# DAWNSci Developer Training

## Table of Contents

DAWNSci Developer Training ........ 1
Eclipse Foundation and DAWSci Eclipse Project ........ 2
What is the DAWNSci Project and why is Diamond involved? ........ 2
Who are the other participants in DAWNSci? ........ 2
Possible GDA project(s) at the foundation ........ 2
Committing to the DAWNSci project ........ 3
Using this tutorial ........ 3
Checking out the Examples ........ 4
ILoaderService ........ 4
What is it? ........ 4
The soft cache ........ 4
IDataset and concrete classes ........ 5
Numpy and MATLAB vs DAWNSci ( *DatasetFactory, Maths, DatasetUtils* ) ........ 5
How to add your own loader by extension point ........ 5
Slicing Data ........ 5
ILazyDataset ........ 5
getSlice(…) and getSliceView(…) ........ 6
IPlottingSystem ........ 6
Introduction ........ 6
Plotting System Lifecycle ........ 7
Overview of pluggable architecture ........ 7
Create, configure, add, remove ........ 8
ITraceSystem ........ 8
IRegionSystem ........ 10
IAxisSystem ........ 11
IAnnotationSystem ........ 11
IToolPageSystem ........ 11
Introduction ........ 11
Creating views with tools ........ 11
........ 12

Adding custom actions to tools ........ 12
Creating your own tool ........ 12
OSGi Services ........ 13
How to declare and use a service without creating dependencies ........ 13
Useful services ........ 13
How to get services and junit tests working together. ........ 17
Exercises ........ 18
Exercise 1 – Create a part with a plot and manipulate an image ........ 18
Exercise 2 – Apply a mask to an image while it updates ........ 18
Exercise 3 – Convert plotted images to a video ........ 18
Exercise 4 – Create a tool to process a region ........ 18
Appendices ........ 19
Appendix 1 – Loaders Registered in DAWN 1.8 ........ 19

# Eclipse Foundation and DAWSci Eclipse Project

## What is the DAWNSci Project and why is Diamond involved?

Diamond Light Source is part of the Eclipse Science Working Group and has a place on the steering committee. The DAWNSci is contributed from Diamond as part of the commitment Diamond has made to the Eclipse Science working group. In this role, DAWNSci defines classes for describing data and mathematics and services for analysis including plotting. The rest of this tutorial shows how some of these designs work. The benefit to Diamond from this relationship is that by publishing the API's, Diamond is able to make a long term commitment to software working in a standardised way. This enables internal and external users of the software to be able to rely on a backwardly compatible and supported portion of the code base. Diamond also benefits because the code is released through the Eclipse Foundation open source mechanism. Through intellectual property checks and experience of license agreement details, the foundation is able to take a would-be open source project and deliver it in a reliable way to the community.

## Who are the other participants in DAWNSci?

It is currently developed by Diamond Light Source and all the registered committers are employees of Diamond Light Source. Because the repository is on the Eclipse github, it is possible for collaborators to fork and complete a pull request, see below. This process has been completed by collaborators such as Kitchwa coders, Itema and Industrial TSI. The other users/repackagers of DAWNSci and members of the Eclipse Science Working Group are potential future committers.

## Possible GDA project(s) at the foundation

It would be possible to publish APIs inside Generic Data Acquisition to the foundation as well. These for instance might include the long term areas of GDA like scanning, detectors and motor connectivity.

### Committing to the DAWNSci project

The procedure for committing to DAWNSci for Diamond Employees and external participators is nearly identical.

1. Committing using a pull request (Preferred)

If you have a change then fork the dawnsci repo onto your github account. Push your change to this fork and complete a pull request to the main repo master branch. One of the existing committers will quickly check and merge in the change.

2. Becoming a committer

You will need to sign the eclipse committer agreement, if you are a Diamond employee; Diamond has already signed the eclipse committer documentation and can be chosen from the drop down list of companies. You will then need to contact our email 'DAWN-DEV@JISCMAIL.AC.UK' asking to become a committer and detailing your eclipse user name. There will be an election which is held on the eclipse web site according to their standard procedure. The voters are the existing committers to the project.

## Using this tutorial

This tutorial is designed to be read as a guide to interacting with the code and in conjunction with the checked out DAWNSci project running in eclipse. It does not get the programmer to complete examples throughout but instead shows places in the code which provide the programmer with knowledge of things available in the DAWNSci product. Armed with this knowledge, or at least where to find it, you can later complete real tasks using DAWNSci, to take a sneak peak look at the Exercises section now.

Wherever you see this arrow, the tutorial is asking you use eclipse to complete a real task. This might be look at a class, write your own or check out code. This is a hands on task but not a programming exercise. The programming exercises are at the end on the tutorial.

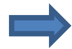
Data Nomenclature: Description of plotted data vs. the type of plot options available

| Data Rank | Description |
|---|---|
| nD | 0 or more dimensions to the data, i.e. any data. Not usually used to mean 0 dimensions. |
| 0, 0D data | Scalar value, plotted as a single point usually with an IRegion called a PointSelection. |
| 1, a 1D array | 1D data plotted for instance as XY Graph, histogram/column, pie chart etc. |
| 2, 2D array or image | 2D data plotted for instance as an Image, a 3D surface (in this case it is 2D data and 3D plotting technology), a vector plot or an array of lines. |
| 3, 3D array for example image stack | 3D data plotted in 3D, for instance an isosurface. |
| 4 and above | Generally sliced to be plotted in a lower dimension. |

## Checking out the Examples

With this course you should have received a target platform and workspace on a USB key allowing you to get started with the DAWNSci without downloading.

