In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pymysql.cursors
from scipy.optimize import curve_fit
from scipy.ndimage import median_filter, gaussian_filter
from importlib import reload
import pandas as pd
boltzmann = 1.38064852e-23
m = 2.69838e-25 #kg
h = 6.62607004e-34
```

In [2]:
```
import databaseLibrary.databaseCommunication as db
```

In [3]:
```
#N_cal = 1.94 #Multiply fit for Beta by N_cal to correct for atom number
N_cal = 1
#omega_x,y,z in Hz, with 2*pi factored out, nu
#f_axial, in Hz, with 2*pi factored out, nu
#T in nK
#returns volume in cm^3
def effectiveVolumeODT(T, omega_x, omega_y, omega_z):
    return 1e6*np.sqrt(8)*(2*np.pi)**(3/2)*((T*10**(-9))*boltzmann/m)**(3/2)*

def effectiveVolumeLattice(T, omega_x, omega_y, omega_z, f_axial):
    a_perp = np.sqrt((h/(2*np.pi))/(m*2*np.pi*f_axial))
    wavelength = (741)*10**(-9)
    return 1e6*np.sqrt(8)*(2*np.pi)**(3/2)*((T*10**(-9))*boltzmann/m)**(2/2)*
            *omega_x*(2*np.pi)*omega_y*(2*np.pi)*omega_z)**(-1)*2*np.sqrt

def effectiveVolumeLattice(T, omega_x, omega_y, omega_z, f_axial):
    wavelength = (741)*10**(-9)
    a_perp_0 = wavelength/(2*np.pi)
    a_perp_z = np.sqrt((h/(2*np.pi))/(m*2*np.pi*f_axial))
    a_perp = 1/(1/a_perp_0+1/a_perp_z)
    #a_perp = a_perp_z
    return 1e6*8*(np.pi)**(3/2)*((T*10**(-9))*boltzmann/m)*((75*10**(-9))*bol
            *omega_x*(2*np.pi)*omega_y*(2*np.pi)*omega_z)**(-1)*np.sqrt(2
```

In [4]:
```
(8e3/effectiveVolumeODT(20, 90, 47, 94))/1e13
```

Out[4]: 1.7110708390025104

In [5]:
```
(((8e3))/effectiveVolumeLattice(210, 290, 290, 94, 500000))/1e13
```

Out[5]: 24.268930281533166

In [6]:
```
m = 162*1.66E-27
k_B = 1.38E-23
mag = 4.38
pixelSize = 6.5
sigma_0 = 4 # micro meters
um = 10**(-6)
ms = 10**(-3)
nK = 10**(-9)
```

In [7]:
```
#delta = detuning in GHz
#V = lattice setpoint in voltage units
#returns axial trap frequency in units of kHz

def trapFreqCalc(delta, V):
    return np.sqrt(-455.993 + 10364.2/(.0173162 + delta))*np.sqrt((60.643*V -
```

In [8]:
```
#omega_z, in kHz, with 2*pi factored out, nu
#returns magnetic field (G) where Zeeman energy matches first vibrational ene

def vibrationalLevelCalc(omega):
    return omega*10**3/(1.74*10**6)
```

In [9]:
```
def exp_dec(t, *p):
    return p[0] * np.exp(-t/p[1])
```

In [10]:
```
def objective_exp(t, *p):
    return p[0]/(1 + p[0]*p[1]*t)


def objective_exp(t, *p):
    return p[0]- p[0]**2*p[1]*t


#def objective_exp(t, *p):
#    return p[0]- p[1]*t
```

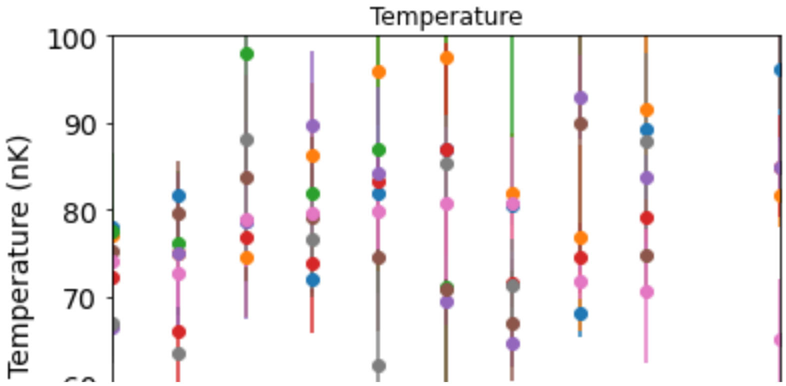
In [11]:
```
def cutDataFrame(df, cutoff, variable = 'BECShapeTime'):
    return df[df[variable] <= cutoff]

#cutoff = 1000
```

ODT

In [12]:
```
imageIDsets = [
              np.arange(214648, 214695 + 1),
              #np.arange(214427, 214474 + 1),
              np.arange(214596, 214643 + 1),
              np.arange(214476, 214523 + 1),
              np.arange(214532, 214594 + 1),
              np.arange(214697, 214744 + 1),
              np.arange(214750, 214797 + 1),
              np.arange(214800, 214862 + 1),
              np.arange(214866, 214907 + 1)
              ]
```

In [13]:
```
scanvars = []

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

cutoff = 10

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var, 'TOF'])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    dfa['temperature_nK'] = m/k_B*((dfa['xWidth']*pixelSize/mag)**2 - sigma_0
    dfa = db.statsFromDataFrame(dfa, varname, fitVariable = 'temperature_nK')

    procdfs.append(dfa)

TList = [procdfs[i]['mean'].iloc[0] for i in range(len(procdfs))]

fig, ax = plt.subplots()
for i, df_run in enumerate(procdfs):
    ax.errorbar(df_run['BECShapeTime'], df_run['mean'],df_run['error'], marke

ax.set_xlabel("Hold time (ms)", fontsize = 14)
ax.set_ylabel(r"Temperature (nK)", fontsize = 14)
ax.tick_params(labelsize = 14)
ax.set_ylim([50, 100])
ax.set_xlim([0, 10])
ax.set_title("Temperature")
```

Out[13]:
```
Text(0.5, 1.0, 'Temperature')
```

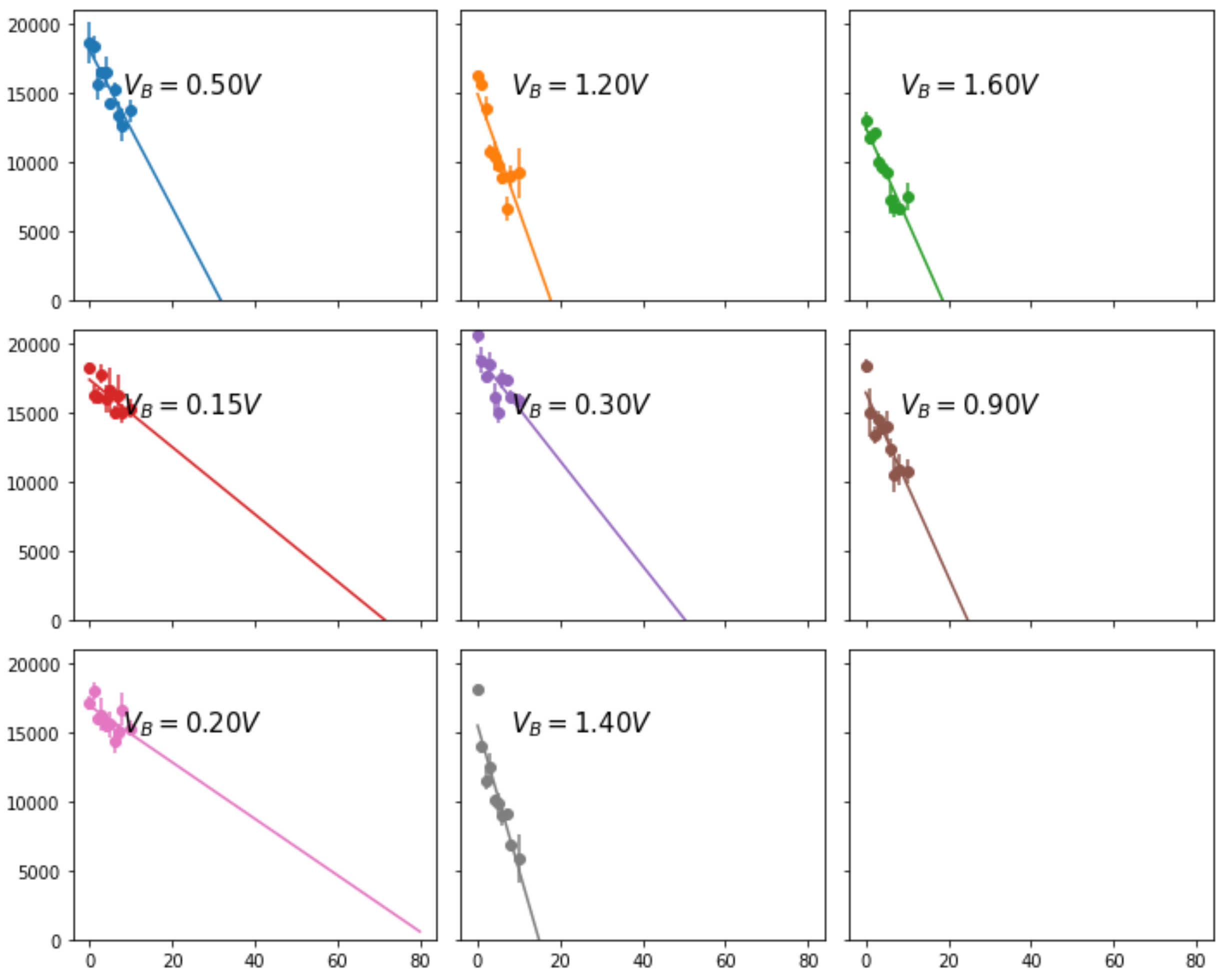


In [14]:

```
scanvars = []
#V = effectiveVolumeODT(20.9, 95, 47, 92.4)

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    df = db.statsFromDataFrame(dfa, varname)
    procdfs.append(df)

