**FRONT END CONTROL SYSTEM**

**Detailed Design**

***Ian Gillingham***

***February 2014***

***Abstract***

*This document describes the detailed design for a standard DIAMOND Front End distributed control system*

***Document History***

|  |  |  |
| --- | --- | --- |
| ***Date*** | ***Rev*** | ***Comment*** |
| *03/08/07* | *1.0* | *First cut* |
| *23/12/09* | *1.1* |  |
| *17/02/14* | *2.0* | *Updating with deprecation of vacuumvalve, adoption of dlsPLC and introduction of dls\_pyqt4widgets.* |
|  |  |  |

**Contents**

[1 INTRODUCTION 3](#__RefHeading__2621_1320577848)

[2 Front End EPICS Application Architecture 3](#__RefHeading__2623_1320577848)

[2.1 Requirements 3](#__RefHeading__2625_1320577848)

[2.2 Deployment 4](#__RefHeading__2627_1320577848)

[2.3 Velocity template generator 4](#__RefHeading__2629_1320577848)

[2.3.1 Substitution File Build Architecture 4](#__RefHeading__2631_1320577848)

[2.4 Support modules 6](#__RefHeading__2633_1320577848)

[2.4.1 PT100 6](#__RefHeading__2635_1320577848)

[2.4.2 Flow meters (Scalers) 6](#__RefHeading__2637_1320577848)

[2.4.3 Motion Control 9](#__RefHeading__2639_1320577848)

[2.4.4 Locum-4 Current Monitor 13](#__RefHeading__2641_1320577848)

[2.4.5 Front End Vacuum 13](#__RefHeading__2643_1320577848)

[3 Front End User Interface Architecture 15](#__RefHeading__2645_1320577848)

[3.1 General design goals 15](#__RefHeading__2647_1320577848)

[3.2 QT 3.3 Implementation 15](#__RefHeading__2649_1320577848)

[3.2.1 Diamond custom Widgets 15](#__RefHeading__2651_1320577848)

[3.2.2 C++ implementation 15](#__RefHeading__2653_1320577848)

[3.2.3 SIP wrapper 17](#__RefHeading__2655_1320577848)

[3.3 QT 4 Implementation 17](#__RefHeading__2657_1320577848)

[3.3.1 Design Overview 18](#__RefHeading__5718_1320577848)

[3.3.2 Design Patterns 18](#__RefHeading__5454_1320577848)

[3.3.3 Making Diamond Qt4 Widgets Accessible from Qt Designer 23](#__RefHeading__5456_1320577848)

[3.3.4 Runtime Environment 24](#__RefHeading__5458_1320577848)

[3.3.5 Developing Widgets for Support Modules 25](#__RefHeading__5460_1320577848)

[3.3.6 Developing Qt Applications with Diamond Qt Python Widgets 26](#__RefHeading__5462_1320577848)

[3.3.7 dls\_pyqt4widgets Directory Structure 28](#__RefHeading__5464_1320577848)

[3.3.8 Deployment 30](#__RefHeading__5476_1320577848)

[3.3.9 Building the User Interface 31](#__RefHeading__5480_1320577848)

[3.4 Python 33](#__RefHeading__2659_1320577848)

[3.4.1 Software Suite Structure 33](#__RefHeading__2661_1320577848)

[4 SNL State Machines 34](#__RefHeading__2667_1320577848)

[4.1.1 Gate Valve Control 34](#__RefHeading__2669_1320577848)

[4.1.2 Beam Path Control 35](#__RefHeading__2671_1320577848)

[4.1.3 Pump Control 35](#__RefHeading__2673_1320577848)

[4.1.4 Status PV Binary Bit Pattern Description 35](#__RefHeading__2675_1320577848)

[4.1.5 Beamline Gateway Control Interface 39](#__RefHeading__33326022)

[4.1.6 dlsPLC 40](#__RefHeading__2677_1320577848)

[5 XBPM Data Handling 42](#__RefHeading__2679_1320577848)

[5.1 Loading initial ID gap and phase lookup tables 42](#__RefHeading__2681_1320577848)

[5.2 Data Flow Diagram for XBPM data handling 44](#__RefHeading__2683_1320577848)

[6 Linux Servers 45](#__RefHeading__4119_1320577848)

[6.1 INIT 45](#__RefHeading__4121_1320577848)

[6.1.1 Script files within the server directory 45](#__RefHeading__4123_1320577848)

[6.2 Controls network IP address 47](#__RefHeading__4125_1320577848)

[6.3 Private network address 47](#__RefHeading__4127_1320577848)

[7 PLC Remote I/O 47](#__RefHeading__4129_1320577848)

[7.1 PLC Register Mapping 47](#__RefHeading__4184_1320577848)

# INTRODUCTION

The Front End domain is located between the cell's Insertion Device and the Beamline optics area. A front-end typically incorporates vacuum systems, personnel safety shutters (machine and beamline), diagnostics, machine protection systems and beam conditioning devices (apertures, beamsplitters, mirrors).

This document describes an outline requirements analysis, essential detailed design aspects and implementation notes.

# Front End EPICS Application Architecture

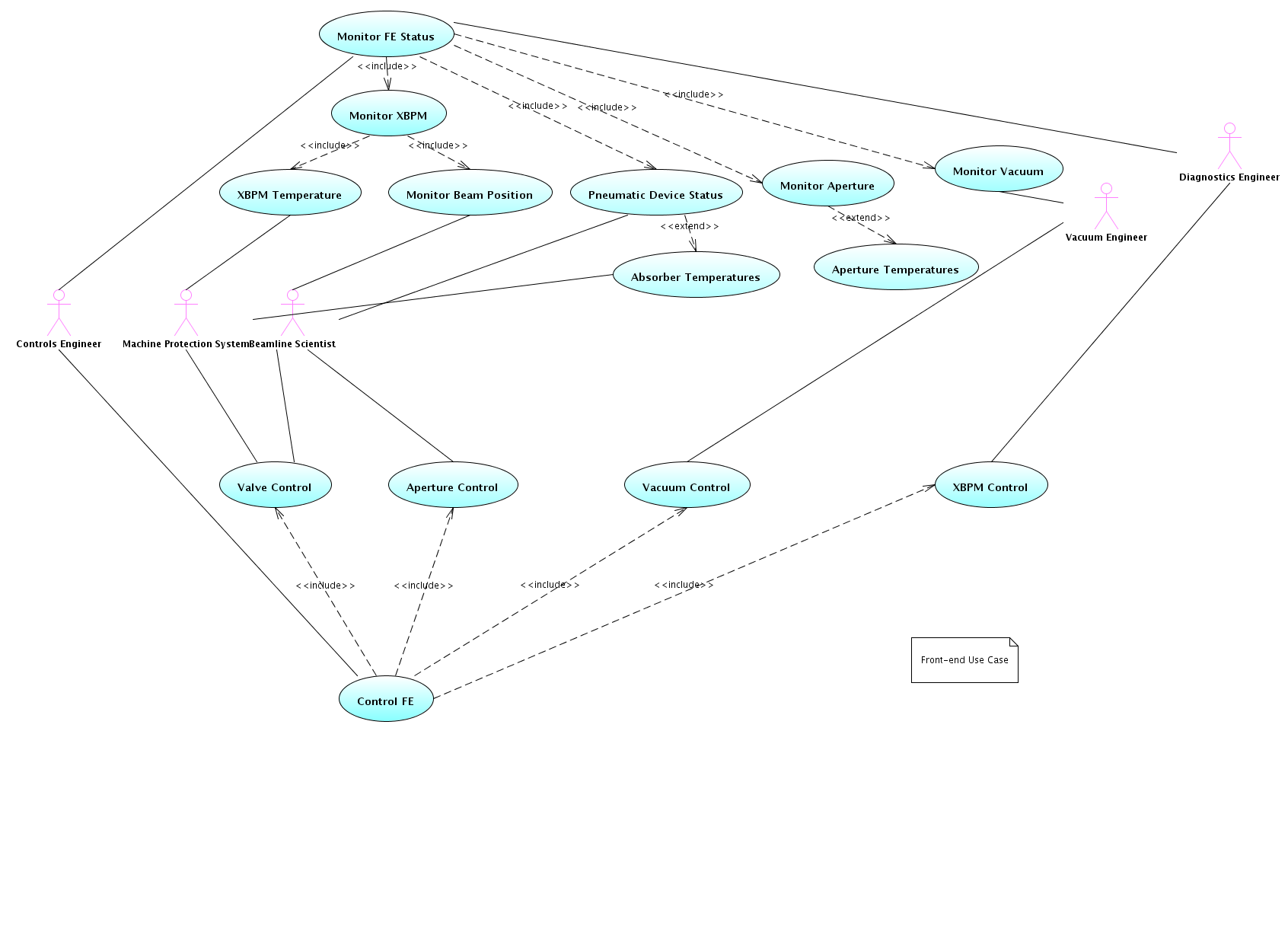
## Requirements

Users of the front-end control system have been identified. They fall into three main categories:

* Machine-side Clients
* The Machine itself
* Beamline-side Clients

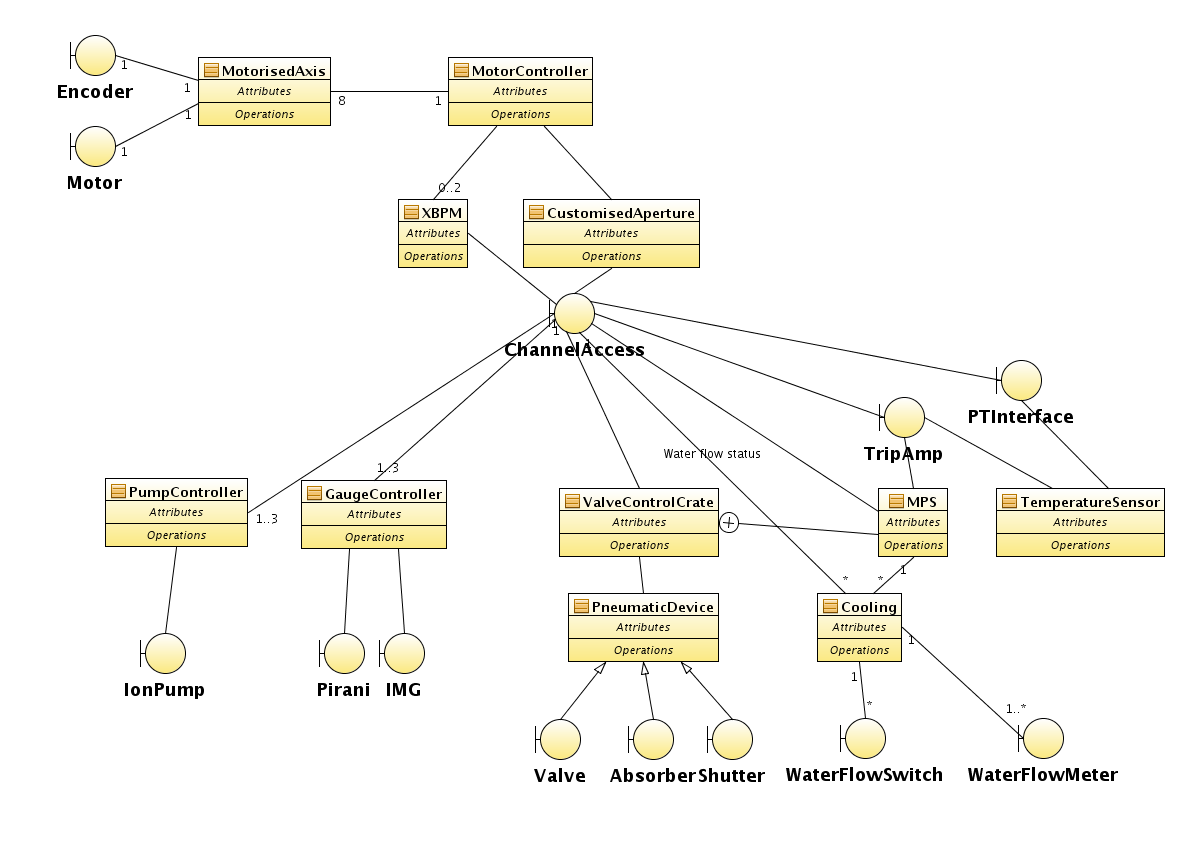
The main user-interfaces are identified in the Use Case diagram in Illustration:

Illustration 1: Front-end control system Use Case diagram



## Deployment

The following class diagram shows the control system's boundary classes (interfaces) and inner classes from the IOC's perspective.



## Velocity template generator

### Substitution File Build Architecture

This section describes the structure of the generic build mechanism for the Front End database substitution file, describing dependencies on external support records.

General design patterns have been identified across all front-ends and implemented in EPICS as a set of templates. The templates have then been further aggregated, as Velocity files (.vm), to model complete objects (such as apertures modules, which have motors and temperature sensors). The Velocity templates are packaged in the FE support module.

The key descriptor for each front-end model is the Db/fe.vm file and located in each front-end IOC build tree. The fe.vm file uses the Velocity engine to generate EPICS database substitution files. These are then, in turn, used by the build system to generate the target EPICS flat database file.

The EPICS database substitution file for each Front End, should reside in /dls\_sw/work/R3.14.8.2/ioc/FExxt/CS/FEApp/Db the file having a name of the form: FExxt-CS-IOC-01.substitutions, where 'xx' is the Front End identification number, e.g. “02” and 't' is the beamline type ('I' or 'B'). During early development, the substitution file was written by hand, but it became apparent that, in order to reduce complexity and expedite maintainability, it was necessary to abstract components, such as apertures, gauges, etc. as templates within their own support structure. This proved to be nigh on impossible with the standard EPICS substitution mechanism, as nested substitutions are not supported. Apache’s Velocity proved to be a suitable vehicle to achieve the desired mechanism.

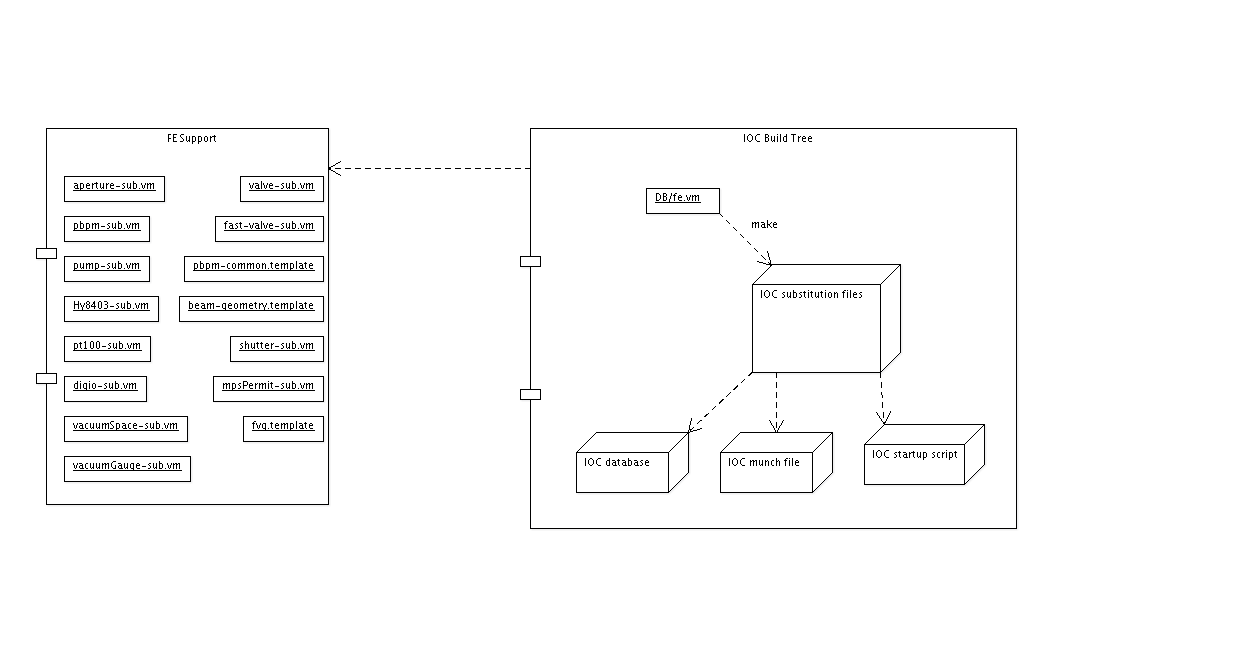
Velocity relies on a build.xml file to control the construction of the output file (in our case the substitution file). This is similar in principle to ‘Makefile’ controlling ‘make’, but now the build.xml file controls ‘ant’, which parses the various levels of Velocity template files and generates the substitution file.

The build.xml file specifies that the control template is control.vm, both files residing in the ioc/…/Db directory. The control.vm file then initiates Velocities parse method $generator.parse(…), the parameters of which specify that the primary velocity template is fe.vm and the output file is FExxI-CS-IOC-01.substitutions.

The fe.vm template file then parses all necessary Velocity templates files in the support structures, first specifying all associated parameters (Velocity references).

This is a relatively crude object orientated approach and parameters are passed to the supporting Velocity templates using simple text object (they are Java classes, but behave as strings to all intents and purposes). It would be very much neater to generate some custom Java classes to encapsulate parameters for the support Velocity templates, but time constraints prevented this. Hopefully these classes may be designed in at a later date.

The build model is shown in context in the UML deployment diagram - Illustration:

Illustration 2: IOC Build Deployment Diagram

## Support modules

### PT100

#### Driver configuration

The VxWorks startup script should call:

Hy8403Configure(cardnum, IPACn, IPID, 0, 2, 2, 0, 1, 4, 2)

where IPACn is returned from ipacEXTAddCarrier(…)

e.g. Hy8403Configure(50, IPAC5, A, 0, 2, 2, 0, 1, 4, 2)

### Flow meters (Scalers)

#### Description

The Frontend water flow meters are turbine blade types. The pulses output from the meters are fed into Hy8512 scaler cards, via an Hy8301 transition board. Pulses are acquired over a period of 30 seconds and the total count output to an EPICS record. The counter is then reset and the next sample initiated.