There is also an online guide to checking out the DAWNSci examples in a workspace so that you can play around with them. This is available at http://www.dawnsci.org/eclipse/getting-started-with-dawnsci



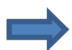
Please either use the USB key or the online instructions to get the DAWNSci examples checked out and compiling now.

## ILoaderService

### What is it?

ILoaderService is an OSGi service for loading any data and making it available in the IDataset format. IDataset is an nD array implementation in Java if you are familiar with numpy, there is a similar class called 'ndarray' if not, it is a class which allows multi-dimensional data to be held and mathematical operations to be executed efficiently. IDataset can pass seamlessly between Java and an ndarray in python and back again. It also has a huge library of mathematics in Java, pipelining tools and of course the plotting relies on it.

#### *LoadingExamples*



Open `org.eclipse.dawnsci.analysis.examples.dataset.LoadingExamples`.

Look at the examples and then run them as a junit plugin tests. This class is not a test but an example, junit plugin simply gives us an easy way to run the code and provide OSGi services.

**NOTE**: ILoaderService is provided by OSGi, other plugins in DAWN provide the service and the test consumes it.

### The soft cache

In the example which you have just run, either an IDataset or an IDataHolder for holding many datasets is returned from the ILoaderService. **IMPORTANT**: The loader service keeps data in a soft reference cache. This makes accessing data fast if it has previously been loaded and the application has not since required memory for anything else that large. However it is possible to accidentally in place modify this data using methods such as imultiply(…) on the dataset. If you need to change the data and use in place methods, please use the clone() method to duplicate the memory of the data first or you might change the value of the cache. However, you should only use clone() if you really need to however because it will duplicate the memory.

The caching can be modified using system properties:

| System Property | Effect |
|---|---|
| uk.ac.diamond.scisoft.analysis.io.nocaching | Set to true to stop all caching (not recommended) |
| uk.ac.diamond.scisoft.analysis.io.weakcaching | Set to true to switch from soft to weak references (expert use only). If nocaching is set to true, this will have no effect. |

### IDataset and concrete classes

Go to the org.eclipse.dawnsci.analysis.dataset plugin and look at the various concrete classes of IDataset. You will see classes such as DoubleDataset and IntegerDataset for holding multi-dimensional arrays of data of these primitives.

Go to the package org.eclipse.dawnsci.analysis.examples.dataset.impl in org.eclipse.dawnsci.analysis.examples. Here there is a subset of some of the tests done in the main test decks of DAWN with IDataset. Look at DoubleDatasetTest and any other tests which take your fancy.

**NOTE:** All java primitives and various aggregates like complex numbers and RBG have dataset implementations.

### Numpy and MATLAB vs DAWNSci ( *DatasetFactory, Maths, DatasetUtils* )

Find `org.eclipse.dawnsci.analysis.examples.dataset.NumpyExamples.` This example shows how to do various operations in DAWNSci that are available in MATLAB and numpy. Right click on the example and choose 'Run As -> JUnit Plug-in Test'. Have a look through the examples; they are a good reference for what we will do later in this tutorial.

| Class | Function |
|---|---|
| Maths | Similar to java.lang.Math but for nD arrays |
| DatasetFactory | Creating filled/empty datasets. |
| DatasetUtils | Dataset manipulation |

**NOTE:** This example shows how DAWNSci compares with numpy and MATLAB

### How to add your own loader by extension point

Many loaders are provided with DAWN, a list of these are shown in appendix 1. It is possible to add your own loader by extension point. An example of registering your own loader can be seen in a plugin contributed to DAWN by DESY called '**de.desy.file.loader**'.

If you are not using the USB key workspace, you can check this plugin out by cloning git@github.com:DawnScience/scisoft-core.git to your examples project and importing just the plugin de.desy.file.loader.

The file format is called 'FIO' and an example of the format is provided in a basic test in the plugin. The loader is defined by extension point, view it now by looking at the class FioLoader.

## Slicing Data

### ILazyDataset

It is all very well writing loaders for data but what if the data is too large to fit in memory? Often large HDF5 files or directories containing stacks of 2D array files (AKA images!) will need to be referenced as a single dataset. In this case, enter stage left: lazy dataset. This is a class which allows data to be described and sliced without holding it all in memory.

Look again at LoadingExamples and the method load2DLazy(). In this example, instead of getting all the data to load out of the data holder, we call getLazyDataset(…) and the holder returns a lazy representation of the data. In the example we are only getting an image which will fit in memory from the holder but if the data were a large stack of images, it means we can immediately get an object which knows the size of the data and can provide slices of it.

### getSlice(…) and getSliceView(…)

Unlike numpy, the standard slice (which is getSlice(…) in DAWNSci) creates a new slice in memory whether the data is already loaded or not. You may use getSliceView() to avoid making a copy when slicing. The table below explains what happens:

| Class | Method | getLazyDataset |
|---|---|---|
| IDataset | getSlice(…) | New memory is assigned to the slice |
| IDataset | getSliceView(…) | The slice is backed by the array of the original IDataset |
| ILazyDataset | getSlice(…) | New memory is assigned to the slice |
| ILazyDataset | getSliceView(…) | A new ILazyDataset is returned which still has not loaded the data until getSlice(…) is called. |

Go to the class `org.eclipse.dawnsci.analysis.examples.dataset.LazyExamples` and run using a junit plugin test. Look at the huge random dataset which can exist but when real data is required from it, it must be sliced. The DAWN user interface uses ILazyDataset extensively to allow users to interface with data larger than will fit in memory. For instance with expressions and an SWT composite to plot slices.

There are some more examples in `org.eclipse.dawnsci.analysis.examples.dataset.SlicingExamples`, these provide some basic example of using a Slice object.

**Advanced:** Look at all the tests in `org.eclipse.dawnsci.analysis.examples.slice` these are a selection of tests from the core of DAWN. They have an introduction to slicing using an iterator and a generator. These classes can be used to walk over large data efficiently.

## IPlottingSystem

### Introduction

The plotting system is an abstract layer between the programmer and the implementation of a plot. It means that different plotting/visualization technologies such as draw2d, javafx and OpenGL can be used without the programmer being exposed to the details of those technologies. Core algorithms should not import and use the plotting system, however at some point there must be a class which uses the mathematical layer and plots information to the user. Often this is a two-way process with the user selecting part of the data and the user interface responding with a new calculation of the algorithm which the user is using.

The plotting system abstraction means that a model view controller or MVC pattern can be used to operate the plotting. In this scenario the model is the algorithm, the view is the plotting system and

the controller is a class which can see both algorithm and plotting, often created in the view part. By abstracting the plotting system this way, it is possible to swap plotting technologies without changing the controller code.

### It's Simple!

Open XYExample from your workspace. This shows creating a simple line plot. Look at the easy code involved with plotting something, this is true for whatever trace type you plot because DAWNSci describes data in a well defined way.

Look at IPlottingSystem and its inherited interfaces, not so simple after all! However the interface is subdivided into separate base interfaces which are discussed in later sections of this chapter.

Run an eclipse application from your workspace and go to the 'Examples' perspective. Look at all the different plotting examples. Now open the 'Data Browsing' perspective. There are files in the 'data' project which have automatically been copied to the workspace of your application. You can open and plot data from these files by double clicking just as you would a Java file.

### Plotting System Lifecycle

IPlottingService is an OSGi service and is injected into the class by telling OSGi to do so (we will look at how this works in a later chapter). From the service, we get an IPlottingSystem instance and this is a class which is used to do plotting.

```
IPlottingService pservice = …; // A service provided by OSGi
IPlottingSystem    system = pservice.createPlottingSystem();

// Later when the part is created, create a plotting system on it.
system.createPlotPart(...);

// Later when the part is disposed
system.dispose();
```

### Overview of pluggable architecture

For an individual plotting system, there can be multiple viewers. Each viewer is used to visualise one or more trace types. The table below details plotting viewers and the trace types that they support. When a user of the API adds a trace of a given type the plotting system will automatically swap the implementation. Viewers are contributed by extension points.

| Internal Class | Traces | Description |
|---|---|---|
| LightWeightPlotViewer | ILineTrace, IVectorTrace, IImageTrace, IImageStackTrace | The main plot viewer ofen seen for XY plots and images and based on draw2d. |
| JRealityPlotViewer | ISurfaceTrace, IMulti2DTrace, ILineStackTrace, IScatter3DTrace | Used mainly for 3D surfaces, based on jreality which is an OpenGL scene graph. |
| FXPlotViewer | IIsosurfaceTrace | Used for isosurfaces only. DAWNSci will also link in the JavaFX 1D plotting. |

## Create, configure, add, remove

The interface IPlottingSystem is large but it is divided into subdivisions which are easier to understand. These subdivisions mostly work by looking at one concept like traces, regions or annotations and allowing the programmer to create, configure and then add the item to the plot.

| Subsystem | What it does |
|---|---|
| ITraceSystem | Manage traces of all supported types. |
| IRegionSystem | Manage regions of many different types. The regions select data on plots and have a wide range of shapes and functions. |
| IAxisSystem | Axes can be retrieved and configured and if the plotting viewer supports it, additional ones added. |
| IAnnotationSystem | Annotations can be created, configured and added or retrieved by name. |
| IPrintablePlotting | Methods to instruct the plotting system to print or export an image to file. |

## ITraceSystem

### *Introduction*

ITraceSystem defines how to create, configure and add traces. The system is used as follows:

```
IXXXTrace trace = system.createXXXTrace(); // XXX is the trace type
trace.setXXP(…);
trace.setXXQ(…);
trace.setXXR(…);
system.addTrace(trace);  // All traces are added with addTrace(…)
```

Open the class `org.eclipse.dawnsci.plotting.examples.XYExample`. This class creates and configures a line trace. Now run the examples by running and eclipse application from the workspace and going to the ‘Examples’ perspective. You can see the plotting of ‘XY Example’.

**NOTE:** All traces work this way: create / configure / add however there are convenience thread safe methods for quickly plotting things. These can also be seen in XYExample. They are called createPlotXXX(…) and updatePlotXXX(…).

### *Listening to events, data change etc.*

A listener may be attached to ITraceSystem which notifies if any new trace is added, you can see the methods in `org.eclipse.dawnsci.plotting.api.trace.ITraceListener`. There are many methods for instance traceAdded and tracesAdded when one or several traces are added and traceRemoved, tracesRemoved.

**Advanced**: one interesting method is traceWillPlot(…). This allows data to be changed before the data goes to the screen. It is useful for instance in making image filtering which does not flicker as it is added to the plot.

### *Appending data to a 1D plot*

The plotting interface can be used for transient image data (see the next section) and transient 1D data. Where 1D data is changing, it can be replotted without creating a new IDataset. This basically circumvents the need to have all your data contained in an IDataset; however once plotted, you may export data from the plotting as an IDataset. Using append pushes the dataset growing code into the plotting which can then determine the most efficient way of appending data (at the time of writing it is using System.arraycopy(…) but this might be avoided in future versions).

Open the class `org.eclipse.dawnsci.plotting.examples.XYUpdateExample` and look at 'XY Update Example' in the examples perspective. Tick the update toggle to get the plotting to start appending.

### *Image plotting*

Introduction

Images are one of the most powerful plotting features in DAWNSci. When an image is plotted, its data is downsampled to approximately fit the plot viewer size then an 8-bit SWT image is created using the IImageService detailed in the next section. This image is then scaled using fast scaling to fit precisely in the current viewer area that the user is viewing. There are different downsampling and histogramming options when completing this process which result in the varied different image options for visualization.

IImageService and IPaletteService

IImageService and IPaletteService create images from intensity data. The colour palette can be extended by extension point. These services are used widely internally to DAWN. Exercise 3 can be solved by using the IImageService to write an image to file for the full data. Look at the methods on IImageService now to frequent yourself with how it works.

IDownsampleService

This service allows datasets to be reduced using schemes and bins. A bin represents a collection of values which will be downsampled into a single value.

| Downsample Mode | Effect |
| --- | --- |
| POINT | Output value is top left value. This type is fast. |
| MEAN | Mean value of bin |
| MAX | Max value of bin |
| MIN | Min value of bin |

Making IImageTrace run fast for updates

In order to make an image update fast, we must set the image to use a fast downsample and also avoid recalculating the histogram as in the below example. Doing this means that the IImageTrace can receive data at quite a fast rate. The plotting system has been connected to MJPG streams running as fast as 100Hz for a 512x512 image and tools are usable.

```
final IPlottingSystem system = …;

final IImageTrace image = system.createImageTrace("fred");
image.setData(Random.rand(new int[]{1024, 1024}), null, true); // Plots the data

image.setDownsampleType(DownsampleType.POINT); // Fast!
image.setRescaleHistogram(false);              // Fast!

// Now calling image.setData(...) at better than 20Hz is possible (depending on
image size…)
```

### *IPlottingFilter*

A plotting filter is a way of registering a filter which, whenever a particular type of data is plotted, responds by transforming it. The transformed data returned is then what is actually plotted. The same effect can be made by using ITraceListener and a traceWillPlot(…) method.

```
IPlottingFilter myIPlottingFilter1 = new AbstactPlottingFilter() {
     public int getRank() {
         return 1;
     }
     protected IDataset[] filter(IDataset x,    IDataset y) {
         return new IDataset[]{null, Maths.square(y)};
     }
 };
 IFilterDecorator dec = PlottingFactory.createFilterDecorator(IPlottingSystem);
 dec.addFilter(myIPlottingFilter1);
 // Maybe, dec.addFilter(myIPlottingFilter2);

 // later, if you need to.
 dec.reset();
```

## IRegionSystem

### *Introduction*

Open the Sector Example in the 'Examples'  perspective of an eclipse application run from your workspace. Look at the yellow sector added to the Duke image. Go to the 'Source' tab of the sector example (at the bottom), the sector here is created in the same way as we have seen for traces; namely Create, Configure and Add.

### *Review of different region types available*

Open the type `org.eclipse.dawnsci.plotting.api.region.IRegion.RegionType`. This is an enumeration of the possible region types available in DAWN. Each region type requires an IROI which is the data which the region has selected or covers. An IROI is separate conceptually from IRegion because it is a selection in the data and so part of the data layer not the user interface. Each IRegion has one or more IROI types it supports.

In the source code, go to the package `org.eclipse.dawnsci.analysis.dataset.roi` and frequent yourself with the data selections comparable to the visual selections. We will return to regions and ROIs in one of the exercises.

Now run the eclipse examples and go to a view with a plotting system on it. Open the menu at the far right of the toolbar of the view. Go to 'Selection Region' and view the various regions that can be added in this way (this is not the total of all possible regions). Add some of the various region types, note that when a region is created, the plotting system shows a custom cursor ready to place the region via a click and drag in the plotting system.

### IAxisSystem

The axis system is used to control axes. With all types of 1D plots, multiple axes are supported. With images and higher plots, multiple axes are not normally supported. Unlike the other systems, axes have the concept of being selected or active. This is required because one may plot data without specifying the axes to use, for simplicity. In order to plot data programmatically to one set of axes and then another, one must create the additional axis and set it as active.

Go to the source code for org.eclipse.dawnsci.plotting.examples.AxisExample and look at how data is first plotted to the default axes, then additional ones are created / selected. When the data is next plotted, it is assigned to the newly selected axes.

### IAnnotationSystem

Annotations are a useful feature for adding points of interest which a user should note and perhaps see the value of. For instance after peaks are fitted, they can also be annotated. Annotations can be created using the same create, configure, add approach of the other features.

Go to the eclipse application and the 'Data Browsing' perspective. Open the file 'metalmix.mca' and plot some of the datasets (Column_1, Column_2 etc.). From the tools on the toolbar of the plot ('XY Plotting Tools') select 'Maths and Fitting->Peak Fitting'. Now drag out an area over the data, a region will appear and be used by the peak fitting tool for the fitting area. Set the number of peaks in the 'Peak Fitting' tool to be three. Annotations are used to mark the peaks.

## IToolPageSystem

### Introduction

One of the most powerful parts of DAWNSci is the tool system which allows custom user interface to be put in a helper part which is displayed alongside the plot. As we saw when choosing peak fitting in the last section, there are many different tools pre-supplied with DAWNSci but it is also possible to create your own.

Unlike the sub-interfaces of IPlottingSystem, IToolPageSystem is a standalone interface. For a given IPlottingSystem it can be retrieved as follows:

```
IToolPageSystem tsystem = (IToolPageSystem)system.getAdapter(IToolPageSystem.class);
IToolPage       tpage    = tsystem.getActiveTool();
```

A tool is added to DAWNSci by using an extension point to declare it.

### Creating views with tools

If you want to have tools automatically appear on a part which is receiving its toolbar from the plotting system, your part has to do one thing – implement getAdpater(…) to return the tool system.

```
@Override
public Object getAdapter(Class clazz) {
    if (system.getAdapter(clazz)!=null) return system.getAdapter(clazz);
    return super.getAdapter(clazz);
}
```

### Adding custom actions to tools

Often an existing tool allows you to choose an area of data but what you would really like to do having seen the data and the result from the tool, is run a custom action. Therefore there is an extension point which adds an action to the toolbar of any tool, allowing custom work to be done. For example the grid tool selects and positions a grid over data, a custom action could be added to the grid tool process data in the grid.

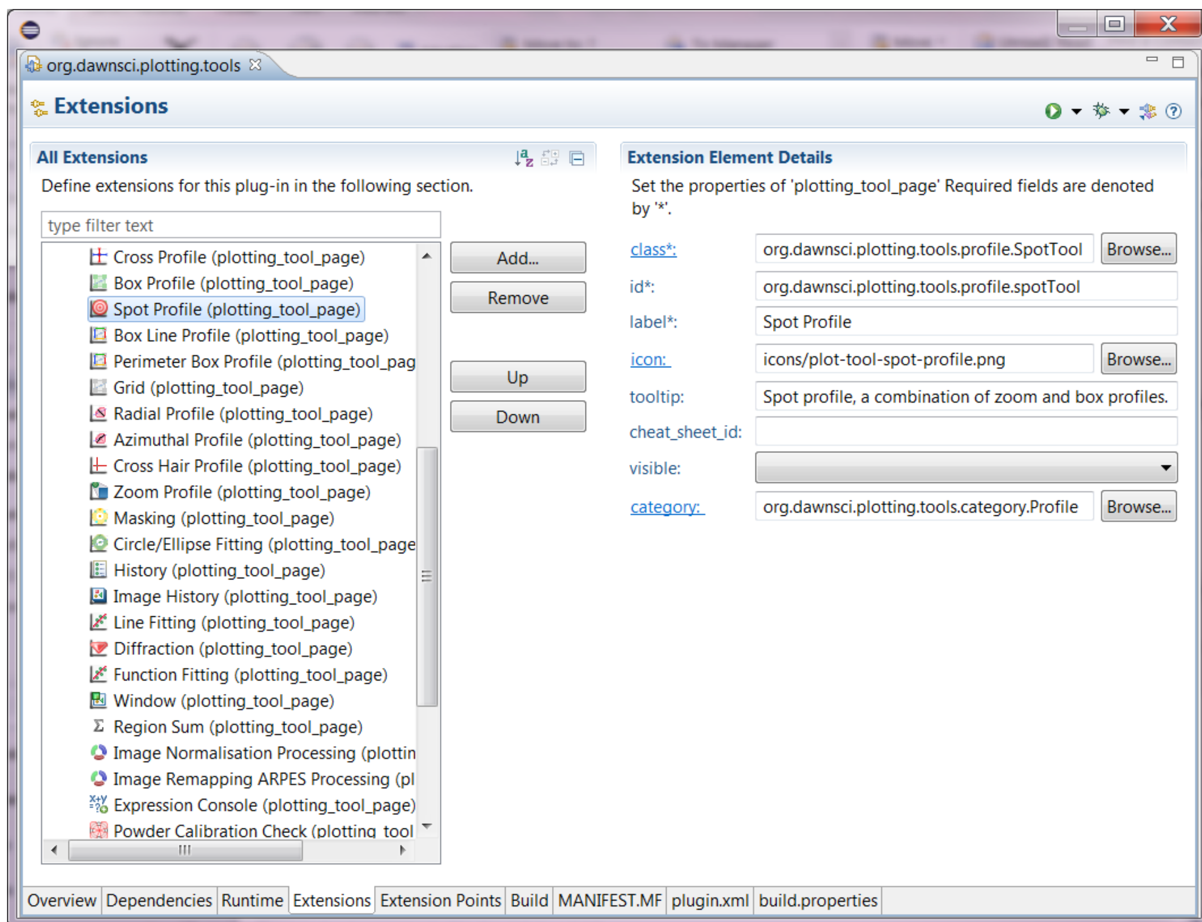
You can use the extension point org.eclipse.dawnsci.plotting.api.toolPageAction to add the action to the tool. **NOTE**: This extension point references a command id, so you must put your code in a command. If the command needs the tool or the plotting system, the IAdapter pattern can be used to get both from the part.

### Creating your own tool

Open the class `org.eclipse.dawnsci.plotting.examples.tool.ExampleTool`. This tool does not actually complete work but it shows the methods which you must implement, namely:

`getToolPageRole, createControl, getControl, activate, deactivate` and usually `dispose`.

To declare the tool you should use the extension point org.eclipse.dawnsci.plotting.api.toolPage which provides the tool with an icon, a tooltip and a category.




# OSGi Services

## How to declare and use a service without creating dependencies

OSGi services work by declaring a bundle as providing a service and/or consuming it in a XML file which is distributed with the bundle. This means that OSGi can determine the available services and their usage without loading the whole Java classpath. In this case DAWN core provides the services and we will be consuming them in our exercises later. The interfaces for the services are defined in a low dependency plugin such as `org.eclipse.dawnsci.analysis.api` which can be imported without creating additional dependencies. This means that the services hide concrete implementations and increase modularity of your product.

## Useful services

### *IConversionService*

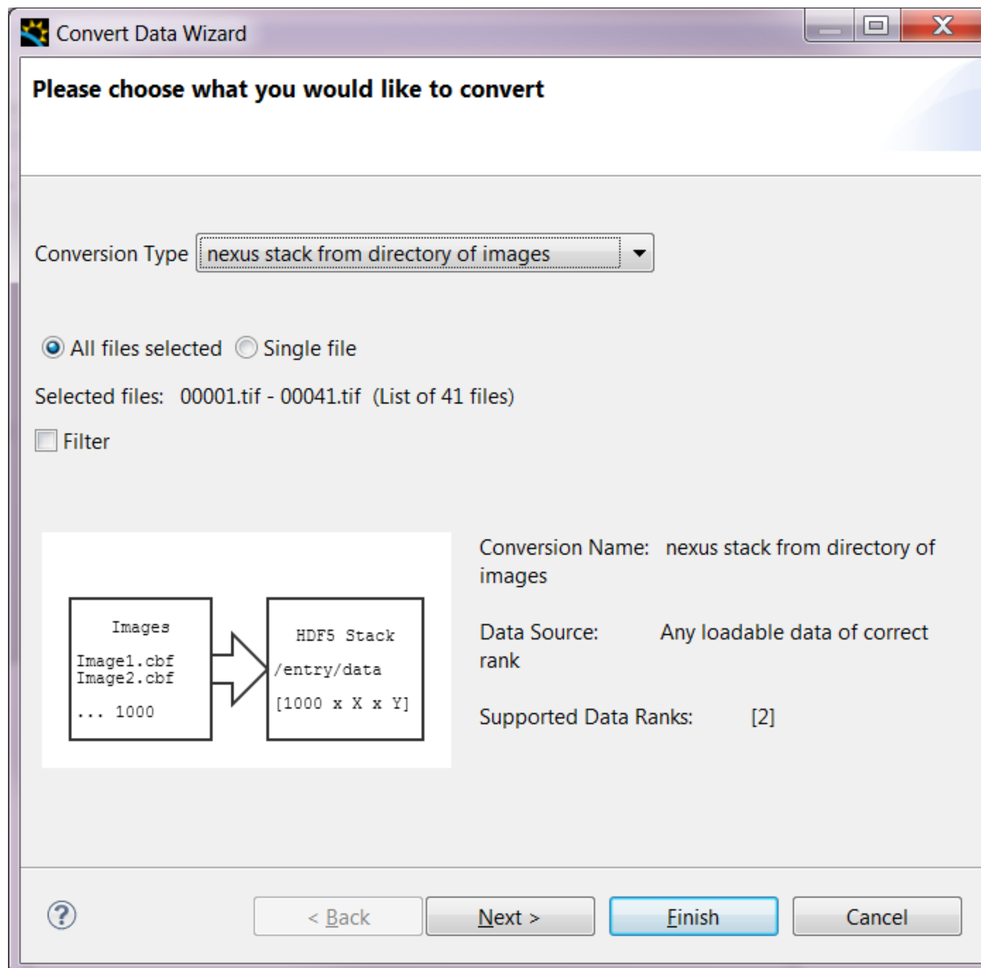
The conversion service is a service which attempts to convert any file format supported by the ILoaderService into any format to which the IConversionService has been provided with a writer. For instance: a directory of images to a video or an HDF5 stack to many text files / image files.

```
IConversionService service = ...;
IConversionContext context = service.open("/dls/path_to_some_hdf5_file.nxs"); // 
regex allowed
context.setDatasetName("/entry1/signal/some_data"); // regex allowed
context.setOutputPath("/dls/some_place_I_want_my_data");

context.setConversionScheme(IConversionContext.ConversionScheme.ASCII_FROM_2D);
service.process(context);
```

The service works in a similar way to previous plotting API. Create a context, configure the context and process the conversion. DAWNSci also provides an RCP wizard for running the service. The wizard is entirely separate from the underlying service so that conversions can be done on servers.

Open an RCP application from your workspace and go to the 'Data Browsing' perspective. Right click on the folder 'data/examples/315029-pilatus100k-files' and run the 'Convert…' action to open the wizard. From the list of conversions choose 'Nexus stack from directory of images' like this:




Now choose next and finish. DAWN will use the conversion service to write the TIFF images in the directory to an HDF5 file called 'ConvertedImageStack.nxs' and open that file. The file contains all the images and allows 3D mathematics to be run on the newly created data.

### *IOperationService*

The operation service is a powerful pipeline system which also comes with a user friendly (simple but fairly linear) interface. The pipeline is executed with a single treaded loop, a pool of threads or with Ptolemy 2 which is an efficient pipeline system. The pipeline is also exportable to Passerelle RCP user interface which is included into DAWN as the 'Workflows' perspective and supports complex graphs for the pipeline, loops, sub-models etc. Passerelle RCP is currently being released as an eclipse project and DAWN distributes an older version but will upgrade to the eclipse version when it is available. Pipelines can also be run from the command line without using a user interface. The persistence service, described in the next section is capable of exporting and reloading operations pipelines.

