In [107]:
```
%config InlineBackend.figure_format='svg'
import numpy
```

In [108]:
```
#load the surf module
import surf
#load the surf_data module
import surf_data

#create instances of these classes
dev=surf.Surf()
devData=surf_data.SurfData()
```

In [109]:
```
#check the pedestal level
#pedestal in mV = value * 2048 / 4096
dev.vped
```

Out[109]:
```
4000
```

In [110]:
```
#change the pedestal level
dev.set_vped(1800)
```

In [111]:
```
#re-check pedestal level
dev.vped
```

Out[111]:
```
1800
```

In [112]:
```
#the pedestal variable is set by reading the on-board DAC. You can access that directly like this
dev.i2c.read_dac()
```

```
Reading from MCP4728...
DAC channel A (RFP_VPED_0):  register is set to 0x9c4, EEPROM is set to 0x9c4
DAC channel B (RFP_VPED_1):  register is set to 0x578, EEPROM is set to 0x578
DAC channel C (RFP_VPED_2):  register is set to 0x578, EEPROM is set to 0x578
DAC channel D (VPED)      :  register is set to 0x708, EEPROM is set to 0x708
```

Out[112]:
```
1800
```

In [34]:
```
#re-take pedestal data
devData.pedestalRun()
```

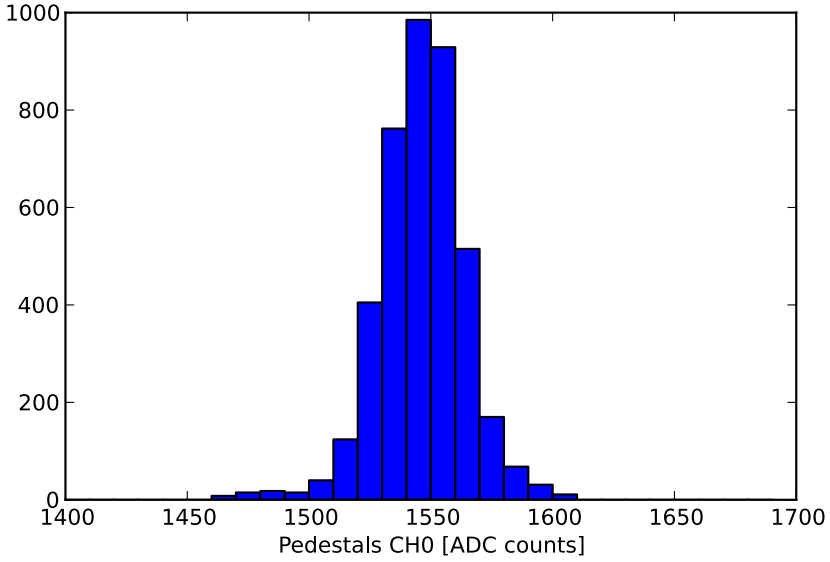
```
Saving pedestals for board CANOES...
```

Out[34]:
```
array([[1548, 1707, 1540, ..., 1625, 1559, 1623],
       [1555, 1724, 1546, ..., 1653, 1572, 1646],
       [1544, 1725, 1520, ..., 1627, 1571, 1643],
       ...,
       [1550, 1767, 1521, ..., 1639, 1599, 1546],
       [1529, 1757, 1549, ..., 1647, 1573, 1566],
       [1543, 1734, 1535, ..., 1602, 1573, 1467]])
```

In [113]:
```
#pedestals are saved to json file: calibrations/surf_calibrations.json
#they are also defined as a class variable:
devData.loadPed()

#histogram the pedestals on CANOES, channel=0
pylab.hist(devData.pedestals[:,0], bins=range(1400, 1700, 10))
pylab.xlabel('Pedestals CH0 [ADC counts]')
```

Out[113]: <matplotlib.text.Text at 0x150ced0c>



In [43]:
```
#log some data to check the baseline
#take 50 events and don't save to a file, otherwise make save=True and specify a filename by filen
ame='xxxxxx'
data = devData.log(50, save=False)

#this automatically subtracts the pedestals from the raw data. To not subtract pedestals, specify
subtract_ped=False
#this automatically unwraps the data given the trigger position. To not unwrap, specify unwrap=Fal
se
```