The Front-end support module has a Velocity file: scaler-sub.vm which generates the appropriate database substitutions.

#### Driver configuration

#### Calibration curve

We need to report flow rate in litres per second.

The internal diameter (d) of the pipe at the flow meter is 32mm -- (1).



In order to convert measured frequency into flow rate, the following measurements have been provided:donecalc

|  |  |
| --- | --- |
| Flow Velocity (m/s) | Frequency (Hz) |
| 0.158 | 5.223 |
| 1.199 | 47.918 |
| 1.774 | 71.45 |
| 2.373 | 94.826 |
| 2.963 | 119.157 |
| 3.548 | 142.366 |
| 4.138 | 165.387 |
| 4.723 | 189.309 |
| 5.307 | 213.024 |
| 5.975 | 239.588 |





### Motion Control

#### PBPM

##### XY Table Description

The FMB PBPM is mounted on an X/Y stage. Each axis has the capability to be adjusted either by hand or by stepper motor control.

###### Limits of Travel

10mm (±5.0mm) of travel in each axis.

###### Motor Type

The stepper motors are PK264M-E2.0B (Oriental Motor). These have a rated maximum current of 1.4A – the SmartDrive D75 cards must be configured appropriately (SW1-4: on on off on)

The drive amplifiers are set to half-stepping, so there are 800 steps per rotation. This correlates to 1.25X10-3 mm/step.

###### Encoder

The encoder step size is -0.001mm in the X axis and +0.001mm in the Y axis.

###### Limit Switches

The X and Y axes have two lower limit switches: one for ‘home’ and the other a little further beyond as a motion interlock.

The X lower limits are towards the motor; the Y lower limits are at maximum Y (upwards).

##### EPICS Parameters

##### OMS VME58 & IO58 Card Settings

The encoders are differential and so the OMS IO58 interface board must have the jumpers removed to enable differential (rather than the default single-ended) operation.

The VME58 card must have all I/Odonecalc polarity (J15) jumpers removed.

Being the second VME58 card in the crate, it’s base address should be set to the default of 0xF000 (note that the first VME58 card is for the customised apertures and should be set to 0xE000)

##### OMS MAXv 4000 Card Settings

As with the VME58, the base address requires setting by configuring jumpers as follows:

OMS MaxV Motor Control VME Card Address Selection

Use short (16 bit) addressing. In this mode, only the top 4 bits of J13 are used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Extended 32 bit | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
| Standard 24 bit | A16 | A17 | A18 | A19 | A20 | A21 | A22 | A23 |
| Short 16 bit | X | X | X | X | A12 | A13 | A14 | A15 |

NOTE: A 1 is given by removing a jumper. 0 is given by inserting a jumper.

Default base address = 0xF000 (short)

|  |  |  |  |
| --- | --- | --- | --- |
| A12 | A13 | A14 | A15 |
| 1 | 1 | 1 | 1 |

Memory size = 4096 (0x1000) bytes

For 2 card system: Second card has base address set to 0xF000 (default)

|  |  |  |  |
| --- | --- | --- | --- |
| A12 | A13 | A14 | A15 |
| 1 | 1 | 1 | 1 |

First card must be set 0x1000 below the second card => 0xE000

|  |  |  |  |
| --- | --- | --- | --- |
| A12 | A13 | A14 | A15 |
| 0 | 1 | 1 | 1 |

For a three card system: First card must be set 0x1000 below the second card => 0xD000

|  |  |  |  |
| --- | --- | --- | --- |
| A12 | A13 | A14 | A15 |
| 1 | 0 | 1 | 1 |

#### Customised Aperture

Customisable apertures in the photon front-end permit beamline scientists to select an area of the photon beam cross-section, to pass on to the beamline optics. All radiation outside of the selected area will be absorbed by the aperture.

Customisable apertures have freedom of motion in both X and Y axes. The range of translation is typically ±5 mm and actuated by computer controlled stepper motors.

##### Template Design

As customisable apertures are common across front-ends, the design pattern has been implemented centrally in the Front End EPICS support module (FE).

An aperture is realised in EPICS via a Velocity descriptor file “aperture-sub.vm” within the FE support package. A front-end IOC build will make reference to this Velocity file for each customisable aperture in the front-end, from within the IOC's Db/fe.vm file. The pattern is a logical collection of two motor controlled axes (X and Y) and upwards of four temperature sensors (PT100).

Parameters are passed to the aperture-sub.vm file at build time as follows:

|  |  |
| --- | --- |
| Macro | Description |
| device | ID of this aperture (e.g. FE02I-AL-APTR-01) |
| CHAN\_LIST | A list of PVs to each Hy8403 ADC channel for each PT100 |
| REFCHAN\_LIST | A list of PVs to each Hy8403 ADC reference channel for each PT100 |
| MOTOR\_CARD | The card number of the associated VME58 or MaxV controller |
| MOTOR\_X\_CHAN | Channel number on the controller card associated with the X axis |
| MOTOR\_Y\_CHAN | Channel number on the controller card associated with the Y axis |
| MOTOR\_X\_AXIS | Logical string identifier for the X axis['X'|'Y'|'Z'|'T'] |
| MOTOR\_Y\_AXIS | Logical string identifier for the Y axis['X'|'Y'|'Z'|'T'] |
| MOTORISED | Switch to decide whether or not to create motor records for this aperture ['true' | 'false'] |
| SIMULATED | Switch to allow motor simulation records to be created. Normally 'false' |
| XUREV | If defined, will override the default UREV value in the common aperture-sub.vm file |

The following is an extract from a front-end fe.vm Velocity file, as an example of creating the customisable aperture instances:

#set( $motor\_controller = "OMS VME58" )

#set ($device = "FE${front\_end}-AL-APTR-02")

#set( $CHAN\_LIST = ["FE${front\_end}-CS-H8403-02:AI00","FE${front\_end}-CS-H8403-02:AI01","FE${front\_end}-CS-H8403-02:AI02","FE${front\_end}-CS-H8403-02:AI03"])

#set( $REFCHAN\_LIST = ["FE${front\_end}-CS-H8403-02:AI07","FE${front\_end}-CS-H8403-02:AI07","FE${front\_end}-CS-H8403-02:AI07","FE${front\_end}-CS-H8403-02:AI07"])

#set( $MOTOR\_CARD = 0)

#set( $MOTOR\_X\_CHAN = 0)

#set( $MOTOR\_Y\_CHAN = 1)

#set( $MOTOR\_X\_AXIS = "X")

#set( $MOTOR\_Y\_AXIS = "Y")

#set( $MOTORISED = true )

#set( $SIMULATED = false )

#set( $XUREV = "0.08" )

#parse("aperture-sub.vm")

##### XY Table Description

Presently there exist three types of moveable apertures:

* Dual undulator beam, shown in Illustration

Each aperture is designed to affect just one of the two photon beams. The large hole in each case permits the other beam to pass through unaffected.

* Single undulator beam, shown in Illustration
* Primary slit aperture, shown in Illustration

The two colinear apertures can be independently translated to create a customisable slit. This slit is then treated as an aggregated single object, which can be translated X and Y.

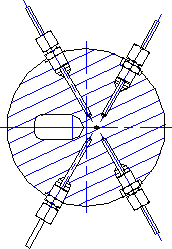


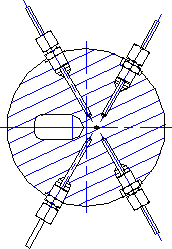
Illustration 4: Dual beam aperture pair

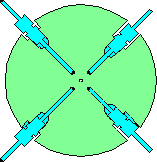
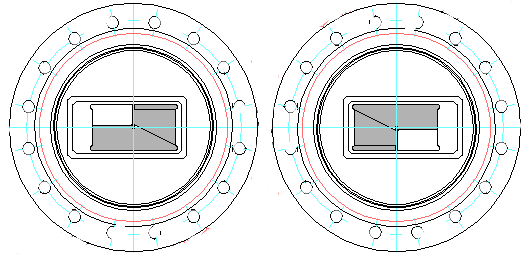
Illustration 3: Single beam aperture

Illustration 5: Defining Slit Apertures (one behind the other)

###### Limits of Travel

Typically ±5 mm in both X and Y.

###### Motor Type

Tyically 1.2A, unipolar stepper motor.

###### Encoder

Renishaw optical, incremental encoder. This varies depending on the resolution requirements.

###### Limit Switches

The low limit switch is high precision (5 µm) and dual purpose, serving as both low limit switch and 'Home' switch.

##### EPICS Parameters

With any stepper motor system, it is important to always drive with sufficient torque to prevent stalling. The torque available from a stepper motor is directly proportional to the number of full steps per second. For stepper motors employed on front-ends, this is typically in the region of between 400 and 1200 pulses per second.

This must be taken into consideration when determining the base velocity (VBAS) and maximum velocity (VMAX).

Most front-end modules (apertures and PBPMs) have a mass of just a few tens of kilograms, so the acceleration can be relatively high (less than 1 mm.s-2). For much larger masses, as found on IR and UV mirror systems, acceleration is more critical and may be in excess of 1 mm.s-2 .

In order to provide and enable signal to the drive amplifier, the Auxilliary output of the OMS cards is used. The Auxilliary line is toggled to energise the required amplifier, by issuing a “AX;AF” premove command (PREM) and “AX:AN” postmove command to switch it off. This example is for the X axis, for other axes it is necessary to use the axis identifier in place of 'X'; so Y axis would be “AY;AF” and “AY;AN”.

The PREM and POST fields of the EPICS motor record are coded in the appropriate Velocity file in the FE support module; such as aperture-sub.vm for apertures and pbpm-sub.vm for PBPMs.

### Locum-4 Current Monitor

The four tungsten blade currents of the standard, undulator front-end PBPM, are converted to a set of corresponding voltages via a current amplifier (ENZ LoCuM4).

The Locum4 has is fully featured and programmable. This is realised via a RS232 serial interface.

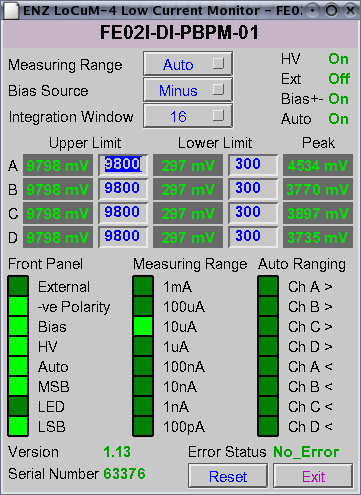
The Locum4 is supported by its own EPICS support module (enzLoCuM4), which provides all communications protocols and EDM screens necessary.

#### Streams Protocol notes

Originally, the EPICS implementation of the Locum4 protocol did a request/receive for all items. Recently, this has been modified to bring all data back in one block, parse it and fan out the resultant items to receiving records.

This significantly streamlined communications.

#### EDM panel



### Front End Vacuum

Vacuum (and other pneumatically actuated) valves are controlled via a six valve control crate, specific to front-end operation.

Each valve control crate has a unique software identifer, which also specifies the domain of the control crate: Front-end valve control crates are divided into two further categories: Insertion Device type (ID) and Bending Magnet type (BM).

The software identifier ranges are as follows:

ID: 0x6100 – 0x61FF

BM: 0x6500 – 0x65FF

These identifier values are used in the feValveGaugeInterlocks.edl EDM panels, to determine which button to make visible for the appropriate gauge interlocks.

# Front End User Interface Architecture

## General design goals

Unlike most other IOCs, the generic front-end control system assumes responsibility across multiple technical areas. These include:

* MPS (MP)
* Vacuum (VA)
* Diagnostics (DI)
* Alignment (AL)
* PSS (PS)
* Radiation Safety (RS)

Displaying all the above information from one perspective proved to be unsatisfactory, so it was considered that views should be available on the main front-end synoptic to switch between perspectives. This was deemed to be best accomplished using some form of 'tabbed' layout. Unfortunately, at the time, EDM didn't provide any built-in mechanism on which this could be readilly implemented. On investigating alternative display frameworks, it was found that QT (the backbone of KDE) could be adopted and using an object oriented language, such as Python, lent itself to integrating well with EPICS channel access libraries.

## QT 3.3 Implementation

### Diamond custom Widgets

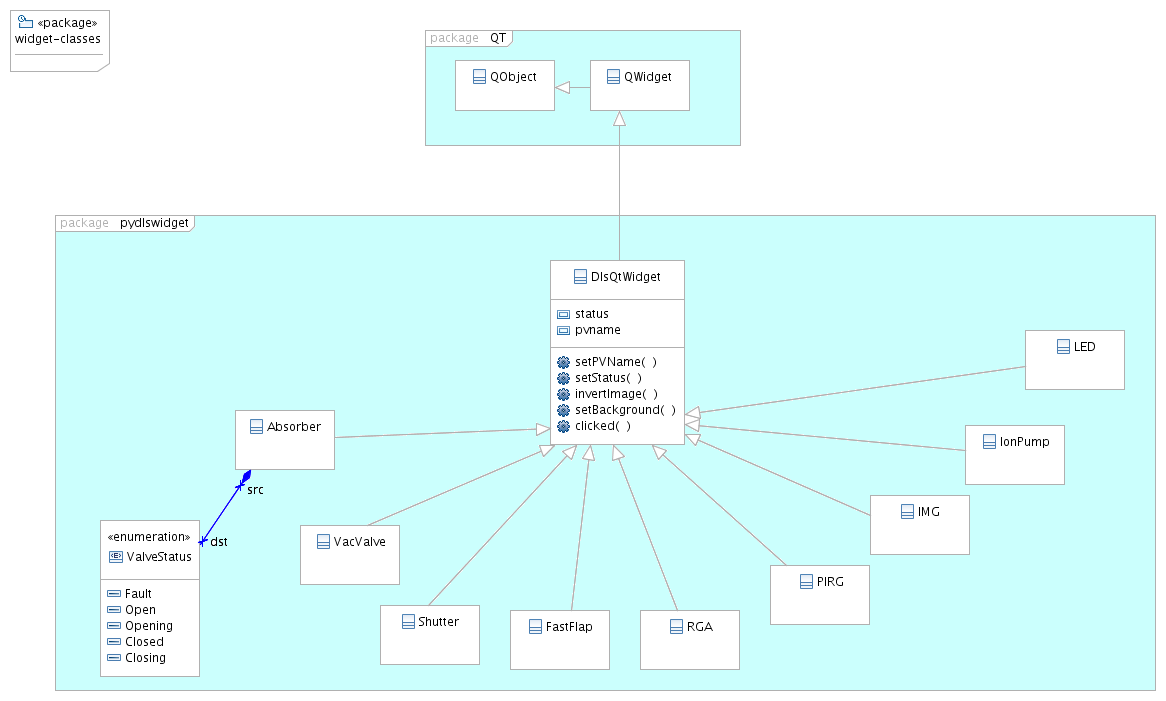
Display frameworks, such as EDM, allow for the building of libraries of both standard and custom control widgets. This needed to be provided in the QT framework. The principal requirements that:

* Must conform to Diamond standards in appearance and functionality.
* Must be EPICS aware and be capable of updating at required rates.

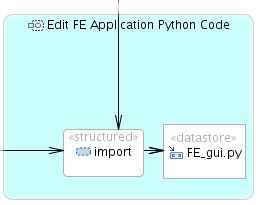
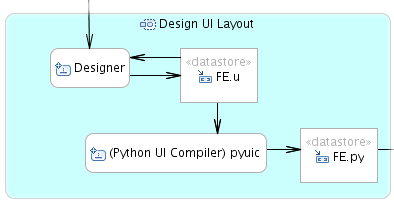
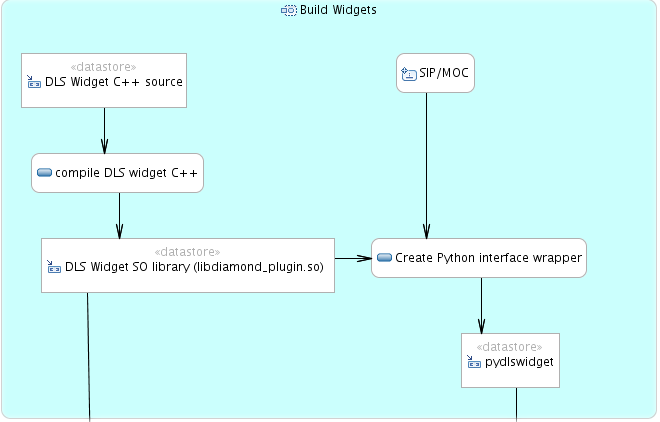
### C++ implementation

The underlying software interface is written in C++ and bound to the C channel access libraries (cothread). The resultant C++ modules are then aggregated into a QT Designer plugin library (libdiamond\_plugin.so) This makes the widgets available from the layout Designer. The C++ widget classes are then wrapped using SIP to create a Python interface (python module pydlswidget), which exposes all the C++ classes and class interfaces as Python equivalents. The Python module (pydlswidget) can then be imported into Python application code and provides full access to the widgets in the QT layout.

The QT layout (user interface file – e.g. FE.ui) must be compiled to be an equivalent Python module, which is also imported into the Python application.



### SIP wrapper



SIP is a tool for generating bindings for C++ classes so that they can be accessed as normal Python classes. SIP is part of the PyQT package, which is a Python interface to the QT framework.

