitative results.

The EFTEM setup was tested on magnetotactic bacteria (M. Stöger-Pollach et al. 2011). Again it showed a raw EMCD signal with good spatial resolution, but no attempt was made at applying the sum rules. The STEM setup has been applied to research on magnetic nanoparticles (Salafranca et al. 2012). STEM was also shown to give quantitative values of m_l/m_s by trading in the real-space resolution for a good signal-to-noise ratio (Thersleff et al. 2015).

But so far no setup with real-space resolution has acquired the + and − spectra simultaneously.

1.3.4 Single pass STEM EMCD

In (Peter Schattschneider, Cécile Hébert, et al. 2008) a setup with many obstacles and many possibilities is presented. The sample can be oriented so that the projection onto the camera

becomes (q_y, E), with the q_x dimension collapsed, and this allows capturing the + and − spectra simultaneously. This was shown to work for parallell illumination, whereby all real-space information is lost, and for a setup with a convergent beam and the sample shifted in height so as to illuminate a larger area and create a diffraction pattern in the image plane. It is noted that this setup cannot easily be extended to scanning over several real-space positions, and is limited in spatial resolution.

From the realization that the (q_y, E) dimensions are present at the camera in normal STEM-EELS mode a setup that can achieve full real-space resolution while acquiring the + and − spectra simultaneously is easily envisioned. The first step towards using this setup is to acquire both dimensions on the camera while scanning the beam, something the standard acquisition software does not do[3].

This proposed q-E STEM setup, when compared to the original double-pass STEM setup, would solve all problems of potentially unrelated + and − spectra. It would also halve beam exposure, potentially a big benefit as beam damage can be substantial (since the low signal to noise ratio necessitates the use of an intense beam).

All setups mentioned here can be summarized with this table:

	Dimension acquisition			*Results*	
	Parallel	Scanned	Reduced	Real-space res.?	Quan. m_l/m_s values?
STEM setup	E	(x, y)	(q_x, q_y) (2 sets)	Yes	With obstacles
EFDIF setup	(q_x, q_y)	E	(x, y)	Impossible	Yes
EFTEM setup	(x, y)	E	(q_x, q_y) (2 sets)	Yes	With obstacles
Publ. q-E setup	(q_y, E)		q_x	Limited, if possible	Possible
Proposed: q-E STEM	(q_y, E)	(x, y)	q_x	Yes	Possible

2 Equipment

All work in this project was done with and for a single microscope, but the use of a widely adopted software system does make the code usable on a wide range of microscopes. See section 4 for details on the requirements. For a detailed list of software and hardware in the four different GMS installations used in development of the code, see appendix A.

2.1 Hardware

The microscope used in this project is a FEI Tecnai F30 ST situated in the cleanroom facility at the Ångström laboratory of Uppsala University. It is equipped with a Gatan Tridiem "GIF", a post-column system of energy filter and camera for electron energy loss spectroscopy (EELS) as well as energy-filtered imaging (EFTEM). The microscope also has Gatan DigiScan STEM capabilities, thereby allowing for spectrum imaging.

The system can achieve a spatial resolution of 1.3 Å and energy resolution of 0.7 eV when doing STEM-EELS.

2.2 Software

The software used for data acquisition as well as consolidated control of almost all subsystems in the TEM is the Gatan Microscopy Suite (GMS). It is directly connected to the GIF, but also connects to the FEI control software as well as an EDS system. The main application of this suite is called DigitalMicrograph (DM). Crucially, for this project, this software system exposes an extensive set of programmability through a language called DigitalMicrograph scripting (DMscript).

[3]The second step is to implement an aperture setup that allows avoiding unwanted parts of q_x. Thersleff and Leifer are working on that.

2.2.1 The Spectrum Imaging hookup system

The Spectrum Imaging plugin (SI) of GMS provides the software needed for acquiring spectroscopic data for a scanned region of the sample, provided the hardware needed (DigiScan and an EELS or EDS system) is available. While this plugin cannot acquire 2D data for each point, as needed for this EMCD project, it does provide a possibility through its hookup system: SI can be instructed to run a custom DMscript function at every point of the complete acquisition.

It should be noted that Gatan provides another plugin called Diffraction Imaging, which should be able to acquire 2D data for a scanned region of the sample — not only in diffraction mode but also in EELS mode: exactly what is needed. At the time of this writing it is unclear to me how much of the created code would be useful in conjunction with that software.

2.2.2 The faux microscope

The full GMS software is intended to be installed with microscope hardware, this is called an "online" installation. An "offline" version of the software which eschews all features connected to hardware control and data acquisition is available[4] for viewing data, post processing and similar tasks. The offline version can be readily used for development testing of any code that does not need access to the hardware features, but this project is about data acquisition.

To facilitate development of code needing the hardware features, there is an undocumented and unsupported "faux mode" which enables the online version of the software to be run without actual microscope hardware. In essence, one needs to install the online version, set up a few things and use the parameter /FAUX when starting DM. More detailed instructions in appendix B.

2.2.3 MATLAB

The MATLAB software from MathWorks provides a coding environment in which many statistical and mathematical tools are readily available. For this reason the pre-study used a set of MATLAB code to analyze the EMCD dataset, and based on time constraints the decision was made to create the tools for extracting and analysing the data in MATLAB rather than porting the existing toolset to DMscript.

3 Introduction to the DMscript language

3.1 The basics of DMscript

In DMscript, you will find a different set of variable types depending on where you read about it.[5] The ones used in this code are four:

```
number
string
image
taggroup
```

Let us create two strings, append the second to the first, and print that to the Result pane of the output window as our Hello World program:

```
string str1, str2
str1= "Hello"
str2= " world!"
Result(str1 + str2 + "\n")
```

[4] Free of charge, even. Gatan's website is www.gatan.com.

[5] To be precise, it depends on how you view the inbuilt objects such as images and taggroups, and on whether or not you include the more obscure types: subareas, and complex and RGB variants of numbers, images and subareas.

Note that the language does not require a special character such as ; for ending each statement, a newline is taken as end of statement. If needed ; can be used to separate statements, while \ can be used to escape the newline in order to write very long statements on multiple lines.

The plus operator is overloaded (i.e. defined, likely in different ways, for different sets of arguments) not only to append strings but also to convert any number to a string if a string is one of the arguments. This is handy:

```
1 string str= "test "
2 number n= 2
3 Result(str + n + "\n")
4 // but confusion can arise
5 Result("test " + 3 + Pi() + "\n")
6 Result(4 + Pi() + " test\n")
```

will produce the following:

```
test 2
test 33.14159
7.14159 test
```

A lot of the DMscript environment is object-oriented. This is a programming paradigm which aims to make code more readable by creating variables called *objects*, which can be operated on by functions called *methods*. The first argument in a call to a method is the object being operated on, and this is accompanied by *dot notation*, where said argument is written as a dot-separated prefix to the method call. A good example is a function to replace text in a string, lines 2 and 3 here are equivalent:

```
1 string str= "String with text to be modified"
2 Replace(str,"to be","that is")
3 str.Replace("to be","that is")
```

Note that DMscript does not include a function to replace text in a string, but one is provided by the library in appendix D.2.4. A library is a collection of functions that can be installed into DM for use in other scripts.

The official documentation for the language is that which is included in the help files for GMS, accessed from DM by typing F1 or using the menu Help > Search..., then navigating to the topic Scripting. Other useful resources include a handbook of tutorials (Bernhard Schaffer 2015), a website (David Mitchell n.d.) and a database of example scripts (*The DM-Script database* n.d.). Note that there is a lot of outdated information on this subject, including some of the scripts in these three sources (especially in the database). For questions that remain, use the dm-script tag on stackoverflow.com (*Stack Overflow* n.d.), where new questions usually receive an answer the same day and often within hours.

3.2 Conventions used in this document

In this text, keywords will show in blue, strings in red and comments in green, as is the default colors in the script windows of DM. Note however that the included editor does not correctly recognise all keywords, notably `taggroup`, `for` and `void` — which this document will do.

I will write functions in CamelCase with a set of parentheses at the end, like so: `Result()`[6]. Variables will be named in all lowercase letters. The language is case agnostic, but due to the very long function names and dot notation inherited from other object-oriented languages this case convention is *highly* recommended to keep things readable.

I will try to use dot notation or normal function notation according to what is most readable, rather than trying to show what is and what isn't objects and methods.

For indentation this document will keep to what I gather is the convention in C and C++, so that functions and loops and other code blocks will look like this:

[6] To find out what arguments you need to put between the parentheses, just copy or write it without arguments — but including parentheses! — into an empty script window in DM (you get one by typing Ctrl+K or clicking File > New Script...) and run that (type Ctrl+Enter or click Execute). There will be an error message, and the Debug pane in the output window will show you the complete argument list, including all overloaded versions.

```
number ExampleFunction(number sumto){
    number i, sum
    for(i=1; i<=sumto; i++){
        /* here goes
        looped code */
        sum += i
    }
    return sum
}
```

Please note that the complete source code, found in appendix, does not completely follow any convention[7] other than variables being named in all lowercase. And most of it should not be seen as an example of good DMscript code.

3.3 Printing the contents of a TagGroup

A `taggroup` is a structure for grouping special variables called "tags". Tags have a label, a value and a type. Taggroups also exist in a variant called `TagList`, which groups the tags in an indexed manner. Both of these are fully nestable, including in each other, whereby one easily creates large tree structures for organizing tags. The available tag types are plentiful, and much more reminiscent of the underlying C++ environment than the more abstract types used by the literal variables of the language. A table that tries to list all the tag types can be found in appendix C.

The included way to view the contents of a taggroup in DM is a GUI browser that can be invoked with the function `TagGroupOpenBrowserWindow()`. This interface requires a lot of clicking to see the tags in the lower levels, so a more efficient way can be to use a script that recursively prints all of the tags to the output window. Writing such a script can also be a good introduction to working with this language.

3.3.1 Recursive functions, variable scope and passing as reference

An example script that recursively prints the contents of a taggroup is provided in the documentation, but that version has a few shortcomings. A major one is that it declares a variable outside the recursive function, to keep track of the current depth:

```
number depth= 0
void ExampleRecursiveFunction(number maxd){
    Result("\n Current depth: " + depth)
    if( depth < maxd ){
        depth++
        ExampleRecursiveFunction(maxd)
    }
    else
        return
}
ExampleRecursiveFunction(3)
Result("\nScript ended\n")
```

It needs to do this because variables created inside a code block, i.e. any pair of `{}`, are local to that block. They are said to have local scope. Variables declared directly in the script file have global scope, which means they are available everywhere in that script.

This means it is not a complete, self-contained function. And therefore it cannot be included in a library, it needs to be saved and run as a separate script file. The way around this problem is to have the function pass this variable to itself as a reference, so that all instances of the function access the same instance of the variable:

```
void ExampleRecursiveFunction2(number maxd, number &depth){
    Result("\n Current depth: " + depth)
```

[7] This is the result of writing code in an unfamiliar language, bringing habits from other languages with you.

```
    if( depth < maxd ){
        depth++
        ExampleRecursiveFunction2(maxd,depth)
    }
    else
        Exit(0)
}
number depth= 0
ExampleRecursiveFunction2(3,depth)
Result("\nScript ended\n")
```

The drawback here is that the function call needs to include a variable, a literal value will not work. This can be remedied by creating a wrapper function:

```
void ExampleWrapper(number maxd){
    number depth= 0
    ExampleRecursiveFunction2(maxd, depth)
}
```

By including this and the above definition of the recursive function, we have a library that can be installed so that the call `ExampleWrapper(3)` could be used from any future code. The wrapper function needs to be placed after the recursive function, as the language is interpreted line by line all functions need to be defined before they are called. (And no, declaring the functions separately before their definition as in C++ does not work. But you could create a class.)

Eagle-eyed readers will have noted a bug. In the first of the three examples above, the function uses the keyword `return` to return control to its caller, and when you run the complete example as a script it will work as expected, ending when the bottom of the file is reached. In the second example, the function `Exit()` is used to directly stop the script — and therefore the closing message is not printed, and the next message printed may end up on the same line! That is not good.

The third of the code snippets above show that functions with return type `void`, meaning they return nothing, can omit the keyword `return` altogether. Yes, that means the first two examples can be made two lines shorter.

3.3.2 Tag types

Another major shortcoming of the example script is that it gives no information at all when it encounters a tag that cannot be handled by the `TagGroupGetTagAsString()` function (which does convert values from most but not all tag types into readable strings). In a clever tag printing implementation, one can resort to printing the available type information:

```
string typestr
number typelength= tg.TagGroupGetTagTypeLength(tag_index)
number i
for( i = 0; i < typelength ; i++){
    typestr += tg.TagGroupGetTagType( tag_index, i ) + ":"
    typestr= typestr.Left(Len(typestr)-1)
    Result("Error, tag not extracted. Tag type "+typestr)
}
```

In an extra clever version, one can implement special messages for types that are known but not printable, or not a good idea to print (such as very long strings). A very useful piece of information for array tags (type1 = 20) is in the last one of the type numbers: it is the number of elements in the array. For a list of the various tag types see appendix C.

In developing code it may be useful to have an extra version that prints all the type information for all the tags, instead of their values or some text messages. My current function for this is available in appendix D.3.1.

3.3.3 Inbuilt taggroups, overloaded functions and tagpaths

With the above ideas, we can create a function that will thoroughly and clearly present the contents (to a desired depth) of a taggroup. But what taggroup? Of course, it can be any taggroup that was created in script, like:

```
taggroup tg= NewTagGroup()
tg.TagGroupSetTagAsNumber("First tag",1)
tg.TagGroupSetTagAsString("Second tag","Two")
PrintTagGroup(tg)
```

But there are also several important taggroups already in DM[8] such as the persistent taggroup containing the Global Tags which holds many settings and much information on the install. Scripts generally save any information they need to keep in here under a tag such as `"_Author:Script"`. There is also a user-specific persistent taggroup containing a few settings, and every image has a set of tags attached to it. These may be accessed like this:

```
PrintTagGroup(GetPersistentTagGroup())
PrintTagGroup(GetUserPersistentTagGroup())
GetFrontImage().ImageGetTagGroup().PrintTagGroup()
```

To create some shortcuts for this we can use *overloading*, defining extra versions of our function for different arguments. For example, to use the persistent taggroup as a default when no argument is passed and to access the image tags by simply passing an image instead of a taggroup:

```
void PrintTagGroup(taggroup tg){
    //Base version of the function goes here
}
void PrintTagGroup()
    PrintTagGroup(GetPersistentTagGroup())
void PrintTagGroup(image img)
    img.ImageGetTagGroup().PrintTagGroup()
```

Note that when a function consists of a single statement, like these very simple overloads, the surrounding `{}` are not needed. Syntactically a code block simply groups several statements into a single statement.

We will also often want to address a single subgroup in an existing taggroup, perhaps several steps down in the tree. In the inbuilt functions, a special string called a tagpath can be used interchangeably with tag labels. This string is simply the labels along the path, separated by colons. The user-specific persistent taggroup contains the following tags:

```
+ Private
:  + Defaults
:  :  + ScriptWindowSize
:  :  :  - Height =  199
:  :  :  - Width  =  634
+ Scripting
:  + CommentColor
:  :  - blue  =  0
:  :  - green =  128
:  :  - red   =  0
:  - fontName =  Courier New
:  - fontSize =  8
:  + KeywordColor
:  :  - blue  =  192
:  :  - green =  0
:  :  - red   =  0
:  - ShowLineNumbers =  true
```

[8] Or, for the advanced user, DM is made up of taggroups. Try for example to open a .dm4 file as a taggroup with `TagGroupLoadFromFile()`.

```
:  - ShowMonoSpacedFonts =  true
:  + StringColor
:  :  - blue  =  0
:  :  - green =  0
:  :  - red   =  192
```

For example, accessing the font size for script windows in two ways:

```
number fs
taggroup tg
GetUserPersistentTagGroup().TagGroupGetTagAsTagGroup("Scripting",tg)
tg.TagGroupGetTagAsNumber("fontSize",fs)
// Accessess the same tag as
taggroup tg2= GetUserPersistentTagGroup()
tg2.TagGroupGetTagAsNumber("Scripting:fontSize",fs)
```

This implies that tag labels cannot contain colons. Tag labels are furthermore case sensitive, and they can contain spaces (although that may become impractical if they need to be handled in another environment).

To overload our function with support for tagpaths we should write

```
void PrintTagGroup( taggroup tg, string tagpath ) {
    taggroup tg2
    if( tg.TagGroupGetTagAsTagGroup(tagpath,tg2) )
        PrintTagGroup(tg2)
    else
        Throw("Specified taggroup doesn't exist")
}
```

The function `TagGroupGetTagAsTagGroup()` returns a boolean, true on success, false on failure. The function `Throw()` throws an error message, also called *exception*, which stops execution of the current function and is passed upwards. It can be handled by `try{};catch{}` in the calling function, or passed on upwards. If it reaches the DM main application it will show an error message with the specified message.

3.3.4 Output: Result(), Debug(), Notes() and speed and versatility

Finally, we may want our function to output the printout to different places. The output window of DM has three panes, Results, Debug and Notes. These can be written to with the functions `Result()`, `Debug()` and `Notes()`. All of these functions will not just write to but also open the output window and make the pane that was written to active, and place a text cursor at the end for quick copying of the just written information.

The Notes pane is meant for the user, nothing in GMS ever writes to this pane. The Debug pane is used for error information, and the Results pane is meant for presenting results to the user. It should be noted that information written to these panes can easily be deleted by the user, and will be lost whenever DM restarts, so any actual measurement results should be saved directly to a file.

To accommodate all use cases it would be good if we had an extra version of our `PrintTagGroup()` function that would return a string variable instead of writing to the result window. But it is not possible to overload a function based on output type: a function knows nothing about how it was called, other than the arguments passed to it. We could make it do both, and just ignore the return value when we don't need it, but we may not want to clutter the result window.

There are two possible ways to solve this: introduce an argument which chooses the mode of operation, a *switch* or *option*, or introduce a second function name. Either way it is efficient to create the base function so that it returns a string, as this makes any additional functions very simple:

```
string PrintTagGroup(taggroup tg){
    //Main function goes here
}
```

```
void PrintTagGroupToResult(taggroup tg)
    Result(PrintTagGroup(tg))
void PrintTagGroup(taggroup tg, string switch){
    if( switch == "-r" || switch == "--result" )
        Result(PrintTagGroup(tg))
    else if( switch == "-d" || switch == "--debug" )
        Debug(PrintTagGroup(tg))
    else if( switch == "-n" || switch == "--notes" )
        Notes(PrintTagGroup(tg))
    else
        Debug("The PrintTagGroup() function was called with an "\
            +"unrecognised option.\nRecognised options are:\n"\
            +"\"-r\" and \"--result\"\n\"-d\" and \"--debug\""\
            +"\n\"-n\" and \"--notes\"\n")
}
```

Yes, that last call to `Debug()` is a bit hard to read. It is four strings which are appended to each other, as the `\` line continuation character cannot be used inside a string — inside a string it is the *escape* character, used to include special things such as newlines and citation marks.

It should also be noted that the approach of collecting all output into a long string instead of doing multiple calls to `Result()` has a marked performance advantage in addition to the versatility advantage. A crude test with an early version of my PrintTagGroup.s printing about a thousand lines took 2 to 3 seconds with one `Result()` call per line, and less than 0.2 s when the output was gathered into a string and printed with a single `Result()` call.

The DMscript language has facilities for measuring real time, but none for CPU time. To see how this is done, look at the source code of the function `ScriptStringTimer()` which is found in the library Linus-functions.s, appendix D.2.4. All code in that library follows the conventions of this document and, while sparse on comments, should be easily readable.

As a final note, the current version of my own `PrintTagGroup()` function can be found in appendix D.2.3. That one is a messy piece of code and not recommended to use, absolutely not recommended reading. I encourage any reader who needs a `PrintTagGroup()` function to write their own, with the help of the above instructions and ideas as well as the example one that can be found in the official documentation (the help files): Scripting > Example Scripts > Tags, TagGroups and TagLists, read Example 4: Print tags to results window. It is a good learning experience.