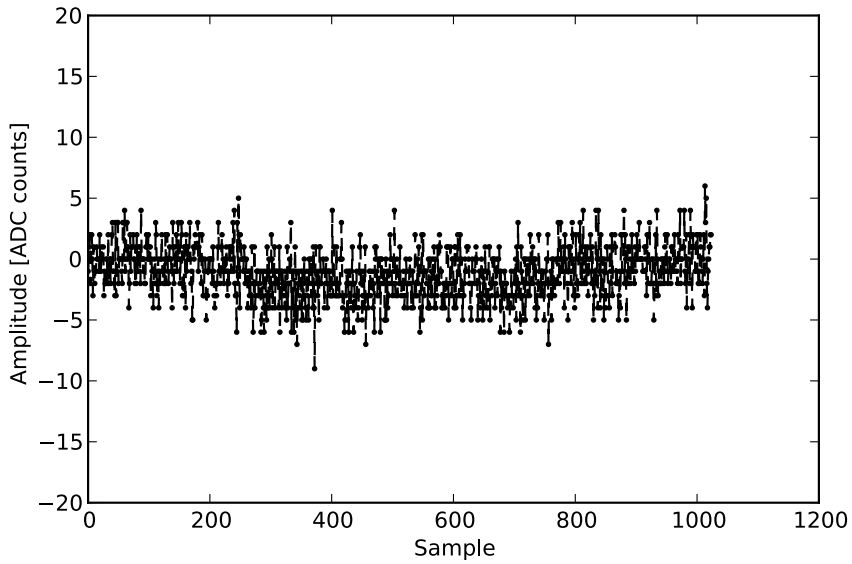
```
logging event...50
```

In [49]:
```
#the data is stored in a list of lists
print 'number of events', len(data), ' -- number of channels', len(data[0]), '-- number of samples
 per channel per event', len(data[0][0])
```

```
number of events 50  -- number of channels 12 -- number of samples per channel per event 1024
```

In [114]:
```
#plot event 5 on channel 0
pylab.plot(data[5][0][:], 'o--', color='black', ms=2)
pylab.xlabel('Sample')
pylab.ylabel('Amplitude [ADC counts]')
pylab.ylim([-20,20])
```

Out[114]: (-20, 20)

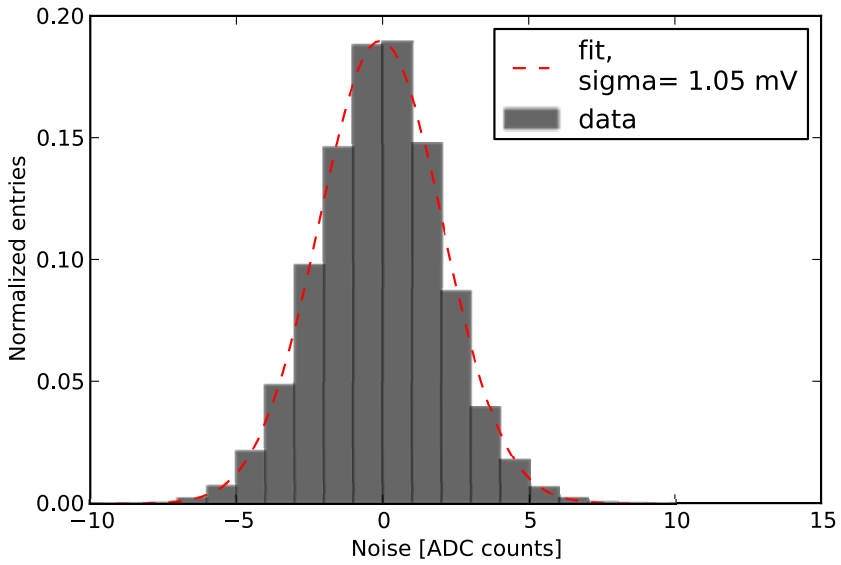


In [115]:
```
#look at the electronics noise on channel 0:
data=numpy.array(data)
pylab.hist(data[:,0,:].flatten(), bins=range(-10, 11, 1), normed=True, color='black', alpha=0.6, l
abel='data')
pylab.xlabel('Noise [ADC counts]')
pylab.ylabel('Normalized entries')

from scipy.stats import norm
mu, std=norm.fit(data[:,0,:].flatten())
x=numpy.linspace(-10, 11, 100)
pdf=norm.pdf(x, mu+0.5, std) #+0.5 to take into account bin edge -> bin center
pylab.plot(x, pdf, '--', color='red', label='fit,\nsigma= {:.2f} mV'.format(std/2)) #just about ex
actly 2 ADC counts per mV with current configuration

pylab.legend()
```

Out[115]: <matplotlib.legend.Legend at 0x14f834ec>



In [83]:
```
#do a pedestal scan, start/stop are pedstal DAC values
#saves to file pedscan.temp (can redefine filename, with filename='xxxxx' argument)

#this takes a few minutes to run
dac_values, lab_values = devData.pedestalScan(start=0, stop=4096, incr=100)
```