## QT 4 Implementation

(see “Diamond PyQt4 Widget Software User Manual” - TDI-CTRL-SMD-PYQT4-001)

### Design Overview

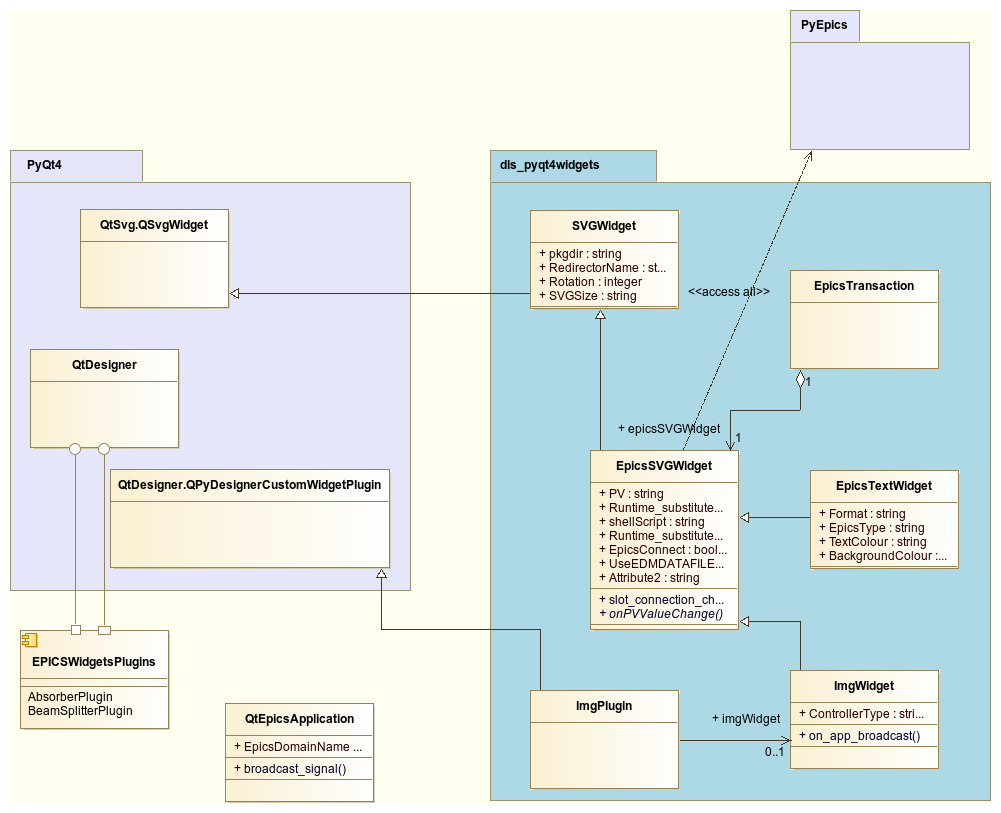
#### Constraints

|  |  |
| --- | --- |
| **Number** | **Constraint** |
| 3.1.1 | In order to preserve continuity and operator familiarity, the user interface components should be similar both functionally and in appearance with the existing EDM style. |
| 3.1.2 | Components (widgets) should adopt a common design pattern where possible. |
| 3.1.3 | EPICS connection status should be visible. |
| 3.1.4 | Device status and severity codes should be handles appropriately. |
| 3.1.5 | Widgets should be EPICS aware at form design time. |
| 3.1.6 | At run-time, the enclosing application should have the ability to change the domain part of every PV on the user interface. |
|  |  |

### Design Patterns

The following describes the common attributes required by the majority of EPICS aware widgets. At the lowest level (level 1), it has been identified that there are graphic attributes common to all widgets. Above level 1 is the EPICS layer (level 2), which encapsulates basic Channel Access connection management and PV callback handling. Above level 2 is further specialisation of EPICS handling, such as text output formatting, text input etc.

This can be visualised in the following diagram:



The PyQt4 package incorporates the user interface design tool QtDesigner along with the Qt Class library, for which there are two available Python interfaces: PyQt4 and PySide. PyQt4 has initially been selected as the wrapper to use for Diamond’s widget set, but PySide has become equally valid and offers an almost identical software interface. The main difference is in licensing, where PySide is Open Source. If desired, it would be straightforward to switch at a later date.

The underlying class to the majority of Diamond widgets is SVGWidget. The class description follows:

**SVGWidget:**

|  |  |  |  |
| --- | --- | --- | --- |
| Class member name | Function or Attribute | Parameters | Description |
| getPkgdir | f |  | in the subclass to provide the FQDN for the derived class module. |
| getSvgFqdn | f | svgfilename | Get the full path name to the SVG file associated with this widget. |
| setAspectRatio | f | x, y | Set the aspect ratio (x:y) for the widget to be honoured during resizing. |
| resizeEvent | f | event | Overrideable. If the aspect ratio has been set then honour it. |
| Author | A |  | Set the widget author name |
| RedirectorName | A |  | Set the name to be used for redirector run-time data. |
| svgUseFilename | f | strFilename | Specify the name of the SVG file to use for the widget. |
| svgLoad | f | Colour | Load the SVG file with the given status colour. The SVG object will be searched for a path with id ‘epicscolour’ and the foreground colour of the path will be set to the given colour. |
| svgChangeStyle | f | Stylename, value | All elements in the SVG object with style identified by Stylename will be changed to the given colour. |
| paintEvent | f | Event | Overridden function from the Qt.QsvgWidget class. This should be further overridden in classes derived from QSvgWidget to manage display of EPICS value/status/severity changes. |
| redirectorPath | f | [optional] nameArg | Returns the string from the redirector for either the class redirector name or nameArg if given. This is equivalent to “configure-ioc show <redirector name>”. |
|  |  |  |  |

EpicsSVGWidget:

This class is derived from SVGWidget and implements the EPICS Channel Access interface.

|  |  |  |  |
| --- | --- | --- | --- |
| Class member name | Function or Attribute | Parameters | Description |
| paintEvent | f | Event | Override of the paintEvent function. In this class, the paintEvent function handles the clipboard interface. If the middle mouse button is pressed (flagged from mousePressEvent() override) then the widget graphics are painted to signify clipboard copy and the value of the associated PV is pasted into the clipboard using the XDND protocol. |
| showUninitialised | f |  | Prior to receiving any EPICS updates, the widget should show a disconnected state - as with EDM. This base class does nothing but act as a pure virtual function which must be overridden in subclasses. |
| epics\_connect | f |  | First disconnects any existing CA connections, sets the widget to display uninitialized, then attempts to connect the given PV name and establish data and connection callbacks. |
| epics\_disconnect | f |  | Disconnects any existing CA connections associated with this class instance. |
| shellScript | A |  | The full pathname to a bash script to be executed when the user clicks on the widget. |
| Pv | A |  | EPICS PV name associated with this class instance. |
| substitutePVDomain | f | newdomain | The full PV name given in the UI file may just be a placeholder to provide the tech-area and device names. If the QApplication object is derived from QtEpicsApplication, then we have the ability to define the EPICS domain name to the application as a run-time argument. |
| substitutePVBase | f | newBase | The full PV name given in the UI file may just be a placeholder to provide the tech-area and device names. This function provides the ability to define the EPICS device base name to the widget at run-time. The device name may be inferred from a full PV name, so we extract the device name as everything before the first ‘:’ |
| Connected | A |  | Returns True if there is a valid CA connection to the PV associated with this class instance. |
| UseEDMenv | A |  | Boolean property to switch between using the preset EDMDATAFILES environment variable or the local support-module-versions file, to invoke EDM screens. |
| onPVValueChange | F |  | Virtual function which must be overridden in the derived class. This gets called whenever the associated PV value changes. |
| onPVConnectionChange | F |  | If the CA connection to the associated PV changes to False, then the widget will display the ‘uninitialised’ graphic state. |
|  |  |  |  |

### Making Diamond Qt4 Widgets Accessible from Qt Designer

PyQt4 includes a common plugin loader for Qt Designer that enables widgets written in Python, with corresponding plugin interfaces defined in the way shown in <REF!!>, to be automatically loaded by Qt Designer when it is run. However, in order for this to work, we need to place the modules containing the custom widget and its plugin class in the appropriate locations in the file system.

By default, modules containing plugin classes can be located in the python directory inside the directory containing the other Qt plugins for Qt Designer. This does not fit well with the software release structure at Diamond, so the PYQTDESIGNERPATH environment variable can be used to refer to the location of the modules containing the plugin classes.

The modules containing the custom widgets themselves only need to be located on one of the standard paths recognized by Python, and can therefore be installed in the user's site-packages directory, or the PYTHONPATH environment variable can be set to refer to their location.

It is essential that **libpythonplugin.so** is installed as a standard Qt Designer plugin. This library makes Qt Designer to be Python aware.

### Runtime Environment

PYTHONPATH must be set to include the location of the widgets in se in the application. The base classes are all located in the dls\_pyqt4widgets support package, but the device specific widgets are located in the relevant support module paths.

As an example, if the base widget library was in /dls\_sw/work/R3.14.11/support/dls\_pyqt4widgets and the RGA support module was at /dls\_sw/work/R3.14.11/support/rga, then the PYTHONPATH environment variable should be set to:

/dls\_sw/work/R3.14.11/support/dls\_pyqt4widgets/dls\_pyqt4widgets: /dls\_sw/work/R3.14.11/support/dls\_pyqt4widgets:/dls\_sw/work/R3.14.11/support/rga/data

PYQTDESIGNERPATH should be set to:

/dls\_sw/work/R3.14.11/support/dls\_pyqt4widgets/designerplugins: :/dls\_sw/work/R3.14.11/support/rga/data

If we intend to use pyEPICS as the Channel Access interface, we need to set:

For x86:

export PYEPICS\_LIBCA=/dls\_sw/epics/R3.14.11/base/lib/linux-x86/libca.so

For x86\_64:

export PYEPICS\_LIBCA=/dls\_sw/epics/R3.14.11/base/lib/linux-x86\_64/libca.so

Under X11 at present it seems necessary to set:

export QT\_GRAPHICSSYTEM=raster

Otherwise the processor usage runs away after an hour or so.

A typical x86\_64 development system would have an environment such as this:

#!/bin/sh

export LD\_LIBRARY\_PATH=/dls\_sw/prod/tools/RHEL6-x86\_64/Python/2-7-3/prefix/lib

# export PYTHONPATH=.:/home/ig43/softdev/python/dls\_pyqt4widgets/dls\_pyqt4widgets:/home/ig43/softdev/python/dls\_pyqt4widgets:/dls\_sw/work/R3.14.11/support/digitelMpc/data:/dls\_sw/work/R3.14.11/support/rga/rgaApp/opi/pyqt/widgets

export PYTHONPATH=/dls\_sw/prod/common/python/dls\_pyqt4widgets/1-16:/dls\_sw/prod/common/python/dls\_pyqt4widgets/1-16/dls\_pyqt4widgets:/dls\_sw/prod/R3.14.11/support/digitelMpc/4-30/data/:/dls\_sw/prod/R3.14.11/support/rga/4-12/data

export PYEPICS\_LIBCA=/dls\_sw/epics/R3.14.11/base/lib/linux-x86\_64/libca.so

export PYQTDESIGNERPATH=/home/ig43/softdev/python/dls\_pyqt4widgets/designerplugins:/dls\_sw/work/R3.14.11/support/digitelMpc/data:/home/ig43/softdev/diamond/rga/data

### Developing Widgets for Support Modules

For support modules which require Qt4 widgets, the can be build in a straightforward manner using the existing dls\_pyqt4widget library.

The RGA support module (rga) is a good working example of implementing Python Qt4 widgets, which are also accessible from Qt Designer.

The following shows the essentials of the Python code structure:

from PyQt4 import QtCore, QtGui, uic

from epics\_epics\_widget import \*

import rgacontrol

# Determine the full path of this custom widget,

# in order to load the SVG file from the same directory

pkgdir = os.path.realpath(os.path.dirname(os.path.abspath(\_\_file\_\_)))

class RgaWidget(EpicsSVGWidget):

"""RgaWidget(EpicsSVGWidget)

Provides a custom widget that shows a vacuum rga.

Various properties are defined so that the user can customize the

appearance of the widget, and change the number and behaviour of the

rga via EPICS.

"""

In this instance, there is a popup dialogue that is presented on clicking the RGA widget. The dialogue is composed using Qt Designer and contains various EPICS aware widgets. The resulting .ui file is compiled during the support module build process (via make) and generates an importable .py file. The dialogue module file in this example is called ‘rgacontrol.py’.

### Developing Qt Applications with Diamond Qt Python Widgets

The development of an application is very straightforward and simply involves a few lines of code to import the necessary forms (composed in Qt Designer – analogous to ‘edm’) and invoke the dialogue.

One important point to note is that, instead of instantiating the standard QApplication class object, one should use the QtEpicsApplication class, which is derived from QApplication. The QtEpicsApplication class allows the setting of the EPICS domain name (e.g. “SR02C”). When a widget is instantiated, it checks to see if the containing application is of type QtEpicsApplication and if so, will look for the derived application’s EPICS domain name, replacing its placeholder domain accordingly. This permits the parameterisation of the EPICS domain name at runtime.

e.g for the Vacuum Lab RGA application – hcui.py:

import sys, signal, os

import logging, logging.handlers

import PyQt4.uic

from PyQt4.QtCore import SIGNAL, SLOT

from app import qtepicsapp

if \_\_name\_\_ == "\_\_main\_\_":

app = qtepicsapp.QtEpicsApplication(sys.argv, epics\_domain\_name = "VALAB")

ui = PyQt4.uic.loadUi('hc.ui')

ui.show()

def quit(\*args, \*\*kwargs):

print 'quit() called'

app.connect(app, SIGNAL("lastWindowClosed()"), app, SLOT("quit()"))

# main loop

app.exec\_()

print 'Terminating \_\_main\_\_'

### dls\_pyqt4widgets Directory Structure

.

|-app

|-designerplugins

|---fe

|-dist

|-dls\_pyqt4widgets

|-----images

#### app/

**qtepicsapp.py:**

Provides the QtEpicsApplication class which has a superclass of QApplication which provides facilities specific to EPICS applications, such as the EPICS domain name to allow scripting of PV names. Child widgets can determine if the parent application is a QtEpicsApplication and use this class to facilitate marshalling. Child widgets can also subscribe to broadcast signals from this QtEpicsApplication instance and from the JSON data, determine any action required by the signal.

**environment.py:**

Provides the Environment class, which will construct EDMDATAFILES and PYTHONPATH environment variables from a list of configure-ioc redirector entries. Useful for python applications to import the desired modules and invoking of EDM displays.

**substitution.py:**

Importable module which provides a set of helper functions to manipulate name substitution.

substitute\_epics\_domain(epics\_domain, pvname)

epics\_domain is the PV prefix to be written over that in pvname.

The resultant string is returned.

substitute\_epics\_field(epics\_field, pvname)

epics\_field is the PV suffix to be written over that in pvname. That is everything to the right of the last ':'

The resultant string is returned.

substitute\_epics\_cell(epics\_cell, pvname)

epics\_cell is the PV cell number to be written over that in pvname.

The resultant string is returned.

#### designerplugins/

A collection of python files, each containing a plugin definition to be registered with Qt Designer. Each widget plugin class is derived from QtDesigner.QPyDesignerCustomWidgetPlugin and acts as a run-time class factory for Designer.

For Python modules to be recognised natively by Qt Designer, it is necessary to ensure that the libpythonplugins.so library is available on LD\_LIBRARY\_PATH. It is also necessary to include the designerplugins directory on the PYQTDESIGNERPATH environment variable.

Good documentation on how to integrate custom widgets with Qt Designer is available on-line.

The contents of designerplugins are only required during application development and if using Qt Designer.

#### dls\_pyqt4widgets/

This directory contains all the commonly used Qt widgets, such as text boxes (edit and display), menu button, indicators etc. Widgets associated with specific support modules should be incorporated inside the support module. In all cases, it will be necessary to include the widget directories in the PYTHONPATH environment variable (as with EDMDATAFILES for EDM widgets).

Many of the widget classes are derived from EpicsSVGWidget, located in the epics\_epics\_widget.py module. It is very straightforward to create EPICS aware custom widgets, as the EpicsSVGWidget class provides all the necessary underlying connection management, callbacks and signals.

The images/ subdirectory contains the SVG and other images required by the widgets.

#### Makefile

Makefile is used (in conjunction with setup.py) when releasing a new version of the widgets.

#### setup.py

setup.py is written as a standard python package installation script. It should rarely, if ever, need modifying and is used by the Makefile at release time on the build server.

### Deployment

This section provides a brief overview on suggested deployment of the widgets in applications. The Diamond Front-end application will be heavily referenced, as this exemplifies all of the workflow described in this document.

#### Application Python Framework

A typical EPICS Qt application will require parsing of command line arguments and invoking a user interface. With much of the functionality residing within the EPICS aware widgets, the framework code can be very small. Occasionally it may be desirable to take advantage of the versatility that Python has to offer, by augmenting the application, either directly in the framework, or by subclassing the user interface form, whether this be classified as a ‘Main Window’ or ‘Dialogue’, which will be described in more detail in section 3.3.9.2.

Examining the ***FEQt4*** support module (which is the Diamond Front-end application framework), the executable application is Python/dls\_FEQT4.py. Aside from the expected imports of PyQt modules, it is very highly advantageous, although not essential, to import qtepicsapp module, which provides the QtEpicsApplication class (see section 3.3.6 for a more detailed description).

require("dls\_pyqt4widgets")

from app import qtepicsapp

Every Qt application has to have an instance of QApplication (or a subclass thereof). By instantiating QtEpicsApplication, it is possible to utilise its understanding of Diamond’s EPICS naming convention, with the ability to get the EPICS domain name and update all the PV names for widgets in the given user interface, with PV names with the placeholder domain replaced by the domain extracted from the name given as a run-time argument.

app = qtepicsapp.QtEpicsApplication(sys.argv, strFE)

In this case, the name of the front-end is passed to the QtEpicsApplication instance and is stored, ready for when widgets are instantiated. A widget derived from EpicsSVGWidget has properties Runtime\_substitute\_domain and Runtime\_substitute\_cell (which are visible and settable in Qt Designer). If Runtime\_substitute\_domain is True then its placeholder PV name (say ‘FE02I-VA-IMG-01:P’) has the domain portion replaced with that of the given name, which if it were say ‘FE23I’ then the widget would connect to the PV ‘FE23I-VA-IMG-01:P’. If Runtime\_substitute\_cell is flagged True, then just the cell number is substituted.

### Building the User Interface

A Qt user interface is very simple to build, using Qt Designer. It can be assembled in code, but here I shall discuss the Designer method.

#### Invoking the User Interface from the Framework

The simplest method to create a user interface is via Qt Designer, which generates a UI file. The UI file is an XML model of the user interface, describing the widgets and their default properties. There are two ways to launch the user interface from the framework code:

1. Compile the UI file using the ‘pyuic4’ tool, the output of which is an importable python module. This module contains a class, which when instantiated will create the user interface.
2. Load the UI file at run-time, via a call to PyQt4.uic.loadUi(), which creates an instance of the user interface, exactly as described in the UI file.
3. Load the UI file as a Python class type via PyQt4.uic.loadUiType(). This has the significant advantage of allowing the returned class to be subclassed and extended functionality incorporated. A good example of this is the RGA support module (‘rga’).

#### Customising User Interface Behaviour

If specialised behaviour is required, then it will be necessary to adopt option (c) from the previous section.

The following is an example from the RGA support module and is cut-down to show the essentials.

rga\_form\_class, rga\_base\_class = PyQt4.uic.loadUiType(rgaui)

class RgaControl(rga\_base\_class, rga\_form\_class):

def \_\_init\_\_(self, parent=None, epics\_device\_name=None):

rga\_base\_class.\_\_init\_\_(self, parent)

self.setupUi(self)

self.connect(self, QtCore.SIGNAL('epics\_connection\_change()'), QtCore.SLOT('slot\_epics\_connection\_changed()'))

In the case of the RGA support module, subclassing was employed to facilitate spectrum plotting, zooming and mass selection.

#### Deployment from Launcher

The standard DLS Launcher application incorporates a Python script, feqt4gui.py, which provides a good example for a suitable method to invoke Qt4 based applications. It is important that a suitable PYTHONPATH is constructed, to permit the application to locate the widgets in use. If EDM screens are called, then it’s also important to construct a suitable EDMDATAFILES environment variable. For Front-end applications, all required support modules are registered with the redirector via ‘configure-ioc’. Each ‘IOC’ is just an entry to any suitable file within the directory to be added to the PYTHONPATH or EDMDATAFILES, usually the ‘data/’ directory of the relevant support module. The Python script launcher reads the redirector entries and depending on how they are flagged (\_use\_in\_python\_path or \_use\_in\_edm\_path or both) will construct the environment variables accordingly.

The Front-end ‘feqt4gui.py’ launcher script contains the following table:

support\_modules = (['dls-qt4widgets-widgets' , \_use\_in\_python\_path],

['dls-qt4widgets-app' , \_use\_in\_python\_path],

['FE-SUPPORT-rga' , \_use\_in\_edm\_path | \_use\_in\_python\_path],

['FE-SUPPORT-devIocStats-soft', \_use\_in\_edm\_path ],

['FE-SUPPORT-devIocStats-vx' , \_use\_in\_edm\_path ],

['FE-SUPPORT-mks937b-img' , \_use\_in\_edm\_path ],

['FE-SUPPORT-mks937a' , \_use\_in\_edm\_path ],

['FE-SUPPORT-TimingTemplates' , \_use\_in\_edm\_path ],

['FE-SUPPORT-digitelMpc' , \_use\_in\_edm\_path | \_use\_in\_python\_path],

['FE-SUPPORT-insertionDevice' , \_use\_in\_edm\_path ],

['FE-SUPPORT-vacuum' , \_use\_in\_edm\_path ],

['FE-SUPPORT-vacuumValve' , \_use\_in\_edm\_path ],

['FE-SUPPORT-vacuumSpace' , \_use\_in\_edm\_path ],

['FE-SUPPORT-motor' , \_use\_in\_edm\_path ],

['FE-SUPPORT-eurotherm2k' , \_use\_in\_edm\_path ],

['feqt4-gui' , \_use\_in\_python\_path]

)

The first item in each tuple specifies the redirector entry and the second entry is a flag to determine whether the path from the redirector should be appended to either PYTHONPATH, EDMDATAFILES or both.

Examination of the source code of Front-end launcher script, feqt4gui.py, show the simplicity of the code and is easily adapted as a basis to launch any Python application.

## Python

### Software Suite Structure

All front-end synoptics are launched via the Python application dls\_FEQT4.py in the *FEQt4* support module.

Within the support module is a collection of Qt .ui files, which have been generated by designer. An XML file (configuration.xml) contains all the information required to launch the appropriate front-end synoptic. One element of each <FrontEnd> entry is <ui></ui> which dictates which ui file should be invoked for the front-end in question.

The is also an XML schema file (configuration.xsd) which helps to impose certain rules on the XML data.

Any time a new front-end if added to the Synchrotron, it will be necessary to make an appropriate entry in configuration.xml.

# SNL State Machines

Each front-end IOC runs a state-machine in the background, to manage the correct order of opening and closing of valves, along with ion pump startup procedures.

State Transition Diagrams

### Gate Valve Control

Statediagram 1: Gate valve control helper state-machine

### Beam Path Control

[NB: For beamline control interface, see 4.1.5 Beamline Gateway Control Interface]

Statediagram 2: Beam block/open control helper state-machine

### Pump Control

### Status PV Binary Bit Pattern Description

#### Valve Status PV ({dom}-VA-SEQ-01:STA)

This 16 bit short tracks the status of the valve open/close sequence state machine.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| ERRNO | | | | X | X | FFLAP | V-02 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| V-01 | X | ARM | CLOSE | OPEN | ABRT | COMP | INIT |

|  |  |
| --- | --- |
| Field | Description |
| INIT | Sequence initiated flag  1 = Initiated  0 = Idle |
| COMP | Sequence completed flag |
| ABRT | Sequence aborted flag |
| OPEN | ‘Opening’ sequence activate |
| CLOSE | ‘Closing’ sequence activate |
| ARM | Fast-Flap arming |
|  |  |
| V-01 | Status relates to VALVE-01 |
| V-02 | Status relates to VALVE-02 |
| FFLAP | Status relates to FVALV-01 |
|  |  |
| ERRNO | Sequence error code |
|  |  |

|  |  |
| --- | --- |
| ERRNO | Error condition |
| 0 | No error |
| 1 | Interlock inhibit |
| 2 | Operation timeout |
| 3 |  |

#### Beam Status PV ({dom}-VA-SEQ-02:STA)

This 16 bit short tracks the status of the beam path open/close sequence state machine.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| ERRNO | | | | X | ABSB-02 | ABSB-01 | SHTR-02 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SHTR-01 | X | X | CLOSE | OPEN | ABRT | COMP | INIT |

Table 1

|  |  |
| --- | --- |
| Field | Description |
| INIT | Sequence initiated flag  1 = Initiated  0 = Idle |
| COMP | Sequence completed flag |
| ABRT | Sequence aborted flag |
| OPEN | ‘Opening’ sequence activate |
| CLOSE | ‘Closing’ sequence activate |
| SHTR-01 | Status relates to SHTR-01 |
| SHTR-02 | Status relates to SHTR-02 |
| ABSB-01 | Status relates to ABSB-01 |
| ABSB-02 | Status relates to ABSB-02 |
| ERRNO | Sequence error code |

|  |  |
| --- | --- |
| ERRNO | Error condition |
| 0 | No error |
| 1 | Interlock inhibit |
| 2 | Operation timeout |
| 3 |  |

#### Pump Status PV ({dom}-VA-SEQ-03:STA)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| ERRNO | | | | X | IONP-05 | IONP-04 | IONP-03 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IONP-02 | IONP-01 | X | STOP | START | ABRT | COMP | INIT |

Table 2

|  |  |
| --- | --- |
| Field | Description |
| INIT | Sequence initiated flag  1 = Initiated  0 = Idle |
| COMP | Sequence completed flag |
| ABRT | Sequence aborted flag |
| START | ‘START’ sequence activate |
| STOP | ‘STOP’ sequence activate |
| IONP-01 | Status relates to IONP-01 |
| IONP-02 | Status relates to IONP-02 |
| IONP-03 | Status relates to IONP-03 |
| IONP-04 | Status relates to IONP-04 |
| IONP-05 | Status relates to IONP-05 |
| ERRNO | Sequence error code |

|  |  |
| --- | --- |
| ERRNO | Error condition |
| 0 | No error |
| 1 | Interlock inhibit |
| 2 | Operation timeout |
| 3 |  |

### Beamline Gateway Control Interface

A state-machine has been deployed which bundles the above into strict control of the front-end beam access from the beamline. The PV **FExxx-CS-BEAM-01** provides the interface to this state-machine. The interface definition is:

This PV is monitored by the vacumm control state-machine on the front-end IOC. When a control request is received via :CTL, the state-machine acts on  
the request in a strictly controlled fashion, reporting back status as states change via the :STS PV.

#### FExxx-CS-BEAM-01:STS

This PV is updated by the state-machine to inform client applications as to the most recent transition.

Bits 16 to 30 define the front-end device last attempted to be actuated by the state-machine. Bits 0 to 15 define the outcome of the attempted actuation.

Bit 31 is set when ever the state-machine is active, i.e. not in either of the stable states OPEN or CLOSED.

For example: 0x00400008 implies that the absorber is open.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***31*** | ***30*** | ***29*** | ***28*** | ***27*** | ***26*** | ***25*** | ***24*** |
| ACTIVE | X | X | X | X | X | X | X |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***23*** | ***22*** | ***21*** | ***20*** | ***19*** | ***18*** | ***17*** | ***16*** |
| X | X | SHTR2 | SHTR1 | ABSB | VALVE2 | VALVE1 | FASTV |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***15*** | ***14*** | ***13*** | ***12*** | ***11*** | ***10*** | ***9*** | ***8*** |
| X | X | X | X | X | X | X | X |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***7*** | ***6*** | ***5*** | ***4*** | ***3*** | ***2*** | ***1*** | ***0*** |
| TIMEOUT | INTERLK | ARM | CLOSE | OPEN | ABORT | DONE | INIT |

#### FExxx-CS-BEAM-01:CTL

This PV is written to by the client application, to request the state-machine to do something.

The permitted values are:

|  |  |  |
| --- | --- | --- |
| Value | Command | Desctription |
| 0 | Idle | The state machine has no requests queued |
| 1 | Open | Request to attempt to open V2, SHTR2, ABSB (and fast valve) |
| 2 | Close | Request to close ABSB, SHTR2, V2 |
| 3 | Abort | Abort any current command |

### dlsPLC

Note that as of January 2014, vacuumValve support module is deprecated. Instead use dls\_PLC and select the appropriate specialised template model for the device being instantiated. The new method makes use of BUSY record for beamline GDA synchronising and as devices differ in their specification of BUSY, it's important to pass in the correct 'donecalc' macro to the template. A sensible default is provided for each template in the support module, but is is possible to override it.



# XBPM Data Handling

Where appropriate, a Front End will contain two X-ray Beam Position Monitors (XBPM – synonymous with PBPM). Each XBPM has four tungsten blades which are situated within the beam tube, as an array around the nominal beam centre.

A change to the proximity of the photon beam will cause a corresponding change to the current flowing through each tungsten blade. The current is measured via a current monitor (Locum-4), which can measure currents in the order of a few pA to several mA.

Illustration 6: XBPM cross sectional view

It is worthy of note that the photon beam is not circular, but commonly elliptical in cross-section; in some instances, the beam will exhibit side-lobes and may also be asymmetrical. This can complicate the determination of the notional beam centre.

The current monitor will amplify the blade currents and emit voltages on four channels, each channel being an analogue of the corresponding blade current.

The voltages are read by the IOC via 16bit ADCs (Hy8401).

The x,y position of the beam is determined thus:

The x and y ordinates are determined in real world distance units by using appropriate Kx, Ky, Offsetx and Offsety coefficients. These are determined by calibration of the XBPM in both X and Y axis and also for varying ID gap distances. The resultant values are stored in a look-up table for each Front End.

## Loading initial ID gap and phase lookup tables

The initial values of the table are manually written to an XML file and the load.py Python script (located in the FE support module) used to establish the tables in EPICS.

The XML file should have the following example format:

<TABLES PV="FE02I-DI-PBPM-01:X">

<KTABLE>

<GAP>

7,8,9,10,11,12,13,14,15

</GAP>

<PHASE>

0,180

</PHASE>

<LOOKUP>

2.37969611709016,

2.20609516650667,

2.15001453412056,

2.13621002243808,

2.16059705156271,

2.20635295237895,

2.26916945099169,

2.36786303200801,

2.52991488149693,

2.37969611709016,

2.20609516650667,

2.15001453412056,

2.13621002243808,

2.16059705156271,

2.20635295237895,

2.26916945099169,

2.36786303200801,

2.52991488149693

</LOOKUP>

</KTABLE>

<OFFSETTABLE>

<GAP>

7,8,9,10,11,12,13,14,15

</GAP>

<PHASE>

0,180

</PHASE>

<LOOKUP>

0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0

</LOOKUP>

</OFFSETTABLE>

</TABLES>

There should be one XML table for each XBPM axis. The <LOOKUP> section is comprised of factors for each GAP element, this then repeated for each PHASE element as appropriate.

In order to load the table into EPICS, run the load.py script thus:

load.py <table.xml> <XBPM PV base for this axis>

e.g. load.py table-FE02I-DI-PBPM-01-X.xml FE02I-DI-PBPM-01:X

Repeat this for each PBPM and each axis.

Once a table is loaded, data are preserved via Autosave/Restore.

## Data Flow Diagram for XBPM data handling



# Linux Servers

As of Phase III installation, front-end control systems have been based on Linux servers running soft-iocs. This has had many benefits, including

* Reduction of rack space.
* Greater diversity in use of remote I/O (RIO), such as EtherCAT and PLC RIO.
* Significantly more flexibility in partitioning system functionality across multiple soft-iocs.
* Reduced development and commissioning time.

## INIT

On booting the Linux server, it will fetch specific start-up instructions from the scripts registered for the server in the INIT support module.

The server will be registered in INIT by its host name, e.g. FE24B-CS-RSERV-01. Note that FE05I and FE21B servers have assumed an old naming convention, where they are named FExxt-CS-SERV-01 instead of the latest standard of FExxt-CS-RSERV-01.

### Script files within the server directory

#### 01-init.sh

This is called at boot run-level 1 to initialise hardwared address etc. An example is shown below:

#!/bin/bash

echo "Starting Init"

sudo /sbin/ip addr add 10.0.0.1/8 brd + dev eth1

sudo /sbin/ip addr add 192.168.0.1/16 brd + dev eth2

sudo /sbin/ip link set dev eth1 up

sudo /sbin/ip link set dev eth2 up

# export MASTER0\_DEVICE=`/sbin/ip addr show dev eth1 | sed -e's/^.\*link[^ ]\* \([^ ]\*\) .\*$/\1/;t;d'`

# echo "MAC=$MASTER0\_DEVICE"

for i in $(seq 10); do

ip link show | grep eth1 | grep "UP,LOWER\_UP" && break || echo "Network interface not ready"

sleep 1

done

sudo /sbin/service ethercat start

caRepeater.sh, conserver.sh, cron-install-crontabList.sh are all soft-linked from the parent directory.

#### ethercat

Assigns the Ethernet port to be used for the EtherCAT network (if applicable).

Note that the MASTER0\_DEVICE MAC address is derived in this script by reading the specified ports (e.g. eth1). An example is shown below:

#--------------------------------------------------------------------------

#

# EtherCAT master sysconfig file

# Master devices.

#

# The MASTER<X>\_DEVICE variable specifies the Ethernet device for a master

# with index 'X'.

#

# Specify the MAC address (hexadecimal with colons) of the Ethernet device

# to

# use. Example: "00:00:08:44:ab:66"

#

# The broadcast address "ff:ff:ff:ff:ff:ff" has a special meaning: It tells

# the master to accept the first device offered by any Ethernet driver.

#

# The MASTER<X>\_DEVICE variables also determine, how many masters will be

# created: A non-empty variable MASTER0\_DEVICE will create one master,

# adding

# a non-empty variable MASTER1\_DEVICE will create a second master, and so

# on.

#

MASTER0\_DEVICE=`/sbin/ip addr show dev eth1 | sed -e's/^.\*link[^ ]\* \([^ ]\*\) .\*$/\1/;t;d'`

#

# Ethernet driver modules to use for EtherCAT operation.

#

# Specify a non-empty list of Ethernet drivers, that shall be used for

# EtherCAT operation.

#

# Except for the generic Ethernet driver module, the init script will try to

# unload the usual Ethernet driver modules in the list and replace them with

# the EtherCAT-capable ones. If a certain (EtherCAT-capable) driver is not

# found, a warning will appear.

#

# Possible values: 8139too, e100, e1000, r8169, generic.

# Separate multiple drivers with spaces.

#

# Note: The e100, e1000, r8169 and generic drivers are not built by default.

# Enable them with the --enable-<driver> configure switches.

#

DEVICE\_MODULES="generic"

#

# Flags for loading kernel modules.

#

# This can usually be left empty. Adjust this variable, if you have problems

# with module loading.

#

#MODPROBE\_FLAGS="-b"

#### soft-iocs

Contains a list of IOCs to be loaded and run on the server. Each IOC must first have an entry in the redirectory table (ioc-configure). Against each IOC is a unique TCP port number, which is used to facilitate a console interface to each running IOC.

An example is shown below:

FE24B-MO-IOC-01 7018

FE24B-ethercat-scanner 7017

FE24B-CS-IOC-01 7016

## Controls network IP address

The address on the Controls network will be assigned via DHCP. This will vary depending on whether it's on the Controls Development or Primary network. Either way, one of the Linux Sysadmins will need to make the entry in the appropriate DHCP server and in DNS. The MAC address of the appropriate Ethernet port will be required.

## Private network address

Most of the local devices, such as valve control creates, GeoBrick, etc. will be on a private (instrumentation) network inside the control rack. The interace allocated for this network is configured via 01-init at boot time (see 6.1.1.1 above). Typically, for front-ends the network address will be 192.168.0.0/16.

# PLC Remote I/O

During 2013, it was decided to rationalise the critical interlocks within the front-end control systems. This incorporates adding intelligence into the PLC to drop the Gobal MPS interlock on an interlock voting scheme.

This has had the additional benefit of removing a significant number of dedicated interface within the control rack, which would have read temperatures and flow switches via IOC hardware, such as VME input or EtherCAT devices.

PLC RIO is handled via the dlsPLC support module.

## PLC Register Mapping