In [1]:
```
# Import libraries
from pandas import DataFrame, concat, date_range, rolling_mean
from sqlalchemy import create_engine, MetaData, Table, select
import matplotlib.pyplot as plt
```

Grab Data from SQL

In this section we use the ***sqlalchemy*** library to grab data from ADEPT. The table we are concerned about is located in the *Marketing* database and called *IEICustomerCounts*. Cindea/Karen are the team members that maintain this table and BI is just mainly using the data in the table. This table is the source of the data behind the customer count numbers Brad in sales uses.

In [2]:
```
# Parameters
ServerName = "RepSer2"
Database = "Marketing"
TableName = "IEICustomerCounts"

# Create the connection
engine = create_engine('mssql+pyodbc://' + ServerName + '/' + Database)
conn = engine.connect()

# Required for querying tables
metadata = MetaData(conn)

# Table to query
tbl = Table(TableName, metadata, autoload=True, schema="dbo")
#tbl.create(checkfirst=True)

# Select all
sql = tbl.select()

# run sql code
result = conn.execute(sql)

# Insert to a dataframe
df = DataFrame(data=list(result), columns=result.keys())

# Convert data types
df.StatusDate = df.StatusDate.astype('datetime64[ns]')
df.CustomerCount = df.CustomerCount.astype('int')
print ' '
print 'Data Types'
print df.dtypes

# Set index to dataframe
df = df.set_index('StatusDate', drop=False)

# Close connection
conn.close()
```

```
Data Types
State                    object
```

```
StatusDate       datetime64[ns]
LDC                      object
Pool                     object
IsCGS                    object
IsUCB                    object
CustomerType             object
Status                   object
CustomerCount             int64
```

In [3]:
```
df.head()
```

Out[3]:

	State	StatusDate	LDC	Pool	IsCGS	IsUCB	CustomerType	Status	CustomerCount
StatusDate									
2011-05-16	GA	2011-05-16 00:00:00	AGLC	SAV	Non-CGS	N/A	Residential	Inactive	896
2011-05-16	GA	2011-05-16 00:00:00	AGLC	SNG	Non-CGS	N/A	Residential	Inactive	1694
2011-05-16	GA	2011-05-16 00:00:00	AGLC	TRA	Non-CGS	N/A	Residential	Inactive	938
2011-05-16	GA	2011-05-16 00:00:00	AGLC	VAL	Non-CGS	N/A	Residential	Inactive	94
2011-05-16	GA	2011-05-16 00:00:00	AGLC	None	CGS	N/A	Residential	Pending	177

Do some Data Munging

This section attempts to clean up the data for analysis.
1. Only select records where the account status is equal to "Active"
2. Make sure the state column is all in upper case
3. Merge (NJ and NY) to NY in the state column
4. Remove any ouliers (I found some zero and other odd results in the data set)

In [4]:
```
# Only grab where Status == Active
df = df[df['Status'] == 'Active']

# Clean State Column, convert to upper case
df.State = df.State.apply(lambda x: x.upper())

# Convert NJ to NY
df.State[df.State == 'NJ'] = 'NY'

# Group by State and StatusDate
Daily = df.groupby(['State','StatusDate']).sum()

# Calculate Outliers
StateYearMonth = Daily.groupby([Daily.index.get_level_values(0), Daily.index.get_level_values(1).year, Daily.index.get_level_values(1).month])
Daily['Outlier'] = StateYearMonth['CustomerCount'].transform( lambda x: abs(x-x.mean()) > 1.60*x.std() )

# Remove Ouliers
Daily = Daily[Daily['Outlier'] == False]
```

The dataframe named ***Daily*** will hold customer counts that have been aggregated per day. The original data (tbl IElCustomerCounts) has multiple records per day. We are left with a data set that is indexed by both the state and the StatusDate. The Oulier column should be equal to zero signifying that the record is not an outlier.