4 Results part one: the code

The main part of this project was creating code that can acquire 2D data from the spectrometer camera for every point of a specified 2D set of points on the sample. The decision to utilize the SI hookup system was made early, to avoid "reinventing the wheel". This means the created code requires the SI system, which in turn requires DigiScan®.

For the code to communicate correctly with the camera, SI in itself should not be using the camera. The way we achieve this is to set up SI for acquiring EDS spectra. Given that the EDS acquisition server could be installed in faux mode on an otherwise real microscope installation, this approach does not actually require EDS hardware — but it does require the EDS acquisition software of GMS. If no EDS system is available, there may be other viable options.

All source code created in this project will be made available at https://github.com/LinusSch, as soon as time allows for needed bugfixes. If there ever is a reasonably stable version, it will also be made available in the DMscript database (*The DM-Script database* n.d.). A current but buggy snapshot is included in appendix D.

Here follows descriptions of the most important parts of the created code, which assumes the reader has a basic familiarity with the language and environment. Such a basic familiarity can be obtained from section 3 of this report, supplemented by the official documentation (included with the install) where needed.

4.1 Managing the SI hookups

The SI hookup system is controlled by populating the taglists at "SI:Acquisition:Scripts:Control" in the persistent taggroup with the function names that the system should call. The documentation says it is ill advised to edit these tags directly, suggesting that writing robust scripts to handle these tags is a safer way. Therefore such scripts were created: the complete code for this spans two files, the SI Hookup Manager library found in appendix D.1.2 and an extremely basic interface, to be installed as a menu command, found in appendix D.2.1.

In the following example the hookup system is set up to run the functions `PixelHelloWorld()` and `PixelHelloTag()` at the beginning of each datapoint acquisition, and the function `SihuClean()` (which clears all these tags) at the end of the complete SI routine. All these functions are part of the created SI Hookup Manager. The printout is produced by calling `SihuPrint()`, which of course utilizes the `PrintTagGroup()` function discussed in the previous section.

```
Printing tags at "SI:Acquisition:Scripts:Control" in the persistent taggroup...
+ Correction
:  + Beginning
:  :  : <empty taglist>
:  + End
:  :  : <empty taglist>
+ Pixel
:  + Beginning
:  :  - [0]  =  PixelHelloWorld
:  :  - [1]  =  PixelHelloTag
:  + End
:  :  : <empty taglist>
+ Spectrum Image
:  + Beginning
:  :  : <empty taglist>
:  + End
:  :  - [0]  =  SihuClean
:  + Pause
:  :  : <empty taglist>
:  + Restart
:  :  : <empty taglist>
```

Note that the function names are specified *excluding* the parentheses in here, and the functions are required to take no arguments. They should not return any values either. Case is unimportant. The functions need to be available, which means any custom functions need to be installed as a library. The function that adds the function names to this list, `SihuAdd()`, checks that a correct hookup point and a correct function name are specified, but does not check arguments and return values.

The created code does provide a way around the function-installed-as-a-library requirement to speed up development: `SihuUseFile()` will insert a wrapper function that calls the inbuilt function `ExecuteScriptFile()` with a specified file path. It is used like this:

```
1 SihuUseFile("Pixel:End","C:\\Users\\Linus\\temp\\myexamplescript.s")
2 SihuPrint()
```

produces the following

```
Printing tags at "SI:Acquisition:Scripts:Control" in the persistent taggroup...
+ Correction
:  + Beginning
:  :  : <empty taglist>
:  + End
:  :  : <empty taglist>
+ Pixel
:  + Beginning
```

```
:  :  : <empty taglist>
:  + End
:  :  - [0]  =  PixelEndRunFile
+ Spectrum Image
:  + Beginning
:  :  : <empty taglist>
:  + End
:  :  : <empty taglist>
:  + Pause
:  :  : <empty taglist>
:  + Restart
:  :  : <empty taglist>
Printing tags at "SI Hookup Manager:Paths" in the User persistent taggroup...
- Pixel End Filepath = C:\Users\Linus\temp\myexamplescript.s
```

and will let you edit and test your code in an almost immediate manner: just save the file and click Start on the SI palette — over and over again. This is essential for efficient development work, uninstalling and reinstalling a library for each test is far too slow[9]. There are currently two drawbacks: this code only supports one file per hookup point, and if you forget to clean up the hookups yourself the next person to try to use a standard SI acquisition may be greeted with one error message per pixel.

In addition to the mentioned functions for addition, cleanup and printout of hookups, the created code provides `SihuRemove()` to remove all hookups for a specific point and `SihuBrowse()` which just opens a tag browser window as a reasonably convenient way to delete just one of the entries at a specific point.

4.2 The q-E STEM library

The main code file of this project, found in appendix D.1.1, aims to provide a complete software package for this acquisition technique. It includes a set of functions covering the following aspects:

- The per-pixel camera acquisition
- A function that runs at the start of the SI acquisition, preparing the camera and the data structure where the acquisitions are stored as well as storing other data about the experiment
- Saving/loading the data to/from a file
- Setting and reading parameters
- Setting the correct hookups
- Installation and uninstallation
- Some testing and development things, such as `PixelHelloCamera()`
- DM startup task
- Show a pixel acquisition as an image in DM

Note that two major parts are missing: cleanup, and a user interface. This is because of time constraints and will be added in the future.

As the library script file is run by DM on startup, one can include a startup task by simply calling it at the end of the file. The startup task of this library currently does nothing more than to make sure correct version information is written to `"qE-STEM:About"`, but more involved tasks are possible.

After installing this library (but first its dependencies, the SI Hookup Manager!) the `"qE-STEM"` taggroup can be found in the global persistent tags:

[9] Unless you write a script to do all that uninstalling/reinstalling, which is perfectly possible.

```
+ _LinusSch
:  - Author =  Linus Schoenstroem (a string tag doesn't handle o with dots)
:  - Author email (may provide support if you ask nicely)
=  linus.schonstrom@tele2.se
:  + Linus-functions
:  + qE-STEM
:  - SI Hookup manager version =  0.5.0b  2016-04-29
```

After additionally doing the install tasks outlined in appendix D.2.2, the `"qE-STEM"` taggroup will contain the following:

```
Printing tags at "_LinusSch:qE-STEM" in the persistent taggroup...
+ About
:  - Installed version =  0.a8.a  2016-04-29
:  - Requires Linus-functions =  v.6
+ Admin
:  + Default settings
:  :  - Binning =  (1,16)
:  :  - Camera area =  (448,0,576,1024)
:  :  - Exposure time (s) =  0.1
:  :  - Processing =  Unprocessed
:  :  - Save directory
=  D:\Linus\Documents\_faktiska dokument\Skolarbete\Exjobb\tests\save
:  :  - Scratch directory
=  D:\Linus\Documents\_faktiska dokument\Skolarbete\Exjobb\tests\scratch
:  - Path to ClickDetectorButton.exe
=  C:\Users\Linus\Linus-temp\ClickDetectorButton.exe
+ PerExperimentData
:  : <empty taggroup>
+ PerPixelData
:  : <empty taglist>
+ Settings
:  : <empty taggroup>
```

The default settings are used as a fallback by `qEReadSettings()` for any settings that are not specified in `"Settings"`. The path for scratch files is meant to be moved out of settings, to the purely administrative level.

To run a q-E STEM acquisition, run first the function `qESetHookups()` which sets the following hookups:

```
Printing tags at "SI:Acquisition:Scripts:Control" in the persistent taggroup...
+ Correction
:  + Beginning
:  :  - [0]  =  ClickDetectorBF
:  + End
:  :  - [0]  =  ClickDetectorOut
+ Pixel
:  + Beginning
:  :  - [0]  =  qEReadout
:  + End
:  :  : <empty taglist>
+ Spectrum Image
:  + Beginning
:  :  - [0]  =  qESetup
:  + End
:  :  - [0]  =  qECleanup
:  + Pause
:  :  : <empty taglist>
:  + Restart
:  :  : <empty taglist>
```

Warning: While a function called `qECleanup()` is in there, as it is defined in code file available in appendix, it does not do anything — cleanup has to be done manually!

Set any desired settings with `qEWriteSettings()` or by directly editing *existing* tags at `"qE-STEM:Settings"`. Then start an SI acquisition with EDS selected under SI Signals — if EELS is selected SI will try to control the camera, which may cause trouble.

Note that the `"qE-STEM"` taggroup, while meant to hold version info and settings, is currently also used as a convenient place to store data and runtime variables. This is not good, as discussed in the first points of section 6.1. While doing an acquisition it contains the following (previously viewed parts not expanded, only one pixel group expanded, taken at the fourth of 18 survey pixels):

```
Printing tags at "_LinusSch:qE-STEM" in the persistent taggroup...
+ About
+ Admin
+ PerExperimentData
:  - Capture resolution =  (1024,8)
:  - Save location
=  D:\Linus\Documents\_faktiska dokument\Skolarbete\Exjobb\tests\save\2016-
05-02_141702_testrun-for-report
:  - Scratch location
=  D:\Linus\Documents\_faktiska dokument\Skolarbete\Exjobb\tests\scratch\2016-
05-02_141702_testrun-for-report
:  - Survey resolution =  (6,3)
+ PerPixelData
:  + [0]
:  :  - data = <array tag, 8192 elements>
:  :  - Pixel number =  0.0
:  :  - Timestamp_1_start =  4.07227e+12
:  :  - Timestamp_2_acquire =  4.07227e+12
:  :  - Timestamp_3_end =  4.07227e+12
:  + [1]
:  + [2]
:  + [3]
+ Runtime variables (please don't touch!)
:  - Binning =  (1,16)
:  - Camera area =  (448,0,576,1024)
:  - Camera ID =  0
:  - Exposure time (s) =  0.1
:  - File =  A
:  - Processing =  1.0
:  - Save directory
=  D:\Linus\Documents\_faktiska dokument\Skolarbete\Exjobb\tests\save\2016-
05-02_141702_testrun-for-report\
:  - Scratch directory
=  D:\Linus\Documents\_faktiska dokument\Skolarbete\Exjobb\tests\scratch\2016-
05-02_141702_testrun-for-report\
+ Settings
```

4.2.1 The acquisition

The most important piece of code in this project: how to read the camera, in a function that can be hooked up at `"Pixel:Beginning"`. For a basic readout, the function to use is `CameraAcquire()`, which returns an image with the acquired readout. Where performance matters the function `CameraAcquireInPlace()`, which acquires the readout into a prepared image variable passed as a reference, is preferable (see section D.4).

These two functions can both be called with none, all, or three different subsets of the acquisition parameters specified:

```
1 camid= IFGetFilterCameraID() //A number identifying the camera
2 exposure= 0.2               //Exposure time, in seconds
3 xBin= 1; yBin= 2             //Binning
```

```
/* A number identifying the camera processing mode: */
proc_enum= CameraGetUnprocessedEnum()
//proc_enum= CameraGetDarkSubtractedEnum()
//proc_enum= CameraGetGainNormalizedEnum()
areaT= 896; areaL= 0; areaB= 1152; areaR= 2048 //Readout area

CameraAcquireInPlace(camid,img)
CameraAcquireInPlace(camid,img,exposure)
CameraAcquireInPlace(camid,img,exposure,xBin,yBin)
CameraAcquireInPlace(camid,img,exposure,xBin,yBin,proc_enum)
CameraAcquireInPlace(camid,img,exposure,xBin,yBin,proc_enum,areaT,areaL,areaB,areaR)
```

The area specification follows one of DMscript's more odd conventions, in which a rectangular area is specified as `top, left, bottom, right` where the top and left numbers are inclusive while the bottom and right numbers are exclusive. This allows for example `sizeX= right - left` to give the correct size. The above example specifies the full width and an eighth of the height of a 2048x2048 pixel camera.

All parameters that are not passed to the acquisition function are, according to the documentation[10], supposed to default to the parameters specified in the Record setting of the GUI. But this has so far been found to not be the case, these functions seem to default to a static set of parameters.

To prepare the image variable, which has to be of correct dimensions, the safest way is to use

```
image img:= CameraCreateImageForAcquire(camid,xBin,yBin,proc_enum,areaT,areaL,areaB,
    areaR)
```

Since this function does take processing into account, it is likely creating an image variable of the most suitable type. This is not required — but on the basis of this function doing it, it is likely preferable.

A bug was found[11] that resides either in `CameraCreateImageForAcquire()` or in `CameraAcquireInPlace()`. When specifying the use of "Cinema Mode", presenting only the middle half of the camera to the GUI controls, this code will throw an error if `areaT` or `areaB` has a value higher than half the height of the camera — but it will still read out the camera with those values as counted from the real top of the camera. Hence only the top half, rather than the middle half, becomes accessible. The workaround is to manually make sure Cinema Mode is not set when doing scripted acquisitions, to my knowledge there is no way of accessing this setting through DMscript.

It should also be noted that the example script for using these functions has a bug: it passes the exposure time rather than the camera ID as the first argument to this function[12].

The code for image acquisition in this project is the `qEReadout()` function of the main library, found in appendix D.1.1.

4.2.2 Viewing an array tag as an image

As the code currently saves the data directly as an array tag rather than in an image variable, a custom function is needed to view it. An array tag cannot be correctly extracted unless its dimensions are known, the dimensions are not available from the tag system itself.

The rest of this section (4.2.2) should be read with caution, as the presented code is currently not working.

Type is available, but as type conversion is liberal in DMscript, it is not needed for basic viewing. A simple general function for this task can thus be written like this:

```
// Show an array tag as an image
// For two-dimensional arrays of unspecified type - TYPE CONVERSION MAY HAPPEN, ONLY
    FOR BASIC VIEWING
```

[10] That is Scripting > Plugins > Camera Control > Scripted Acquisition, heading Acquisition.

[11] On the real microscope and faux microscope 1, faux microscope 2 does not have Cinema Mode (see appendix A for details).

[12] The script somehow still runs, at least on faux microscope 2 (see appendix A for details).

```
void TagGroupShowArrayTagAsImage(taggroup tg, string tagpath, number size_x, number
    size_y) {
    string label= TagPathExtractLabel(tagpath)
    image img= RealImage("Image created from tag "+label,4,size_x,size_y)
    number r= tg.TagGroupGetTagAsArray(tagpath,img)
    if(r)
        img.ShowImage()
    else
        Throw("Reading the array failed")
}
```

where `TagPathExtractLabel()` is another custom function, which simply removes anything up to and including the last colon of a string. Both these functions are made available by the library found in appendix D.2.4.

This general function is called by the specific function of the q-E STEM library that shows the acquired q-E data for a specified survey pixel:

```
// Shows the camera readout from one pixel
void qEShowPixel(number i) {
    taggroup qetags, pixeldata
    TagGroupGetTagAsTagGroup(GetPersistentTagGroup(),"_LinusSch:qE-STEM",qetags)
    qetags.TagGroupGetTagAsTagGroup("PerPixelData:["+i+"]",pixeldata)
    number size_x, size_y
    qetags.TagGroupGetTagAsShortPoint("PerExperimentData:Capture resolution",size_x,
        size_y)
    pixeldata.TagGroupShowArrayTagAsImage("data",size_x,size_y) // This does not take
        type into account, which works for viewing but should not be used to extract
        the data
}
```

Note that the dimensions were saved in a separate tag, to make sure the data can be extracted.

4.2.3 STEM detector control

In order to do drift correction at least one STEM detector needs to be receiving signal. In the aperture setup used for the pre-study this posed a problem, since the objective aperture blocks the HAADF detector while the BF and ADF detector would block the spectrometer.

DMscript functions to insert and retract the STEM detectors was found not to exist, even though the functionality is readily available by clicking buttons in the GUI. This is indeed not unusual: DMscript was despite its name not conceived as a way to automate DM tasks.

> **Aside: the history of DMscript**[a] At the very beginning of DM, in the eighties, DMscript was included with a specific purpose: to do math on the acquired data. Soon, Gatan engineers came to use it to quickly create whatever special functionality a customer asked for. This led to great extensions of the language, whereby it became an effective development tool internally. Eventually most of GMS came to be written in DMscript!
>
> Then a couple of important changes in computer hardware happened at the same time as new microscopic and spectroscopic techniques asked for a lot more performance, unobtainable by an interpreted language. Since almost everything was to be ported to C++ Gatan did consider dropping the DMscript language completely — the original purpose does make less sense in a time when most customers expect GUIs for everything. But part of the customer base, who had clued in to the possibilities, had come to rely on this inadvertently created and mostly undocumented feature. The decision was made to try to keep the "power user" functionality, document it and even extend it.
>
> This is where it stands: it is essentially a programming language about 30 years old, more or less reborn as a requested feature. It is not made to duplicate, automate and extend the functionality of DM — it is there as a complement, but is often (and often not) able to automate and extend it as well. But this may change: change has been a constant in

its history.

[a] This summary is based on personal communications with Bernhard Schaffer.

As this detector control functionality was found missing, and Gatan staff agreed it would be good to have, this functionality has been added to GMS 3.1[13]. But the available microscope runs GMS 2.3, so a workaround was created as well. Beware: this workaround may not be a working solution in its current state, see further down.

This workaround utilizes a third party software called AutoIT[14], a scripting language for general Windows automation. With this language a very simple script was created, found in appendix D.5.1, which literally clicks the button (provided that the button is at the specified location in the screen). This has the major drawback that if the user is moving the mouse while it happens, it fails. The current version needs to have the correct coordinates for any specific installation set in the source code, and then be compiled into a command-line executable before use in conjunction with DM. A future version will take the coordinates as argument, so that no compiling is needed.

The workaround is made possible by the DMscript function `LaunchExternalProcess()`, which can call any Windows command-line program including arguments. The complete function making the call to insert the BF detector in the q-E STEM suite is written like this:

```
1 void ClickDetectorBF() {
2     OpenAndSetProgressWindow("q-E STEM Acquisition","Clicking to insert BF","detector.
          DON'T TOUCH MOUSE")
3     number r
4     string exepath
5     TagGroupGetTagAsString(GetPersistentTagGroup(),"_LinusSch:qE-STEM:Admin:Path to
          ClickDetectorButton.exe",exepath)
6     r= LaunchExternalProcess(exepath+" \"BF\"")
7 }
```

and is accompanied by an analogous `ClickDetectorOut()`. Note that while this does wait for the external process to return, and does fetch its return value, that AutoIT script does not even know whether or not it correctly clicked the button.

The next problem is that inserting and retracting the sensor takes an amount of time. Ideally the script would wait for this to complete before returning control to SI, but by default all scripts are run in the same thread as the main DM application, freezing it until the script has completed! Putting a pause into the above code therefore delays DM's reaction to the clicked button. The SI acquisition itself is run in a threaded manner, so the DM application is able to react when control is returned to SI. But SI at the same time marches on — and herein lies the untested part, as the test would need to be done with a sample in the real microscope: is the detector insertion and retraction fast enough that drift correction will work and not too much spectroscopic data is lost?

If not, one has to write a threaded script that calls `SIInvokeButton()` to pause and resume SI. There is a function called `IFGetConfiguration()` which gets you a small taggroup including a tag labeled `"Extra detector inserted"`, but this tag was found not to correlate to detector positions. Hence it would also be needed to experimentally find the necessary wait time.

4.3 Matlab

4.3.1 Importing the data

An existing frontend (written by T. Thersleff) for the dmread.m utility written by Andreas Korinek was modified to extract the data from the saved taggroups. This code is available in appendix D.6.1. While dmread.m was written for the dm4 file format, internally that is just a specifically structured taggroup — and so the gtg file format is read in the exact same way.

Showing a string imported as one of these tags requires a little conversion, see appendix D.6.2.

[13] Which, as of 2016-05-08, is not yet released (but the beta is available for download).
[14] Available free of charge at www.autoitscript.com.

4.3.2 Other tools

A few ad-hoc tools for visualizing were also created. The most important one, showesi.m, can be found in appendix D.6.3. It takes a 2D array and shows it with the same orientation and a similar contrast as DM would. Another tool extends on this with semi-interactive browsing through a set of images, esisliceplayer.m which can be found in appendix D.6.4.

There is also the code for the analysis presented in section 5.2.1, and a generalization of esisliceplayer.m to browse through that data in appendix D.6.5. The latter theoretically enables browsing through the results of any Matlab code that takes a number and writes to the current figure but does currently not work well with showesi.m.

Code for extracting the EMCD spectra and signal from these datasets was just started on.

5 Results part two: testing

Several smaller tests were performed throughout the development of the code, the results of which are apparent from the current state of the code as discussed in sections 4 and 6. Two larger-scale tests produced results that are discussed here: one performance test of the various camera readout commands including more or less processing, and one complete test of the output of the code in its current state. The complete test also included measuring the performance, but those results have not been analyzed.

5.1 Camera commands and readout processing

There are several different ways to do a camera readout in DMscript:

```
1 img:= CameraAcquire(camid)
2 CameraAcquireInPlace(camid,img)
3 img:= CameraAcquire(camid,exposure,xBin,yBin,processing,areaT,areaL,areaB,areaR)
4 CameraAcquireInPlace(camid,img,exposure,xBin,yBin,processing,areaT,areaL,areaB,areaR)
```

Calling these two functions without explicitly passing all or some of the acquisition parameters to them is, according to the documentation, supposed to default to settings specified in the GUI[15]. This was found to not be the case, the default set of parameters remained unchanged regardless of what was set in the GUI.

A third function named `SSCUnprocessedBinnedAcquire()` is mentioned in e.g. Schaffer's handbook (Bernhard Schaffer 2015). By its name it is clear that this function is more specific, hence it was tested with hopes of better performance. But this function was found broken: it produces images with incorrect dimensions and disordered data when binning.

Take note of the parameter `processing`. This can be set so that the acquisition functions provide the camera readout in three very different ways:

Unprocessed This gives an image containing the raw data readout from the camera, no processing applied. Note however that depending on camera setup the output can be at least 16-bit unsigned integer or 32-bit floating point, possibly other things too.

Dark subtracted This gives an image where the *dark reference* has been subtracted from the raw data. The possibility for negative data points means that the output cannot be unsigned integers, that same camera setup with this option produces 16-bit signed integer. The dark reference is an image acquired with the camera blocked or the electron beam off, otherwise the same conditions as the current acquisition. It is essentially a measurement of the noise produced by the camera system, and subtracting it cancels that out.

Gain normalized This includes a second stage, also crucial to get close to measuring how many electrons the camera was actually hit by. It multiplies the dark subtracted image with the *gain reference*. The gain reference is created by an acquisition routine with the camera evenly illuminated by a spread out electron beam, and some normalization. It corrects for gain (amplification) differences across the camera, which arise partly from

[15] Specifically, the "Record" setting of the "camera acquire" dialog.

pixel-to-pixel differences in the CCD and electronics but mainly from the *scintillator*. The scintillator is a layer on top of the CCD which produces the visible light that the CCD can measure when hit by electrons.

These processing steps are likely to have the largest influence on the time taken to run the acquisition function, so these were investigated first. The complete code used for these tests is available in appendix D.4.

It was found that gain normalizing takes a considerable amount of time. For a 0.1 s exposure, time taken is up from 0.25 s to 0.45 s. Dark subtraction takes a little time, the difference is less than 0.01 s.

No version takes a consistent amount of time, all of them has a significant number of acquisitions taking more than 0.05 s extra, which for everything but the gain normalized version is more than four times the width of the main grouping in a histogram. This is concerning since it may mean the probe stays an inconsistent amount of time at each point of the sample, and while it would have no impact on any acquisition (they are all shuttered by independent timers[16]), it may have an impact on beam damage.

The difference from whether or not the image variable was declared and/or initialized outside the loop is virtually nonexistent, for the main group. This may be due to compiler optimizations resulting in the actual hardware instructions executed being more similar than the source code would suggest, maybe even the exact same.

That said, there were some notable observations. The test with the variable not even declared (outside the loop) does have the main group very slightly extended towards longer times. Somehow, the test with the variable declared but not initialized shows significantly fewer of the slow outliers. However, when it was later run with a colon (assign rather than copy), it looked just like the other two again, so it may be a fluke.

Regardless, the `CameraAcquireInPlace()` function shows consistently better performance, including a tighter grouping of times. While it was tested only with variable creation outside the loop, it clearly outperformed `CameraAcquire()` with similar conditions. Basic statistics for all tests without processing:

Variable was...	Mean [s]	st. dev. [s]	# of slow outliers
initialized	0.254	0.022	107
declared	0.245	0.013	32
not even declared	0.253	0.021	88
declared - assign	0.253	0.021	105
`CameraAcquireInPlace()`	0.243	0.016	57

5.2 What the readout looks like

A conventionally prepared sample of an iron thin film on magnesium oxide substrate was set up in the microscope as a basic test run of the created code. The experiment was not optimized for EMCD analysis, and it was limited to a linescan rather than a 2D mapping in order to have a more manageable dataset. While no EMCD signal has been extracted, the q-E acquisitions do show regular spectra with identifiable oxygen and iron peaks.

Figure 3 shows the survey image. A lot of drift was observed with other experiments on the same sample done directly before this experiment, but drift correction was accidentally left at once every row — for a single row this results in just once, at the beginning of the experiment. Additionally, that one failed, so weak relations between the survey image and the acquired data are to be expected.

One discrepancy that is unexpected and unexplained is that the specified length of the linescan in the survey image is 93.17 nm, while the scale of the in-experiment captured ADF signal says that the scan was made over a distance of 93.99 nm. Figure 4 compares the captured signal with an extraction of the specified line in the survey image.

[16] Except maybe a basic STEM imaging detector.

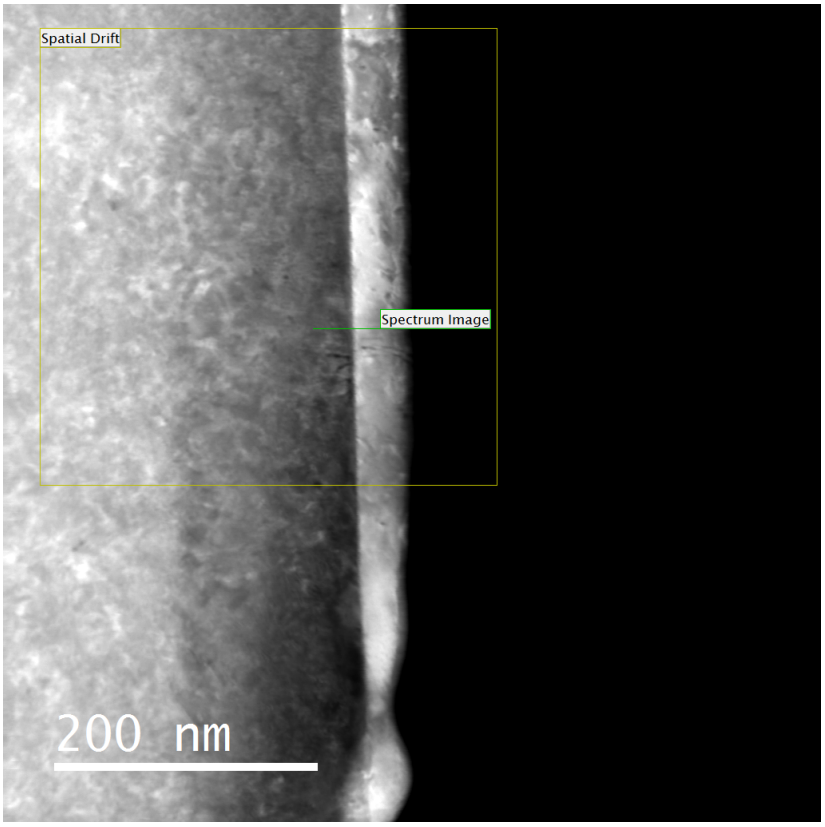


Figure 3: Survey image of the test experiment.

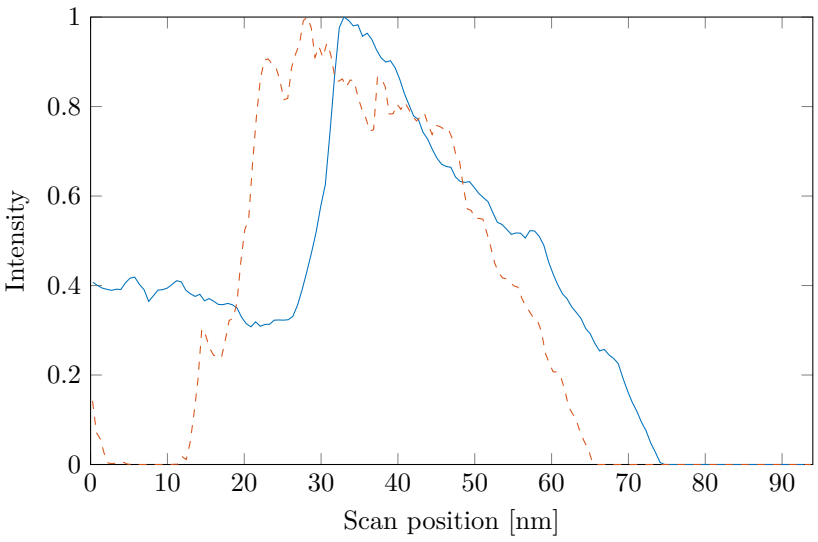
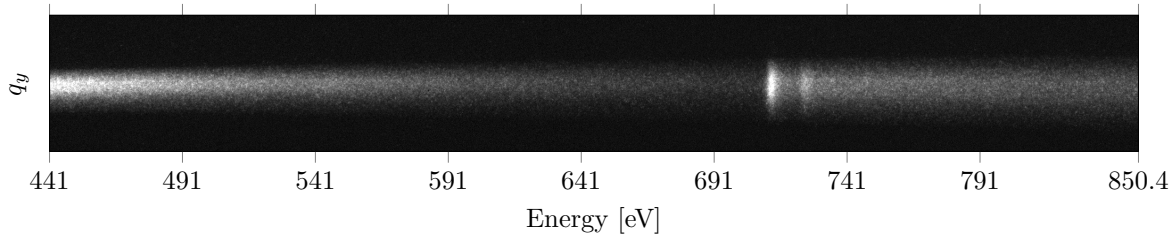
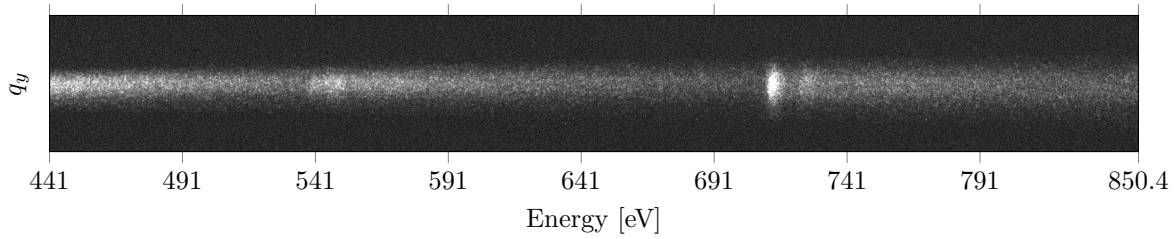


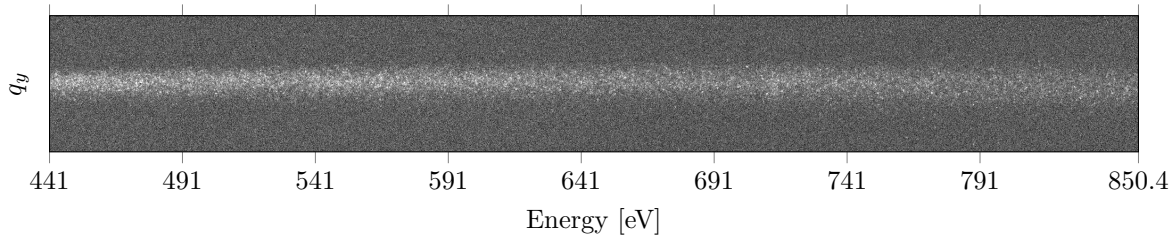
Figure 4: Comparison of data extracted from survey image (blue) with the ADF signal acquired during the linescan (red, dashed). Peak normalized intensity.



(a) Spectrum capture from point 55, the beginning of a region where no oxygen is showing. Iron peaks (708, 721 eV) are clearly visible.



(b) Spectrum capture from point 127, close to the edge of the sample. Both oxygen (540 eV) and iron (708, 721 eV) peaks are visible.



(c) Spectrum capture from point 134, showing almost no signal. All data from here up to the end of acquisition at point 185 are virtually the same.

Figure 5: Three examples of the q-E spectrum captures.

From the survey image it can be seen that the linescan covered three regions. Measurements in the image showed the substrate region to be about 30 nm, the film region to be about 39 nm and the vacuum region to be about 24 nm.

Figure 5a shows the spectrum capture from point 55, representative of the captures 55 – 123 (circa 28 – 63 nm into the scan) as it shows no oxygen peak. This mostly corresponds to the iron film region. Figure 5b shows the capture of point 127, towards the end of the iron film region. Here the oxygen peak reappears: at the edge of the sample, the iron is oxidized.

Figure 5c shows the capture from point 134 — from this point up to the end at 185, all captures are very similar as almost no electrons are reaching the camera. These 52 survey points correspond to 26 nm in the scale of the survey image. The difference to the measurement of the vacuum region in the survey image can be attributed to drift, this is backed up by the in-experiment ADF signal (which actually indicates the difference would be much larger, see figure 4).

5.2.1 The spectrum captures are curved

To find out if there was a curve to the spectrum captures or not, a simple algorithm was used: for each vertical line of the picture, find the centre of the signal, then analyse if those 2048 points are on a straight line or on a curved line.

Finding the centre of the signal by identifying the signal part through histogram counts proved prone to errors due to the noisy nature of the data. Instead, a basic centre-of-mass algorithm was used:

```
centreline= zeros(1,sizeX);
```

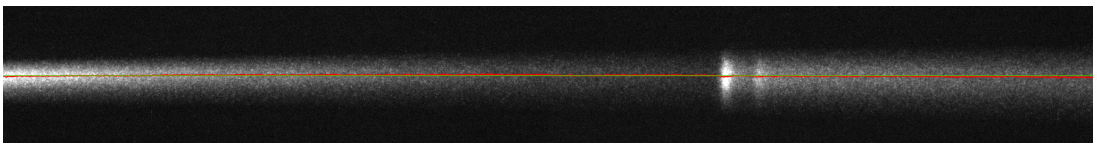


Figure 6: Spectrum capture of point 63, one of the most visibly curved. The calculated centre is shown as a red line, with its mean as a straight green line on top to help identify the curve.

```
for i= 1:sizeX
    line= array(i,:);
    momenta= line .* [1:sizeY];
    centreline(i)= sum(momenta)/sum(line);
end
```

This algorithm has one important artifact to keep in mind when analyzing its output. A stronger signal at the same position, when not perfectly centred, will create a larger deviation in the value taken to be the centre. Ideally this investigation should be redone with this artifact corrected for.

The complete function used, including unsuccessful attempts at finding the edges of the data as well, is available in appendix D.6.6. It was used on all 133 spectrum captures with data in them, after dark subtraction and gain normalization.

```
>> for i=1:133; allcenters(:,i)= plotLinesOnqE(grmultiplied(:,:,i),'nop');end
```

The result of this is 133 noisy centrelines, but none of them varies more than a few pixels, except for two singular outliers (at capture 19 line 798 and capture 106 line 916). When shown on the same scale as the image itself both noise and any trends are very small, see figure 6. When shown on an enhanced scale it is reasonably clear that the signal is centered slightly lower towards the edges of the image, slightly higher in the middle. For some captures this is more clear, for others less clear, which may or may not be an indication that the amount of curve varies.

In an attempt to put a number on this slight curve, a circular arc centered at $x = 1024.5$ was fitted to an average of all the centreline data. The fit is shown in figure 7. It was found to have a radius close to 25 thousand pixels, more than 12 times the width of the camera. This equates to a height difference of just over 2 px between the middle and the edges. While this is likely not the correct model, a quick calculation showed that even at the edges this theoretical tilt of the spectrum will create at most a few pixels worth of blurring — when vertically summing all of the signal. At the L edges, when positioned as in this experiment, it would mean that the $+$ and $-$ spectra are shifted in x relative to each other with something between one and two pixels.

In the EMCD signal this effect is likely below the noise floor, but it should be investigated. The effect presumably arises from aberrations in the GIF, which means an aberration-corrected GIF may solve the problem entirely.

5.2.2 Possibly inconsistent positioning in y

When browsing through the plots of the centreline data, it was noted that not only the curve but also the overall y position of the data seemingly changes slightly between captures. To plot this:

```
>> plot(mean(allcenters),'.')
```

The resulting scatter plot has to be interpreted with caution. All trend seen in this diagram very closely correlates to overall signal strength, which as mentioned is an artifact of the algorithm used. This is highlighted by figure 8 which plots the two on top of each other, extended with the data from the empty captures 134 – 185.

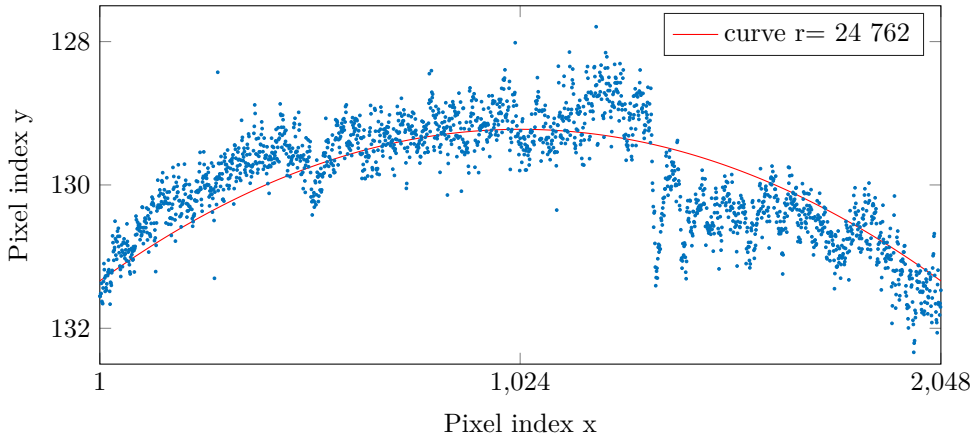


Figure 7: The circular arc fitted to the averaged centreline data. Note that this is a greatly enhanced scale, compare figure 6, which is actually slightly more curved.

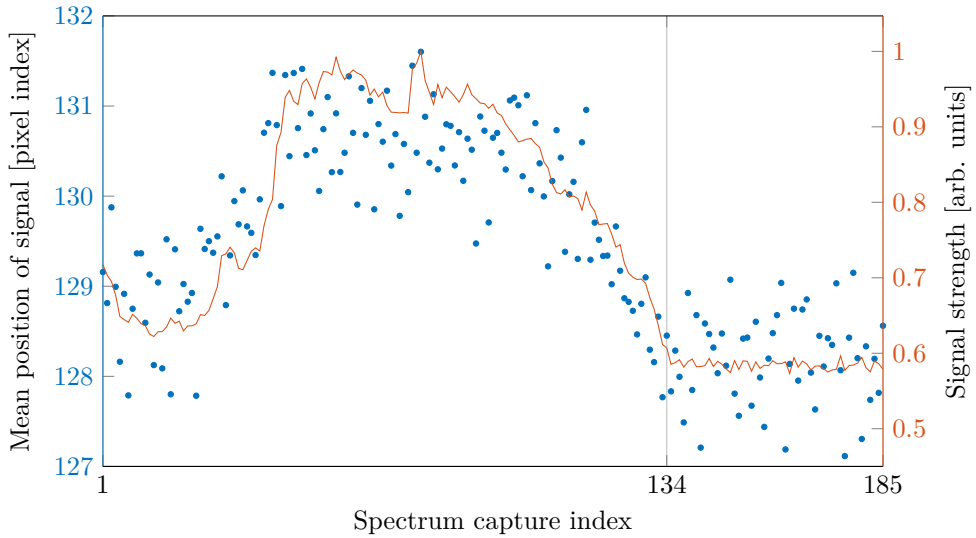


Figure 8: Possibly inconsistent spectrum positioning (blue dots). Note that the trend in the positional data closely correlates to the signal strength (red line), an artifact of the algorithm used. In the region from capture 134 to capture 185, where there is no trend, no correlation was found. The spread of the positional data in that region is about 2.5 px.

In the region from capture 134 to capture 185, where there is no trend, no correlation was found. The spread of the positional data in that region is about 2.5 px. It is unknown whether this is an actual instability in the positioning of the signal on the camera, or just a result of the noise in the data.

6 Future work

6.1 Known bugs and other things to correct

- The highly important `qECleanup()` function needs to be written. Without this, and the would-be temporary data storage situated in the persistent taggroup, the situation easily arises that DM writes this very large dataset to disk on shutdown and loads it on startup — not good at all.
  - A good cleanup routine does not avoid the problem in case of a crash. In a farther future, storage of this data should be completely moved out of the persistent tags.
- Both data and runtime variables are currently vulnerable to inadvertent deletion or modification. Same solution as above: they are to be moved out of the persistent tags.
- `qESetup()` uses the inbuilt `DateStamp()` function for part of the folder name, but this is not consistent across Windows localization settings. For some locales its output includes slashes, resulting in a folder structure ranging from mildly (in the case of yyyy/mm/dd) to very confusing.
  - This will soon be fixed, as creating a consistent `DateStamp("ISO")` overload is a known easy and good solution.
- The `qEShowPixel()` function does not work properly.
- Saving and loading data includes the version info of the q-E STEM suite, which is good, but under the label "Installed version", which is not good. A solution is planned, but needs a lot of code.
- Save and scratch root paths are labeled as save and scratch directories.
- Scratch root path should not be included in the general settings.
- Saving the data to disk during the course of the experiment is an important feature to avoid losing potentially useful data to a computer or DM crash, but the current code for this is unusable.
  - It should be appending to the saved files instead of rewriting them, as the datasets are very large.
  - Even then, it should be ran in a background thread. There is a correct way to do this, and there is also a simple hack that might work, but there is no point in doing this until the function works correctly (appending).
- Default camera settings are not portable.
- Make `PrintTagGroup()` not print overly long taglists and array tags.
- If possible, handling the Cinema mode bug mentioned in section 4.2.1.
- It needs to be verified whether the automated mouseclick solution for STEM detector control works or not.
- The Matlab code for browsing through data should print its status to the figure window rather than the command window.
- Last but not least all code, especially qE-library.s, needs better structuring and commenting — especially if more people are to work on it.

6.2 Missing features

Planned features for the q-E STEM library include:

- Gathering the data into a multidimensional `dm4` file.
- Automated dark reference acquisition.
- Integrated dark subtraction and gain normalization.
- Making use of the included detector control functionality in GMS 3.1 when available.

The Matlab code for browsing through the images is planned to support 4D datasets. There also needs to be code to extract the EMCD signal from the data, and analyze it.

6.2.1 User interface

Arguably the largest missing feature of the current codebase is a user interface for the q-E STEM library. This could be made in several ways, of which the simplest to code would be a set of one-question dialogs — but such an interface would be very frustrating to use. A simple CLI such as the one created for the SI Hookup Manager is reasonably easy to create, but still impractical to use. An advanced CLI, maybe as a palette window, would be extremely efficient to use but may be accordingly difficult to create. A single-dialog GUI can be a good solution, but usually needs a lot of user testing for a good design.

It should also be noted that the very simple CLI for the SI Hookup Manager is — while usable — far from optimal, and will eventually be improved upon.

6.3 Performance

6.3.1 Analyze timestamps

As can be seen from the taggroup printout in section 4.2, and the source code of `qEReadout()`, the data acquired includes three timestamps per pixel. These are meant to analyze the real-world performance of the code, on any installed system, to guide future development. These timestamps need to be interpreted with the inbuilt DMscript function `CalcHighResSecondsBetween()`, and the code for this has not been created yet. Ideally the function for this should also create the relevant histograms, so that it can additionally be used as a diagnostic tool in case of problems.

6.3.2 Matlab code performance

The created Matlab code is a wrapper around an already-built `dm4` file reader. This currently creates two copies of the data in memory, which can be a large performance problem with large datasets. Merging these two parts can eliminate this as well as opening the possibility to make sure the file is read in a sequential manner. Additionally the code is currently looping through the survey dimensions, they should optimally be vectorized.

7 Discussion

7.1 Performance

7.1.1 Integers or floats from the camera

The camera on the real microscope is set up to output 32-bit floating point numbers also when not applying any processing. Given that this is double the amount of data from the default, presumably native, 16-bit unsigned integers it is well advised to investigate if changing this has an effect on performance.

7.1.2 Classful programming vs tags

As discussed previously, using the persistent tags to store data and runtime variables is not good. It should be replaced by creating one or several persistent objects. Since the code is therefore making the move to classful programming, one might investigate any performance differences between structuring variables with the tag system and through creating more objects.

7.1.3 Threading ideas

Classful programming in DMscript also provides facilities for threaded execution. An obvious candidate for a background thread is saving the data to disk while the acquisition is running. It may also be fruitful to have a background thread running that can minimize the amount of code which needs to be in the `qEReadout()` hookup. A class that provides a multi-dimensional image variable could potentially also do this, but may have trouble writing the data to disk while the acquisition is running.

7.1.4 Low-level and hardware ideas

From the fact that a 0.1 s exposure results in about 0.25 s of time taken for the camera readout functions, there is theoretically room for improving the performance with new low-level and/or hardware subsystems. Considering the performance of the GMS suite as a whole and the variation seen in the time taken, it is unlikely that pure software work could bring any improvement.

The speed of the hardware and firmware involved may be bottlenecked at one of several places:

- Recocking the shutter
- CCD readout subsystem
- On-camera intermediate storage
- Firmware layer to read from intermediate storage and transfer to computer
- Link to computer

To develop any of this is likely to take a lot of resources, but the recent advances in availability of high-speed cameras for normal light applications may provide some useful pieces of technology.

7.2 Usability and other concerns for wider adoption

In its current state it is easily seen that the created software package has severe usability problems. But even after that which is discussed in section 6 has been corrected, magnetic experiments in the TEM is still inaccessible compared to most TEM techniques.

This is in large part because of stringent experimental requirements, but also because the result is not visible while at the microscope. With further software development this could be remedied. A more direct view of the EMCD signal could help the operator greatly, and therefore be an important step towards making magnetic experiments in the TEM commonplace.

It may also be possible to automate, or in other ways help, parts of the complicated aligning and orienting required through software. Many tools for automatic tuning of the TEM in different situations and setups are available, it is likely that such tools could be utilized and combined to create a dedicated q-E setup helper tool.

As a final note, future usability of this technique depends not only on the accessibility of the acquisition itself but also the accessibility of the analysis. To develop a complementary software package with readily available tools and options for denoising, background subtraction, EMCD signal extraction and application of the sum rules is a logical next step.

8 Conclusions

The results of this work show that:

- The widely used Gatan Microscopy Suite® provides, through the Spectrum Imaging hookup system, a functional framework for custom STEM acquisitions
- The proposed q-E STEM acquisition setup (for details see section 1.3.4) can be achieved with this system, through the code presented here
- The captured data has aberrations which are relatively minor (their effect is likely to be masked by noise)

The created acquisition routine is currently not practical to use, but correcting this is straightforward code work outlined in section 6.

The DMscript language proved powerful and flexible enough that all obstacles could be overcome or worked around. Any needed functionality can be created, it just may require a considerable amount of programming. Documentation issues can be a hindering factor but is generally offset by good support.

All things considered a complete and readily usable software system, including facilities for live monitoring of the resulting signal, is certainly possible. Such a system could even be extended to include not just q-E STEM but all known setups for magnetic dichroism in EELS. However, it would be a large project. The author intends to create an open-source project with such a system as the ultimate goal — any interested collaborators are welcome.

9 Acknowledgements

First and foremost, I want to thank the people who have been involved in this exam work, sharing their knowledge and skills:

Thomas Thersleff, my supervisor, who had the idea for the project itself, introduced me to the subject and operated the microscope.

Ján Rusz, who kindly helps out in my attempts to understand the theory of this over and over again, and provided the data for figure 2.

Bernhard Schaffer, who has been very helpful with technical and even historical questions on the DMscript language, and told me about running DM in faux mode.

Klaus Leifer, supporting professor, who has answered many questions and entertained enthusiastic discussions.

Emil Eriksson, whose helpful comments have improved the report.

A few people that were more tangentially involved also deserves a mention:

Nadia Mohammadi, who gave me a schedule and some hope.

Mats Olof-Ors, who listens to all my rambling thoughts.

Åsa Kassman, who provided guidance on the administrational part of doing an exam work.

Last and certainly not least, there are three people to whom I am endlessly grateful. While not as directly involved, they continually play crucial roles in my ability to do not just this exam work but indeed any work at all — I would likely not even be at a university without them:

Johan Schönström, my father, my primary sounding board, and main support in many different matters.

Simon Schönström, my brother, who resets my brain when it does the equivalent of a Windows BSOD.

Sofia Kontos, who gives me the courage and ability to do the things I want to do.

References

Bernhard Schaffer (2015). *How To Script - Digital Micrograph Scripting handbook*. URL: http://digitalmicrograph-scripting.tavernmaker.de/HowToScript_index.htm (visited on 05/06/2016).

Calmels, L. et al. (2007). "Experimental application of sum rules for electron energy loss magnetic chiral dichroism". In: *Phys. Rev. B* 76.6, p. 060409. ISSN: 1098-0121, 1550-235X. URL: http://link.aps.org/doi/10.1103/PhysRevB.76.060409 (visited on 02/06/2014).

David Mitchell. *DMScripting.com - Home*. URL: http://www.dmscripting.com/index.html.

Hébert, C. and P. Schattschneider (2003). "A proposal for dichroic experiments in the electron microscope". In: *Ultramicroscopy* 96.3-4, pp. 463–468. ISSN: 03043991. URL: http://linkinghub.elsevier.com/retrieve/pii/S0304399103001086 (visited on 02/07/2014).

Lidbaum, Hans, Ján Rusz, Andreas Liebig, et al. (2009). "Quantitative Magnetic Information from Reciprocal Space Maps in Transmission Electron Microscopy". In: *Phys. Rev. Lett.* 102.3, p. 037201. ISSN: 0031-9007, 1079-7114. URL: http://link.aps.org/doi/10.1103/PhysRevLett.102.037201 (visited on 02/07/2014).

Lidbaum, Hans, Ján Rusz, Stefano Rubino, et al. (2010). "Reciprocal and real space maps for EMCD experiments". In: *Ultramicroscopy* 110.11, pp. 1380–1389. ISSN: 03043991. URL: http://linkinghub.elsevier.com/retrieve/pii/S0304399110002044 (visited on 02/07/2014).

Rubino, Stefano et al. (2012). "EMCD Techniques". In: *Linear and Chiral Dichroism in the Electron Microscope*. Ed. by Peter Schattschneider. Singapore: Pan Stanford Publishing Pte. Ltd., pp. 149–173. ISBN: 978-981-4267-48-9.

Rusz, Ján et al. (2007). "Sum rules for electron energy loss near edge spectra". In: *Phys. Rev. B* 76.6, p. 060408. URL: http://link.aps.org/doi/10.1103/PhysRevB.76.060408 (visited on 06/02/2014).

Salafranca, Juan et al. (2012). "Surfactant Organic Molecules Restore Magnetism in Metal-Oxide Nanoparticle Surfaces". In: *Nano Lett.* 12.5, pp. 2499–2503. ISSN: 1530-6984, 1530-6992. URL: http://pubs.acs.org/doi/abs/10.1021/nl300665z (visited on 02/07/2014).

Schattschneider, Peter, Cécile Hébert, et al. (2008). "Magnetic circular dichroism in EELS: Towards 10nm resolution". In: *Ultramicroscopy* 108.5, pp. 433–438. ISSN: 03043991. URL: http://linkinghub.elsevier.com/retrieve/pii/S030439910700174X (visited on 02/07/2014).

Schattschneider, Peter, Michael Stöger-Pollach, et al. (2008). "Detection of magnetic circular dichroism on the two-nanometer scale". In: *Phys. Rev. B* 78.10. ISSN: 1098-0121, 1550-235X. URL: http://link.aps.org/doi/10.1103/PhysRevB.78.104413 (visited on 02/07/2014).

Schattschneider, P. et al. (2006). "Detection of magnetic circular dichroism using a transmission electron microscope". In: *Nature* 441.7092, pp. 486–488. ISSN: 0028-0836, 1476-4679. URL: http://www.nature.com/doifinder/10.1038/nature04778 (visited on 02/07/2014).

Stack Overflow. URL: http://stackoverflow.com/questions/tagged/dm-script.

Stöger-Pollach, M. et al. (2011). "EMCD real space maps of Magnetospirillum magnetotacticum". In: *Micron* 42.5, pp. 456–460. ISSN: 09684328. URL: http://linkinghub.elsevier.com/retrieve/pii/S0968432811000047 (visited on 02/07/2014).

The DM-Script database. URL: http://portal.tugraz.at/portal/page/portal/felmi/DM-Script.

Thersleff, Thomas et al. (2015). "Quantitative analysis of magnetic spin and orbital moments from an oxidized iron (1 1 0) surface using electron magnetic circular dichroism". In: *Sci. Rep.* 5, p. 13012. ISSN: 2045-2322. URL: http://www.nature.com/articles/srep13012 (visited on 08/20/2015).

A Details on the tested software and hardware systems

This table includes details on the four GMS installations used in the development of the created code. There are three installations used on the same computer: first an offline installation, then two faux microscope installations. This computer received regular Windows updates throughout its use. The update situation on the real microscope is not known.

	Real microscope	Offline	Faux microscope 1	Faux microscope 2
Time period	2015-08 – 2016-05	2015-08 – 2015-11	2015-11 – 2015-12	2016-01 – 2016-05
GMS	2.31.691.0	2.31.734.0	2.31.734.0	2.32.888.0
Windows	7 x64	7 x64	7 x64	10 x64
CPU	2* Intel® Xeon® E5-2620	Intel® Core™ i5-2540m	Intel® Core™ i5-2540m	Intel® Core™ i5-2540m
GIF	Tridiem®, 863.3KP		Tridiem®, unknown version	Tridiem®, unknown version
CCD resolution	2048x2048		1024x1024	2048x2048

B DM faux microscope install instructions

Disclaimer: I am by no means an authority on this thing, and I can absolutely not be held accountable for any trouble caused by trying to follow these instructions. The software is not mine and the functionality mentioned here is not an official feature — there is no support!

This means that if you do not know how to troubleshoot and fix your installation yourself if it goes wrong, you should avoid using this. It is not like Gatan is going to ignore you if you have a problem — but of course you are expected to test and prove that your problem is not related to running in faux mode first (requiring a second, normal, install).

The information in here is mainly from a helpful person called Bernhard Schaffer, who also gave me the FauxDigiscan.dm3 file. The rest of my knowledge is from experimenting.

1. Install an *online* copy of Digital Micrograph.
   (a) This means you need the *online* license. The license installer will install the license file found in the folder it is run from.
   (b) You probably want to install all parts you have a license for, so that your faux setup is as close as possible to your real microscope.
2. If applicable, install the EDS Acquisition Server.
   (a) In the installer, there is a standard feature selection screen where the category "EDS Clients" holds the parts individual to separate detector systems. Choose the one labeled "Gatan Faux Detector" or similar.
3. Put the file FauxDigiscan.dm3 in the same folder as DigitalMicrograph.exe (by default that would be C:\Program Files\Gatan\).
4. Create a new or edit the existing shortcut to Digital Micrograph so that it passes the `/faux` parameter to the application. Remember that the parameter needs to be outside the citation marks around the path to the application, and separated by a space.
5. The first time you start it you will be presented by a setup dialog asking, among other things, what GIF to simulate. To be able to choose this correctly you need to know the 3-digit name of the thing ("Tridiem" is not specific enough).
6. Acquiring a gain reference for the faux camera has a trick to it, since you can't adjust the intensity. When the procedure asks you to adjust the intensity, note the count you're getting in the picture, then abort and run again with an adjusted target intensity.

By now you should have a working faux microscope install, to the extent that I know to be possible. You should be able to read cameras, run DigiScan, run EELS and EDS acquisitions, Spectrum Imaging and so on — all with imaginary junk data of course, but it works good for testing the software.

One good use of this is to create window layouts for use on the microscope, since you now have all the same palette tools as in the real install. However, this requires you to know how to handle registry entries (that includes knowing how to recover a seriously crashed Windows install, in case you forgot). Comfortable with that? Okay then: the layouts are (for GMS 2.3 at least) stored in HKCU\SOFTWARE\Gatan\DigitalMicrograph\.

Good luck. Do write me if you encounter any technical problems, I want to learn about them and I may be able to help. No promises though.

C Table of DMscript tag types

Type 0	Subtypes, delimited to make sense	Name of type	Size [B]	Size [b]	Comment
		TagGroup			
2		Short	2	16	signed integer
3		Long	4	32	
3		RGB	4		same but different, one byte unused
3		RGBNumber	4		just an alias?
4		UInt16	2	16	
5		UInt32	4	32	
6		Float	4	32	single precision
7		Double	8	64	
7		Number	8	64	alias?, double is standard
8		Boolean	1		
15	0:2_0:7,0:7	Complex	16		double is standard
15	0:2_0:7,0:7	ComplexNumber	16		alias?
15	0:2_0:6,0:6	FloatComplex	8		single precision
15	0:2_0:6,0:6	FloatPoint	8		same but different
15	0:2_0:11,0:11	LongPoint	16		
15	0:2_0:2,0:2	ShortPoint	4		
15	0:3_0:2,0:2,0:2	RGBUint16	6		actually three numbers (contrast RGB)
15	0:4_0:6,0:6,0:6,0:6	FloatRect	16		
15	0:4_0:11,0:11,0:11,0:11	LongRect	32		
15	0:4_0:2,0:2,0:2,0:2	ShortRect	8		
20	4_#	Text	2 /c.	16 /c.	# is number or elements (characters)
20	4_#	String	2 /c.	16 /c.	alias?
20	7_#	Array	8 /e.	64 /e.	double
Arrays corresponding to the rest of the image types, not in the TagGroup documentation					
20	6_#	Array, realimage(4)	4 /e.	32 /e.	single
20	15_0:2_0:7,0:7_#	Array, compleximage(16)	16 /e.		complex doubles
20	15_0:2_0:6,0:6_#	Array, compleximage(8)	8 /e.		complex singles
20	8_#	Array, binaryimage()	1 /e.	8 /e.	Note the weird size.
20	10_#	Array, integerimage(1,0)	1 /e.	8 /e.	unsigned 8-bit integers
20	9_#	Array, integerimage(1,1)	1 /e.	8 /e.	signed, read by ...AsString()
20	4_#	Array, integerimage(2,0)	2 /e.	16 /e.	unsigned shorts, same thing as a string
20	2_#	Array, integerimage(2,1)	2 /e.	16 /e.	signed shorts
20	5_#	Array, integerimage(4,0)	4 /e.	32 /e.	unsigned longs
20	3_#	Array, integerimage(4,1)	4 /e.	32 /e.	signed longs
20	15_3:3_0:10,1:10,2:10_#	Array, rgbimage(3)	3 /e.		Image is called RGB UInt8
20	3_#	Array, rgbimage(4)	4 /e.		Image is called RGB
20	15_3:3_0:4,1:4,2:4_#	Array, rgbimage(6)	6 /e.		Image is called RGB UInt16
20	15_4:4_0:4,1:4,2:4,3:4_#	Array, rgbimage(8)	8 /e.		Image is called RGBA UInt16
Here starts fully undocumented land					
15	0:8_0:5,0:5,0:3,0:3,0:3,0:3,0:3,0:3		32=2*4+6*4		Global:Private:PageSetup:General

D Complete listing of created source code

D.1 Main DMscript code

D.1.1 qE-library.s, The main code file

```
/* q-E STEM library
 *
 * This is a DMscript library containing the functions needed for
 * two-dimensional CCD readout on each pixel of a Spectrum Imaging (SI)
 * acquisition routine in EELS mode, intended for the q-E single-pass
 * STEM EMCD experiment. These are internal functions, they are used by
 * means of the separate script ?.s (not created) that provides a simple
 * user interface. Advanced users are encouraged to play with the code,
 * and please send any ideas for improvement to me, I'll be happy!
 *
 * Beware of the qEInstall() function, it will delete the qE-STEM
 * taggroup. Same goes for qEUninstall(). You can easily backup
 * the complete taggroup with qESaveData(filepath).
 *
 * Requires the SI hookup manager. You would want that one anyway.
 *
 * These functions uses a TagGroup for common variables instead of
 * a class, since the SI hookup system only takes function names.
 * Or perhaps because I haven't had the time to figure it out.
 *
 * Written by Linus Schoenstroem from 2015-10-29 to 2016-
 *
 * This version is outdated and not licensed. Please download the
 * current, correctly licensed version at github.com/linussch
 *
 * Last edit: 2016-04-29
 */

/* Changelog

major.minor.fix[status]
a means alpha (not complete), b means beta (not tested) and x means knowingly broken

The first development versions have a leading a, as they are incomplete at a large
    scale.

0.a0.x 2015-10-29 Created file. Started on sifcaSetup(), ReadCameraToTGArray() and
    WriteTGArrayToDisk().
0.a0.x ?          Finished ReadCameraToTGArray(), created PixelHelloWorld(), started
    on PixelHelloTag()
0.a0.a 2015-11-12 Finished PixelHelloTag()
0.a1.x 2015-11-21 Started on ShowTGArrayAsImage()
0.a1.b 2015-11-23 Finished that
0.a1.0 2015-11-23 Edited sifcaSetup() and others for the data group to be a list, this
     was the first "working" version
0.a2.a 2015-12-07 Added camera settings to sifcaSetup()
0.a2.x 2015-12-08 Added CameraTimeHist() to be used with camera-test-call.s
0.a2.b 2015-12-09 Finished that (added the histogram menu call)
0.a2.0 2015-12-11 Tested on microscope, added sifcaClearData() and found a need for
    sifcaCameraSetup()
0.a3.x 2016-01-29 Started rewrite of ReadCameraToTGArray(), adding timestamps
0.a3.x 2016-02-01 Moved CameraTimeHist() over to the script now named test-camera-
    calls.s (and rewrote it as PlotAndHistFromNumberTags())
0.a3.x 2016-02-01 Continued rewrite of ReadCameraToTGArray(), using
    CameraAcquireInPlace()
0.a3.a 2016-02-02 Finished rewrite of ReadCameraToTGArray(), but did not test it.
    Reorganized the file. Saved this as sifca-library-backup.s
```

```
0.a3.b 2016-02-03 Actually finished the rewrite, there was a bug in yesterdays version. Also rewrote ShowTGArrayAsImage() accordingly. Saved over sifca-library-backup.s
0.a4.x 2016-02-12 Started moving all the tags, started adding the user interface functions, changed the name from SIFCA to EMCD.
0.a4.x 2016-02-16 More tag-moving, added emcdWriteSettings() and emcdReadOrCreateSettings(). Also worked on emcdSetup() and pondered over cleanup and ShowTGArrayAsImage().
0.a4.a 2016-02-18 Added saving an example .dm4 to emcdSetup(). Did a bit of this and a bit of that, I think this version works, but haven't tested at all.
0.a5.a 2016-02-19 Added emcdInstall(), emcdSetOnStartup(), ClickBFDetectorIn(), ClickDetectorOut(), emcdPause(), emcdUninstall(), renamed WriteTGArrayToDisk() -> emcdRunningBackup().
0.a5.a 2016-02-20 Renamed ClickBFDetectorIn() -> ClickDetectorBF() and ReadCameraToTGArray() -> emcdReadout(), replaced ShowTGArrayAsImage() with TagGroupShowArrayTagAsImage() and emcdShowPixel(), changed name of TagGroup "Temp data" to "PerPixelData" and added "PerExperimentData"
0.a5.a 2016-02-21 Ironed out many bugs, including function order. Created TagPathExtractLabel(). Remade directory settings into rootpaths, working but not finished.
0.a6.a 2016-02-26 Swatted a couple bugs, added GetSISurveyResolution(), and put some better error handling in emcdSetup().
0.a7.a 2016-04-28 Renamed everything: function prefix emcd -> qE, taggroup "_EMCD" -> "_LinusSch:qE-STEM". Moved TagPathExtractLabel() and TagGroupShowArrayTagAsImage() to Linus-functions.s. Modified PixelHelloWorld(), removed tag use.
0.a8.a 2016-04-29 Moved PixelHelloWorld() and PixelHelloTag() to si-hookup-manager.s.

*/

// This function runs on every DM startup. SET VERSION NUMBER HERE AS WELL!
void qESetOnDMStartup() {
    taggroup linus
    linus= TagGroupGetOrCreateTagGroup(GetPersistentTagGroup(),"_LinusSch")
    linus.TagGroupSetTagAsString("qE-STEM:About:Installed version","0.a8.a 2016-04-29")
    linus.TagGroupSetTagAsString("qE-STEM:About:Requires Linus-functions","v.6")
    if( ! linus.TagGroupDoesTagExist("Author") )
        linus.TagGroupSetTagAsString("Author","Linus Schoenstroem (a string tag doesn't handle o with dots)")
    if( ! linus.TagGroupDoesTagExist("Author email (may provide support if you ask nicely)") )
        linus.TagGroupSetTagAsString("Author email (may provide support if you ask nicely)","linus.schonstrom@tele2.se")
}

// I declare all functions here so that I can reference them out of order
// Oops, that doesn't work... I get "undeclared variable" on the second line of these. I'll just reorder the functions for now, and solve this later.
/*
void qEReadout()
void ClickDetectorBF()
void ClickDetectorOut()
void qEPause()
void qERunningBackup()
void qESetup()
void qEReadSettings(number &exposure, number &xBin, number &yBin, string &processing, number &areaT, number &areaL, number &areaB, number &areaR, string &savedir, string &scratchdir)
void qEWriteSettings(number exposure, number xBin, number yBin, string processing, number areaT, number areaL, number areaB, number areaR, string savedir, string scratchdir)
void qESetHookups()
void qEClearData()
void qEShowPixel(number i)
void qEDataToDM4()
```

```
92  void qELoadData(string filepath)
93  void qESaveData(string filepath)
94  void qEInstall(string detector_exe_path,string default_save_directory, string
        default_scratch_directory)
95  void qEUninstall()
96  void PixelHelloCamera()
97  */
98
99  /*
100  * ****************************
101  * Core, functions for acquistion
102  * ****************************
103  */
104
105 // Reads the camera and puts the readout in a TagGroup array
106 // NB: This is the key part, but it is only readout and assumes everything is in order
107 // To be hooked up at Pixel Beginning
108 void qEReadout()
109     {
110     number t_start = GetHighResTickCount()
111
112     number i = SIGetAcquisitionPixelNumber()
113     OpenAndSetProgressWindow("q-E STEM acquisition","Acquiring pixel "+i,"")
114
115     taggroup runtime, data, thispixel
116     TagGroupGetTagAsTagGroup(GetPersistentTagGroup(),"_LinusSch:qE-STEM:Runtime
            variables (please don't touch!)",runtime)
117     TagGroupGetTagAsTagGroup(GetPersistentTagGroup(),"_LinusSch:qE-STEM:PerPixelData",
            data)
118     thispixel= NewTagGroup()
119     data.TagGroupInsertTagAsTagGroup(i,thispixel)
120
121     thispixel.TagGroupSetTagAsNumber("Pixel number",i)
122     thispixel.TagGroupSetTagAsNumber("Timestamp_1_start",t_start)
123
124     number camid,exposure,xBin,yBin,proc_enum,areaT,areaL,areaB,areaR
125     runtime.TagGroupGetTagAsNumber("Camera ID",camid)
126     runtime.TagGroupGetTagAsNumber("Exposure time (s)",exposure)
127     runtime.TagGroupGetTagAsShortPoint("Binning",xBin,yBin)
128     runtime.TagGroupGetTagAsNumber("Processing",proc_enum)
129     runtime.TagGroupGetTagAsShortRect("Camera area",areaT,areaL,areaB,areaR)
130
131     image img:= CameraCreateImageForAcquire(camid,xBin,yBin,proc_enum,areaT,areaL,areaB
            ,areaR)
132
133     thispixel.TagGroupSetTagAsNumber("Timestamp_2_acquire",GetHighResTickCount())
134     CameraAcquireInPlace(camid,img,exposure,xBin,yBin,proc_enum,areaT,areaL,areaB,areaR
            )
135     thispixel.TagGroupSetTagAsNumber("Timestamp_3_end",GetHighResTickCount())
136     thispixel.TagGroupSetTagAsArray("data",img)
137     }
138
139 /* These two wait functions does not work, but are kept because this bug may be solved
        at a later date
140 // Waits for the BF detector to be inserted
141 // To be hooked up at Drift Correction Beginning
142 void WaitForBFDetectorIn()
143     {
144     number b
145     while( !(ShiftDown() && OptionDown() && SpaceDown()) ){
146         IFGetConfiguration().TagGroupGetTagAsBoolean("Extra detector inserted",b) //Bug
                : this tag does not actually update, so this function does not work
147         if(b)
```

```
148             break
149         else{
150             OpenAndSetProgressWindow("q-E STEM Acquisition","Waiting for BF detector","
                    to be inserted")
151             sleep(.1)
152             }
153         }
154     }
155 
156 // Waits for the BF detector to be retracted
157 // To be hooked up at Drift Correction End
158 void WaitForBFDetectorOut()
159     {
160     number b
161     while( !(ShiftDown() && OptionDown() && SpaceDown()) ){
162         IFGetConfiguration().TagGroupGetTagAsBoolean("Extra detector inserted",b) //Bug
                : this tag does not actually update, so this function does not work
163         if(!b)
164             break
165         else{
166             OpenAndSetProgressWindow("q-E STEM Acquisition","Waiting for BF detector","
                    to be retracted")
167             sleep(.1)
168             }
169         }
170     }
171 (end of bad functions) */
172 
173 // Calls the external ClickDetectorButton.exe for inserting the BF detector
174 // To be hooked up at Drift Correction Beginning
175 void ClickDetectorBF()
176     {
177     OpenAndSetProgressWindow("q-E STEM Acquisition","Clicking to insert BF","detector.
            DON'T TOUCH MOUSE")
178     number r
179     string exepath
180     TagGroupGetTagAsString(GetPersistentTagGroup(),"_LinusSch:qE-STEM:Admin:Path to
            ClickDetectorButton.exe",exepath)
181     r= LaunchExternalProcess(exepath+" \"BF\"")
182     }
183 // Oh crap how do I stop SI from continuing until the detector is in without freezing
        DM
184 // Oh crap I do that by calling
185 //      SIInvokeButton("Start/Stop",2) for pause
186 //      SIInvokeButton("Start/Stop",3) for resume
187 // from a separate thread...
188 // Let's just try to ignore the problem for now. With luck, detector insertion/
        retraction can be triggered between this hookup script and the next, or at least
        is quick enough that drift correction as well as acquired data though damaged will
         work. A test with clicking the stop button seemed to indicate that no, the click
        does not register between the hookup scripts, but it was not easy to see and
        should probably be tested on the real thing anyway.
189 
190 // Calls the external ClickDetectorButton.exe for retracting the detector
191 // To be hooked up at Drift Correction End
192 void ClickDetectorOut()
193     {
194     OpenAndSetProgressWindow("q-E STEM Acquisition","Clicking to insert BF","detector.
            DON'T TOUCH MOUSE")
195     number r
196     string exepath
197     TagGroupGetTagAsString(GetPersistentTagGroup(),"_LinusSch:qE-STEM:Admin:Path to
            ClickDetectorButton.exe",exepath)
```

```
198     r= LaunchExternalProcess(exepath+" \"out\"")
199     }
200 
201 // A simple pause (just for testing for now)
202 void qEPause()
203     sleep(5)
204 
205 // Saves the qE-STEM taggroup to a file
206 // Belongs in the group of functions handling the data, needs to be here for now since
      qERunningBackup() uses it
207 void qESaveData(string filepath)
208     {
209     taggroup tg
210     TagGroupGetTagAsTagGroup(GetPersistentTagGroup(),"_LinusSch:qE-STEM",tg)
211     tg.TagGroupSaveToFile(filepath)
212     }
213 
214 // Saves data to two alternating files while running (this will be moved to a
      background thread when I learn how to do that)
215 // To be hooked up at Drift Correction Beginning
216 void qERunningBackup()
217     {
218     string filepath, scratchdir, file
219     taggroup runtime
220     TagGroupGetTagAsTagGroup(GetPersistentTagGroup(),"_LinusSch:qE-STEM:Runtime
          variables (please don't touch!)",runtime)
221     runtime.TagGroupGetTagAsString("Scratch directory",scratchdir)
222     runtime.TagGroupGetTagAsString("File",file)
223 
224     if( file == "A" )
225         runtime.TagGroupSetTagAsString("File","B")
226     else
227         runtime.TagGroupSetTagAsString("File","A")
228 
229     qESaveData(scratchdir+"qE-STEM-running-backup-"+file)
230     }
231 
232 // Reading the current or default settings
233 // Used both by qESetup at experiment start and by the user interface
234 void qEReadSettings(number &exposure, number &xBin, number &yBin, string &processing,
      number &areaT, number &areaL, number &areaB, number &areaR, string &savedir,
      string &scratchdir)
235     {
236     taggroup settings= TagGroupGetOrCreateTagGroup(GetPersistentTagGroup(),"_LinusSch:
          qE-STEM:Settings")
237     taggroup defaults= TagGroupGetOrCreateTagGroup(GetPersistentTagGroup(),"_LinusSch:
          qE-STEM:Admin:Default settings")
238     if( ! settings.TagGroupGetTagAsNumber("Exposure time (s)",exposure) )
239         defaults.TagGroupGetTagAsNumber("Exposure time (s)",exposure)
240     if( ! settings.TagGroupGetTagAsShortPoint("Binning",xBin,yBin) )
241         defaults.TagGroupGetTagAsShortPoint("Binning",xBin,yBin)
242     if( ! settings.TagGroupGetTagAsString("Processing",processing) )
243         defaults.TagGroupGetTagAsString("Processing",processing)
244     if( ! settings.TagGroupGetTagAsShortRect("Camera area",areaT,areaL,areaB,areaR) )
245         defaults.TagGroupGetTagAsShortRect("Camera area",areaT,areaL,areaB,areaR)
246     if( ! settings.TagGroupGetTagAsString("Save directory",savedir) )
247         defaults.TagGroupGetTagAsString("Save directory",savedir)
248     if( ! settings.TagGroupGetTagAsString("Scratch directory",scratchdir) )
249         defaults.TagGroupGetTagAsString("Scratch directory",scratchdir)
250     }
251 
252 // Gets the SI survey resolution, if possible
253 void GetSISurveyResolution(number &sx, number &sy)
```

```
    {
    if( SIGetMode() == "2D Array" )
        {
        if( ! SIGetFieldValue(101,sx) )
            Throw("Unexpected error: could not get survey resolution.")
        if( ! SIGetFieldValue(111,sy) )
            Throw("Unexpected error: could not get survey resolution.")
        }
    else if( SIGetMode() == "LineScan" )
        {
        sy= 1
        if( ! SIGetFieldValue(201,sx) )
            Throw("Unexpected error: could not get survey resolution.")
        }
    else if( SIGetMode() == "MultiPoint" )
        Throw("SI is in MultiPoint mode, survey resolution is undefined")
    else
        Throw("Unexpected error: did not recognize SI mode.")
    }

// Setup for acquisition, this function creates the runtime variables taggroup
// used by the above functions. Also runs CameraPrepareForAcquire. Settings are
// copied from the settings taggroup created by qEWriteSetup() as found below.
// To be hooked up at SI Beginning
void qESetup()
    {
    taggroup qetags,data,runtime
    TagGroupGetTagAsTagGroup(GetPersistentTagGroup(),"_LinusSch:qE-STEM",qetags)
    // Check that the data TagList exists, create it if not

    // Creating the tags used at runtime...
    runtime= TagGroupGetOrCreateTagGroup(qetags,"Runtime variables (please don't touch
        !)")
    runtime.TagGroupSetTagAsString("File","A")
    number camID= IFGetFilterCameraID()
    runtime.TagGroupSetTagAsShort("Camera ID",camID)
    CameraSetActiveCameraID(camID)
    // ...by reading settings...
    number exposure,xBin,yBin,areaT,areaL,areaB,areaR
    string processing,savedir,scratchdir
    qEReadSettings(exposure,xBin,yBin,processing,areaT,areaL,areaB,areaR,savedir,
        scratchdir)
    // ...creating the experiment directories...
    string idnote
    if( ! GetString("Enter an identification note for filenames:","",idnote) )
        idnote= ""
    string datestamp= DateStamp() // BUG: this datestamp function returns strings with
        slashes in them on systems with certain locales, custom datestamp function
        planned
    datestamp.Replace(" ","_")
    datestamp.Replace(":","")
    savedir += "\\" + datestamp + "_" + idnote
    scratchdir += "\\" + datestamp + "_" + idnote
    CreateDirectory(savedir)
    CreateDirectory(scratchdir)
    qetags.TagGroupSetTagAsString("PerExperimentData:Save location",savedir)
    qetags.TagGroupSetTagAsString("PerExperimentData:Scratch location",scratchdir)
    savedir += "\\"
    scratchdir += "\\"
    // ...parsing the processing parameter...
    number proc_enum
    if( StringToLower(processing) == "unprocessed" )
        proc_enum= CameraGetUnprocessedEnum()
```

```
313     else if( StringToLower(processing) == "dark subtracted" )
314         proc_enum= CameraGetDarkSubtractedEnum()
315     else if( StringToLower(processing) == "gain normalized" )
316         proc_enum= CameraGetGainNormalizedEnum()
317     else
318         {
319         SIInvokeButton("Start/Stop",0) //This stops the acquisition, an error in a
                hookup script will not
320         Throw("qESetup() did not recognise the setting for camera readout processing")
321         }
322     // ...and, finally, storing all that.
323     runtime.TagGroupSetTagAsNumber("Exposure time (s)",exposure)
324     runtime.TagGroupSetTagAsShortPoint("Binning",xBin,yBin)
325     runtime.TagGroupSetTagAsNumber("Processing",proc_enum)
326     runtime.TagGroupSetTagAsShortRect("Camera area",areaT,areaL,areaB,areaR)
327     runtime.TagGroupSetTagAsString("Save directory",savedir)
328     runtime.TagGroupSetTagAsString("Scratch directory",scratchdir)
329
330     // Prepare camera, save example .dm4 and save the resolution to a tag
331     // In future should save all that is needed to tags and save the example .dm4 only
            to scratchdir
332     CameraPrepareForAcquire(camID)
333     image img:= CameraCreateImageForAcquire(camid,xBin,yBin,proc_enum,areaT,areaL,areaB
            ,areaR)
334     CameraAcquireInPlace(camid,img,exposure,xBin,yBin,proc_enum,areaT,areaL,areaB,areaR
            )
335     img.SaveAsGatan(savedir+"ExamplePictureForCalibrationData.dm4")
336     qetags.TagGroupSetTagAsShortPoint("PerExperimentData:Capture resolution",img.
            ImageGetDimensionSize(0),img.ImageGetDimensionSize(1))
337     // Saving the survey image as a separate .dm4, and its resolution to another tag
338     //img.DeleteImage()
339     //img= SIGetSurveyImage()
340     //img.SaveAsGatan(savedir+"ExperimentSurveyImage.dm4")
341     GetImageFromID(SIGetSurveyImageID()).SaveAsGatan(savedir+"ExperimentSurveyImage.dm4
            ")
342     // Saving the survey resolution to a tag
343     number sx,sy
344     try
345         GetSISurveyResolution(sx,sy)
346     catch
347         {
348         sx= 0
349         sy= 0
350         ShowAlert(GetExceptionString()+"\n\nTag is set to zero. You have to save the
                survey resolution MANUALLY!",1)
351         break
352         }
353     qetags.TagGroupSetTagAsShortPoint("PerExperimentData:Survey resolution",sx,sy)
354     }
355
356 // Cleanup: removes runtime variables, saves data, clears scratch (preferrably not if
        any errors occurred), optionally clears data, optionally clears hookups
357 void qECleanup()
358     {
359     // Read in savedir and scratchdir first
360     // Clear runtime variables
361     // Save data
362     // Ask user to check data and decide whether to keep the scratch (the scratch
            should be kept if errors occurred, it includes runtime variables and in the
            future also the test .dm4 and other things useful for debugging
363     // User decides to clear or not to clear data
364     // User decides to clear hookups or run another q-E STEM experiment
365     }
```

```
366
367 /*
368  * ************************
369  * User interface functions
370  * ************************
371  */
372
373 // Writing the settings (to call when clicking OK in the setup dialog)
374 // This function allows invalid settings to be written, error checking and handling is
        done in the dialog script
375 void qEWriteSettings(number exposure, number xBin, number yBin, string processing,
        number areaT, number areaL, number areaB, number areaR, string savedir, string
        scratchdir)
376     {
377     taggroup settings= TagGroupGetOrCreateTagGroup(GetPersistentTagGroup(),"_LinusSch:
            qE-STEM:Settings")
378     settings.TagGroupSetTagAsNumber("Exposure time (s)",exposure)
379     settings.TagGroupSetTagAsShortPoint("Binning",xBin,yBin)
380     settings.TagGroupSetTagAsString("Processing",processing)
381     settings.TagGroupSetTagAsShortRect("Camera area",areaT,areaL,areaB,areaR)
382     settings.TagGroupSetTagAsString("Save directory",savedir)
383     settings.TagGroupSetTagAsString("Scratch directory",scratchdir)
384     }
385
386 // Setting the appropriate SI hookups, this (and qECleanup()) relies on SI Hookup
        Manager by Linus Schoenstroem
387 void qESetHookups()
388     {
389     sihuClean()
390     sihuAdd("Spectrum Image:Beginning","qESetup")
391     sihuAdd("Pixel:Beginning","qEReadout")
392     sihuAdd("Correction:Beginning","ClickDetectorBF")
393     //sihuAdd("Correction:Beginning","qERunningBackup")
394     sihuAdd("Correction:End","ClickDetectorOut")
395     sihuAdd("Spectrum Image:End","qECleanup")
396     sihuPrint()
397     }
398
399 /*
400  * ***************************
401  * Functions handling the data
402  * ***************************
403  */
404
405 // Clear data
406 void qEClearData()
407     {
408     taggroup data= TagGroupGetOrCreateTagList(GetPersistentTagGroup(),"_LinusSch:qE-
            STEM:PerPixelData") //This fails when the tag exists but is not a taglist
409     data.TagGroupDeleteAllTags()
410     data= TagGroupGetOrCreateTagGroup(GetPersistentTagGroup(),"_LinusSch:qE-STEM:
            PerExperimentData")
411     data.TagGroupDeleteAllTags()
412     }
413
414 // Shows the camera readout from one pixel
415 void qEShowPixel(number i) {
416     taggroup qetags, pixeldata
417     TagGroupGetTagAsTagGroup(GetPersistentTagGroup(),"_LinusSch:qE-STEM",qetags)
418     qetags.TagGroupGetTagAsTagGroup("PerPixelData:["+i+"]",pixeldata)
419     number size_x, size_y
420     qetags.TagGroupGetTagAsShortPoint("PerExperimentData:Capture resolution",size_x,
            size_y)
```

```
421     pixeldata.TagGroupShowArrayTagAsImage("data",size_x,size_y) // This does not take
            type into account, which works for viewing but should not be used to extract
            the data
422 }
423
424 // Consolidates data from TagGroup arrays into a single 4-dim
425 // image for use inside DigitalMicrograph
426 void qEDataToDM4()
427     {
428     }
429
430 // Loads a saved qE-STEM taggroup
431 // BUG: includes version info as "installed" :P
432 void qELoadData(string filepath)
433     {
434     taggroup global,load
435     global= GetPersistentTagGroup()
436     global.TagGroupDeleteTagWithLabel("_LinusSch:qE-STEM")
437     load= NewTagGroup()
438     load.TagGroupLoadFromFile(filepath)
439     global.TagGroupSetTagAsTagGroup("_LinusSch:qE-STEM",load)
440     }
441
442 /*
443  * ***********************************
444  * Functions for install and uninstall
445  * ***********************************
446  */
447
448 // The install function, intended to be run after installing this library or when
        changes have been made.
449 // CAUTION: it deletes things.
450 // This function sets up the tag structure used by this library, including any install
        specific things
451 // like paths to external programs and default paths. Also includes preset default
        acquisition settings.
452 void qEInstall(string detector_exe_path,string default_save_directory, string
        default_scratch_directory)
453     {
454     taggroup global
455     global= GetPersistentTagGroup()
456     global.TagGroupDeleteTagWithLabel("_LinusSch:qE-STEM")
457
458     global.TagGroupSetTagAsTagGroup("_LinusSch:qE-STEM:Settings",NewTagGroup())
459     global.TagGroupSetTagAsTagGroup("_LinusSch:qE-STEM:PerPixelData",NewTagList())
460     global.TagGroupSetTagAsTagGroup("_LinusSch:qE-STEM:PerExperimentData",NewTagGroup()
            )
461     global.TagGroupSetTagAsString("_LinusSch:qE-STEM:Admin:Path to ClickDetectorButton.
            exe",detector_exe_path)
462     qESetOnDMStartup()
463
464     taggroup defaults
465     defaults= NewTagGroup()
466     number exposure,xBin,yBin,areaT,areaL,areaB,areaR
467     string processing
468
469         exposure= .1
470         xBin= 1
471         yBin= 16
472         processing= "Unprocessed"
473         areaT= 768
474         areaB= 1280
475         areaL= 0
```

```
476         areaR= 2048
477
478     defaults.TagGroupSetTagAsNumber("Exposure time (s)",exposure)
479     defaults.TagGroupSetTagAsShortPoint("Binning",xBin,yBin)
480     defaults.TagGroupSetTagAsString("Processing",processing)
481     defaults.TagGroupSetTagAsShortRect("Camera area",areaT,areaL,areaB,areaR)
482     defaults.TagGroupSetTagAsString("Save directory",default_save_directory)
483     defaults.TagGroupSetTagAsString("Scratch directory",default_scratch_directory)
484
485     global.TagGroupSetTagAsTagGroup("_LinusSch:qE-STEM:Admin:Default settings",defaults
            )
486     }
487 // ...and uninstall
488 void qEUninstall()
489     {
490     taggroup global
491     global= GetPersistentTagGroup()
492     global.TagGroupDeleteTagWithLabel("_LinusSch:qE-STEM")
493     }
494
495 /*
496  * ******************************
497  * Functions for testing purposes
498  * ******************************
499  */
500
501 // PixelHelloCamera, reads the camera but does not put the data anywhere
502 void PixelHelloCamera()
503     {
504     number i, t
505     i= SIGetAcquisitionPixelNumber()
506     Result("Pixel number "+i+" ...")
507     t= GetHighResTickCount()
508
509     image img= CameraAcquire(IFGetFilterCameraID())
510
511     t= CalcHighResSecondsBetween(t,GetHighResTickCount())
512     Result("was acquired in "+Format(t,"%11.9f")+" seconds\n")
513     }
514
515 /*
516  * ******************************
517  * And finally, calling the function to run on startup (scripts installed as libraries
          are executed at startup)
518  * ******************************
519  */
520 qESetOnDMStartup()
```