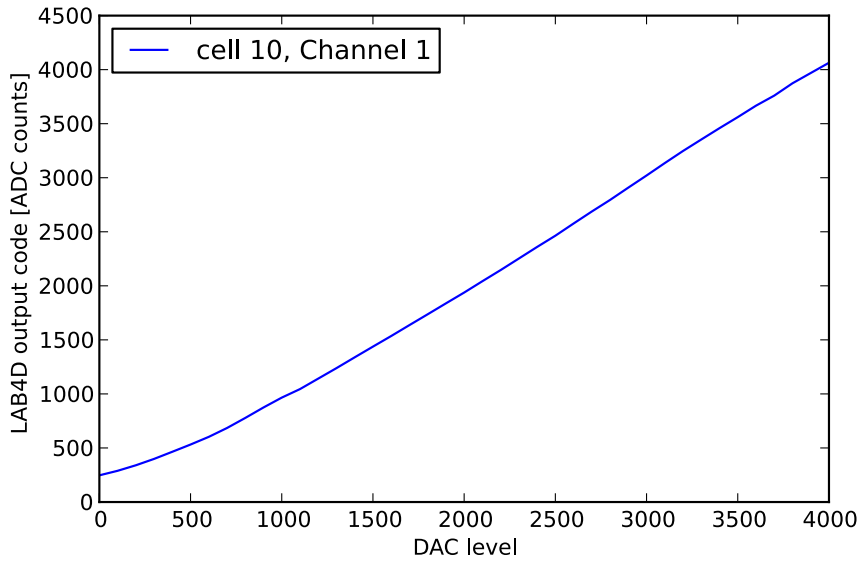
```
logging event...120
```

In [116]:
```
#dac_values = list of values of the pedestal DAC
#lab_values = list of array of pedestal values
print 'size of pedestal scan output array (number of scan points, number of cells per channel, num
ber of channels) =', numpy.array(lab_values).shape

#plot the tranfser curve for storage cell 10 on Channel 1:
pylab.plot(dac_values, numpy.array(lab_values)[:,10,1], label='cell 10, Channel 1')
pylab.xlabel('DAC level')
pylab.ylabel('LAB4D output code [ADC counts]')
pylab.legend(loc='upper left')
```

```
size of pedestal scan output array (number of scan points, number of cells per channel, number o
f channels) = (41, 4096, 12)
```

Out[116]:
```
<matplotlib.legend.Legend at 0x14a5c48c>
```



In [101]:
```
#code exists to generate a linear-interpolated LUT based on the output file from the pedestal scan
#however, it is not yet implemented in the readout or handled to a calibration file...TO DO!

#it can be generated using this:
surf_lut = devData.makeSurfLUT('calibrations/pedscan.temp', pedscan_start=0, pedscan_interval=100)
```

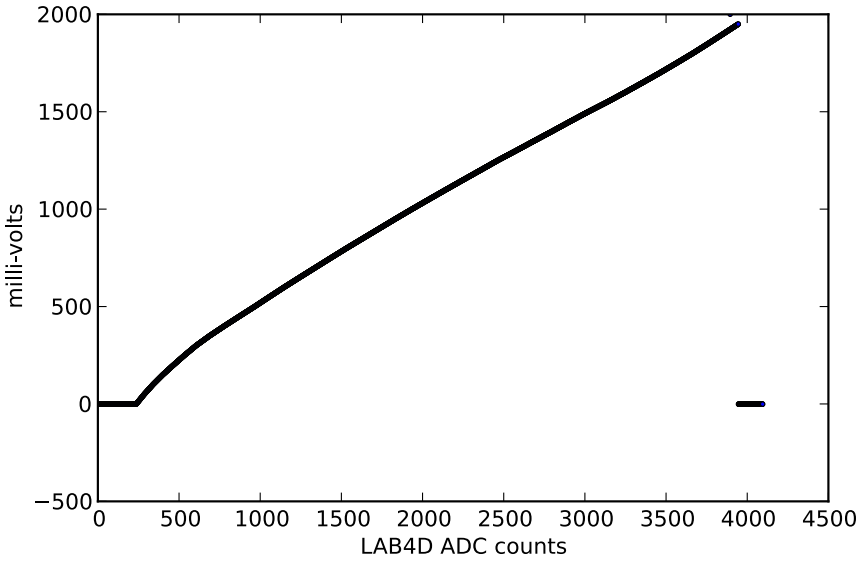
In [102]:
```
#this LUT is made by taking the mean response of each channel.
#It does not generate a LUT for each storage cell, which is probably the way to do this in the end
.
surf_lut.shape
```

Out[102]:
```
(12, 4096)
```

In [117]:
```
#plot the LUT for channel 1:
pylab.plot(surf_lut[1], 'o', ms=2)
pylab.xlabel('LAB4D ADC counts')
pylab.ylabel('milli-volts')

#note: values of -1 indicate the LUT is poorly defined (no mapping)
```

Out[117]: <matplotlib.text.Text at 0x14f8fe4c>



In [ ]: