# Policy Statement

Modified Recommendations for Use of Palivizumab for Prevention of Respiratory Syncytial Virus Infections

Release Date: December 2009


Bibliographic citation: doi:10.1542/peds.2009-2345

Comparable Guideline: This statement updates and replaces the 2003 AAP statement and the 2006 Red Book and is consistent with the 2009 Red Book recommendations. Pediatrics 2009;124:000

# Deficiencies:
* not implemented: conditionals 1.2, 5.2, and 5.3; RSV season for Florida
* conditionals 8.2, 8.3, and 8.4 are combined into a single rule (should be split into separate rules).
* should look for additional evidence of hemodynamically significant heart disease
  (e.g. digoxin, diuretics, baseline pulse ox below some threshold)

# Assumptions:
* Only one patient object can be in working memory at a time

#created on: Oct 1, 2010

package edu.chop.cbmi.cds.RSV;

#list any import classes here.
import edu.chop.cbmi.cds.Patient;
import edu.chop.cbmi.cds.Patient.*;
import edu.chop.cbmi.cds.Schedule;
import edu.chop.cbmi.cds.RSVSeason;
import edu.chop.cbmi.cds.FuzzyDate;
import edu.chop.cbmi.cds.Localization;
import edu.chop.cbmi.cds.DocFact;
import edu.chop.cbmi.cds.DocFact.*;
import java.util.ArrayList;
import java.util.Arrays;
import org.joda.time.*
import org.joda.time.format.*;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

#declare any global variables here

# RSV fact dictionary

# Describe candidate reasons why a patient may or may not be eligible for RSV prophylaxis.
# Specify the number of doses recommended and the reason.
declare RSVEligibleCandidate
  startDate : DateTime           # start of season for this eligible candidate
  doses : Integer
  reason : String
  reasonBrief : String
end

# Describe allergies to RSV vaccine or its ingredients
declare RSVAllergy
  notedDate : DateTime           # date when allergy was first noted
end
# Describe the final reason why a patient is or is not eligible for RSV prophylaxis.  
# Specify the number of doses recommended and the reason.

decclare RSVEligibleFinal
    startDate : DateTime           # start of season for this final eligibility
    doses : Integer
    reason : String
    reasonBrief : String
end

# Describe the insurance approval status
# Specify the number of doses recommended and the reason.

decclare RSVApproval
    startDate : DateTime           # start of season for this approval
    date: DateTime                 # date of this insurance status
    doses : Integer                # doses approved
    status : String                # current status
end

# Identify patients with chronic lung disease meeting eligibility criteria for RSV prophylaxis.  
# Include ICD9 codes: 770.7

decclare RSVChronicLungDisease
    description : String
    diagnoses : String     # describe the diagnoses that give rise to this fact
    icd9 : String
end

# Identify patients with airway abnormalities or neuromuscular disease  
# meeting eligibility criteria for RSV prophylaxis.  
# Include ICD9 codes: NNN.NN

decclare RSVAirwayNeuromuscularDisease
    description : String
    diagnoses : String     # describe the diagnoses that give rise to this fact
    icd9 : String
    type : String
end

# Identify patients with stridor, which may indicate airway abnormalities  
# meeting eligibility criteria for RSV prophylaxis.  
# Include ICD9 codes: 786.1

decclare RSVStridor
    description : String
    diagnoses : String     # describe the diagnoses that give rise to this fact
    icd9 : String
end

# Identify patients with congenital heart disease meeting eligibility criteria for RSV prophylaxis.  
# Include ICD9 codes: NNN.NN

decclare RSVCongenitalHeartDisease
    description : String
    diagnoses : String     # describe the diagnoses that give rise to this fact
# Identify patients receiving cardiac medications
# Include diuretics, afterload reducing agents, inotropes
declare RSVCardiacMedications
  seasonStart : DateTime           # start of season that applies for this medication
  description : String             # list of qualifying medications that were recently filled
end

# Identify documented hypoxemia
declare RSVHypoxemia
  seasonStart : DateTime           # start of season that applies for this medication
  pulseOx : Double                # pulse ox recorded
end