In [5]:
```
Daily.head()
```

Out[5]:

		CustomerCount	Outlier
State	**StatusDate**		
FL	**2009-09-05**	9948	0
	2009-09-12	9937	0
	2009-09-19	9919	0
	2009-09-26	9916	0
	2009-10-03	9924	0

We create a seperate dataframe named ***ALL*** which groups the Daily dataframe by StatusDate. We are essentially getting rid of the State column. The ***Max*** column represents the maximum customer count per month. The Max column is used to smooth out the graph.

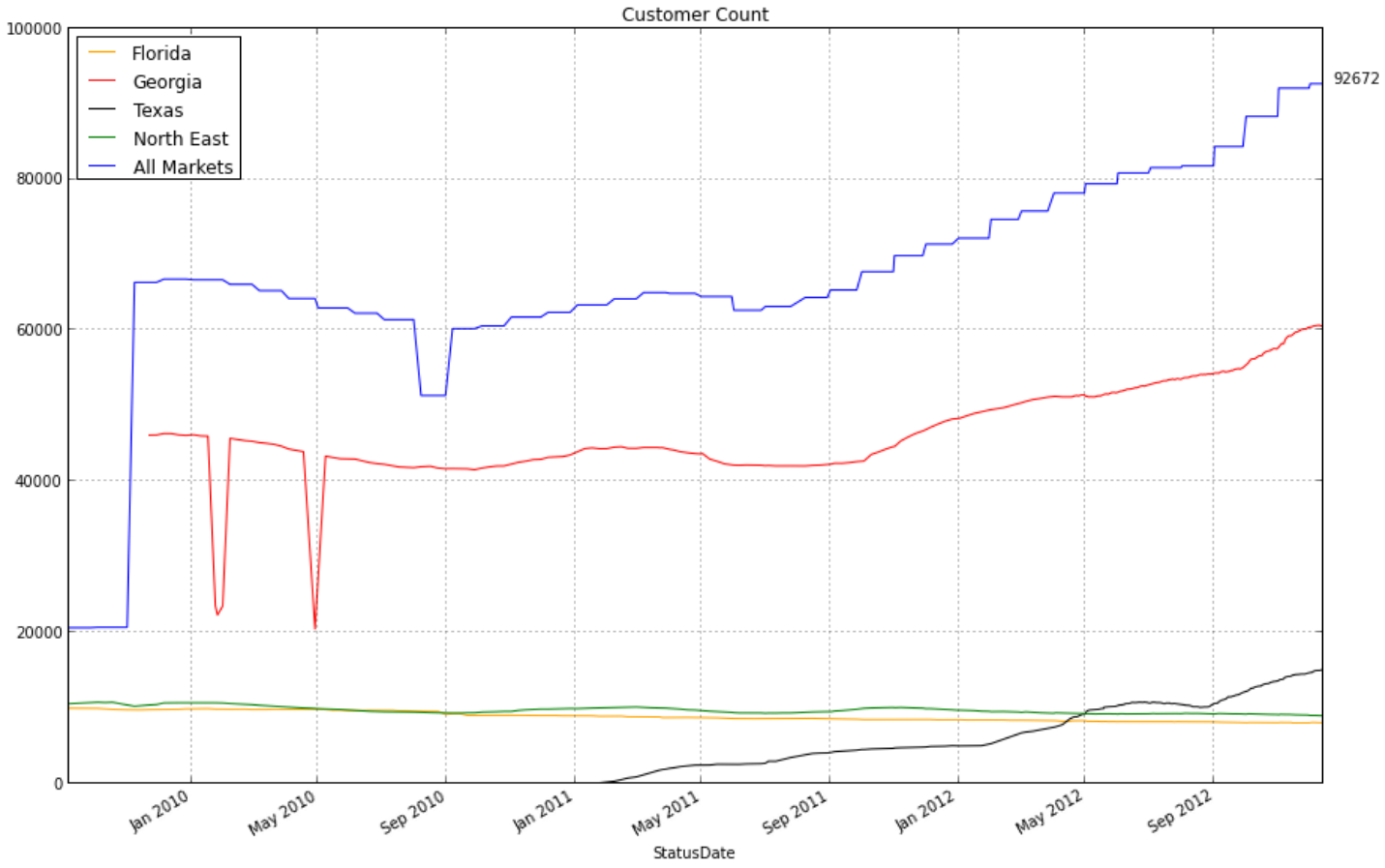
In [6]:
```
# Combine all markets

# Get the max customer count by Date
ALL = DataFrame(Daily['CustomerCount'].groupby(Daily.index.get_level_values(1)).sum())
ALL.columns = ['CustomerCount'] # rename column

# Group by Year and Month
YearMonth = ALL.groupby([lambda x: x.year, lambda x: x.month])

# What is the max customer count per Year and Month
ALL['Max'] = YearMonth['CustomerCount'].transform(lambda x: x.max())
ALL.head()
```

Out[6]:

	CustomerCount	Max
StatusDate		
2009-09-05	20501	20626
2009-09-12	20543	20626
2009-09-19	20560	20626
2009-09-26	20626	20626
2009-10-03	20671	20671

Graph Customer Counts per Market

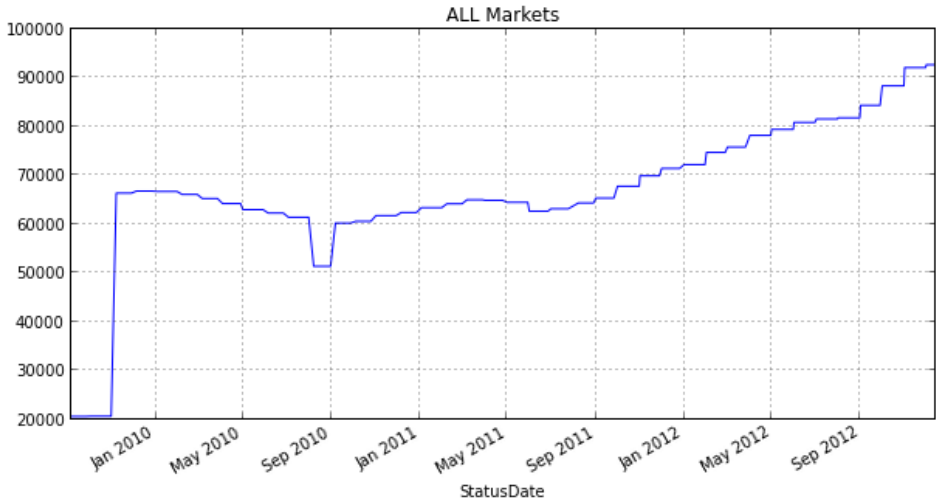
In [7]:
```
fig, axes = plt.subplots(figsize=(15, 10))
fig.subplots_adjust(hspace=1.0) ## Create space between plots

Daily.ix['FL']['CustomerCount']['2009':].fillna(method='pad').plot(color='orange', label='Florida')
Daily.ix['GA']['CustomerCount']['2009':].fillna(method='pad').plot(color='red', label='Georgia')
Daily.ix['TX']['CustomerCount']['2009':].fillna(method='pad').plot(color='black', label='Texas')
Daily.ix['NY']['CustomerCount']['2009':].fillna(method='pad').plot(color='green', label='North East')
ALL['Max']['2009':].plot(color='blue', label='All Markets')

plt.title('Customer Count')
plt.legend(loc='best')

# code to get the label on the right side of the graph
Graphs = [ALL['Max']['2009':]]
for var in (Graphs):
    plt.annotate('%i' % var.max(), xy=(1, var.max()), xytext=(8, 0),
                 xycoords=('axes fraction', 'data'), textcoords='offset points',rotation=0)
```

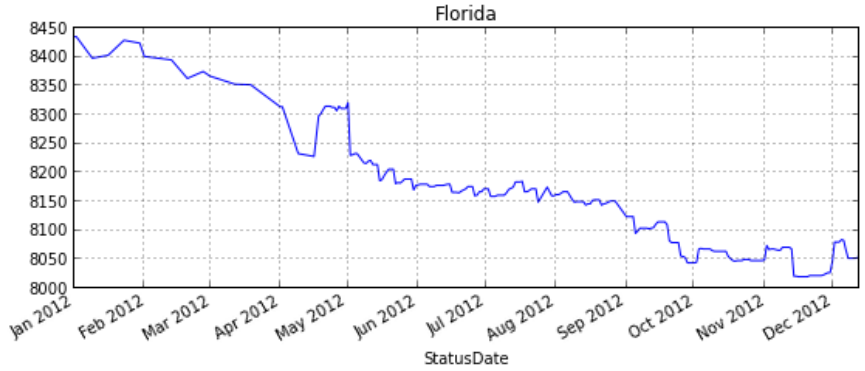


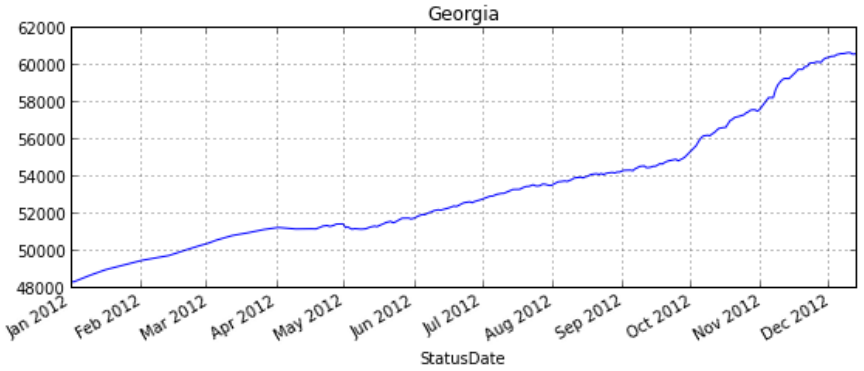
Graph Customer Counts, create sub plots per Market

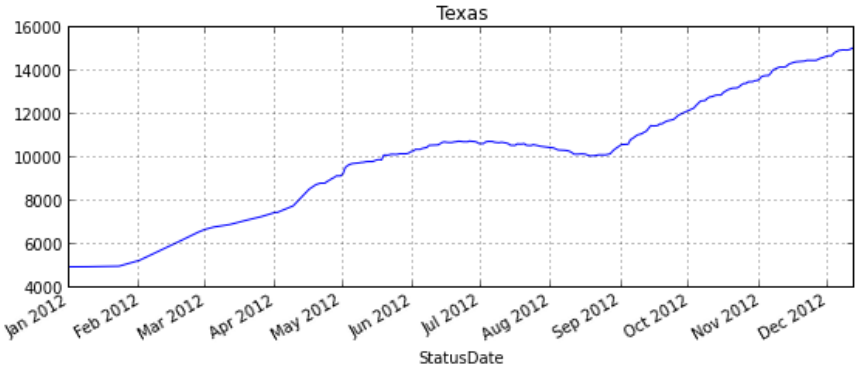
In [8]:
```
# First Graph
ALL['Max'].plot(figsize=(10, 5));plt.title('ALL Markets')

# Last four Graphs
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
fig.subplots_adjust(hspace=1.0) ## Create space between plots

Daily.ix['FL']['CustomerCount']['2012':].fillna(method='pad').plot(ax=axes[0,0]); axes[0,0].set_title('Florida')
Daily.ix['GA']['CustomerCount']['2012':].fillna(method='pad').plot(ax=axes[0,1]); axes[0,1].set_title('Georgia')
Daily.ix['TX']['CustomerCount']['2012':].fillna(method='pad').plot(ax=axes[1,0]); axes[1,0].set_title('Texas')
Daily.ix['NY']['CustomerCount']['2012':].fillna(method='pad').plot(ax=axes[1,1]); axes[1,1].set_title('North East')
```