# # sort by value of second_var
# order = np.argsort(scanvars)
# procdfs = [procdfs[i] for i in order]
# scanvars = [scanvars[i] for i in order]
```

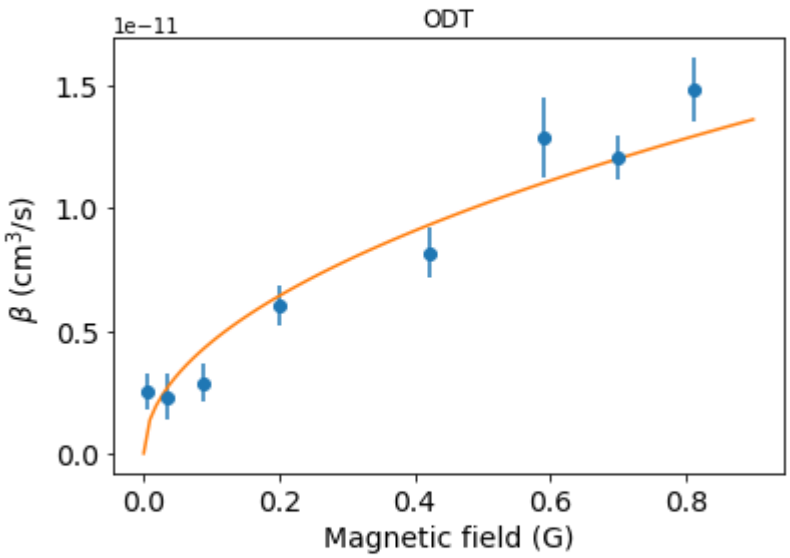
In [15]:

```
fig, ax = plt.subplots(3, 3, figsize = (10, 8), sharey = True, sharex = True)
ax = ax.flatten()

fit_a = []
fit_a_err = []

p_init = np.array([13E3, 0.005/1e3])
p_lower = np.array([5E3, 0.001/1e4])
p_upper = np.array([25E3, 0.2/1e3])

for i in range(len(scanvars)):
    if np.any(procdfs[i]["error"] == 0.0):
            popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdf
    else:
        #popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i
        popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i]
    fit_a.append(popt[1]*effectiveVolumeODT(TList[i], 95, 47, 92.4) *N_cal)
    fit_a_err.append(np.sqrt(np.diag(pcov)[1])*effectiveVolumeODT(TList[i], 9
    tdum = np.linspace(0, 80, 500)
    ax[i].plot(tdum, objective_exp(tdum, *popt), color = "C%i" % i)
    ax[i].errorbar(procdfs[i][varname], procdfs[i]["mean"], procdfs[i]["error
    t = ax[i].text(8, 15000, (r"$V_B = %.2f V$" % scanvars[i]), fontsize = 15

ax[0].set_ylim([0, 21e3])
# ax[-1].axis('off')
fig.tight_layout()
```



In [16]:
```
def sqrtfit(x, *p):
    return p[0]*np.sqrt(x)
```

In [17]:
```
Vtry = effectiveVolumeODT(TList[0], 95, 47, 92.4)
fig, ax = plt.subplots()
Bfield = 0.556 * (np.array(scanvars) - 0.14)
ax.errorbar(Bfield, np.array(fit_a)*1e3, np.array(fit_a_err)*1e3, marker = 'o

xdum = np.linspace(0, .9, 100)
popt, pcov = curve_fit(sqrtfit, Bfield, np.array(fit_a)*1e3, sigma = np.array
plt.plot(xdum, sqrtfit(xdum, *popt))

ax.set_xlabel("Magnetic field (G)", fontsize = 14)
ax.set_ylabel(r"$\beta$ (cm$^{3}$/s)", fontsize = 14)
ax.set_title("ODT")
ax.tick_params(labelsize = 14)
plt.savefig("dataOut/ODT.png", dpi = 300)
```



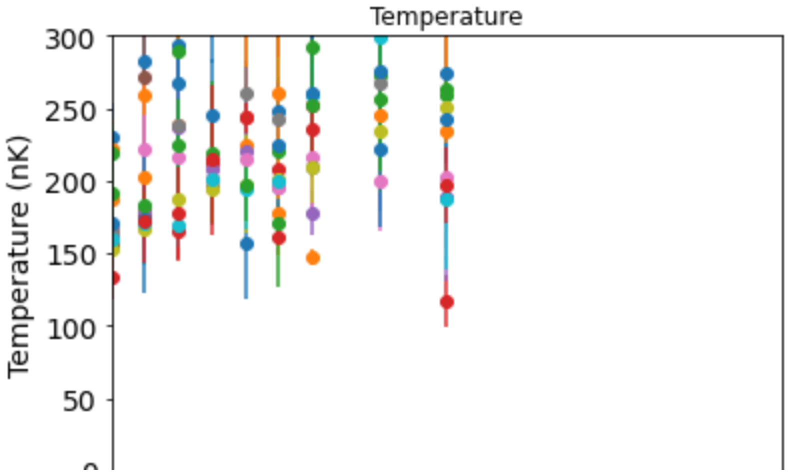
In [18]:
```
dfOut = pd.DataFrame({'Bfield' : Bfield, 'beta' : np.array(fit_a)*1e3, 'beta_
```

14.25 GHz, 1.9V (40.002nK, 130Hz)

In [19]:

```
imageIDsets = [np.arange(221206, 221262 + 1),
               np.arange(221264, 221320 + 1),
               np.arange(221431, 221493 + 1),
               np.arange(221539, 221601 + 1),
               np.arange(221649, 221711 + 1),
               np.arange(221713, 221775 + 1),
               np.arange(221779, 221841 + 1),
               np.arange(221849, 221911 + 1),
               np.arange(221923, 221985 + 1),
               np.arange(221987, 222049 + 1),
               np.arange(222124, 222186 + 1),
               np.arange(222275, 222337 + 1),
               np.arange(222423, 222485 + 1),
               np.arange(222359, 222421 + 1)
              ]

scanvars = []

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

cutoff = 5

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var, 'TOF'])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    dfa['temperature_nK'] = m/k_B*((dfa['xWidth']*pixelSize/mag)**2 - sigma_0
    dfa = db.statsFromDataFrame(dfa, varname, fitVariable = 'temperature_nK')

    procdfs.append(dfa)

TList = [procdfs[i]['mean'].iloc[0] for i in range(len(procdfs))]

fig, ax = plt.subplots()
for i, df_run in enumerate(procdfs):
    ax.errorbar(df_run['BECShapeTime'], df_run['mean'],df_run['error'], marke

ax.set_xlabel("Hold time (ms)", fontsize = 14)
ax.set_ylabel(r"Temperature (nK)", fontsize = 14)
ax.tick_params(labelsize = 14)
ax.set_ylim([0, 300])
ax.set_xlim([0, 10])
ax.set_title("Temperature")
```

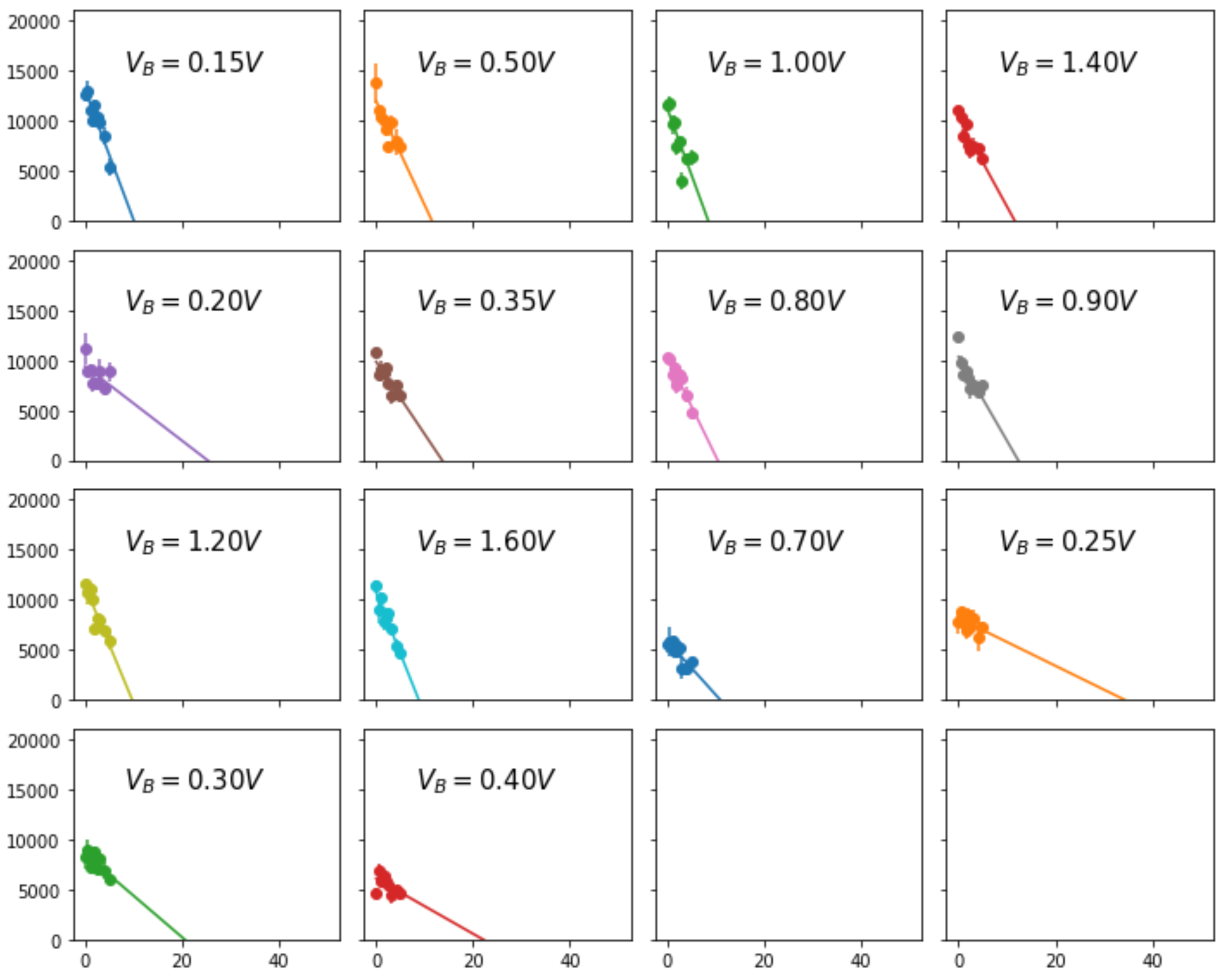
Out[19]: Text(0.5, 1.0, 'Temperature')



In [20]:

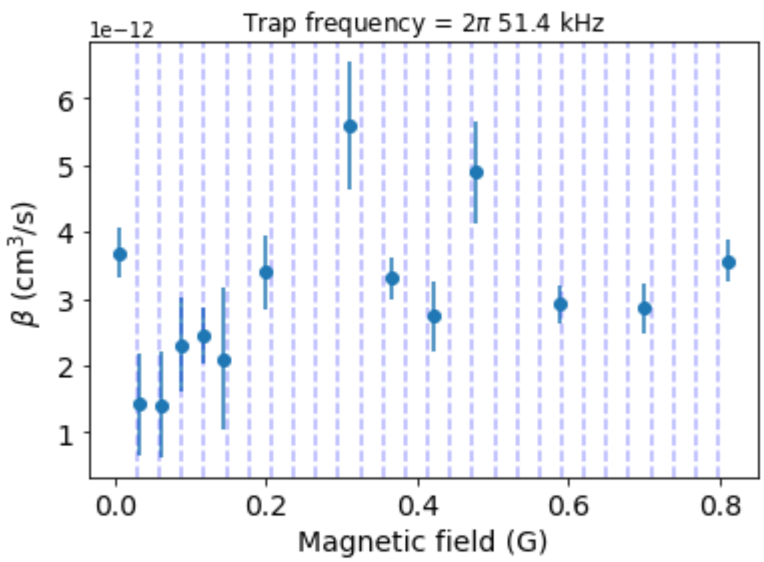
```
scanvars = []
#V = effectiveVolumeLattice(40.002, 130, 130, 92.4, 1000*trapFreqCalc(14.25,

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    df = db.statsFromDataFrame(dfa, varname)
    procdfs.append(df)

# # sort by value of second_var
# order = np.argsort(scanvars)
# procdfs = [procdfs[i] for i in order]
# scanvars = [scanvars[i] for i in order]
```

In [21]:
```
fig, ax = plt.subplots(4, 4, figsize = (10, 8), sharey = True, sharex = True)
ax = ax.flatten()

fit_a = []
fit_a_err = []

p_init = np.array([13E3, 0.005/1e3])
p_lower = np.array([5E3, 0.001/1e4])
p_upper = np.array([25E3, 0.2/1e3])

for i in range(len(scanvars)):
    if np.any(procdfs[i]["error"] == 0.0):
            popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdf
    else:
        # popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[
        popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i]
    fit_a.append(popt[1]*effectiveVolumeLattice(TList[i], 130, 130, 92.4, 100
    fit_a_err.append(np.sqrt(np.diag(pcov)[1])*effectiveVolumeLattice(TList[i
    tdum = np.linspace(0, 50, 500)
    ax[i].plot(tdum, objective_exp(tdum, *popt), color = "C%i" % i)
    ax[i].errorbar(procdfs[i][varname], procdfs[i]["mean"], procdfs[i]["error
    t = ax[i].text(8, 15000, (r"$V_B = %.2f V$" % scanvars[i]), fontsize = 15

ax[0].set_ylim([0, 21e3])
# ax[-1].axis('off')
fig.tight_layout()
```



In [22]:
```
def sqrtfit(x, *p):
    return p[0]*np.sqrt(x)
```

In [23]:
```
fig, ax = plt.subplots()
f = trapFreqCalc(14.25, 1.9)
B_1v = vibrationalLevelCalc(f)

Bfield = 0.556 * (np.array(scanvars) - 0.14)
ax.errorbar(Bfield, np.array(fit_a)*1e3, np.array(fit_a_err)*1e3, marker = 'o

[ax.axvline(x=n*B_1v, color='b', ls='--', alpha = 0.3) for n in np.arange(1,

ax.set_xlabel("Magnetic field (G)", fontsize = 14)
ax.set_ylabel(r"$\beta$ (cm$^{3}$/s)", fontsize = 14)
ax.tick_params(labelsize = 14)
ax.set_title("Trap frequency = $2\pi$ " + str(round(f, 1)) + " kHz")
plt.savefig("dataOut/Bscan1.png", dpi = 300)
```

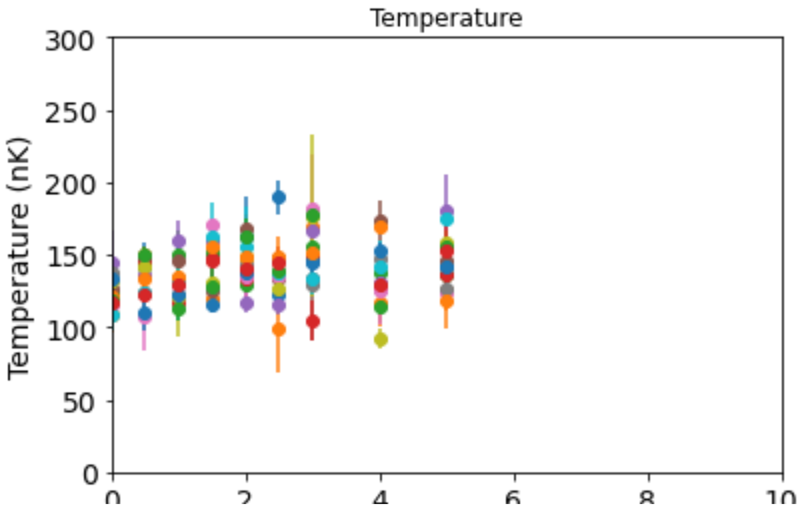


In [24]:
```
dfOut = dfOut.append(pd.DataFrame({'Bfield' : Bfield, 'beta' : np.array(fit_a
```

14.25 GHz, 0.1V (44.03nK, 150Hz)

In [25]:
```
imageIDsets = [#np.arange(222560, 222631 + 1),
               np.arange(223599, 223661 + 1),
               np.arange(222637, 222699 + 1),
               np.arange(222706, 222768 + 1),
               np.arange(222774, 222836 + 1),
               np.arange(222856, 222917 + 1),
               np.arange(222919, 222981 + 1),
               np.arange(222986, 223048 + 1),
               np.arange(223053, 223115 + 1),
               np.arange(223119, 223181 + 1),
               np.arange(223239, 223301 + 1),
               np.arange(223305, 223367 + 1),
               np.arange(223372, 223434 + 1),
               np.arange(223437, 223499 + 1),
               np.arange(223507, 223569 + 1)
              ]

scanvars = []

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

cutoff = 5

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var, 'TOF'])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    dfa['temperature_nK'] = m/k_B*((dfa['xWidth']*pixelSize/mag)**2 - sigma_0
    dfa = db.statsFromDataFrame(dfa, varname, fitVariable = 'temperature_nK')

    procdfs.append(dfa)

TList = [procdfs[i]['mean'].iloc[0] for i in range(len(procdfs))]

fig, ax = plt.subplots()
for i, df_run in enumerate(procdfs):
    ax.errorbar(df_run['BECShapeTime'], df_run['mean'],df_run['error'], marke

ax.set_xlabel("Hold time (ms)", fontsize = 14)
ax.set_ylabel(r"Temperature (nK)", fontsize = 14)
ax.tick_params(labelsize = 14)
ax.set_ylim([0, 300])
ax.set_xlim([0, 10])
ax.set_title("Temperature")
```

Out[25]:
```
Text(0.5, 1.0, 'Temperature')
```

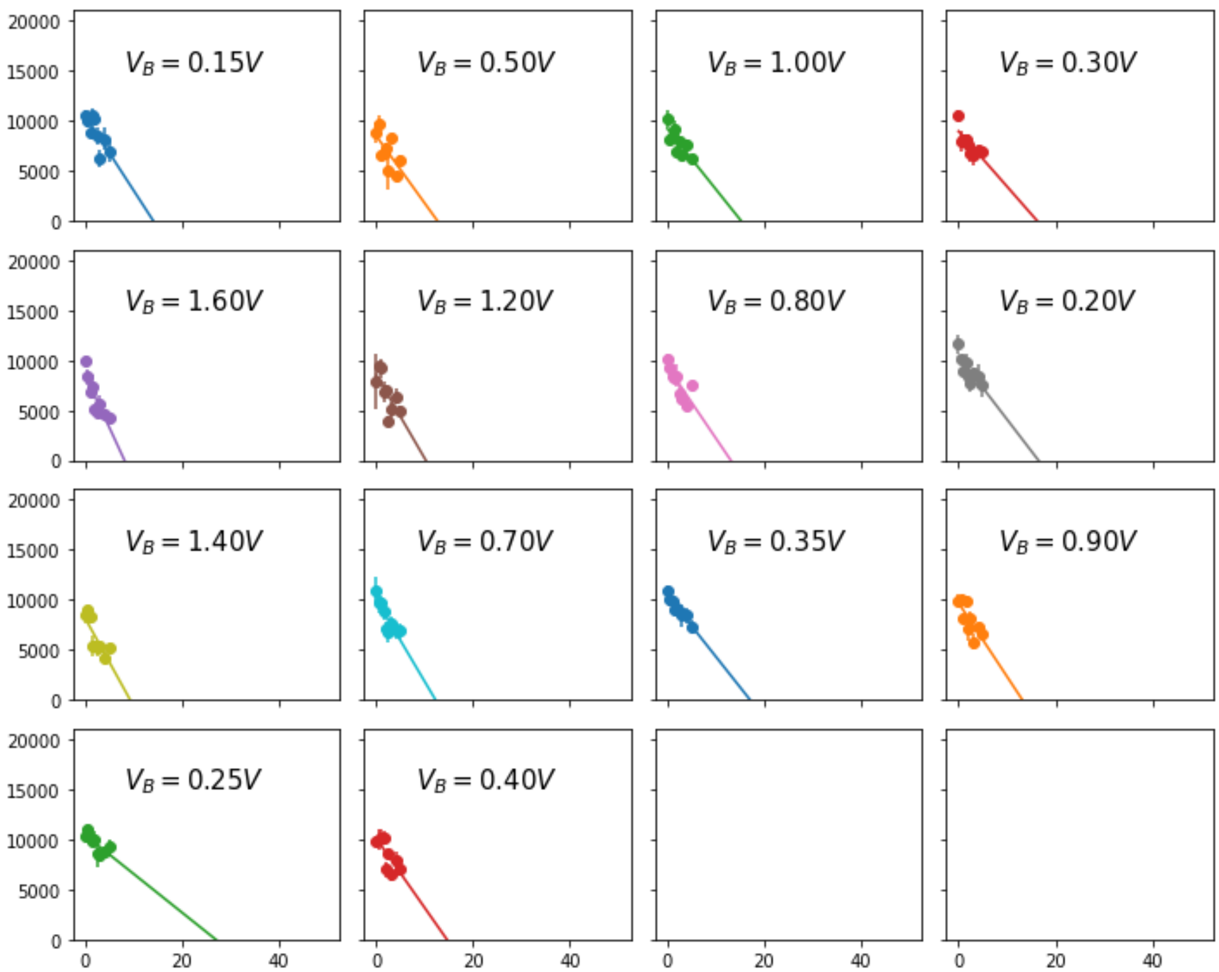


In [26]:

```
scanvars = []
#V = effectiveVolumeLattice(44.03, 150, 150, 92.4, 1000*trapFreqCalc(14.25, .

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    df = db.statsFromDataFrame(dfa, varname)
    procdfs.append(df)

# # sort by value of second_var
# order = np.argsort(scanvars)
# procdfs = [procdfs[i] for i in order]
# scanvars = [scanvars[i] for i in order]
```

In [27]:

```
fig, ax = plt.subplots(4, 4, figsize = (10, 8), sharey = True, sharex = True)
ax = ax.flatten()

fit_a = []
fit_a_err = []

p_init = np.array([13E3, 0.005/1e3])
p_lower = np.array([5E3, 0.001/1e4])
p_upper = np.array([25E3, 0.2/1e3])

for i in range(len(scanvars)):
    if np.any(procdfs[i]["error"] == 0.0):
            popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdf
    else:
        #popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i
        popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i]
    fit_a.append(popt[1]*effectiveVolumeLattice(TList[i], 150, 150, 92.4, 100
    fit_a_err.append(np.sqrt(np.diag(pcov)[1])*effectiveVolumeLattice(TList[i
    tdum = np.linspace(0, 50, 500)
    ax[i].plot(tdum, objective_exp(tdum, *popt), color = "C%i" % i)
    ax[i].errorbar(procdfs[i][varname], procdfs[i]["mean"], procdfs[i]["error
    t = ax[i].text(8, 15000, (r"$V_B = %.2f V$" % scanvars[i]), fontsize = 15

ax[0].set_ylim([0, 21e3])
# ax[-1].axis('off')
fig.tight_layout()
```

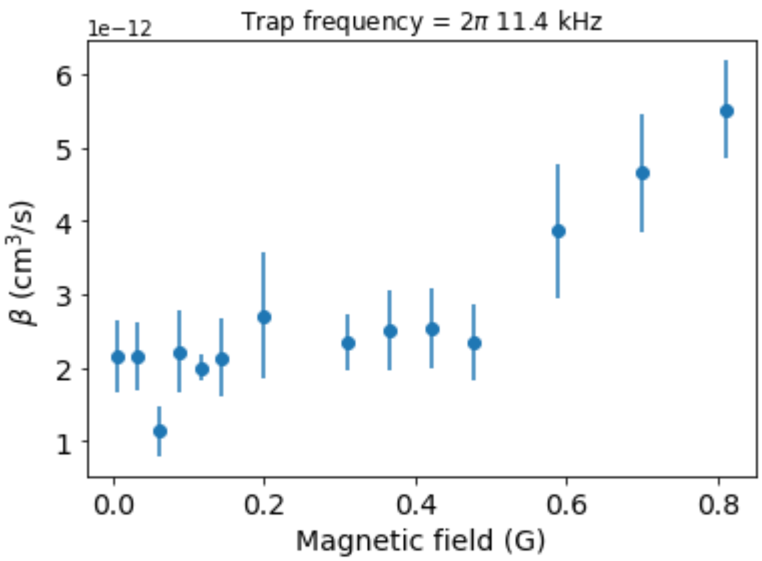


In [28]:

```
def sqrtfit(x, *p):
    return p[0]*np.sqrt(x)
```

In [29]:

```
fig, ax = plt.subplots()
Bfield = 0.556 * (np.array(scanvars) - 0.14)
ax.errorbar(Bfield, np.array(fit_a)*1e3, np.array(fit_a_err)*1e3, marker = 'o

# xdum = np.linspace(0, 1.5, 100)
# popt, pcov = curve_fit(sqrtfit, np.array(scanvars)-0.14, np.array(fit_a)*1e
# plt.plot(xdum+0.14, sqrtfit(xdum, *popt))
# ax.plot(xdum+0.14, 0.005*xdum)

f = trapFreqCalc(14.25, 0.1)
B_1v = vibrationalLevelCalc(f)
#plt.axvline(x=B_1v, color='b', ls='--')
[ax.axvline(x=n*B_1v, color='b', ls='--', alpha = 0) for n in np.arange(1, 12


ax.set_xlabel("Magnetic field (G)", fontsize = 14)
ax.set_ylabel(r"$\beta$ (cm$^{3}$/s)", fontsize = 14)
ax.tick_params(labelsize = 14)
ax.set_title("Trap frequency = $2\pi$ " + str(round(f, 1)) + " kHz")
plt.savefig("dataOut/Bscan2.png", dpi = 300)
```



In [30]:

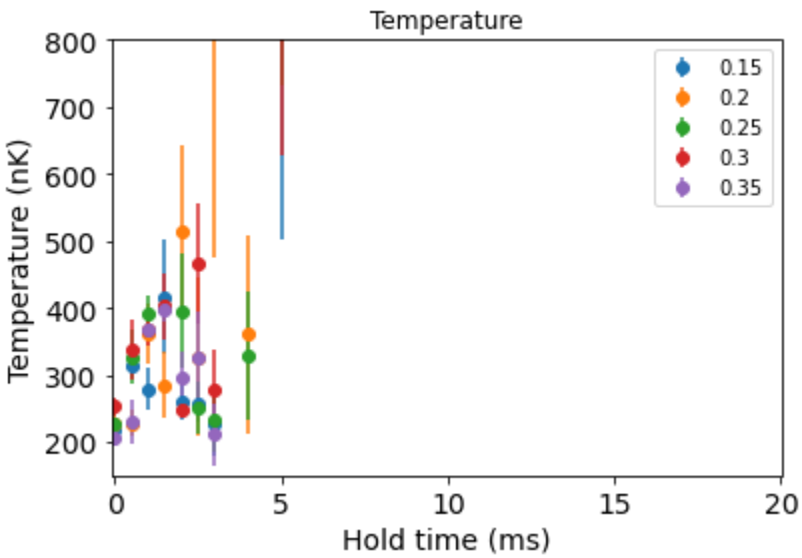
```
dfOut = dfOut.append(pd.DataFrame({'Bfield' : Bfield, 'beta' : np.array(fit_a
```

0.75 GHz, 1.0V (66.67nK, 150Hz)

In [31]:

```
imageIDsets = [#np.arange(223770, 223832 + 1)
               np.arange(223856, 223918 + 1),
               np.arange(223921, 223983 + 1),
               np.arange(223988, 224050 + 1),
               np.arange(224056, 224118 + 1),
               #np.arange(224121, 224183 + 1),
               np.arange(224186, 224248 + 1),
    np.arange(224251, 224313 + 1),
    np.arange(224316, 224378 + 1),
    np.arange(224381, 224443 + 1),
    np.arange(224447, 224509 + 1),
    np.arange(224524, 224586 + 1),
    np.arange(224588, 224644 + 1),  # excluded some data here
    np.arange(224653, 224715 + 1),
    np.arange(224716, 224778 + 1),
    np.arange(224779, 224841 + 1)
              ]

#cutoff = 8

scanvars = []

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

cutoff = 5

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var, 'TOF'])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    dfa['temperature_nK'] = m/k_B*((dfa['xWidth']*pixelSize/mag)**2 - sigma_0
    dfa = db.statsFromDataFrame(dfa, varname, fitVariable = 'temperature_nK')

    procdfs.append(dfa)

TList = [procdfs[i]['mean'].iloc[0] for i in range(len(procdfs))]

# sort by value of second_var
order = np.argsort(scanvars)
procdfs = [procdfs[i] for i in order]
scanvars = [scanvars[i] for i in order]
TList = [TList[i] for i in order]

fig, ax = plt.subplots()
for i, df_run in enumerate(procdfs[:5]):
    ax.errorbar(df_run['BECShapeTime'], df_run['mean'],df_run['error'], marke
ax.legend()

ax.set_xlabel("Hold time (ms)", fontsize = 14)
```

Out[31]: Text(0.5, 1.0, 'Temperature')

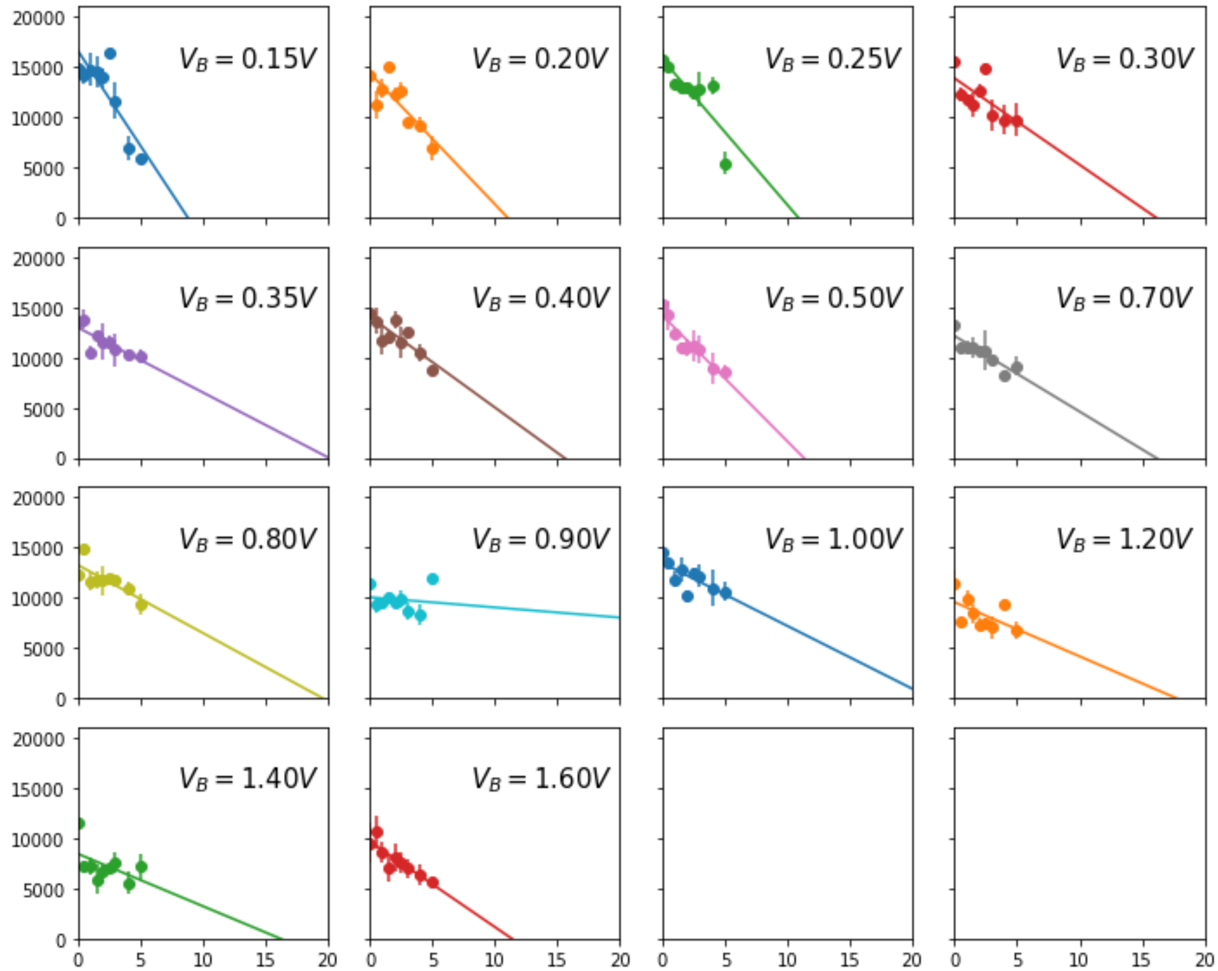


In [32]:

```
scanvars = []
#V = effectiveVolumeLattice(66.67, 150, 150, 92.4, 1000*trapFreqCalc(.75, 1))

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    df = db.statsFromDataFrame(dfa, varname)
    procdfs.append(df)

# sort by value of second_var
order = np.argsort(scanvars)
procdfs = [procdfs[i] for i in order]
scanvars = [scanvars[i] for i in order]

# sort by value of second_var
#order = np.argsort(scanvars)
#procdfs = [procdfs[i] for i in order]
#scanvars = [scanvars[i] for i in order]
```

In [33]:

```
fig, ax = plt.subplots(4, 4, figsize = (10, 8), sharey = True, sharex = True)
ax = ax.flatten()

fit_a = []
fit_a_err = []

p_init = np.array([13E3, 0.005/1e3])
p_lower = np.array([5E3, 0.0001/1e4])
p_upper = np.array([25E3, 0.4/1e3])

p_init = np.array([13E3, 0.005/1e3])
p_lower = np.array([1E3, 0.001/1e3])
p_upper = np.array([20E3, 0.1/1e3])

for i in range(len(scanvars)):
    if np.any(procdfs[i]["error"] == 0.0):
            popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdf
    else:
        #popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i
        popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i]
    fit_a.append(popt[1]*effectiveVolumeLattice(TList[i], 150, 150, 92.4, 100
    fit_a_err.append(np.sqrt(np.diag(pcov)[1])*effectiveVolumeLattice(TList[i
    tdum = np.linspace(0, 50, 500)
    ax[i].plot(tdum, objective_exp(tdum, *popt), color = "C%i" % i)
    ax[i].errorbar(procdfs[i][varname], procdfs[i]["mean"], procdfs[i]["error
    t = ax[i].text(8, 15000, (r"$V_B = %.2f V$" % scanvars[i]), fontsize = 15

ax[0].set_ylim([0, 21e3])
ax[0].set_xlim([0, 20])
# ax[-1].axis('off')
fig.tight_layout()
```

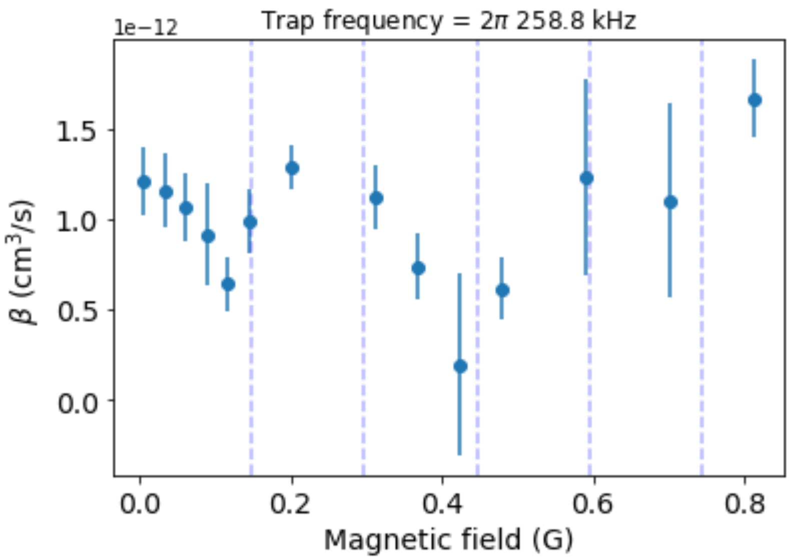


In [34]:

```
def sqrtfit(x, *p):
    return p[0]*np.sqrt(x)
```

In [35]:

```
fig, ax = plt.subplots()
Bfield = 0.556 * (np.array(scanvars) - 0.14)
ax.errorbar(Bfield, np.array(fit_a)*1e3, np.array(fit_a_err)*1e3, marker = 'o

# xdum = np.linspace(0, 1.5, 100)
# popt, pcov = curve_fit(sqrtfit, np.array(scanvars)-0.14, np.array(fit_a)*1e
# plt.plot(xdum+0.14, sqrtfit(xdum, *popt))
# ax.plot(xdum+0.14, 0.005*xdum)

f = trapFreqCalc(0.75, 1)
B_1v = vibrationalLevelCalc(f)
[ax.axvline(x=n*B_1v, color='b', ls='--', alpha = 0.3) for n in np.arange(1,

ax.set_xlabel("Magnetic field (G)", fontsize = 14)
ax.set_ylabel(r"$\beta$ (cm$^{3}$/s)", fontsize = 14)
ax.tick_params(labelsize = 14)
ax.set_title("Trap frequency = $2\pi$ " + str(round(f, 1)) + " kHz")
plt.savefig("dataOut/Bscan3.png", dpi = 300)
```



In [36]:
```
dfOut = dfOut.append(pd.DataFrame({'Bfield' : Bfield, 'beta' : np.array(fit_a
```

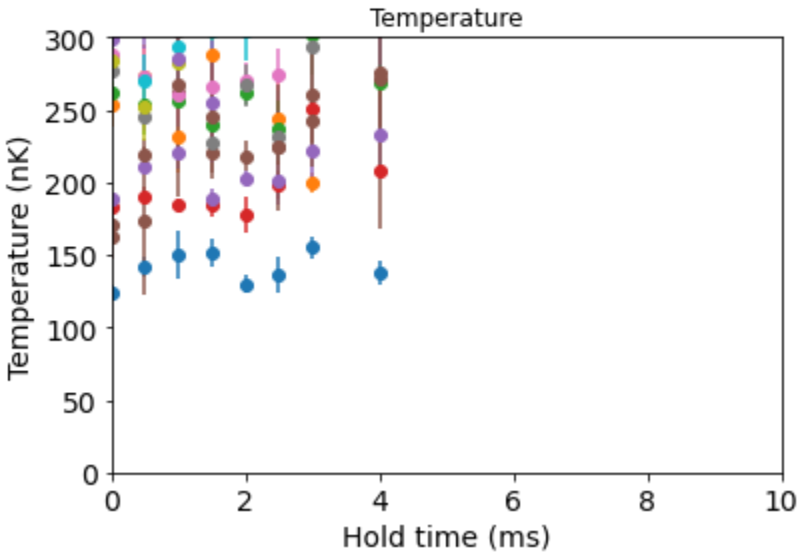
Scanning trap frequency at 1V Magnetic Field Setpoint

In [37]:

```
imageIDsets = [np.arange(222706, 222768 + 1),
               #np.arange(223921, 223983 + 1),
               #np.arange(230806, 230877 + 1),
               np.arange(230895, 230964 + 1),
               #np.arange(230978, 231046 + 1),
               np.arange(231349, 231417 + 1),
               #np.arange(231051, 231119 + 1),
               #np.arange(231131, 231199 + 1),
               #np.arange(231208, 231276 + 1),
               np.arange(231279, 231347 + 1),
               np.arange(232334, 232402 + 1),
               np.arange(232303, 232469 + 1),
               np.arange(232477, 232545 + 1),
               np.arange(232548, 232616 + 1),
               #np.arange(232653, 232721 + 1),
               np.arange(232724, 232790 + 1),
               np.arange(232857, 232925 + 1),
               #np.arange(233018, 233086 + 1),
               #np.arange(233101, 233169 + 1),
               np.arange(233170, 233238 + 1),
               np.arange(233253, 233321 + 1),
               np.arange(233341, 233412 + 1),
               np.arange(233414, 233482 + 1),
               np.arange(233554, 233622 + 1),
               np.arange(222124, 222186 + 1)
               #np.arange(236769, 236784 + 1)
              ]

trapFreqs = [trapFreqCalc(14.25, .1), #Compensation: 1V (150 Hz)
             #258.8,
             #200.1,
             trapFreqCalc(.75, .4), #Compensation: 1.6V (140 Hz)
             #114.2,
             trapFreqCalc(6.75, 1.5), #Compensation: 1.4V (145 Hz)
             #74.3,
             #52.4,
             trapFreqCalc(6.75, .3), #Compensation: 1V (130 Hz)
             trapFreqCalc(6.75, 1), #Compensation: 1V (115 Hz)
             trapFreqCalc(6.75, 0.5), #Compensation: 1V (125 Hz)
             trapFreqCalc(0.75, 0.2), #Compensation: 1.4V (145 Hz)
             trapFreqCalc(0.75, 0.4), #Compensation: 1.6V (140 Hz)
             #trapFreqCalc(0.75, 0.6),
             trapFreqCalc(0.75, 0.8), #Compensation: 1.8V (135 Hz)
             trapFreqCalc(0.75, 0.6), #Compensation: 1.8V (145 Hz)
             #trapFreqCalc(0.75, 0.5),
             trapFreqCalc(0.75, 1.5), #Compensation: 3V (185 Hz)
             trapFreqCalc(0.75, 2.0), #Compensation: 4V (230 Hz)
             trapFreqCalc(.75, 3.5), #Compensation: 6V (290 Hz)
             trapFreqCalc(.75, 2.5), #Compensation: 5V (265 Hz)
             trapFreqCalc(.75, 1), #Compensation: 2V (150 Hz)

             trapFreqCalc(14.25, 0.4), #Compensation: 1V (135 Hz)

            ]


oscLengths = [np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tra
              np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tra
```

```
                np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tra
                np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tra
                np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tra
                np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tra
                np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tra
                np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tra
                np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tra
                #np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * tr

              ]


#T_list = [34.08, 55.90, 52.48, 42.25, 42.14, 40.52, 54.95, 55.90, 59.26, 59.