# Identify patients with immune deficiencies who may be eligible for RSV prophylaxis.
# Include ICD9 codes: NNN.NN
declare RSVImmuneDeficiency
  description : String
  diagnoses : String     # describe the diagnoses that give rise to this fact
  icd9 : String
end

# Identify patients with cystic fibrosis (not a criteria for eligibility).
# Include ICD9 codes: NNN.NN
declare RSVCysticFibrosis
  description : String
  diagnoses : String     # describe the diagnoses that give rise to this fact
  icd9 : String
end

# Identify patients who may be at increased risk of exposure to RSV, and describe the reason
declare RSVExposureRisk
  reason : String
end

# Describe RSV immunization dose schedule including dates when given, dates recommended in the future, and the status of that does (e.g. given, late, now, or future).
declare RSVImmunization
  date : DateTime
  seasonStart : DateTime           # indicate start date of season for this immunization
  doseNum : Integer                # indicate which number dose this is in the season
  status : String                  # may be "first", "given", "late" (implies given, but later than ideal interval)
end

# Identify patients who had breakthrough RSV disease during this season, describe when the disease was first noted by either diagnosis code NNN.NN or lab test result
declare RSVBreakthrough
  date : DateTime
end
# Identify patients who are in the hospital
declare RSVHospitalized
date : DateTime
end

# Describe special directives that may affect RSV eligibility or dose schedule.
describe RSVImperative
directive : String
end

# Identify the last dose given
describe RSVLastDose
last : RSVImmunization     # last dose given on or before eval date
end

# describe best possible dose schedule for this season based on actual reality
describe RSVActualSchedule
seasonStart : DateTime       # indicate start date of season for this immunization
doseNum : Integer            # dose number for this dose
interval : Interval           # acceptable interval based on actual or ideal date of
previous dose
date : DateTime               # either actual or ideal date for vaccine
dateRange : String            # text description of acceptable interval
dateActual : String           # text description of actual date
status : String                # may be 'No appointment' 'None in range' 'Scheduled' or
'Given'
projectedWeightKG : Double    # projected weight on ideal date
projectedWeight : String       # formatted text description of weight estimate
order : String
end

#describe appointment dates when synagis can be given
describe RSVAppointment
date : DateTime               # date of appointment
status : String                # status of appointment (scheduled, no show, cancelled, ...)
end

#describe season offset for testing purposes (move season earlier by N months
describe RSVSeasonOffset
months : Integer               # number of months to offset season for testing (e.g. -7
for testing vs 0 for production)
end

# report final RSV recommendations to calling application. Describe the RSV season
# dates that apply for this patient, the final eligibility results, any special
# directives that may affect eligibility or schedule, and the recommended
# schedule (including both past doses this season, and future doses recommended)
# note that there may be multiple occurrences of this object during analysis for those
# patients eligible to receive synagis during 2 seasons
describe RSVRecommendation
seasons : ArrayList           # describe RSV season(s) that patient may be eligible to
receive doses
startThisSeason : DateTime  # start date for this season
schedule : ArrayList  # ideal immunization schedule
scheduleActual : ArrayList  # actual immunization schedule
scheduleActualThisSeason : ArrayList  # convenience field with sorted imm schedule for this season
givenDoses : ArrayList  # observed immunization doses
imperatives : ArrayList  # guideline imperatives for this season
nextDoseNum : Integer  # describe next dose number (if any) that should be given
summaryStatement : String  # clinician summary of current RSV immunization status
eligibleReasonThisSeason : String  # convenience field to report eligibility reason this season
eligibleReasonBriefThisSeason : String  # convenience field to report abbreviated eligibility reason this season
eligibleDosesThisSeason : Integer  # convenience field to report doses eligible this season
flag : Boolean  # indicate whether or not this row should be flagged for action
chartReviewQuery : String  # query string in format param1=value1&param2=value2&...
isOnList : Boolean  # indicate whether or not patient is on an RSV eligibility smart list
showRsvOnly : Boolean  # indicate if display should be restricted to RSV only

# Localization for testing purposes: start season 7 months earlier
rule "localize season for testing"
  ruleflow-group "localization"
  when
    $p : Patient()
    # determine if this is development environment
    exists (Identifier(type == Identifier.ENVIRONMENT_ID, id matches "DEV.*") from $p.getIdentifiers())
    then
      Localization fact = new Localization("RSV");
      fact.setDecisionVariable("RSVSeason-Test");
      #insert(fact);  # disable season shifting
  end

# if Testing, start season 7 months earlier
rule "season for testing"
  ruleflow-group "rsv-season"
  when
    exists Localization(guideline == "RSV", decisionVariable == "RSVSeason-Test")
    then
      RSVSeasonOffset fact = new RSVSeasonOffset();
      fact.setMonths(-6);
      insert(fact);
  end

# if Production, start season on time
rule "season for production"
  ruleflow-group "rsv-season"
  when
    not Localization(guideline == "RSV", decisionVariable == "RSVSeason-Test")
    then
RSVSeasonOffset fact = new RSVSeasonOffset();
fact.setMonths(0);
insert(fact);
end

# Decision Variable: Onset of RSV Season
# Value: If Location = Southeast Florida - July 1
#         If Location = North-central or southwest Florida - 9/15
#         Else 11/1
rule "RSV Season in US except Florida"
  ruleflow-group "rsv-season"
  when
    # find patients not in florida
    $p: Patient($birthDate: birthDate, state != "FL")
    # determine offset months from recommendation due to localization
    RSVSeasonOffset($months: months)
  then
    # "year" of RSV season defined by year when the season starts. Through the end of
    # season from the prior year is the season of interest.
    # iterate from birth date through age 2 to consider all possible seasons
    int startYear = ($birthDate.getMonthOfYear() <= 4 ? $birthDate.getYear() - 1 :
                    $birthDate.getYear());
    for(int i = 0; i < 3; i++) {
      DateTime startDate = new DateTime(startYear+i,11+$months,1,0,0,0,0);
      insert(new RSVSeason(new Interval(startDate, startDate.plusMonths(5)),
                            startDate.minusMonths(3)));
    }
  end

# Criteria 1. Infants with CLD (Page 4, Column 1, Paragraph 3) - Conditional
# Infants with CLD
# Infants with CLDz <24 mo (at start of season)
# who received medical therapy (O2, inhaled meds or diuretics)
# for CLD within 6 mo prior to start of season should receive up to up to 5 doses
rule "Eligible for 5 doses due to chronic lung disease"
  ruleflow-group "rsv-risk-eligibility"
  when
    # find patients with chronic lung disease as a risk factor
    $p: Patient()
    $cldz: RSVChronicLungDisease()
    # determine the start date for the relevant RSV season. be sure patient was born
    # before the season end
    RSVSeason($startSeason: startDate, $endSeason: endDate > ($p.getBirthDate()))
    # check to make sure age < 24 months at start of season
    # TODO: clarify, if child reaches 24 months during season is immunization stopped
    $p.ageMonthsAt($startSeason.minusDays(1))
    # check to see if at least one prescription related to chronic lung disease was active
    # within the 6 month period preceding the season
    # qualifying prescriptions: supplemental oxygen, bronchodilator, diuretic or
    # chronic corticosteroid therapy
exists (Prescription(endDate == null || endDate >= ($startSeason.minusMonths(6)),
    pharmClass matches "(?ism).*\b(?:diuretics?|corticosteroids?|oxygen|antiasthmatics?)\b.*" ||
    generic matches "(?ism).*\b(?:oxygen?)\b.*")
    from $p.getPrescriptions())

then
    # eligible for 5 doses
    RSVEligibleCandidate fact = new RSVEligibleCandidate();
    fact.setStartDate($startSeason);
    # calculate patient age in months at the end of the season to determine maximum
    doses possible
    fact.setDoses((int)Math.min(5, $p.ageMonthsAt($endSeason) + 1));
    fact.setReason("chronic lung disease on treatment");
    insert(fact);
end