D.1.2 si-hookup-manager.s, A crucial part

```
1  /* SI hookup manager
2   *
3   * A small library written by Linus Schoenstroem, largely as
4   * outlined in Gatan's documentation.
5   *
6   * This version is outdated and not licensed. Please download the
7   * current, correctly licensed version at github.com/linussch
8   *
9   * Requires string.Replace() and PrintTagGroup()
10  *
11  * Created: 2015-11-10
12  * Last edited: 2016-04-29
```

```
 *
 * This should really have more error and sanity checking.
 *
 */

/* Changelog

major.minor.fix[status]
a means alpha (not complete), b means beta (not tested) and x means knowingly broken

0.1.0b 2015-11-10 Created first version
0.1.0  2015-11-23 Verified working
0.2.0x 2015-11-27 Renamed functions, started replacing unnecessary wrapper with useful
     script file execution
0.2.0b 2015-11-30 Finished the script file execution system
0.3.0a 2015-12-04 Added a primitive sihuPrint(), requiring PrintTagGroup() v.1.0.0a3
0.3.0  2015-12-16 Solved bug in sihuPrint(), updated dependencies, added sihuBrowse()
0.3.1b 2016-02-04 Added a blank line in sihuPrint(), added some TODO notes
0.4.0  2016-04-28 Write version info to global tags on startup, StringReplace() ->
    Replace(), removing beta status
0.5.0b 2016-04-29 Moved PixelHelloWorld() and PixelHelloTag() from qE-library.s to
    here

*/

// This function runs on every DM startup. SET VERSION NUMBER HERE AS WELL!
void sihuSetOnDMStartup() {
    taggroup linus
    linus= TagGroupGetOrCreateTagGroup(GetPersistentTagGroup(),"_LinusSch")
    linus.TagGroupSetTagAsString("SI Hookup manager version","0.5.0b 2016-04-29")
    if( ! linus.TagGroupDoesTagExist("Author") )
        linus.TagGroupSetTagAsString("Author","Linus Schoenstroem (a string tag doesn't
             handle o with dots)")
    if( ! linus.TagGroupDoesTagExist("Author email (may provide support if you ask
        nicely)") )
        linus.TagGroupSetTagAsString("Author email (may provide support if you ask
            nicely)","linus.schonstrom@tele2.se")
}

// Printing the state of things
void sihuPrint()
    {
    Result("\n")
    PrintTagGroup("SI:Acquisition:Scripts:Control",3)
    try
        PrintTagGroup("User:SI Hookup Manager:Paths")
    catch
        {
        Result("\nNo paths specified.")
        break
        }
    }

// Managing the tags directly
void sihuBrowse()
    {
    taggroup sicontrol
    GetPersistentTagGroup().TagGroupGetTagAsTagGroup("SI:Acquisition:Scripts:Control",
        sicontrol)
    // This browser window function doesn't pause script execution, normally a feature
        but
    // it becomes a bug when used with the simple CLI
```

```
    sicontrol.TagGroupOpenBrowserWindow("Browsing tags at SI:Acquisition:Scripts:
        Control",0)
    }

// Cleanup
// TODO: Check that it is a valid hookup tag, no confusing the user
void sihuRemove(string toremove)
    {
    taggroup sicontrol, list
    sicontrol= GetPersistentTagGroup().TagGroupGetOrCreateTagGroup("SI:Acquisition:
        Scripts:Control")
    if( sicontrol.TagGroupGetTagAsTagGroup(toremove,list) )
        list.TagGroupDeleteAllTags()
    }
void sihuClean()
    {
    sihuRemove("Correction:Beginning")
    sihuRemove("Correction:End")
    sihuRemove("Pixel:Beginning")
    sihuRemove("Pixel:End")
    sihuRemove("Spectrum Image:Beginning")
    sihuRemove("Spectrum Image:End")
    sihuRemove("Spectrum Image:Pause")
    sihuRemove("Spectrum Image:Restart")
    }

// Adds a hookup script.
// TODO: Sanity check - duplicate?
void sihuAdd(string place,string functionname)
    {
    taggroup sicontrol, list
    sicontrol= GetPersistentTagGroup().TagGroupGetOrCreateTagGroup("SI:Acquisition:
        Scripts:Control")

    if(place == "Correction:Beginning" || place == "Correction:End" || \
           place == "Pixel:Beginning" || place == "Pixel:End" || \
           place == "Spectrum Image:Beginning" || place == "Spectrum Image:End" || \
           place == "Spectrum Image:Pause" || place == "Spectrum Image:Restart" )
        list= sicontrol.TagGroupGetOrCreateTagList(place)
    else
        Throw("You specified a tag not known as a SI hookup script tag")

    if(!DoesFunctionExist(functionname))
        Throw("The specified function name doesn't exist. Check that you did not
            include the parentheses.")

    list.TagGroupInsertTagAsString(list.TagGroupCountTags(),functionname)
    }

// Replaces all hookup scripts (at given point) with a wrapper with a generic name.
// Said wrapper executes a script file given by a path in a tag in the
// user persistent group, which is also set by this function.
// TODO: overloaded version to use already stored path
void sihuUseFile(string place, string filepath)
    {
    if(place == "Correction:Beginning" || place == "Correction:End" || \
           place == "Pixel:Beginning" || place == "Pixel:End" || \
           place == "Spectrum Image:Beginning" || place == "Spectrum Image:End" || \
           place == "Spectrum Image:Pause" || place == "Spectrum Image:Restart" )
        {}
    else
        Throw("You specified a tag not known as a SI hookup script tag")

```

```
126     taggroup sicontrol, list, paths
127     sicontrol= GetPersistentTagGroup().TagGroupGetOrCreateTagGroup("SI:Acquisition:
            Scripts:Control")
128     list= sicontrol.TagGroupGetOrCreateTagList(place)
129     paths= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup Manager:
            Paths")
130
131     string tagname, wrappername
132     tagname= place
133     tagname.Replace(":"," ")
134     tagname += " Filepath"
135     wrappername= place
136     wrappername.Replace(":","")
137     wrappername.Replace(" ","")
138     wrappername += "RunFile"
139
140     sihuRemove(place)
141     list= sicontrol.TagGroupGetOrCreateTagList(place)
142     list.TagGroupInsertTagAsString(0,wrappername)
143     paths.TagGroupSetTagAsString(tagname,filepath)
144     }
145
146 // The wrapper scripts
147 void CorrectionBeginningRunFile()
148     {
149     string filepath
150     taggroup paths= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup
            Manager:Paths")
151     paths.TagGroupGetTagAsString("Correction Beginning Filepath",filepath)
152     ExecuteScriptFile(filepath)
153     }
154 void CorrectionEndRunFile()
155     {
156     string filepath
157     taggroup paths= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup
            Manager:Paths")
158     paths.TagGroupGetTagAsString("Correction End Filepath",filepath)
159     ExecuteScriptFile(filepath)
160     }
161 void PixelBeginningRunFile()
162     {
163     string filepath
164     taggroup paths= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup
            Manager:Paths")
165     paths.TagGroupGetTagAsString("Pixel Beginning Filepath",filepath)
166     ExecuteScriptFile(filepath)
167     }
168 void PixelEndRunFile()
169     {
170     string filepath
171     taggroup paths= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup
            Manager:Paths")
172     paths.TagGroupGetTagAsString("Pixel End Filepath",filepath)
173     ExecuteScriptFile(filepath)
174     }
175 void SpectrumImageBeginningRunFile()
176     {
177     string filepath
178     taggroup paths= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup
            Manager:Paths")
179     paths.TagGroupGetTagAsString("Spectrum Image Beginning Filepath",filepath)
180     ExecuteScriptFile(filepath)
181     }
```

```
void SpectrumImageEndRunFile()
    {
    string filepath
    taggroup paths= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup
        Manager:Paths")
    paths.TagGroupGetTagAsString("Spectrum Image End Filepath",filepath)
    ExecuteScriptFile(filepath)
    }
void SpectrumImagePauseRunFile()
    {
    string filepath
    taggroup paths= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup
        Manager:Paths")
    paths.TagGroupGetTagAsString("Spectrum Image Pause Filepath",filepath)
    ExecuteScriptFile(filepath)
    }
void SpectrumImageRestartRunFile()
    {
    string filepath
    taggroup paths= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup
        Manager:Paths")
    paths.TagGroupGetTagAsString("Spectrum Image Restart Filepath",filepath)
    ExecuteScriptFile(filepath)
    }

// PixelHelloWorld
void PixelHelloWorld()
    {
    number i= SIGetAcquisitionPixelNumber()
    Result("Pixel number "+i+" says 'Thankyou Sofia'\n")
    }

// PixelHelloTag
void PixelHelloTag()
    {
    number i= SIGetAcquisitionPixelNumber()
    taggroup data= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"SI Hookup
        Manager:Hello from the pixels")
    data.TagGroupInsertTagAsString(i,"Pixel number "+i+" says 'Thankyou Sofia'")
    }

// Running the startup function
sihuSetOnDMStartup()
```

D.2 Supporting DMscript code

D.2.1 sihu-simple-ui.s

```
// Essentially a miniature CLI for the SI Hookup manager
//
// Written by Linus Schoenstroem on 2016-02-04
//
// This version is outdated and not licensed. Please download the
// current, correctly licensed version at github.com/linussch
//
// There is one known bug: it doesn't wait for you to finish browsing
// before reopening the dialog... the *modal* dialog :(

string i = "print"
while( !(ShiftDown() && OptionDown() && SpaceDown()) )
    {
    if(!GetString(\
```

```
15              "Functions available:\n"+\
16              "exit, q\n"+\
17              "print\n"+\
18              "clean\n"+\
19              "browse\n"+\
20              "add( \"HookupPath\", \"functionname\" )\n"+\
21              "remove( \"HookupPath\" )\n"+\
22              "usefile( \"HookupPath\", \"FilePath\" )",i,i))
23      exit(0)
24
25  if(i=="exit" || i=="q" || i==":q")
26      exit(0)
27
28  else if(i=="print"||i=="clean"||i=="browse")
29      ExecuteScriptString("sihu"+i+"()")
30
31  else if(i.len() > 8)  //Can't extract the substrings to check if input isn't long
        enough
32      {
33      if( i.left(4)=="add(" || i.left(7)=="remove(" || i.left(8)=="usefile(" )
34          {
35          try
36              ExecuteScriptString("sihu"+i)
37          catch
38              {
39              ShowAlert(GetExceptionString(),1)
40              break
41              }
42          }
43      else
44          ShowAlert("Invalid command",1)
45      }
46
47  else
48      ShowAlert("Invalid command",1)
49  }
```

D.2.2 qE-install-myX220.s

```
1  // Example of installation tasks for the q-E STEM library.
2  string root= "D:\\Linus\\Documents\\_faktiska dokument\\Skolarbete\\Exjobb\\tests\\"
3  string exepath= "C:\\Users\\Linus\\Linus-temp\\ClickDetectorButton.exe"
4  qEInstall(exepath,root+"save",root+"scratch")
5  GetPersistentTagGroup().TagGroupSetTagAsShortRect("_LinusSch:qE-STEM:Admin:Default
       settings:Camera area",448,0,576,1024)
6  PrintTagGroup("_LinusSch:qE-STEM",-3)
```