T_list = [ 44.03055249,  57.82075724,  75.73953845,  46.37899156,
        62.89561661,  52.27345216,  75.91874759,  57.82075724,
        70.14875499,  76.24154524, 107.34449701, 130.41976528,
       202.53963013, 174.18055734,  66.67119409,  40.34705835]

omega_transverse = [150, 140, 145, 130, 115, 125, 145, 140, 135, 145, 185, 23

scanvars = []

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

cutoff = 4

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var, 'TOF'])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    dfa['temperature_nK'] = m/k_B*((dfa['xWidth']*pixelSize/mag)**2 - sigma_0
    dfa = db.statsFromDataFrame(dfa, varname, fitVariable = 'temperature_nK')

    procdfs.append(dfa)

TList = [procdfs[i]['mean'].iloc[0] for i in range(len(procdfs))]

fig, ax = plt.subplots()
for i, df_run in enumerate(procdfs):
    ax.errorbar(df_run['BECShapeTime'], df_run['mean'],df_run['error'], marke

ax.set_xlabel("Hold time (ms)", fontsize = 14)
```

Out[37]: Text(0.5, 1.0, 'Temperature')

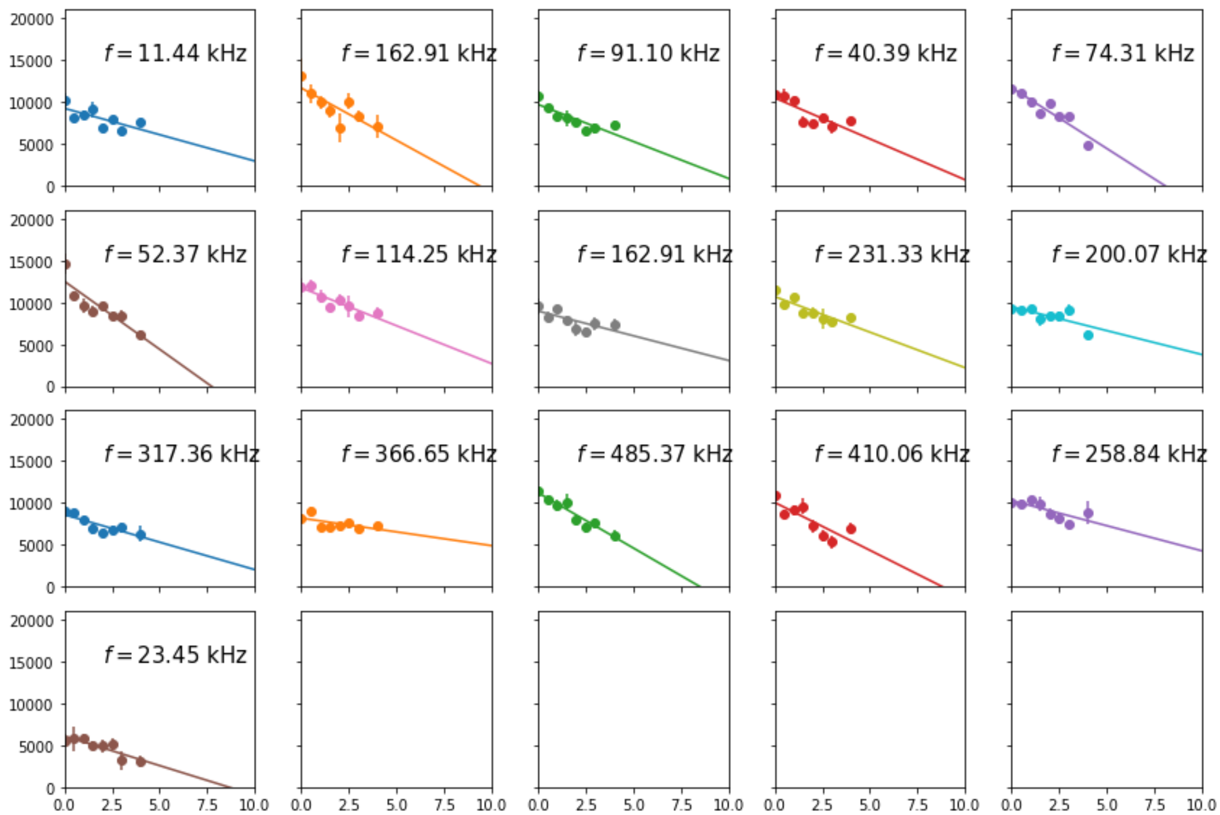


In [38]:

```
scanvars = []

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    df = db.statsFromDataFrame(dfa, varname)
    procdfs.append(df)
```

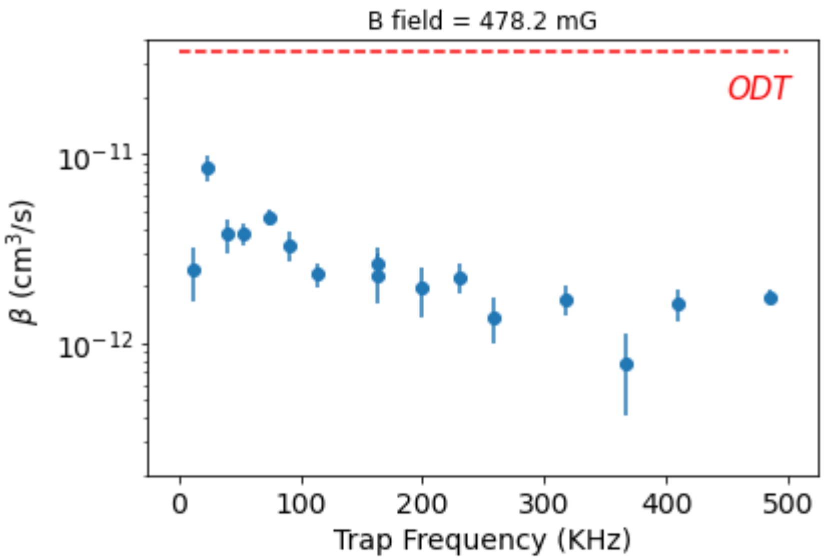
In [39]:

```
fig, ax = plt.subplots(4, 5, figsize = (12, 8), sharey = True, sharex = True)
ax = ax.flatten()

fit_a = []
fit_a_err = []

p_init = np.array([1e-8, 0.004/1e3, 13E3])
p_lower = np.array([1e-14, 0.001/1e5, 5E3])
p_upper = np.array([1e-7, 0.2/1e3, 25E3])

p_init = np.array([13E3, 0.005/1e3])
p_lower = np.array([5E3, 0.001/1e4])
p_upper = np.array([25E3, 0.2/1e3])

for i in range(len(scanvars)):
    if np.any(procdfs[i]["error"] == 0.0):
            popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdf
    else:
        # popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[
        popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i]
    fit_a.append(popt[1]*N_cal*effectiveVolumeLattice(TList[i], omega_transve
    fit_a_err.append(np.sqrt(np.diag(pcov)[1])*N_cal*effectiveVolumeLattice(T
    #fit_a.append(popt[1]*N_cal)
    #fit_a_err.append(np.sqrt(np.diag(pcov)[1])*N_cal)
    tdum = np.linspace(0, 80, 500)
    ax[i].plot(tdum, objective_exp(tdum, *popt), color = "C%i" % i)
    ax[i].errorbar(procdfs[i][varname], procdfs[i]["mean"], procdfs[i]["error
    t = ax[i].text(2, 15000, (r"$f = %.2f$ kHz" % trapFreqs[i]), fontsize = 1

ax[0].set_ylim([0, 21e3])
ax[0].set_xlim([0, 10])
# ax[-1].axis('off')
fig.tight_layout()
```



In [40]:

```
fig, ax = plt.subplots()

#ax.hlines(N_cal*3.1509741486501117e-12, 0, 500, color = 'r', linestyle = '--
ax.hlines(3.5e-11, 0, 500, color = 'r', linestyle = '--')
ax.text(450, 2e-11, ("ODT"), fontsize = 15, color = 'r', fontstyle = 'italic'

#ax.axhspan((3.1509741486501117e-12 - 0.5 * 5.961356082343591e-13)*N_cal, (3.

ax.errorbar(trapFreqs, np.array(fit_a)*1e3,np.array(fit_a_err)*1e3, marker =

ax.set_xlabel("Trap Frequency (KHz)", fontsize = 14)
ax.set_ylabel(r"$\beta$ (cm$^{3}$/s)", fontsize = 14)
ax.set_yscale('log')
ax.tick_params(labelsize = 14)
ax.set_ylim([2e-13, 4e-11])
ax.set_title("B field = " + str(round(1000*0.556 * (1 - 0.14), 1)) + " mG")
plt.savefig("dataOut/OmegaScan1.png", dpi = 300)
```

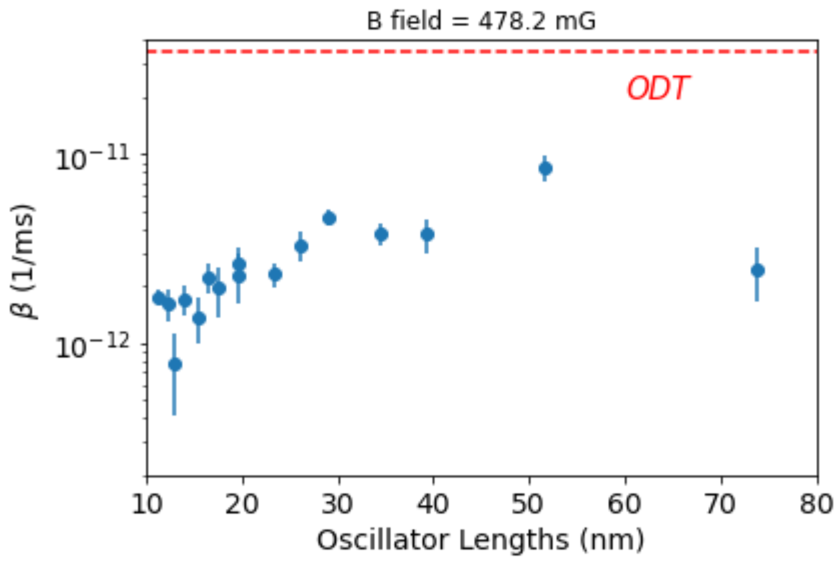


In [41]:

```
dfOut = dfOut.append(pd.DataFrame({'Bfield' : 0.556 * (1 - 0.14), 'beta' : np
```

In [42]:

```
fig, ax = plt.subplots()

#ax.hlines(N_cal*3.1509741486501117e-12, 0, 80, color = 'r', linestyle = '--'
ax.hlines(3.5e-11, 0, 80, color = 'r', linestyle = '--')
ax.text(60, 2e-11, ("ODT"), fontsize = 15, color = 'r', fontstyle = 'italic')

#ax.axhspan(N_cal*(3.1509741486501117e-12 - 0.5 * 5.961356082343591e-13), N_c

ax.errorbar(oscLengths, np.array(fit_a)*1e3,np.array(fit_a_err)*1e3, marker =

ax.set_xlabel("Oscillator Lengths (nm)", fontsize = 14)
ax.set_ylabel(r"$\beta$ (1/ms)", fontsize = 14)
ax.set_yscale('log')
ax.tick_params(labelsize = 14)
ax.set_xlim([10, 80])
ax.set_ylim([2e-13, 4e-11])
ax.set_title("B field = " + str(round(1000*0.556 * (1 - 0.14), 1)) + " mG")
plt.savefig("dataOut/OmegaScan1bis.png", dpi = 300)
```



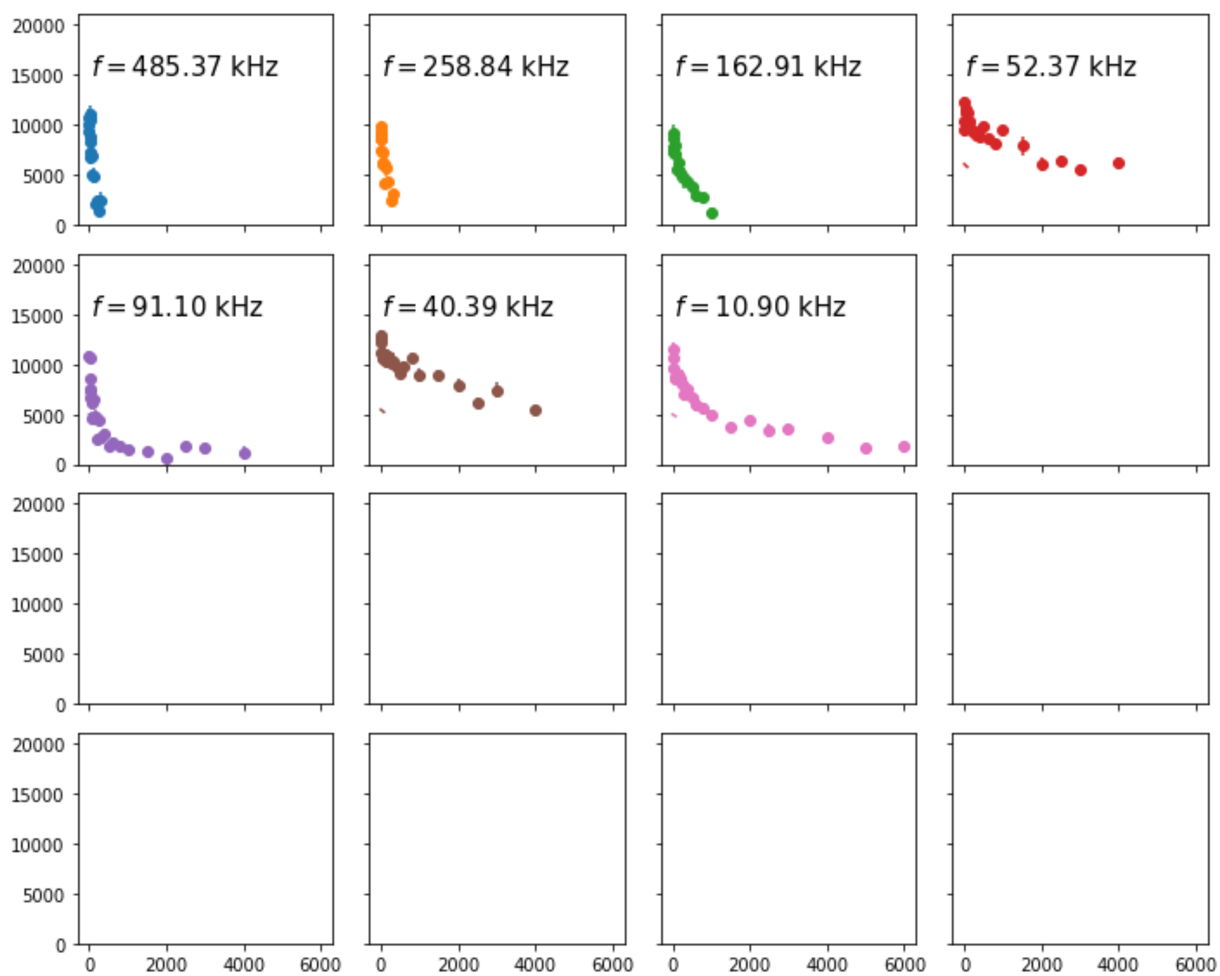
-8 Lifetimes

In [43]:

```
imageIDsets = [np.arange(235850, 235900 + 1),
               np.arange(235951, 236001 + 1),
               np.arange(236012, 236065 + 1),
               np.arange(236066, 236134 + 1),
               np.arange(236139, 236207 + 1),
               np.arange(236208, 236276 + 1),
               np.arange(236277, 236342 + 1)
              ]

trapFreqs = [trapFreqCalc(0.75, 3.5),
             trapFreqCalc(0.75, 1.0),
             trapFreqCalc(0.75, 0.4),
             trapFreqCalc(6.75, 0.5),
             trapFreqCalc(6.75, 1.5),
             trapFreqCalc(6.75, 0.3),
             trapFreqCalc(14.75, 0.1)
            ]


scanvars = []

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    df = db.statsFromDataFrame(dfa, varname)
    procdfs.append(df)
```

In [44]:

```
fig, ax = plt.subplots(4, 4, figsize = (10, 8), sharey = True, sharex = True)
ax = ax.flatten()

fit_a = []
fit_a_err = []

p_init = np.array([13E3, 0.005/1e3])
p_lower = np.array([5E3, 0.001/1e4])
p_upper = np.array([25E3, 0.2/1e3])

for i in range(len(scanvars)):
    if np.any(procdfs[i]["error"] == 0.0):
            popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdf
    else:
        popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i]
    fit_a.append(popt[1]*effectiveVolumeLattice(T_list[i], omega_transverse[i
    fit_a_err.append(np.sqrt(np.diag(pcov)[1])*effectiveVolumeLattice(T_list[
    tdum = np.linspace(0, 80, 500)
    ax[i].plot(tdum, objective_exp(tdum, *popt), color = "C%i" % i)
    ax[i].errorbar(procdfs[i][varname], procdfs[i]["mean"], procdfs[i]["error
    t = ax[i].text(8, 15000, (r"$f = %.2f$ kHz" % trapFreqs[i]), fontsize = 1

ax[0].set_ylim([0, 21e3])
# ax[-1].axis('off')
fig.tight_layout()
```



Scanning trap frequency at 1.6V Magnetic Field Setpoint

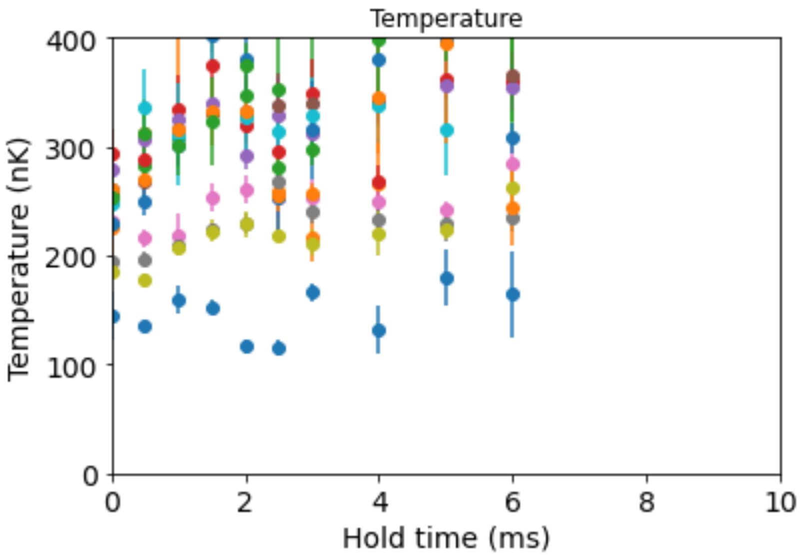
In [45]:
```
trapFreqCalc(0.75, 0.8)
```

Out[45]: 231.33097492336526

In [46]:
```
imageIDsets = [np.arange(222855, 222917 + 1),
               np.arange(223988, 224050 + 1),
               np.arange(231451, 231519 + 1),
               np.arange(231522, 231590 + 1),
               np.arange(231597, 231665 + 1),
               np.arange(231671, 231739 + 1),
               np.arange(231745, 231813 + 1),
               np.arange(231898, 231966 + 1),
               np.arange(231969, 232037 + 1),
               np.arange(232042, 232110 + 1),
               np.arange(232187, 232255 + 1),
               np.arange(232259, 232326 + 1),
               np.arange(232665, 232733 + 1)
               ]

trapFreqs = [trapFreqCalc(14.25, .1),
             trapFreqCalc(.75, 1),
            trapFreqCalc(0.75, 0.6),
            trapFreqCalc(0.75, 0.4),
            trapFreqCalc(0.75, 0.2),
            trapFreqCalc(6.75, 1.5),
            trapFreqCalc(6.75, 1.0),
            trapFreqCalc(6.75, 0.5),
            trapFreqCalc(6.75, 0.3),
            trapFreqCalc(0.75, 0.8),
            trapFreqCalc(0.75, 0.5),
            trapFreqCalc(.75, .4),
            trapFreqCalc(.75, 3.5)]
```

In [47]:
```
scanvars = []

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

cutoff = 6

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var, 'TOF'])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    dfa['temperature_nK'] = m/k_B*((dfa['xWidth']*pixelSize/mag)**2 - sigma_0
    dfa = db.statsFromDataFrame(dfa, varname, fitVariable = 'temperature_nK')

    procdfs.append(dfa)