# Criteria 2. Infants Gestational Age < 32 weeks (Page 4, Column 2, Paragraph 2)
# Infants <= 28 6/7 weeks gestational age and are 0-11 mo old at start of RSV season are
# considered at risk through entire RSV season and eligible for 5 doses
# ALSO...
# Infants >= 29 0/7 weeks and <= 31 6/7 weeks gestational age and are 0-5 mo old at start
# of RSV season are considered at risk through entire RSV season and eligible for 5 doses
rule "Eligible for 5 doses based on gestational age"
ruleflow-group "rsv-risk-eligibility"
when
    # find patients with gestational age < 32 weeks (<= 31 weeks 6 days)
    $p: Patient($gestAge: gestAge < 32)
    # determine the start date for the relevant RSV season. be sure patient was born
    before the season end
    $s: RSVSeason($startSeason: startDate, $endSeason: endDate > ($p.getBirthDate()))
    # calculate patient age in months at the start of the season
    $ageMonthsStart: Integer() from $p.ageMonthsAt($startSeason.minusDays(1))
    # test to see if age < 6 months at season start, or < 12 months for subgroup with
    gest age < 29 weeks (<= 28 weeks 6 days)
    eval($ageMonthsStart < 6 || $gestAge < 29 && $ageMonthsStart < 12)
then
    # eligible for 5 doses, provide a brief plain english explanation including
    patients gestational age, and age at season start
    RSVEligibleCandidate fact = new RSVEligibleCandidate();
    fact.setStartDate($startSeason);
    # calculate patient age in months at the end of the season to determine maximum
    doses possible
    fact.setDoses((int)Math.min(5, $p.ageMonthsAt($endSeason) + 1));
    fact.setReason("gestational age " + $gestAge.intValue() + " weeks");
    insert(fact);
end

# Criteria 3. Infants Gestational Age > 32 & < 35 weeks (Page 4, Column 2, Paragraph 3)
# Premature infants with a gestational age of 32 wk 0 d to 34 wk 6 d
# with at least 1 risk factor and born [no more than] 3 mo before or during RSV season
# are eligible for a maximum of 3 doses
rule "consider 3 doses based on risk and gestational age"
ruleflow-group "rsv-risk-eligibility"
when
# find patients with gestational age between 32 0/7 and 34 6/7 weeks
$\text{p: Patient(}$gestAge: gestAge < 35, gestAge >= 32$)$
# find the relevant RSV season that the child was born before or during (if any)
and determine the season end date
$\text{RSVSeason(}$startSeason: startDate, $\text{endSeason: endDate > ($p.getBirthDate())}$)$
# check to make sure patient born less than 3 months before the season start
$\text{ageMonthsStart: Integer(intValue < 3) from}$
$\text{p.ageMonthsAt(}$startSeason.minusDays(1)$)$
# calculate patient age in months at the end of the season
$\text{ageMonthsEnd: Integer(intValue >= 0) from}$
$\text{p.ageMonthsAt(}$endSeason$)$
# possible number of doses (up to 3)
$\text{doses: Integer() from Math.min(Math.min(3-}$ageMonthsStart, $\text{ageMonthsEnd +1), 3)$ then}
# see dependent rules
end

rule "eligible up to 3 doses" extends "consider 3 doses based on risk and gestational age"
ruleflow-group "rsv-risk-eligibility"
when
# check for presence of at least one RSV risk factor
$\text{riskList: ArrayList(size > 0) from collect (RSVExposureRisk())}$
then
# eligible for one dose per age in months while age < 3 months -- up to a maximum of 3 doses
$\text{RSVEligibleCandidate fact = new RSVEligibleCandidate();}$
$\text{fact.setStartDate(}$startSeason$);$  
$\text{fact.setDoses(}$doses$); # up to 3 doses
$\text{String riskReason = "";}$
$\text{for(RSVExposureRisk risk : (ArrayList<RSVExposureRisk>)$riskList$) riskReason =}$
  riskReason + ", " + risk.getReason();
$\text{fact.setReason("Gestational age " +}$gestAge.intValue() + " weeks" + riskReason);
insert(fact);
end

rule "not eligible up to 3 doses" extends "consider 3 doses based on risk and gestational age"
ruleflow-group "rsv-risk-eligibility"
when
# verify no RSV exposure risk documented
not RSVExposureRisk()
then
# eligible for one dose per age in months while age < 3 months -- up to a maximum of 3 doses
$\text{RSVEligibleCandidate fact = new RSVEligibleCandidate();}$
$\text{fact.setStartDate(}$startSeason$);$  
$\text{fact.setDoses(0);}$  
$\text{fact.setReason("Gestational age " +}$gestAge.intValue() + " weeks" + ", may be eligible for " + $doses + " doses if risk factors present");}$
insert(fact);
end