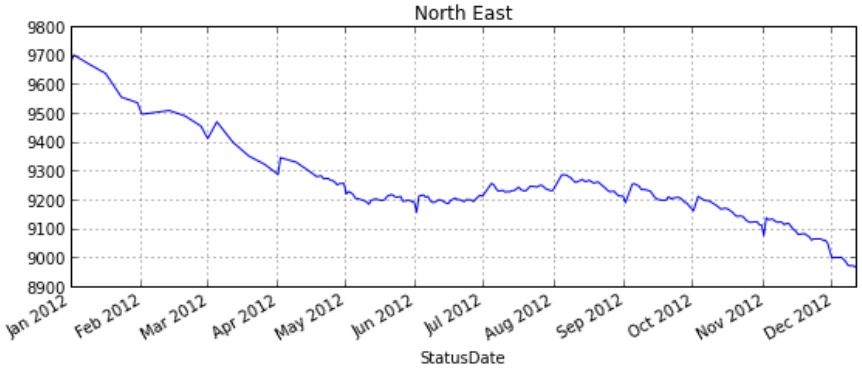
Out[8]: <matplotlib.text.Text at 0x92ecfd0>











Add the BHAG numbers to the "ALL" dataframe

12/31/2012 - 92,000 customers
12/31/2013 - 138,600 customers
12/31/2014 - 200,000 customers

In [9]:
```
# Create the BHAG dataframe
data = [92000,138600,200000]
idx = date_range(start='12/31/2012', end='12/31/2014', freq='A')
BHAG = DataFrame(data, index=idx, columns=['BHAG'])
BHAG
```

Out[9]:

	BHAG
2012-12-31	92000
2013-12-31	138600
2014-12-31	200000

In [10]:
```
# Combine the BHAG and ALL data sets
combined = concat([ALL,BHAG], axis=0)
combined.tail()
```

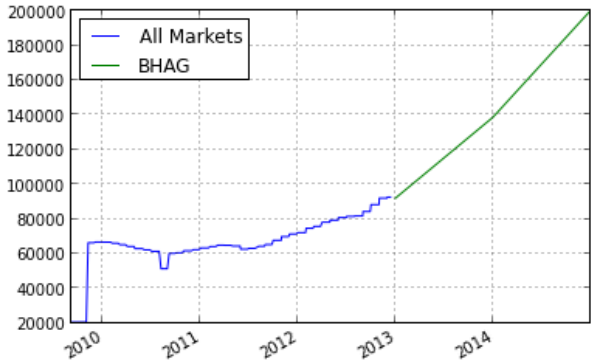
Out[10]:

	BHAG	CustomerCount	Max
2012-12-12	NaN	92672	92672
2012-12-13	NaN	83783	92672
2012-12-31	92000	NaN	NaN
2013-12-31	138600	NaN	NaN

2014-12-31	200000	NaN	NaN

In [11]:
```
combined['Max'].plot(color='blue', label='All Markets')
combined['BHAG'].plot(color='green', label='BHAG')
plt.legend(loc='best')
```

Out[11]: <matplotlib.legend.Legend at 0x938ac50>



Do some basic Forecasting

In [12]:
```
# Group by Year and then get the max value per year
Year = combined.groupby(lambda x: x.year).max()

# Add a column representing the percent change per year
Year['YR_PCT_Change'] = Year['Max'].pct_change(periods=1)
Year
```

Out[12]:

	BHAG	CustomerCount	Max	YR_PCT_Change
2009	NaN	66794	66794	NaN
2010	NaN	66705	66705	-0.001332
2011	NaN	71426	71426	0.070774
2012	92000	92672	92672	0.297455
2013	138600	NaN	NaN	NaN
2014	200000	NaN	NaN	NaN

To get next year's end customer count we will assume our current growth rate remains constant. We then will increase this years customer count by that amount and that will be our forecast for next year.

In [13]:
```
(1 + Year.ix[2012,'YR_PCT_Change']) * Year.ix[2012,'Max']
```

Out[13]:
```
120237.72273401843
```