TList = [procdfs[i]['mean'].iloc[0] for i in range(len(procdfs))]

fig, ax = plt.subplots()
for i, df_run in enumerate(procdfs):
    ax.errorbar(df_run['BECShapeTime'], df_run['mean'],df_run['error'], marke

ax.set_xlabel("Hold time (ms)", fontsize = 14)
ax.set_ylabel(r"Temperature (nK)", fontsize = 14)
ax.tick_params(labelsize = 14)
ax.set_ylim([0, 400])
ax.set_xlim([0, 10])
ax.set_title("Temperature")
```

Out[47]: Text(0.5, 1.0, 'Temperature')

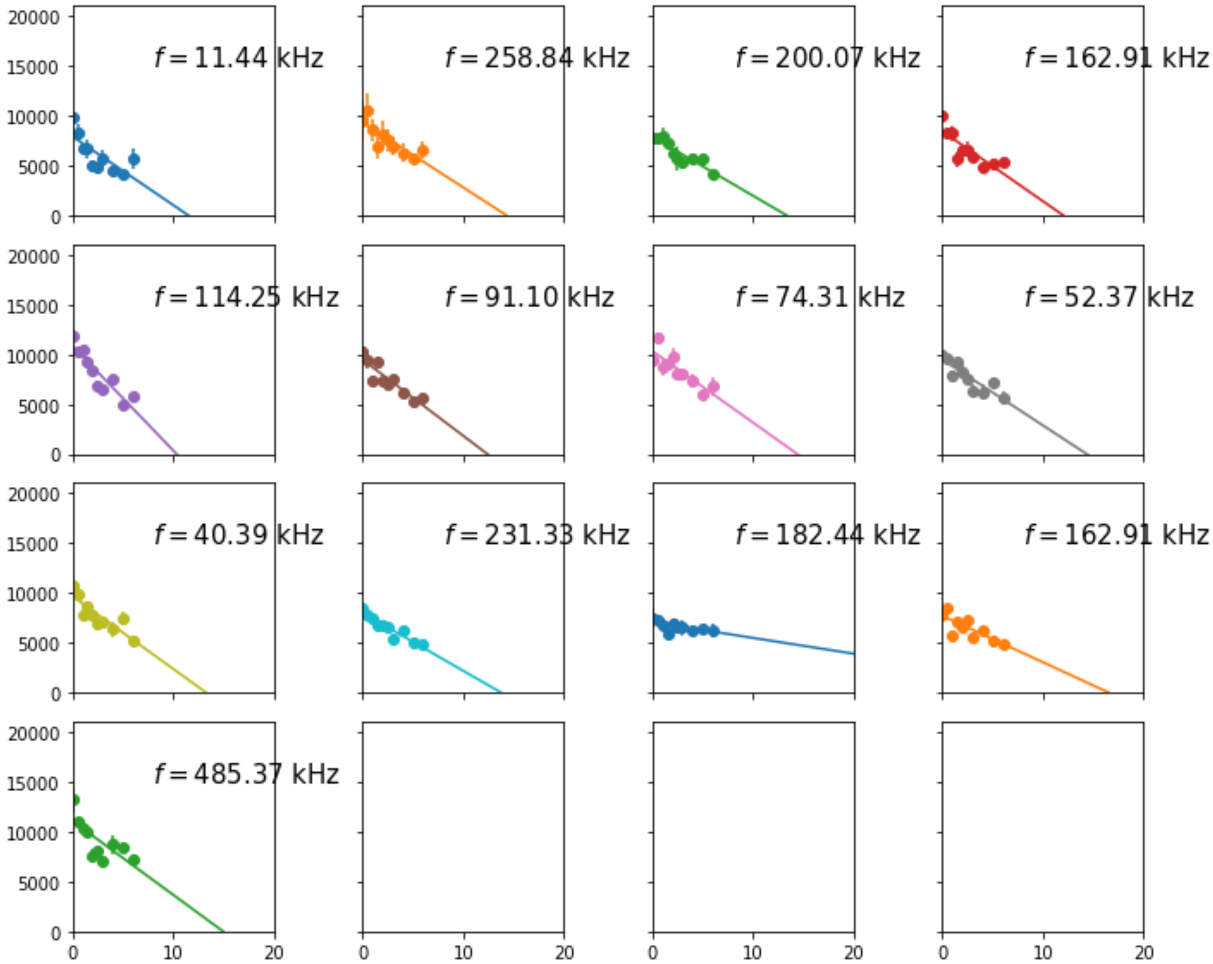


In [48]:

```
scanvars = []

varname = "BECShapeTime"
second_var = "drBField"
procdfs = []

for k, imageIDs in enumerate(imageIDsets):
    df = db.createDataFrameVec(imageIDs, [varname, second_var])
    scanvars.append(df[second_var][0])

    # Get rid of invalid data points
    todrop = np.arange(len(df))[df["nCount"] < 0]
    dfa = df.drop(index = todrop)

    # cut at low time
    dfa = cutDataFrame(dfa, cutoff = cutoff, variable = 'BECShapeTime')

    df = db.statsFromDataFrame(dfa, varname)
    procdfs.append(df)
```

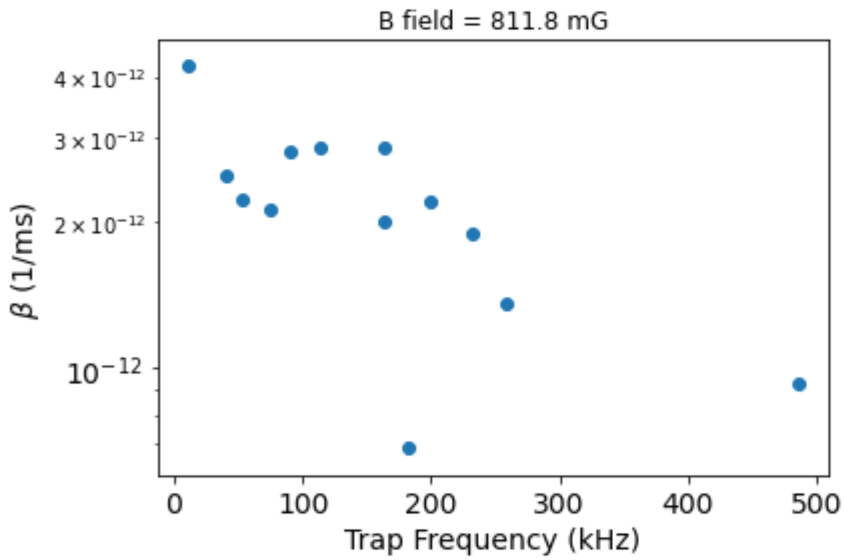
In [49]:

```
fig, ax = plt.subplots(4, 4, figsize = (10, 8), sharey = True, sharex = True)
ax = ax.flatten()

fit_a = []
fit_a_err = []

p_init = np.array([13E3, 0.005/1e3])
p_lower = np.array([5E3, 0.001/1e4])
p_upper = np.array([25E3, 0.2/1e3])

for i in range(len(scanvars)):
    if np.any(procdfs[i]["error"] == 0.0):
            popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdf
    else:
        #popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i
        popt, pcov = curve_fit(objective_exp, procdfs[i][varname], procdfs[i]
    fit_a.append(popt[1]*N_cal*effectiveVolumeLattice(TList[i], 150, 150, 92.
    fit_a_err.append(np.sqrt(np.diag(pcov)[1])*N_cal*effectiveVolumeLattice(2
    tdum = np.linspace(0, 80, 500)
    ax[i].plot(tdum, objective_exp(tdum, *popt), color = "C%i" % i)
    ax[i].errorbar(procdfs[i][varname], procdfs[i]["mean"], procdfs[i]["error
    t = ax[i].text(8, 15000, (r"$f = %.2f$ kHz" % trapFreqs[i]), fontsize = 1
ax[0].set_ylim([0, 21e3])
ax[0].set_xlim([0, 20])
# ax[-1].axis('off')
fig.tight_layout()
```



In [50]:

```
fig, ax = plt.subplots()
ax.errorbar(trapFreqs, np.array(fit_a)*1e3, np.array(fit_a_err)*1e3, marker =

ax.set_xlabel("Trap Frequency (kHz)", fontsize = 14)
ax.set_yscale('log')
ax.set_ylabel(r"$\beta$ (1/ms)", fontsize = 14)
ax.tick_params(labelsize = 14)
#ax.set_ylim([2e-13, 4e-12])
ax.set_title("B field = " + str(round(1000*0.556 * (1.6 - 0.14), 1)) + " mG")
plt.savefig("dataOut/OmegaScan2.png", dpi = 300)
```



In [51]:
```
dfOut = dfOut.append(pd.DataFrame({'Bfield' : 0.556 * (1.6 - 0.14), 'beta' :
```

In [52]:
```
dfOut.to_pickle('dataOut/dataRuns.pickle')
```

-8 Measurements

In [53]:
```
imageIDsets = [np.arange( + 1),
```

```
  File "<ipython-input-53-83ac62ab533c>", line 1
    imageIDsets = [np.arange( + 1),
                                   ^
SyntaxError: unexpected EOF while parsing
```

Pierre's sandbox 🏝️

In [ ]:
```
L
```

In [ ]:
```
L = np.array(dfa[dfa['BECShapeTime'] == 0]['xWidth'])
f = lambda x: np.std(x)

mean = np.mean(L)
std = 0
for l in L:
    std += (l-mean)**2

print(L.std())
print(np.std(L))
print(f(L))
print(np.sqrt(std/3))
print(np.sqrt(std/2))
```

In [ ]:
```
L
```

In [ ]:
```
def homeStd(x):
    mean = np.mean(x)
    return np.sqrt(np.sum((x - mean)**2)/(np.shape(x)[0]-1))

def numpyStd(x):
    return np.std(x)

def standardErrorOfMean(x):
    mean = np.mean(x)
    return np.sqrt(np.sum((x - mean)**2))/(np.shape(x)[0]-1)

def meanError2(x):
    return homeStd(x)/np.sqrt(np.shape(x)[0]-1)
```

In [ ]:
```
dfa.groupby(['BECShapeTime']).agg([np.std, 'std', numpyStd, homeStd, standard
```

In [ ]:
```
dfc = dfa[dfa['BECShapeTime'] == 0]
dfc.head()
```

In [ ]:
```
np.array(fit_a)[6]*1e3
```

In [ ]:
```
np.array(fit_a_err)[6]*1e3
```

In [ ]:
```
Bfield
```

In [ ]:
```
np.sqrt( h / (2 * 3.14159265359) / (m * 2 * 3.14159265359 * trapFreqCalc(14.2
```

In [ ]:
```
trapFreqCalc(14.25, .1)
```

In [ ]:
```
oscLengths
```

In [ ]:
```

```