# Criteria 4. Infants with congenital abnormalities of the airway or neuromuscular
disease.  
# Immunoprophylaxis may be considered for infants (< 1 year) who have either significant congenital abnormalities of the airway or a neuromuscular condition that compromises handling of respiratory tract secretions. 

rule "Eligible for 5 doses due to abnormalities of the airway or neuromuscular disease" 
  ruleflow-group "rsv-risk-eligibility" 
  when 
    # find patients with congenital abnormalities of the airway or neuromuscular disease 
    $p: Patient()
    RSVAirwayNeuromuscularDisease($description: description, $diagnoses: diagnoses)
    # determine the start date for the relevant RSV season. be sure patient was born before the season end
    RSVSeason($startSeason: startDate, $endSeason: endDate > ($p.getBirthDate()))
    # check to make sure age < 12 months at start of season 
    # TODO: clarify, if child reaches 12 months during season is immunization stopped
    $ageMonthsStart: Integer(intValue < 12) from
    $p.ageMonthsAt($startSeason.minusDays(1))
    then 
      # eligible for 5 doses
      RSVEligibleCandidate fact = new RSVEligibleCandidate();
      fact.setStartDate($startSeason);
      # calculate patient age in months at the end of the season to determine maximum doses possible 
      fact.setDoses((int)Math.min(5, $p.ageMonthsAt($endSeason) + 1));
      fact.setReason($description + " (" + $diagnoses + ")");
      fact.setReasonBrief($description);
      insert(fact);
    end 

# Stridor may be an indication of airway abnormality, but is insufficient by itself to justify RSV 
rule "May be eligible due to stridor" 
  ruleflow-group "rsv-risk-eligibility" 
  when 
    # find patients with cystic fibrosis 
    $p: Patient()
    RSVStridor()
    # determine the start date for the relevant RSV season. be sure patient was born before the season end 
    RSVSeason($startSeason: startDate, $endSeason: endDate > ($p.getBirthDate()))
    then 
      # patients with cystic fibrosis are not eligible
      RSVEligibleCandidate fact = new RSVEligibleCandidate();
      fact.setStartDate($startSeason);
      fact.setDoses(0)
      fact.setReason("Patients with stridor may be eligible if critical airway issues exist");
      insert(fact);
    end 

# Criteria 5. Infants and children with CHD. 
# Infants and children with CHD: Children who are 24 months of age or younger with
# hemodynamically significant cyanotic or acyanotic CHD may benefit from palivizumab prophylaxis.

rule "Eligible for 5 doses due to congenital heart disease"
  ruleflow-group "rsv-risk-eligibility"
  when
    # find patients with congenital heart disease
    $p: Patient()
    $risk: RSVCongenitalHeartDisease($description: description, $diagnoses: diagnoses)
    # determine the start date for the relevant RSV season. be sure patient was born before the season end
    $s: RSVSeason($startSeason: startDate, $endSeason: endDate > ($p.getBirthDate()))
    # check to make sure age < 24 months at start of season
    # TODO: clarify, if child reaches 24 months during season is immunization stopped
    $ageMonthsStart: Integer(intValue < 24) from $p.ageMonthsAt($startSeason.minusDays(1))
  then
    # see dependent rules for qualifiers regarding eligibility
  end

rule "congenital heart disease on medications" extends "Eligible for 5 doses due to congenital heart disease"
  ruleflow-group "rsv-risk-eligibility"
  when
    # identify presence of recent cardiac medications for this season
    RSVCardiacMedications(seasonStart == $startSeason, $medications: description)
  then
    # eligible for 5 doses
    RSVEligibleCandidate fact = new RSVEligibleCandidate();
    fact.setStartDate($startSeason);
    # calculate patient age in months at the end of the season to determine maximum doses possible
    fact.setDoses((int)Math.min(5, $p.ageMonthsAt($endSeason) + 1));
    fact.setReason($description + " (" + $diagnoses + "); " + $medications);
    fact.setReasonBrief($description + " on medications");
    insert(fact);
  end