Using the operations service from code instead of the user interface we have:

```
IOperationService service = ...;  // OSGI injection
IOperationContext context = service.createContext();
context.setData(...);     // ILazyDataset
context.setSlicing(...); // Information about how to slice the dataset
context.setSeries(...);  // The operations
service.execute(context);
```

Here the service argument is an array of configured IOperations which do the work on the data as it is sliced out of an ILazyDatset in chunks defined by the slicing argument. To create an operation it must be found and its model set - then it is ready to use in a pipeline. Operations and their associated models can be added to DAWNSci by using the extension point 'org.eclipse.dawnsci.analysis.api.operation'.

```
IOperation add  = service.findFirst("add");
// Or
add = service.create("uk.ac.diamond.scisoft.analysis.processing.addOperation");
add.setModel(new ValueModel(101));

context.setSeries(add, …);  // The operations
```

Open an RCP application from your workspace and go to the 'Processing' perspective. Click and drag the file 'data/examples/pow_M99S5_1_0001.cbf' from 'Project Explorer' to 'Data Slice View'. In the wizard that pops up, click the finish button and keep the default settings. In the 'Processing' view click the yellow star, you get a drop down of all possible operations. Add an 'Image Threashold'; in the 'Input' view you will see your original image and in the 'Output' view the result of the pipeline as far as currently selected in the 'Processing' view. The green run button in the 'Data Slice View' submits whole directories or large data stacks to be processed (once they are added in and their slicing set up). It is also possible to submit the pipeline to be run on a cluster, DAWN provides servers for receiving processing pipelines from Activemq and running them remotely from the user interface.

### *IPersistenceService*

The persistence service is a class for saving data to a standard file format called HDF5, specifically the persisted files will be nexus compliant. Chunks of data, axes, masks, regions and operation pipelines can be saved using the service. The API is similar to previous we have seen;, create a persistence file, set various things in it and close it.

```
IPersistenceService persist = … // OSGI injection
IPersistentFile file = null;
try {
    file = persist.createPersistentFile(…);
    file.setData(…);// IDataset
    file.setAxes(…);
    file.addMask(…);// One or more masks
    file.setROIs(…);// All the region locations
} finally{
    if(file!= null) file.close();
}
```

As well as the API above the persistence is available in many tools in the user interface. Most popup a wizard which saves and loads persisted data.

Open an RCP application from your workspace and go to the 'Data Browsing' perspective. Open the file 'data/examples/pow_M99S5_1_0001.cbf' and choose the tool 'Masking'. Create a mask on the image by enabling lower and upper mask at -1 and 20 respectively and ensuring the mask is applied. If you have done it correctly, part of the image will be replaced with a green colour. Now click on the 'regions' radio button and add a region, for example a box, somewhere in the image. We will now persist the file by clicking the 'Export Mask to file' button on the masking tool's toolbar. Choose a new file in the 'data' project and name it 'mask.nxs' then hit 'Next'. On the next screen export everything you can to the persistence file. Don't forget to give the mask a name before hitting 'Finish'. You can open and import the file produced. For now, open the file and look at the various things saved about the current state of your data, go to the 'Tree' tab of mask.nxs and look at everything stored in the file. **NOTE**: If you reimport 'mask.nxs' you will get the saved mask and any regions that you exported reimported. You will not get the original data reimported.

### *IExpressionService*

DAWNSci wraps an expression engine called Apache-JEXL. JEXL stands for Java Expression Language and has a powerful syntax exposed to users in diverse places in DAWN where starting a whole python or Jython interpreter is not warranted. DAWNSci abstracts JEXL to an OSGi service because, in the past, we have used other expression engines and swapability has been important.

```
final IExpressionService service = …;
final IExpressionEngine  engine  = service.getExpressionEngine();
engine.createExpression(…);// Expression e.g. x*x, x==10.0  etc…

final Map<String, Object>    values = new HashMap<>();
// Fill values with the values to evaluate the expression.
engine.setLoadedVariables(values);
Object value = engine.evaluate();
// The value of the evaluated expression for instance a Double or Boolean.
```

### *IMacroService*

The macro service is a way of exporting commands as they are run to a python console. DAWN uses py4j to allow objects in the DAWNSci API, for instance IPlottingSystem, to be controlled from python. The IMacroService is notified as the user uses the DAWN interface and echos the equivalent commands to the python console or script file if it recording a macro. In order for this approach to work in all cases, the macro service must be called from all DAWN code to echo commands as the UI is used. This task lags behind all possible user interface in the product however many things can be scripted, for instance IPlottingSystem, IConversionService and IOperationService.

**NOTE**: To do the next hands on, you will need to install python. If you do not want to do this, please skip this section and continue to the exercises. No exercise involves python programming in this tutorial.

To record and run macros it is best to install python, recommended is python 2.7, and also numpy. Once you have done this run an eclipse application from your workspace and go to 'Window->Preferences' then search for python in the box above the tree of preferences. Go to 'Python

Interpreter' and hit 'Quick Auto-config' which should find your python. Then press Ok on the preferences form – Pydev will now link to your interpreter.

Now go to the 'Data Browsing' perspective. Go to the 'Console' view and hit 'PyDev Console' then choose the 'Python Console' radio button, press ok. You should see something like this:

```
>>>#Configuring Environment, please wait
PyDev console: using default backend (IPython not available).
pydev debugger: warning: psyco not available for speedups (the debugger
will still work correctly, but a bit slower)
>>>import scisoftpy as dnp;import sys;sys.executable=''
h5py could not be imported No module named h5py
NAPI not supported yet
Could not import nexus
Could not import python image library
>>>
```

In the console find the toolbar action called 'Record Macro'. It will start a server for interacting with the python console and ask you to restart if you have never recorded a macro before. Choose yes and the server will be turned on and the eclipse application restart. Repeat the process of starting a python console and pressing the 'Record Macro' button, now it will stay on and you are ready to start recording actions.

Now open the file 'data/examples/pow_M99S5_1_0001.cbf' from the project explorer. You will see the macro commands being echoed to the console for doing this.

These commands are echoed to the console because a programmer in DAWN figured out the equivalent numpy/python/java object commands and reported them using the IMacroService. You may also report your actions to allow python to be easily used.

## How to get services and junit tests working together.

DAWNSci currently provides services into junit tests in two ways; either by creating a hard dependency on a service implementation or creating a mock façade to the service. In the former case the test is put in its own plugin or fragment and not included in the product. Therefore the test fragment is free to import any plugin including where a concrete implementation of a service exists. Instead of getting the service from OSGi the service is assigned using an @BeforeTest annotation to assign the implementation of the service then using the concrete services constructor.

# Exercises

Please attempt to solve each exercise by what you have learned in the previous sections. There is also a suggested answer in the package org.eclipse.dawnsci.plotting.examples.exercises.

## Exercise 1 – Create a part with a plot and manipulate an image

- Create a new RCP view with a plot using the plotting service.
- Open the image 'org.eclipse.dawnsci.analysis.examples/pow_M99S5_1_0001.cbf' and plot it in your view.
- Create a button in the view which when pressed continuously removes the last 10 rows of the image, puts them at the start and replots the result. You will need sleep between replots.

**ANSWER** @see org.eclipse.dawnsci.plotting.examples.exercises.Exercise1

## Exercise 2 – Apply a mask to an image while it updates

- Add an ITraceListener to your plot
- In the traceUpdated(…) method, create a BooleanDataset mask the same size as the image and has the value of 'false' when the value in the original image is less than or equal to -1.

**ANSWER** @see org.eclipse.dawnsci.plotting.examples.exercises.Exercise2

## Exercise 3 – Convert plotted images to a video

- When the mask has been applied, export the plotting system image to a PNG file in a directory somewhere. Downsample the image to ¼ size. **HINT** You will need IImageService to do the export to PNG and the ImageServiceBean from the IImageTrace. You will need IDownsampleServce to downsample the dataset.
- When the user stops running the thread, convert the image files into a single nexus file using the conversion service.

**ANSWER** @see org.eclipse.dawnsci.plotting.examples.exercises.Exercise3

## Exercise 4 – Create a tool to process a region

- In this exercise we will attach a tool to any plotting system showing an image.
- Create the new tool to respond to any rectangular region added to the plotting system.
- When a region is added or moved, calculate a new mask which is the same threshold mask previously created by applying only to pixels contained in the region added.
- Plot the masked image in the tool.

**ANSWER** @see org.eclipse.dawnsci.plotting.examples.exercises.Exercise4

# Appendices

## Appendix 1 – Loaders Registered in DAWN 1.8

```
Hard coded loaders available in DAWN. Loaders may also be added by extension
point.

registerLoader("npy",  NumPyFileLoader.class);
registerLoader("img",  ADSCImageLoader.class);
registerLoader("osc",  RAxisImageLoader.class);
registerLoader("cbf",  CBFLoader.class);
registerLoader("img",  CrysalisLoader.class);
registerLoader("tif",  PixiumLoader.class);
registerLoader("jpeg", JPEGLoader.class);
registerLoader("jpg",  JPEGLoader.class);
registerLoader("mccd", MARLoader.class);
registerLoader("mar3450", MAR345Loader.class);
registerLoader("pck3450", MAR345Loader.class);
registerLoader("mrc", MRCImageStackLoader.class);

// There is some disagreement about the proper nexus/hdf5
// file extension at different facilities
registerLoader("nxs",  NexusHDF5Loader.class);
registerLoader("nexus",NexusHDF5Loader.class);
registerLoader("h5",   HDF5Loader.class);
registerLoader("hdf",  HDF5Loader.class);
registerLoader("hdf5", HDF5Loader.class);
registerLoader("hd5",  HDF5Loader.class);
registerLoader("mat",  HDF5Loader.class);

registerLoader("tif",  PilatusTiffLoader.class);
registerLoader("png",  PNGLoader.class);
registerLoader("raw",  RawBinaryLoader.class);
registerLoader("srs",  ExtendedSRSLoader.class);
registerLoader("srs",  SRSLoader.class);
registerLoader("dat",  DatLoader.class);
registerLoader("xy",  DatLoader.class);
registerLoader("dat",  ExtendedSRSLoader.class);
registerLoader("dat",  SRSLoader.class);
registerLoader("csv",  CSVLoader.class);
registerLoader("txt",  DatLoader.class);
registerLoader("txt",  SRSLoader.class);
registerLoader("txt",  RawTextLoader.class);
registerLoader("mca",  DatLoader.class);
registerLoader("mca",  SRSLoader.class);
registerLoader("mca",  RawTextLoader.class);
registerLoader("tif",  TIFFImageLoader.class);
registerLoader("tiff", TIFFImageLoader.class);
registerLoader("zip",  XMapLoader.class);
registerLoader("edf",  PilatusEdfLoader.class);
registerLoader("pgm",  PgmLoader.class);
registerLoader("f2d",  Fit2DLoader.class);
registerLoader("msk",  Fit2DMaskLoader.class);
registerLoader("mib", MerlinLoader.class);
registerLoader("bmp", BitmapLoader.class);

registerUnzip("gz",  GZIPInputStream.class);
registerUnzip("zip", ZipInputStream.class);
registerUnzip("bz2", CBZip2InputStream.class);
```