D.2.3 PrintTagGroup.s

```
1  /*
2   * Prints all tags to the result window
3   *
4   * void PrintTagGroup( taggroup tg, string tagpath, number max_depth )
5   *
6   * All arguments optional:
7   *  - without argument max_depth, uses 0 (prints only top level)
8   *  - without argument tagpath, prints from the root of the specified taggroup
9   *  - without argument tg, prints from the persistent taggroup
10  *
11  * As a shortcut for the lazy prepending "User:" to a tagpath when called
```

```
 * without a specified taggroup prints from the user persistent taggroup.
 *
 * A negative max_depth uses infinite depth (prints all levels).
 *
 * Written by Linus Schoenstroem from 2015-10-20 onwards
 * Last edited: 2016-02-02
 *
 * This version is outdated and not licensed. Please download the
 * current, correctly licensed version at github.com/linussch
 *
 * A known problem is that not all TagTypes can be recognized and extracted
 * A known bug is that when printing a large amount of tags, some characters disappear
       from output
 */

/* When you script it may be advisable to use the "internal" function
 * PrintTagGroupRecursing() directly, as this allows you to direct the
 * output string to wherever you want. This function needs you to initialize
 * its depth counter as a number set to zero, look at how it is called in
 * the most simple case PrintTagGroup( taggroup tg ) below for an example.
 */

/* Changelog

major.minor.fix[status]
a means alpha (not complete), b means beta (not tested) and x means knowingly broken

           2015-10-20 The beginning. I think.
(many undocumented development versions)
0.?.?      2015-11-21 No idea what the number on this one is, but it is working
1.0.0a     2015-12-01 Works as a function, converted to bulk printing, prints type
     string of unreadable tags
1.0.0a2    2015-12-02 Added string tagpath functionality, negative as a short for
     infinite depth
1.0.0a3    2015-12-02 Added string call version with headers
1.0.0a3    2015-12-03 Updated comments, description and example script
1.0.0a3    2015-12-04 Added numbering of the alpha versions...
1.0.0a4    2015-12-15 Removed label "Global" from string call, updated help, added
     printing persistent group when none specified

*/

void PrintTagGroupRecursing( taggroup tg, number max_depth, number &depth, string &
     output)
    {
    number count = tg.TagGroupCountTags()
    number islist = tg.TagGroupIsList()
    number i

    string cr = "\n"
    for( i = 0; i < depth; i++)
        cr= cr+": "

    if(count==0)
        output += islist ? (cr+": <empty taglist>") : (cr+": <empty taggroup>")
    else
        {
        for( i = 0; i < count; i++ )
            {
            output += (cr)
            string label= islist ? "["+i+"] " : tg.TagGroupGetTagLabel(i)+"\t" //More
                intelligent aligning could make output prettier
            number type= tg.TagGroupGetTagType(i,0)
```

```
69
70              if( type == 0 ) {
71                  // tag is a TagGroup
72                  output += ("+ "+label )
73                  if( max_depth > depth ) {
74                      taggroup tg2
75                          tg.TagGroupGetIndexedTagAsTagGroup(i,tg2)
76                      depth += 1
77                      PrintTagGroupRecursing(tg2,max_depth,depth,output)
78                      depth -= 1
79                      }
80                  continue
81                  }
82
83              else if( type == 20 && (tg.TagGroupGetTagType(i,1) == 6 || tg.
                     TagGroupGetTagType(i,1) == 7) ){
84                  // tag is an array
85                  // TODO: recognise more array types, print type and number of elements
86                  output += ("- "+label+"= "+"An array, can only be read if you know the
                         dimensions")
87                  continue
88                  }
89
90              else {
91                  // tag should be printable as a string
92                  // TODO: not print overly long strings
93                  string value
94                  number r= tg.TagGroupGetIndexedTagAsString( i, value ) //Indexes work
                         also with labeled tags
95                  if(r)
96                      output += ("- "+label+"= "+value)
97                  else {
98                      string typestr
99                      number typelength = tg.TagGroupGetTagTypeLength(i)
100                     number i2
101                     for( i2 = 0; i2 < typelength ; i2++)
102                         typestr += tg.TagGroupGetTagType( i, i2 ) + ":"
103                     typestr= typestr.left(len(typestr)-1)
104                     output += ("- "+label+"Error, tag not extracted. Tag type "+typestr)
105                     }
106                 continue
107                 }
108             }
109         }
110     }
111 // Main functions for operating directly on taggroup
112 void PrintTagGroup( taggroup tg, number max_depth )
113     {
114     if( max_depth < 0 )
115         max_depth= Infinity()
116     number depth= 0
117     string output
118     PrintTagGroupRecursing( tg, max_depth, depth, output)
119     Result(output)
120     }
121 void PrintTagGroup( taggroup tg )
122     {
123     number depth= 0
124     string output
125     PrintTagGroupRecursing( tg, 0, depth, output)
126     Result(output)
127     }
128 // Overloading for operating on part of a taggroup
```

```
void PrintTagGroup( taggroup tg, string tagpath, number max_depth )
    {
    taggroup tg2
    if( tg.TagGroupGetTagAsTagGroup(tagpath,tg2) )
        PrintTagGroup(tg2,max_depth)
    else
        Throw("Specified taggroup doesn't exist")
    }
void PrintTagGroup( taggroup tg, string tagpath )
    {
    taggroup tg2
    if( tg.TagGroupGetTagAsTagGroup(tagpath,tg2) )
        PrintTagGroup(tg2,0)
    else
        Throw("Specified taggroup doesn't exist")
    }
// Main function for queries on taggroups that are not loaded
void PrintTagGroup( string tagpath, number max_depth )
    {
    taggroup tg
    if( StringToLower(tagpath) == "-u" ){
        Result("\nPrinting the User persistent taggroup...")
        tg= GetUserPersistentTagGroup()
        }
    else if( StringToLower(tagpath) == "" ){
        Result("\nPrinting the persistent taggroup...")
        tg= GetPersistentTagGroup()
        }
    else {
        if( StringToLower(tagpath.Left(3)) == "-u " ){
            tagpath= tagpath.Right( Len(tagpath) - 3 )
            if ( ! TagGroupGetTagAsTagGroup(GetUserPersistentTagGroup(),tagpath,tg) )
                Throw("Specified taggroup doesn't exist")
            Result("\nPrinting tags at \""+tagpath+"\" in the User persistent taggroup
                ...")
            }
        else {
            if ( ! TagGroupGetTagAsTagGroup(GetPersistentTagGroup(),tagpath,tg) )
                Throw("Specified taggroup doesn't exist")
            Result("\nPrinting tags at \""+tagpath+"\" in the persistent taggroup...")
            }
        }
    if( max_depth < 0 )
        max_depth= Infinity()
    number depth= 0
    string output
    PrintTagGroupRecursing( tg, max_depth, depth, output)
    Result(output)
    }
void PrintTagGroup( string tagpath )
    PrintTagGroup(tagpath,0)
// Finally, print the persistent taggroup if called with none specified
void PrintTagGroup( number max_depth )
    PrintTagGroup(GetPersistentTagGroup(), max_depth)
void PrintTagGroup()
    PrintTagGroup(GetPersistentTagGroup())
```

D.2.4 Linus-functions.s

```
/*
 * Some DMscript functions by Linus Schoenstroem
 *
```

```
 * This version is outdated and not licensed. Please download the
 * current, correctly licensed version at github.com/linussch
 *
 * Last edit:  2016-04-28
 *
 *      v.1     2015-10-29     Included StringCountSubstring() and ImageGetSizeString()
 *      v.2     2015-11-30     Added Replace(string, find, put)
 *      v.3     2015-12-15     Changed the name of that one to StringReplace() to avoid
      mixups
 *      v.4     2016-01-27     Added ScriptStringTimer()
 *      v.5     2016-04-20     Renamed StringReplace() back to Replace(), readability
      improvements (case and indent convention)
 *      v.6     2016-04-28     Moved TagPathExtractLabel() and
      TagGroupShowArrayTagAsImage() from qE-library.s to here. Added setting version
      info in global tags on DM startup.
 */

// This function runs on every DM startup, it is called at the end of this file.
// It puts information about this library into the global tags for easy reference.
void LinusSchSetOnDMStartup() {
    taggroup linus
    linus= TagGroupGetOrCreateTagGroup(GetPersistentTagGroup(),"_LinusSch")
    linus.TagGroupSetTagAsString("Linus-functions:Installed version","v.6 2016-04-28")
    if( ! linus.TagGroupDoesTagExist("Author") ) // This tag may be set by another of
        my libraries
        linus.TagGroupSetTagAsString("Author","Linus Schoenstroem (a string tag doesn't
            handle o with dots)")
}

/* StringCountSubstring(str,sub)
 *
 * Counts the number of occurances of a substring in a string. Uses
 * Find(), which returns the index of the first occurrence of a sub-
 * string in a string, or -1 if no occurrence.
 *
 * Written by Linus Schoenstroem 2015-10-28
 */
number StringCountSubstring(string str, string sub) {
    number i, c
    c= 0
    i= str.Find(sub)
    while(i != -1) {
        c++
        str= str.Right(str.Len()-(i+1))
        i= str.Find(sub)
    }
    return c
}

/* ImageGetSizeString(image)
 *
 * Written by Linus Schoenstroem 2015-10-21
 */
string ImageGetSizeString( image img ) {
    number dim= img.ImageGetNumDimensions()
    string size
    number i
    for(i=0; i<dim; i++) {
        number len= img.ImageGetDimensionSize(i)
        size= size+len
        if( i+1 < dim )
            size= size+"x"
    }
```

```
    return size
}

/* Replace(string, find, put)
 *
 * Written by Linus Schoenstroem 2015-11-30
 * Last edit: 2015-12-15
 */
number Replace(string &str, string oldsub, string newsub) {
    number i, c
    string part1, part2
    c= 0
    i= str.Find(oldsub)
    while(i != -1) {
        c++
        part1= str.Left(i)
        part2= str.Right(str.Len() - (i+oldsub.Len()))
        str= part1 + newsub + part2
        i= str.Find(oldsub)
    }
    return c
}

/* ScriptStringTimer(script_string, count)
 *
 * Executes the script_string, measuring how much real time it takes,
 * count number of times. Note that these results can be far from
 * repeatable.
 *
 * Written by Linus Schoenstroem 2016-01-27
 * Last edit: 2016-01-27
 */
void ScriptStringTimer(string script_string, number count) {
    number tstart,tend,tps,tres,i,sumtime
    sumtime= 0
    tps= GetHighResTicksPerSecond()
    tres= GetHighResTickResolution()

    Result("\n Testing the given script "+count+" times")
    for(i=0;i<count;i++) {
        tstart= GetHighResTickCount()
        ExecuteScriptString(script_string)
        tend= GetHighResTickCount()
        Result("\n      Time taken = ")
        Result(Format(CalcHighResSecondsBetween(tstart,tend),"%12.9f"))
        Result(" +/- "+Format((tres/tps),"%12.9f")+" sec")
        sumtime += tend-tstart
    }
    Result("\n Mean time for the given script on these "+count+" runs = ")
    Result(Format(CalcHighResSecondsBetween(0,sumtime/count),"%12.9f"))
}
void ScriptStringTimer(string script_string)
    ScriptStringTimer(script_string,5)

// Extract the label of a tag from a tagpath
// In exact, remove everything before and including the last colon of a string
string TagPathExtractLabel(string tagpath) {
    number i= tagpath.Find(":")
    if( i == -1 )
        return tagpath
    else
        return tagpath.Right( Len(tagpath) - i )
}
```

```
124
125 // Show an array tag as an image
126 // For now only this version, for two-dimensional arrays of unspecified type - TYPE
    CONVERSION MAY HAPPEN, ONLY FOR BASIC VIEWING
127 void TagGroupShowArrayTagAsImage(taggroup tg, string tagpath, number size_x, number
    size_y) {
128     string label= TagPathExtractLabel(tagpath)
129     image img= RealImage("Image created from tag "+label,4,size_x,size_y)
130     number r= tg.TagGroupGetTagAsArray(tagpath,img)
131     if(r)
132         img.ShowImage()
133     else
134         Throw("Reading the array failed")
135 }
136
137 // Sets version info in global tags on startup
138 LinusSchSetOnDMStartup()
```

D.3 Crucial DMscript development tools

D.3.1 PrintTagTypes.s

```
1  /*
2   * Prints the types of a set of tags to the result window
3   *
4   * void PrintTagTypes( taggroup tg, string tagpath, number max_depth )
5   *
6   * All arguments optional:
7   *  - without argument max_depth, uses 0 (prints only top level)
8   *  - without argument tagpath, prints from the root of the specified taggroup
9   *  - without argument tg, prints from the persistent taggroup
10  *
11  * As a shortcut for the lazy prepending "User:" to a tagpath when called
12  * without a specified taggroup prints from the user persistent taggroup.
13  *
14  * A negative max_depth uses infinite depth (prints all levels).
15  *
16  * Created by Linus Schoenstroem at 2016-02-03 as a special version of PrintTagGroup.s
17  * Last edited: 2016-02-03
18  *
19  * This version is outdated and not licensed. Please download the
20  * current, correctly licensed version at github.com/linussch
21  */
22
23 /* Changelog
24
25 major.minor.fix[status]
26 a means alpha (not complete), b means beta (not tested) and x means knowingly broken
27
28 1.0.0      2016-02-03 Created, as a slight modification of PrintTagGroup.s
29
30 */
31
32 void PrintTagTypesRecursing( taggroup tg, number max_depth, number &depth, string &
    output)
33     {
34     number count = tg.TagGroupCountTags()
35     number islist = tg.TagGroupIsList()
36     number i
37
38     string cr = "\n"
39     for( i = 0; i < depth; i++)
```

```
        cr= cr+": "

    if(count==0)
        output += islist ? (cr+": <empty taglist>") : (cr+": <empty taggroup>")
    else
        {
        for( i = 0; i < count; i++ )
            {
            output += (cr)
            string label= islist ? "["+i+"] " : tg.TagGroupGetTagLabel(i)+"\t" //More
                intelligent aligning could make output prettier
            number type= tg.TagGroupGetTagType(i,0)

            if( type == 0 ) {
                // tag is a TagGroup
                output += ("+ "+label )
                if( max_depth > depth ) {
                    taggroup tg2
                        tg.TagGroupGetIndexedTagAsTagGroup(i,tg2)
                    depth += 1
                    PrintTagTypesRecursing(tg2,max_depth,depth,output)
                    depth -= 1
                    }
                continue
                }

            else {
                // Printing the tagtype string
                string typestr
                number typelength = tg.TagGroupGetTagTypeLength(i)
                number i2
                for( i2 = 0; i2 < typelength ; i2++)
                    typestr += tg.TagGroupGetTagType( i, i2 ) + ":"
                typestr= typestr.left(len(typestr)-1)
                output += ("- "+label+" = "+typestr)
                }
            }
        }
    }
// Main functions for operating directly on taggroup
void PrintTagTypes( taggroup tg, number max_depth )
    {
    if( max_depth < 0 )
        max_depth= Infinity()
    number depth= 0
    string output
    PrintTagTypesRecursing( tg, max_depth, depth, output)
    Result(output)
    }
void PrintTagTypes( taggroup tg )
    {
    number depth= 0
    string output
    PrintTagTypesRecursing( tg, 0, depth, output)
    Result(output)
    }
// Overloading for operating on part of a taggroup
void PrintTagTypes( taggroup tg, string tagpath, number max_depth )
    {
    taggroup tg2
    if( tg.TagGroupGetTagAsTagGroup(tagpath,tg2) )
        PrintTagTypes(tg2,max_depth)
    else
```

```
102         Throw("Specified taggroup doesn't exist")
103     }
104 void PrintTagTypes( taggroup tg, string tagpath )
105     {
106     taggroup tg2
107     if( tg.TagGroupGetTagAsTagGroup(tagpath,tg2) )
108         PrintTagTypes(tg2,0)
109     else
110         Throw("Specified taggroup doesn't exist")
111     }
112 // Main function for queries on taggroups that are not loaded
113 void PrintTagTypes( string tagpath, number max_depth )
114     {
115     taggroup tg
116     if( StringToLower(tagpath) == "user" ){
117         Result("\nPrinting types of tags in the User persistent taggroup...")
118         tg= GetUserPersistentTagGroup()
119         }
120     else if( StringToLower(tagpath) == "" ){
121         Result("\nPrinting types of tags in the persistent taggroup...")
122         tg= GetPersistentTagGroup()
123         }
124     else {
125         if( StringToLower(tagpath.Left(5)) == "user:" ){
126             tagpath= tagpath.Right( Len(tagpath) - 5 )
127             if ( ! TagGroupGetTagAsTagGroup(GetUserPersistentTagGroup(),tagpath,tg) )
128                 Throw("Specified taggroup doesn't exist")
129             Result("\nPrinting types of tags at \""+tagpath+"\" in the User persistent
                    taggroup...")
130             }
131         else {
132             if ( ! TagGroupGetTagAsTagGroup(GetPersistentTagGroup(),tagpath,tg) )
133                 Throw("Specified taggroup doesn't exist")
134             Result("\nPrinting types of tags at \""+tagpath+"\" in the persistent
                    taggroup...")
135             }
136         }
137     if( max_depth < 0 )
138         max_depth= Infinity()
139     number depth= 0
140     string output
141     PrintTagTypesRecursing( tg, max_depth, depth, output)
142     Result(output)
143     }
144 void PrintTagTypes( string tagpath )
145     PrintTagTypes(tagpath,0)
146 // Finally, print the persistent taggroup if called with none specified
147 void PrintTagTypes( number max_depth )
148     PrintTagTypes(GetPersistentTagGroup(), max_depth)
149 void PrintTagTypes()
150     PrintTagTypes(GetPersistentTagGroup())
```

D.3.2 small-scriptrunner.s

```
1 string root, dir, filename, fullpath
2
3 dir=       "kodtest"
4 filename=  "current.s"
5
6 root= "D:\\Linus\\Documents\\_faktiska dokument\\Skolarbete\\Exjobb\\code"
7
8 /*
```

```
 * Small script to run script files from disk
 *
 * By Linus Schoenstroem on 2015-11-21
 *
 */

if(len(dir)==0)
    fullpath= root+"\\"+filename
else
    fullpath= root+"\\"+dir+"\\"+filename

ExecuteScriptFile(fullpath)
```

D.4 Scripts for testing camera readout

D.4.1 camera-setup-for-test.s

```
// This script sets camera settings in the qE-STEM taggroup, for use with
// the script test-camera-calls.s or other testing purposes.
//
// Written by Linus Schoenstroem 2015-12-07
// Last edit: 2016-04-28, Updated taggroup name
//
// This version is outdated and not licensed. Please download the
// current, correctly licensed version at github.com/linussch

taggroup qetags= TagGroupGetOrCreateTagGroup(GetUserPersistentTagGroup(),"_LinusSch:qE
    -STEM")
number camID,exposure,xBin,yBin,processing,areaT,areaL,areaB,areaR
camID= CameraGetActiveCameraID()

/* Either set everything... */
exposure=  .1
xBin=  1
yBin=  16

processing= CameraGetUnprocessedEnum()
//processing= CameraGetDarkSubtractedEnum()
//processing= CameraGetGainNormalizedEnum()

areaT= 256
areaB= 768

areaL= 0
areaR= 2048

/* ...or get everything - just uncomment this line to overwrite the values above */
//CameraGetDefaultParameters(camID,exposure,xBin,yBin,processing,areaT,areaL,areaB,
    areaR)

/* Finally, store it */
    qetags.TagGroupSetTagAsShort("Camera ID",camID)
    qetags.TagGroupSetTagAsNumber("Exposure time (s)",exposure)
    qetags.TagGroupSetTagAsShortPoint("Binning",xBin,yBin)
    qetags.TagGroupSetTagAsNumber("Processing",processing)
    qetags.TagGroupSetTagAsShortRect("Camera area",areaT,areaL,areaB,areaR)
    // The following line is somewhat wrong, and there should be a sanity check
    qetags.TagGroupSetTagAsFloatPoint("Capture resolution",(areaR-areaL)/xBin,(areaB-
        areaT)/yBin)

/* And show it was done */
PrintTagGroup("_LinusSch:qE-STEM",1)
```

