MEMORANDUM

To: Commission File No. S7-08-10

From: Katherine Hsu
Senior Special Counsel
Office of Rulemaking
Division of Corporation Finance
U.S. Securities and Exchange Commission

Date: July 22, 2010

Re: Proposing Release on Asset-Backed Securities (Release Nos. 33-9117; 34-61858)

On July 15, 2010, Paula Dubberly, Katherine Hsu, and Rolaine Bancroft of the Division of Corporation Finance; Gregg Berman of the Division of Trading and Markets; and Adam Glass and Josh White of the Division of Risk, Strategy and Financial Innovation met with Michael Megliola, Lulu Young, and Lee B. Stephens of The Bank of New York Mellon Corporation. Among the topics discussed was the Commission's April 7, 2010 proposing release regarding asset-backed securities. A handout of a sample waterfall model written in Python is attached to this memorandum.

Attachment

```
# Copyright 2010 The Bank of New York Mellon Corporation

    #
    # 1 of 3: this function sets the notional amounts of
    #         non-principal-bearing bond classes.
    #
    def setNotionalAmounts(self, ndx, absNdx):
        AIO.setNotionalAmount(A1.getProjBalStatedInitial())

    #
    # 2 of 3: this function sets the pass-through rates
    #         for bond classes that have complex coupons.
    #
    def setNextPtr(self, ndx, absNdx):

        netWac = pool.getProjNetWAC()

        if absNdx <= 52:
            A1.setNextPtr( min ( netWac, 0.0375 ) )
            AIO.setNextPtr( max ( netWac - 0.0375, 0.0) )
        else:
            A1.setNextPtr( max ( netWac - 0.0050, 0.0 ) )
            AIO.setNextPtr( 0.0050 )

    #
    # 3 of 3: this function processes payments for each
    #         projected monthly payment period
    #
    def project(self, ndx, absNdx):

        global srPrepayPctPrior

        #
        # step-down test (a value of True indicates that the test is passed)
        #

        # step-down component 1: average delinquent balance

        if subCerts.getProjBalStatedInitial() > 0.0:
            delinqRatio = model.getAverageHistoricalDelinqBal() / subCerts.getProjBalStatedInitial()
            stepdown1 = ( delinqRatio < 0.50 )
        else:
            stepdown1 = False

        # step-down component 2: cumulative loss amount

        if absNdx < 84:
            stepdown2 = ( pool.getProjCumLossAmount() <= 0.30 * subCerts.getProjBalStatedInitial() )
        elif absNdx < 96:
            stepdown2 = ( pool.getProjCumLossAmount() <= 0.35 * subCerts.getProjBalStatedInitial() )
        elif absNdx < 108:
            stepdown2 = ( pool.getProjCumLossAmount() <= 0.40 * subCerts.getProjBalStatedInitial() )
        elif absNdx < 120:
            stepdown2 = ( pool.getProjCumLossAmount() <= 0.45 * subCerts.getProjBalStatedInitial() )
        else:
            stepdown2 = ( pool.getProjCumLossAmount() <= 0.50 * subCerts.getProjBalStatedInitial() )

        stepdownIsAllowed = ( stepdown1 & stepdown2 )


        #
        # two-times test (a value of True indicates that the test is passed)
        #

        srPctOrig = min ( A1.getCutoffBalance() / pool.getCutoffBalance() , 1.0 )
        subPctOrig = 1.0 - srPctOrig

        srPct = min ( A1.getProjBalStatedInitial() / pool.getProjBalStatedInitial() , 1.0 )
        subPct = 1.0 - srPct

        twoTimesTestIsPassed = ( subPct >= subPctOrig * 2.0 )
```

```
#
# senior prepayment percentage
#

#
# Language from PSA (page 23) is difficult to reconcile:
#
# Provisions begins...
#
# With respect to any Distribution Date occurring before the
# Distribution Date in May 2017, 100%.  Except as provided herein,
# the Senior Prepayment Percentage for any Distribution Date
# occurring in or after May 2017 shall be as follows:
#
# ...then later states:
#
# Notwithstanding the above, if the Two Times Test is satisfied,
# on any Distribution Date (i) before the Distribution Date in
# May 2013, the Senior Prepayment Percentage shall equal the
# Senior Percentage for such Distribution Date plus 50% of an
# amount equal to the 100% minus the Senior Percentage for such
# Distribution Date and (ii) on or after the Distribution Date
# in May 2013, the Senior Prepayment Percentage shall equal the
# Senior Percentage for such Distribution Date.
#
# (the limitation on the exception is not consistent with the
# scope of the "notwithstanding")
#

if twoTimesTestIsPassed:

    if absNdx < 36:
        srPrepayPct = srPct + 0.50 * ( 1.0 - srPct )
    else:
        srPrepayPct = srPct

elif (absNdx >= 84) & (srPct > srPctOrig):

    srPrepayPct = 1.0

elif stepdownIsAllowed:

    if absNdx < 84:
        srPrepayPct = 1.0
    elif absNdx < 96:
        srPrepayPct = srPct + 0.70 * subPct
    elif absNdx < 108:
        srPrepayPct = srPct + 0.60 * subPct
    elif absNdx < 120:
        srPrepayPct = srPct + 0.40 * subPct
    elif absNdx < 132:
        srPrepayPct = srPct + 0.20 * subPct
    else:
        srPrepayPct = srPct
else:
    srPrepayPct = srPrepayPctPrior

srPrepayPctPrior = srPrepayPct


#
# subordinate prepayment percentage
#

subPrepayPct = 1.0 - srPrepayPct
```

```
#
# senior & subordinate principal distribution amounts
#

srPda1 = srPct * pool.getAmountAvailable(Acct.PRINCIPAL_SCHEDULED)
subPda1 = subPct * pool.getAmountAvailable(Acct.PRINCIPAL_SCHEDULED)

srPda2 = srPrepayPct * pool.getAmountAvailable(Acct.PRINCIPAL_PREPAID)
subPda2 = subPrepayPct * pool.getAmountAvailable(Acct.PRINCIPAL_PREPAID)

srPda3 = 0.0
subPda3 = 0.0
for loan in pool.getItems():
    x = loan.pPrinRecdRecovery
    y = srPrepayPct * loan.pBalStatedInitial
    srShare = min ( x, y )
    srPda3 += srShare
    subPda3 += x - srShare


#
# this limitation to "Senior Prepayment Percentage" is enforced
# here...
#
# In addition, if on any Distribution Date the allocation to the
# Senior Certificates then entitled to distributions of principal
# of full and partial Principal Prepayments and other amounts
# in the percentage required above would reduce the aggregate of
# the Class Principal Amounts of those Certificates to below zero,
# the Senior Prepayment Percentage for such Distribution Date shall
# be limited to the percentage necessary to reduce that Class
# Principal Amount to zero.
#

srPda = srPda1 + srPda2 + srPda3

if (srPda > A1.getProjBalStatedInitial()):
    excess = srPda - A1.getProjBalStatedInitial()
    srPda -= excess
    subPda2 += excess

subPda = subPda1 + subPda2 + subPda3

minSubAmt = pool.getCutoffBalance() * 0.0075

if ( subCerts.getProjBalStatedInitial() < minSubAmt ):

    balB1 = 0.0
    balB2 = 0.0
    balB3 = 0.0
    balB4 = 0.0

elif ( B2.getProjBalStatedInitial() + B3.getProjBalStatedInitial() + B4.getProjBalStatedInitial() < minSubAmt ):

    balB1 = B1.getProjBalStatedInitial()
    balB2 = 0.0
    balB3 = 0.0
    balB4 = 0.0

elif ( B3.getProjBalStatedInitial() + B4.getProjBalStatedInitial() < minSubAmt ):

    balB1 = B1.getProjBalStatedInitial()
    balB2 = B2.getProjBalStatedInitial()
    balB3 = 0.0
    balB4 = 0.0

elif ( B4.getProjBalStatedInitial() < minSubAmt ):

    balB1 = B1.getProjBalStatedInitial()
    balB2 = B2.getProjBalStatedInitial()
    balB3 = B3.getProjBalStatedInitial()
    balB4 = 0.0
```

```
else:

    balB1 = B1.getProjBalStatedInitial()
    balB2 = B2.getProjBalStatedInitial()
    balB3 = B3.getProjBalStatedInitial()
    balB4 = B4.getProjBalStatedInitial()

subPdaBal = balB1 + balB2 + balB3 + balB4

if subPdaBal > 0.0:

    subPdaB1 = subPda * ( balB1 / subPdaBal )
    subPdaB2 = subPda * ( balB2 / subPdaBal )
    subPdaB3 = subPda * ( balB3 / subPdaBal )
    subPdaB4 = subPda * ( balB4 / subPdaBal )

else:

    subPdaB1 = 0.0
    subPdaB2 = 0.0
    subPdaB3 = 0.0
    subPdaB4 = 0.0

# as provided in "Subordinate Principal Distribution Amount", pay any
# remaining principal collections to senior bonds:
srPda += subPda - ( subPdaB1 + subPdaB2 + subPdaB3 + subPdaB4 )


#
# distributions (see Pooling Agreement, section 5.01)
#

pool.pay(Acct.ANY_AND_ALL, Acct.INTEREST, Limit.CURRENT_AND_DEFERRED, [ srCerts, AIO ])
srPda -= pool.pay(Acct.ANY_AND_ALL, Acct.PRINCIPAL, srPda, [ LTR, R, A1 ])
pool.pay(Acct.ANY_AND_ALL, Acct.INTEREST, Limit.CURRENT_AND_DEFERRED, [ B1 ])
pool.pay(Acct.ANY_AND_ALL, Acct.PRINCIPAL, subPdaB1, [ B1 ])
pool.pay(Acct.ANY_AND_ALL, Acct.INTEREST, Limit.CURRENT_AND_DEFERRED, [ B2 ])
pool.pay(Acct.ANY_AND_ALL, Acct.PRINCIPAL, subPdaB2, [ B2 ])
pool.pay(Acct.ANY_AND_ALL, Acct.INTEREST, Limit.CURRENT_AND_DEFERRED, [ B3 ])
pool.pay(Acct.ANY_AND_ALL, Acct.PRINCIPAL, subPdaB3, [ B3 ])
pool.pay(Acct.ANY_AND_ALL, Acct.INTEREST, Limit.CURRENT_AND_DEFERRED, [ B4 ])
pool.pay(Acct.ANY_AND_ALL, Acct.PRINCIPAL, subPdaB4, [ B4 ])

# as provided in "Subordinate Principal Distribution Amount", pay any
# remaining senior principal distribution amount to subs, in order:
pool.pay(Acct.ANY_AND_ALL, Acct.PRINCIPAL, srPda, [ B1, B2, B3, B4])

pool.pay(Acct.ANY_AND_ALL, Acct.OTHER, Limit.UNLIMITED, [ R ])
```