rule "congenital heart disease with hypoxemia" extends "Eligible for 5 doses due to congenital heart disease"
  ruleflow-group "rsv-risk-eligibility"
  when
    # verify presence of hypoxemia within 6 months before season
    exists RSVHypoxemia(seasonStart == $startSeason)
    Number($pulseOxMin: intValue) from accumulate (RSVHypoxemia($pulseOx: pulseOx, seasonStart == $startSeason), min($pulseOx))
  then
    # see dependent rules
  end

rule "congenital heart disease with hypoxemia on no medications" extends "congenital heart disease with hypoxemia"
  ruleflow-group "rsv-risk-eligibility"
  when
# Patients with unrepaired or partially intervened-upon cyanotic congenital cardiac defects
# should be considered eligible even in the absence of documented medications or hypoxemia.
# This includes but may not be limited to: tricuspid atresia, pulmonary atresia,
# tetralogy of Fallot, Ebstein’s anomaly, total anomalous pulmonary venous return/connection,
# truncus arteriosus, transposition of the great arteries, and HLHS.
# Also, any patient with congestive heart failure should be included
rule "cyanotic congenital cardiac defects or heart failure" extends "Eligible for 5 doses
due to congenital heart disease"
  ruleflow-group "rsv-risk-eligibility"
  when
    # verify not on cardiac medications
    not RSVCardiacMedications(seasonStart == $startSeason)
    # verify no documented hypoxemia
    not RSVHypoxemia(seasonStart == $startSeason)
    # enumerate some cardiac conditions for which eligibility is likely even in the
    # absence of cardiac medications or documented hypoxemia
    eval($description.matches("(?ism).*\b(?:atresia|tetralogy|fallot|tof|ebstein.*anomaly|total.*anom.*pulm.*ven\w*\{|rc\}|truncus\transposition|tgalhypoplas\w*|hlhs|hrhs|double.*outlet|dorv|common.*cavc|congestive.*failure|chf)\b.*"))
  then
    # eligible for 5 doses
    RSVEligibleCandidate fact = new RSVEligibleCandidate();
    fact.setStartDate($startSeason);
    # calculate patient age in months at the end of the season to determine maximum
doses possible
    fact.setDoses((int)Math.min($p.ageMonthsAt($endSeason) + 1));
    fact.setReason($description + " (" + $diagnoses + "); hypoxemia (" + $pulseOxMin + 
"%)");
    fact.setReasonBrief($description + " and hypoxemia");
    insert(fact);
  end

# Criteria 6. Immunocompromised children.
# Palivizumab prophylaxis has not been evaluated in randomized trials in
# immunocompromised children. Although specific recommendations for immunocompromised
# children cannot be made, infants and young children with severe immunodeficiency
# (eg, severe combined immunodeficiency or advanced AIDS) may benefit from prophylaxis
# NOTE: we interpret "infants and children" to be children less than 24 months old at
# the start of the season, and we presume 5 doses of eligibility
rule "Eligible for 5 doses due to immune deficiency"
  ruleflow-group "rsv-risk-eligibility"
  when
    # find patients with immune deficiency
    $p: Patient()
    $risk: RSVImmuneDeficiency($description: description, $diagnoses: diagnoses)
    # determine the start date for the relevant RSV season. be sure patient was born
    before the season end
    RSVSeason($startSeason: startDate, $endSeason: endDate > ($p.getBirthDate()))
    # check to make sure age < 24 months at start of season
    # TODO: clarify, if child reaches 24 months during season is immunization stopped
    $ageMonthsStart: Integer(intValue < 24) from
    $p.ageMonthsAt($startSeason.minusDays(1))
    then
      # eligible for 5 doses
      RSVEligibleCandidate fact = new RSVEligibleCandidate();
      fact.setStartDate($startSeason);
      # calculate patient age in months at the end of the season to determine maximum
      doses possible
      fact.setDoses((int)Math.min(5, $p.ageMonthsAt($endSeason) + 1));
      fact.setReason("Patients with cystic fibrosis may not be eligible for Palivizumab");
      insert(fact);
    end