D.4.2 test-camera-calls.s

```
// This script reads the camera, recording how much time it took into
// a tag, for testing purposes. The camera settings are assumed to be
// available in the qE-STEM taggroup as set by camera-setup-for-test.s.
//
// Written by Linus Schoenstroem 2015-12-07
//
// This version is outdated and not licensed. Please download the
// current, correctly licensed version at github.com/linussch
//
// Last edit: 2016-04-28, Updated taggroup name

    /* Function to plot and create a histogram of numbers stored in tags */
    void PlotAndHistFromNumberTags(taggroup data)
        {
        image plot, hist
        number i,count,value

        count= data.TagGroupCountTags()
        plot:= RealImage("",4,count,1)
        for(i=0;i<count;i++)
            {
            data.TagGroupGetIndexedTagAsNumber(i,value)
            plot[i,0]= value
            }
        plot.ShowImage()
        ChooseMenuItem("Analysis","","Histogram")
        }

/* Fetching settings */
    taggroup qetags
    number camid,exposure,xBin,yBin,processing,areaT,areaL,areaB,areaR,sizeX,sizeY

    TagGroupGetTagAsTagGroup(GetUserPersistentTagGroup(),"_LinusSch:qE-STEM",qetags)
    qetags.TagGroupGetTagAsNumber("Camera ID",camid)
    qetags.TagGroupGetTagAsNumber("Exposure time (s)",exposure)
    qetags.TagGroupGetTagAsShortPoint("Binning",xBin,yBin)
    qetags.TagGroupGetTagAsNumber("Processing",processing)
    qetags.TagGroupGetTagAsShortRect("Camera area",areaT,areaL,areaB,areaR)
    qetags.TagGroupGetTagAsShortPoint("Capture resolution",sizeX,sizeY)

//  Result("\n"+camid+" "+exposure+" "+binX+" "+binY+" "+processing+" "+areaT+" "+areaL
     +" "+areaB+" "+areaR+" "+sizeX+" "+sizeY)

/* Preparing */
number i
image img
taggroup data

qetags.TagGroupGetTagAsTagGroup("Temp data",data)
img:= CameraCreateImageForAcquire(camid,xBin,yBin,processing,areaT,areaL,areaB,areaR)
CameraPrepareForAcquire(camid)

/* Main loop. See to that only the camera command you want to test is
 * uncommented, and start at a low count. The data is not automatically
 * cleared between runs of this script, so as to enable easy creation of
 * a larger dataset as needed.
 */
for(i=0;i<20;i++)
```

```
    {
    number t
//  i= SIGetAcquisitionPixelNumber()
    OpenAndSetProgressWindow("Testing camera acquisition","Test "+(i+1),"")

    t= GetHighResTickCount()

//  image img= CameraAcquire(camid)
//  image img:= CameraAcquire(camid)
//  img= CameraAcquire(camid)
//  img= CameraAcquire(camid,exposure)
//  img= CameraAcquire(camid,exposure,xBin,yBin)
//  img= CameraAcquire(camid,exposure,xBin,yBin,processing)
//  CameraAcquireInPlace(camid,img)
//  CameraAcquireInPlace(camid,img,exposure)
//  CameraAcquireInPlace(camid,img,exposure,xBin,yBin)
//  CameraAcquireInPlace(camid,img,exposure,xBin,yBin,processing)
    CameraAcquireInPlace(camid,img,exposure,xBin,yBin,processing,areaT,areaL,areaB,
        areaR)
//  img:= CameraAcquire(camid,exposure,xBin,yBin,processing,areaT,areaL,areaB,areaR)

    t= CalcHighResSecondsBetween(t,GetHighResTickCount())
    data.TagGroupInsertTagAsNumber(i,t)
    }
img.ShowImage()
PrintTagGroup("_LinusSch:qE-STEM",1)
PlotAndHistFromNumberTags(data)
OpenAndSetProgressWindow("","","")
```

D.5 AutoIT code

D.5.1 ClickDetectorButton.au3

```
; ClickDetectorButton
;
; Clicks the buttons for detector insertion and retraction in
; Digital Micrograph 2.3, provided that they are in the expected
; place. Exits 0 unless called erroneously: 1 if not called with
; exactly one argument, 2 if said argument is not one of the
; strings "out", "BF" or "ADF". (case agnostic, though)
;
; This code is not reliable. It relies on no one touching the
; mouse, and it does not know whether it succeded in pressing
; the right button or not. Use only with extreme caution!
;
; Written by Linus Schoenstroem 2016-02
;
; This version is outdated and not licensed. Please download the
; current, correctly licensed version at github.com/linussch

 Const $xout = 3449    ; I think the screens are 1920 + 1920 = 3840
 Const $xBF =  2406
 Const $xADF = 3750
 Const $y   =  -580
 Const $b   =  "primary"

If $CmdLine[0] <> 1 Then  Exit 2  ; Error 2 = b10, wrong number of arguments

If $CmdLine[1] = "out" Then
   MouseClick( $b, $xout, $y)
ElseIf $CmdLine[1] = "BF" Then
   MouseClick( $b, $xBF, $y)
```

```
ElseIf $CmdLine[1] = "ADF" Then
   MouseClick( $b, $xADF, $y)
Else
    Exit 6  ; Error 2 = b110, unknown argument
EndIf

 Exit 0
```

D.6 Matlab code

D.6.1 getqE.m

```
function [data, energy, othertags] = getqE(varargin)
%Imports the data from a qE-STEM TagGroup into Matlab.
%
%Gives you a GUI file browser when called without arguments. Can also be
%called with a structure (as read in by dmread.m) or a filepath as
%argument.
%
%Not done yet - not even checking the arguments for errors...
%
%Partly from Thomas Thersleff's getDM4ESI.m, the rest written by Linus
%Schoenstroem. Essentially a special-case frontend for Andreas Korinek's
%dmread.m.
%
% This version is outdated and not licensed. Please download the
% current, correctly licensed version at github.com/linussch

%Last edit: 2016-04-28 by Linus, Updated taggroup name
%
%   I added the 4-dimensional case (earlier). It works, but I have not
%   checked the error handling.

%% Parsing inputs (always the clever part)
if nargin < 1
    [FileName,PathName] = uigetfile({'*.gtg', 'DigitalMicrograph TagGroup file'; ...
                                     '*.*','All Files'}, ...
                                     'Select the file');
    complete = dmread([PathName,FileName]);
elseif isstruct(varargin{1})
    complete= varargin{1};
else
    fString = varargin{1};
    complete = dmread(fString);
end

%% Construct the data array
dimX= complete.PerExperimentData.CaptureResolution.Value{2}
dimY= complete.PerExperimentData.CaptureResolution.Value{1}
dim3= complete.PerExperimentData.SurveyResolution.Value{2}
dim4= complete.PerExperimentData.SurveyResolution.Value{1}
if dim3 == 0
    othertags= complete;
    error(['This dataset does not have survey resolution set. If';...
           'you specified the "othertags" output argument that ';...
           'structure will now contain the complete dataset. '])
end

part1= 'complete.PerPixelData.Unnamed';
part2d= '.data.Value';
part2pn= '.PixelNumber.Value';

```

```
if dim4 == 1
    %% Linescan case, only 3 dimensions
    data= zeros(dimX,dimY,dim3);
    for i= 1:dim3
        pn= i-1;   % Pixel number
        tagpath= [part1,int2str(pn),part2d];
        data(:,:,i)= reshape(eval(tagpath),dimX,dimY);
        % Checks if pixel numbering is fishy
        tagpath= [part1,int2str(pn),part2pn];
        saved_pn= eval(tagpath);
        if saved_pn ~= pn
            warning(['Tag number ',int2str(pn),' says PixelNumber= ',int2str(saved_pn),
                 ...
                    '. This is array page (:,:,',int2str(i),').'])
        end
    end
else
    %% Mapping case, 4 dimensions
    data= zeros(dimX,dimY,dim3,dim4);
    for ii= 1:dim4
        n_prev_lines= (ii-1)*dim3 % Number of pixels on previous lines
        for i= 1:dim3
            pn= i-1+n_prev_lines % Pixel number
            tagpath= [part1,int2str(pn),part2d];
            data(:,:,i,ii)= reshape(eval(tagpath),dimX,dimY);
            % Checks if pixel numbering is fishy
            tagpath= [part1,int2str(pn),part2pn];
            saved_pn= eval(tagpath);
            if saved_pn ~= pn
                warning(['Tag number ',pn,' says PixelNumber= ',int2str(saved_pn), ...
                        '. This is array page (:,:,',int2str(i),',',int2str(ii),').'])
            end
        end
    end
end

%% Energies and the rest
% Get energy vector (not existing for now)

end
```

D.6.2 ShowDMTagString.m

```
function string= ShowDMTagString(str)
% Converts a field in a structure that was originally a tag of type string
% in DigitalMicrograph, as read in by Andreas Korinek's dmread.m, into a
% Matlab character array.
%
% Written by Linus Schoenstroem, 2016-04-21
c= struct2cell(str);
string= char(c{1}');
```

D.6.3 showesi.m

```
function showesi( array )
%Shows our readouts as close as easily possible to how Digital Micrograph
%would show them.
%
%Written by Linus Schoenstroem
%
```

```
% This version is outdated and not licensed. Please download the
% current, correctly licensed version at github.com/linussch

% Last edit 2016-04-28 by Linus

% Turns off the warning about docked figures having magnification set to
% fit
warning('off','images:imshow:magnificationMustBeFitForDockedFigure')

d= reshape(array,1,[]);  % Turns the 2D array into a vector for the
low= prctile(d,.1);      % percentile functions, which sets the limits
high= prctile(d,99.9);   % for the display contrast

array= shiftdim(array,1); % swaps the dimensions as imshow() expects an
imshow(array,[low high]); % array which is y by x
```

D.6.4 esisliceplayer.m

```
function esisliceplayer( data )

% Interactively plays the slices of a 3d data cube
%
% Prints liberally to the command window, for now. Performance ain't very
% good yet, but control is.

% Written by Linus Schoenstroem on 2016-03-03 - 2016-03-04
%
% This version is outdated and not licensed. Please download the
% current, correctly licensed version at github.com/linussch

    % Turns off the warning about docked figures having magnification set to fit
    warning('off','images:imshow:magnificationMustBeFitForDockedFigure')

    update_time= .1;  % The time between updates, in seconds
    run= 1;
    direction= 1;

    h= figure('KeyPressFcn',@keyboardcontrol);
    set(gca,'Position',[0 0 1 1])

    i= 1;
    imax= size(data,3);
    while run
        if direction == 0
            pause(.1)
            continue
        end
        updateesi(data(:,:,i));
        disp(num2str(i))
        pause(update_time);
        % Going to the next slice
        i= i+direction;
        if i == imax || i == 1
            disp 'Reached the end, pausing'
            direction= 0;
        end
    end
    set(h,'KeyPressFcn','')
    disp 'Sliceplayer was stopped.'

    % Function to update the image
    function updateesi( array )
```

```
        d= reshape(array,1,[]);  % Turns the 2d array into a vector for the
        low= prctile(d,.1);      % percentile functions, which sets the limits
        high= prctile(d,99.9);   % for the display contrast

        array= shiftdim(array,1); % swaps the dimensions as imshow() expects an
        imshow(array,[low high]); % array which is y by x
    end

    % Function to catch keyboard commands
    function keyboardcontrol( ~, call )
        char = int8(call.Character');
        if isempty(char)
            return;
        end;
        switch char
            case 27    % Esc
                run= 0;
            case 28    % Left
                direction= -1;
                disp 'Backwards'
            case 29    % Right
                direction= 1;
                disp 'Forward!'
            case {32,48}  % Space & Number 0
                direction= 0;
                disp 'Sliceplayer paused'
            case num2cell(49:57)  % Numbers 1 to 9
                char= char - 48;
                switch char
                    case 1, update_time= .05;
                    case 2, update_time= .1;
                    case 3, update_time= .2;
                    case 4, update_time= .5;
                    case 5, update_time= 1;
                    case 6, update_time= 2;
                    case 7, update_time= 5;
                    case 8, update_time= 10;
                    case 9, update_time= 20;
                end
                disp(['Update time was set to ',num2str(update_time)])
            case {43,30}  % Plus & Up
                update_time= update_time *.5;
                disp(['Update time was set to ',num2str(update_time)])
            case {45,31}  % Minus & Down
                update_time= update_time *2;
                disp(['Update time was set to ',num2str(update_time)])
            otherwise  % Unused key
                return
        end
    end
end
```

D.6.5 LinusPlotPlayer.m

```
function LinusPlotPlayer( f_h, istart, istop )

% Interactively plays any plot command (that plots in the current figure),
% passed as a function handle with a single argument (which is varied
% between imin and imax)
%
% Prints liberally to the command window, for now. Performance ain't very
% good yet, but control is.
```

```
9
10  % Written by Linus Schoenstroem on 2016-03-10
11  % Adapted from esisliceplayer.m
12  %
13  % This version is outdated and not licensed. Please download the
14  % current, correctly licensed version at github.com/linussch
15  %
16  % Bugfix on 2016-04-08: including the endpoints, support for starting
17  % backwards
18
19      % Input check (TODO: check types)
20      narginchk(3,3)
21
22      % Turns off the warning about docked figures having magnification set to fit
23      warning('off','images:imshow:magnificationMustBeFitForDockedFigure')
24
25      % Starting parameters
26      update_time= .1;  % The time between updates, in seconds
27      run= 1;
28      if istart < istop
29          direction= 1;
30          imin= istart;
31          imax= istop;
32      elseif istart > istop
33          direction= -1;
34          imax= istart;
35          imin= istop;
36      else
37          error('Unexpected error, istart == istop ?')
38      end
39
40      % Initiating the figure
41      h= figure('KeyPressFcn',@keyboardcontrol);
42
43      % Main loop
44      i= istart;
45      while run
46          if direction == 0
47              pause(.1)
48              continue
49          end
50          disp(num2str(i))
51          f_h(i);
52          pause(update_time);
53          % Going to the next slice
54          i= i+direction;
55          if i > imax || i < imin
56              disp 'Reached the end, pausing'
57              i= i-direction;
58              direction= 0;
59          end
60      end
61      set(h,'KeyPressFcn','')
62      disp 'The plot player was stopped.'
63
64      % Function to catch keyboard commands
65      function keyboardcontrol( ~, call )
66          char = int8(call.Character');
67          if isempty(char)
68              return;
69          end;
70          switch char
71              case 27    % Esc
72
```

```
                run= 0;
            case 28    % Left
                direction= -1;
                disp 'Backwards'
            case 29    % Right
                direction= 1;
                disp 'Forward!'
            case {32,48}  % Space & Number 0
                direction= 0;
                disp 'Sliceplayer paused'
            case num2cell(49:57)  % Numbers 1 to 9
                char= char - 48;
                switch char
                    case 1, update_time= .05;
                    case 2, update_time= .1;
                    case 3, update_time= .2;
                    case 4, update_time= .5;
                    case 5, update_time= 1;
                    case 6, update_time= 2;
                    case 7, update_time= 5;
                    case 8, update_time= 10;
                    case 9, update_time= 20;
                end
                disp(['Update time was set to ',num2str(update_time)])
            case {43,30}  % Plus & Up
                update_time= update_time *.5;
                disp(['Update time was set to ',num2str(update_time)])
            case {45,31}  % Minus & Down
                update_time= update_time *2;
                disp(['Update time was set to ',num2str(update_time)])
            otherwise  % Unused key
                return
        end
    end
end
```

D.6.6 plotLinesOnqE.m

```
% Trying to find and draw where the signal is, both center and edges.
%
% It is a custom analysis for the q-E STEM readout.
%
% Writes over current figure.
%
%   centreline= plotLinesOnqE(array,options)
%
% Second argument is optional. It is either a string that contains any one,
% or a combination, of:
%   noi     to not show the image, just the lines
%   noe     to skip trying to find the edges of the data (just centre)
%   nos     to skip smoothing of lines depicting edges
%   noh     to not show straight helper lines
%   nop     to not plot at all (useful for getting the centreline data)
%
% ...or a number, in which case the analysis of that single vertical line
% of the image is shown instead. Or a vector, in which case all lines given
% by the elements of that vector will be shown, pausing .3 seconds on each
% plot.
%
% Private code, not intended/licensed/fit for public use.
%
% Last edit: 2016-04-28, v0.3.1
```

```
25
26  % Version history
27  %
28  %   2016-03-21 v0.2   Swapped in centre-of-mass calculation model
29  %   2016-04-08 v0.2.1 Bugfix: I had called a variable plot
30  %   2016-04-11 v0.3   More legible lines
31  %   2016-04-28 v0.3.1 Name change
32
33  function out= plotLinesOnqE(array,options)
34      % Input check
35      narginchk(1,2)
36      if nargin == 1
37          options= '';
38      end
39      [sizeX,sizeY]= size(array);
40
41      % Parameters
42      bins= 30;
43      noisepart= .6;
44      excludesingles= 1;
45
46      %% Special cases
47      % The analyse one line case
48      if isnumeric(options)
49          if isscalar(options)
50              analyseoneline(options)
51          else
52              for ii= 1:length(options)
53                  analyseoneline(options(ii))
54                  pause(.3)
55              end
56          end
57          return
58      end
59      % The no plot case
60      if ~ isempty(strfind(options,'nop'))
61          centrelinef(0)
62          out= centreline;
63          return
64      end
65      %% Main
66      if isempty(strfind(options,'noi')) % Unless asked not to,
67          showesi(array);                % shows the image.
68      else                               % Otherwise,
69          clf                            % clears the figure properties and
70          xlim([1 sizeX])                % creates something suitable
71          set(gca,'ydir','reverse')
72      end
73
74      centrelinef(1)
75      if isempty(strfind(options,'noe'))
76          edgelinesf()
77      end
78      if nargout==1
79          out= centreline;
80      end
81
82      %% Subfunctions
83
84      function centrelinef(doplot)
85          % I'm looping the vertical lines (seemed easiest)
86          centreline= zeros(1,sizeX);
87          for i= 1:sizeX
```

```
88          line= array(i,:);
89          % This time, let's find the centre of mass instead
90          momenta= line .* [1:sizeY];
91          centreline(i)= sum(momenta)/sum(line);
92      end
93      if doplot
94          % Plotting that line (and a straight helper unless asked not to)
95          hold on
96          plot(centreline,'LineWidth',2,'Color',[1 0 0])
97          if isempty(strfind(options,'noh'))
98              y= mean(centreline);
99              plot([1 sizeX],[y y],'LineWidth',1,'Color',[0.294 0.808 0])
100         end
101         hold off
102     end
103 end
104
105 % It may be possible to fit the centreline to a circular arc,
106 %   arc= @(x,r,oy)r+oy-realsqrt(r^2-(x-1024.5).^2)
107 % with starting values r= 25000 and oy= mean(centreline)
108
109 function edgelinesf()
110     % I'm looping the vertical lines (seemed easiest)
111     edgelines= zeros(2,sizeX);
112     for i= 1:sizeX
113         line= array(i,:);
114         [N,edges]= histcounts(line,bins);
115         % Extracting edges
116         % The lowest-valued pixels are the noise
117         ind= find(cumsum(N) > sizeY*noisepart,1);
118         indices= find( line > edges(ind+1) );
119         if excludesingles == 1
120             % And the signal part is coherent - indices are consecutive
121             % (this gets rid of some outliers)
122             indices= indices( diff(indices)==1 );
123         end
124         try
125             edgelines(:,i)= [indices(1) indices(end)];
126         catch ME
127             ME= addCause(ME,MException('MATLAB:plotLinesOnqE:foundnoindices',['Did
                    not find coherent set of signal pixels on i= ' num2str(i) ', see
                    plot']));
128             disp('Plotting problem...')
129             analyseoneline(i)
130             rethrow(ME)
131         end
132     end
133
134     % Smoothing the lines (unless asked not to)
135     if isempty(strfind(options,'nos'))
136     end
137
138     % Plotting the edgelines and straight helpers
139     hold on
140     plot(edgelines(1,:),'LineWidth',2,'Color',[1 0 0])
141     plot(edgelines(2,:),'LineWidth',2,'Color',[1 0 0])
142     if isempty(strfind(options,'noh'))
143         y= mean(edgelines(1,:));
144         plot([1 sizeX],[y y],'LineWidth',1,'Color',[0.294 0.808 0])
145         y= mean(edgelines(2,:));
146         plot([1 sizeX],[y y],'LineWidth',1,'Color',[0.294 0.808 0])
147     end
148     hold off
```

```
    end

    function analyseoneline(linenumber)
        line= array(linenumber,:);
        plot(line)
        xlim([1 sizeY])
        hold on
        [N,edges]= histcounts(line,bins);
        % Extracting edges
        % The lowest-valued pixels are the noise
        ind= find(cumsum(N) > sizeY*noisepart,1);
        indices= find( line > edges(ind+1) );
        if excludesingles == 1
            % And the signal part is coherent - indices are consecutive
            % (this gets rid of any certain outliers)
            indices= indices( diff(indices)==1 );
        end
        plot(indices,line(indices),'*')
        % And the centre, calculated as centre of mass
        momenta= line .* [1:sizeY];
        centre= sum(momenta)/sum(line);
        plot([centre centre],ylim,'--')
        hold off
    end

end
```