# Criteria 7. Patients with cystic fibrosis
# A recommendation for routine prophylaxis in patients with cystic fibrosis cannot be made
rule "Not eligible due to cystic fibrosis"
  ruleflow-group "rsv-risk-eligibility"
  when
    # find patients with cystic fibrosis
    $p: Patient()
    RSVCysticFibrosis()
    # determine the start date for the relevant RSV season. be sure patient was born
    before the season end
    RSVSeason($startSeason: startDate, $endSeason: endDate > ($p.getBirthDate()))
    then
      # patients with cystic fibrosis are not eligible
      RSVEligibleCandidate fact = new RSVEligibleCandidate();
      fact.setStartDate($startSeason);
      fact.setDoses();
      fact.setReason("Patients with cystic fibrosis may not be eligible for Palivizumab");
      insert(fact);
    end

# Conditional: 8.1 Breakthrough RSV infection
# Action: Continue until a maximum number of doses have been administered
# Reason: This recommendation is based on the observation that infants at
# high risk may be hospitalized more than once in the same season with RSV lower
# respiratory tract disease and the fact that more than 1 RSV strain often
# cocirculates in a community.
rule "continue prophylaxis if breakthrough disease"
   ruleflow-group "rsv-imperative"
   when
      # find patients with breakthrough disease
      RSVBreakthrough($date: date)
   then
      # continue prophylaxis
      RSVImperative fact = new RSVImperative();
      fact.setDirective("RSV disease noted on " + DateTimeFormat
      .forPattern("MM/dd/yyyy").print($date) + ", patients with breakthrough disease should continue to receive RSV prophylaxis");
      insert(fact);
   end

# Conditional: 8.2 Hospitalized infants who qualify for prophylaxis during the RSV season

# Action: receive the first dose of palivizumab 48 to 72 hours before discharge or promptly after discharge
# Conditional: 8.3 Hospitalized during course
# Action: receive that dose as scheduled while they remain in the hospital
# Conditional: 8.4 Infection control
# Directive: RSV is known to be transmitted in the hospital setting and to cause serious disease in infants at high risk. Among hospitalized infants, the major means of reducing RSV transmission is strict observance of infection-control practices, including prompt initiation of precautions for RSV-infected infants.
# If an RSV outbreak occurs: If in a high-risk unit (eg, PICU or NICU or stem cell transplantation unit), primary emphasis should be placed on proper infection control practices, especially hand hygiene. No data exist to support palivizumab use in controlling outbreaks of health care associated disease, and palivizumab use is not recommended for this purpose
rule "hospitalized patients"
   ruleflow-group "rsv-imperative"
   when
      # find hospitalized patients
      RSVHospitalized()
   then
      # if no prior doses: should received first dose 48 to 72 hours before discharge
      # if at least one prior dose: receive next dose on schedule
      # RSV is known to be transmitted in the hospital setting
      RSVImperative fact = new RSVImperative();
      fact.setDirective("hospitalized patients should receive their first dose 48 to 72 hours before discharge or promptly after discharge; " +
      "patients who have received prior doses, should receive the next dose on schedule during hospitalization; " +
      "RSV is known to be transmitted in the hospital setting");
      insert(fact);
   end

# Imperative: 8.5 Palivizumab does not interfere with response to vaccines.
rule "palivizumab does not interfere with response to vaccines"
   ruleflow-group "rsv-imperative"
   when
      $p: Patient()
then
# if no prior doses: should received first dose 48 to 72 hours before discharge
# if at least one prior dose: receive next dose on schedule
RSVImperative fact = new RSVImperative();
fact.setDirective("Palivizumab does not interfere with response to vaccines");
insert(fact);
end

# Ensure that all eligibility candidates have both a detailed and a brief reason
rule "use detailed reason as brief"
ruleflow-group "rsv-risk-eligibility"
when
  $fact: RSVEligibleCandidate($reason: reason != null, reasonBrief == null)
then
  modify($fact) { setReasonBrief($reason); }
end

rule "use brief reason as detailed"
ruleflow-group "rsv-risk-eligibility"
when
  $fact: RSVEligibleCandidate(reason == null, $reason: reasonBrief != null)
then
  modify($fact) { setReason($reason); }
end

# Determine final eligibility based on candidate eligibilities
# aggregate all the reasons that justify the maximum number of doses
rule "Determine final eligibility"
ruleflow-group "rsv-final-eligibility"
when
  RSVSeason($seasonStart: startDate)
# verify no allergy to RSV noted before this season
  not RSVAllergy(notedDate <= $seasonStart)
# verify that final eligibility has not already been set for this season
  not RSVEligibleFinal(startDate == $seasonStart)
# accumulate eligibility information to maximize eligible number of doses
  $fact: RSVEligibleFinal() from accumulate (
    # Identify all candidate explanations for RSV eligibility
    RSVEligibleCandidate($doses: doses, $reason: reason, $reasonBrief:
    reasonBrief, startDate == $seasonStart),
    # Initialize with "base case": Does not meet AAP criteria for Palivizumab
    init (
      RSVEligibleFinal eligible = new RSVEligibleFinal();
      eligible.setStartDate($seasonStart);
      eligible.setDoses(0);
      eligible.setReason("Does not meet AAP criteria for Palivizumab");
      eligible.setReasonBrief(eligible.getReason()); ),
    action (
      # If candidate eligibility exceeds dose recommendation of current eligibility,
      # then take that dose recommendation and reason
      if ($doses > eligible.getDoses()) {
        eligible.setDoses($doses);
        eligible.setReason($reason);
      }
    )
  )
end

eligible.setReasonBrief($reasonBrief);

# If candidate eligibility equals dose recommendation of current eligibility,
# then add that reason to the current eligibility
else if ($doses == eligible.getDoses()) {
    eligible.setReason(eligible.getReason() + "; " + $reason);
    eligible.setReasonBrief(eligible.getReasonBrief() + "; " + $reasonBrief);
}

result({ eligible })

then

insert($fact);

end

# update season information with eligibility
rule "RSV update season information with eligibility"
ruleflow-group "rsv-final-eligibility"
when
    # find season that has not yet had eligibility information described
    $s: RSVSeason($seasonStart: startDate, eligibleDoses == null)
    # obtain eligibility information
    $e: RSVEligibleFinal(startDate == $seasonStart)
then
    modify($s) {
        setEligibleDoses($e.getDoses()),
        setEligibleReasonBrief($e.getReasonBrief()),
        setEligibleReason($e.getReason());
    }
end

# update season information with insurance approval information
rule "RSV update season information not approved"
ruleflow-group "rsv-final-eligibility"
when
    # find season that has not yet had approval information described, but may be eligible
    $s: RSVSeason($seasonStart: startDate, approvedDoses == null)
    # verify no approval information
    not RSVApproval(startDate == $seasonStart)
then
    modify($s) { setApprovedDoses(0), setInsuranceStatus(RSVSeason.NOT_SUBMITTED); }
end

# update season information with insurance approval information if not eligible
rule "RSV update season information not eligible"
ruleflow-group "rsv-final-eligibility"
when
    # find season that has not yet had approval information described
    $s: RSVSeason($seasonStart: startDate, approvedDoses == null, eligibleDoses == 0)
    # verify no approval information
    not RSVApproval(startDate == $seasonStart)
then
    modify($s) { setApprovedDoses(0), setInsuranceStatus(RSVSeason.NOT_ELIGIBLE); }
end
# update season information with insurance approval information if it is found
rule "RSV update season information approval status"
ruleflow-group "rsv-final-eligibility"
  when
    # find season that has not yet had approval information described
    $s: RSVSeason($seasonStart: startDate, approvedDoses == null)
  # identify approval statuses this season
    RSVApproval($date: date, startDate == $seasonStart, $doses: doses != null,
    $status: status)
  # verify no more recent status
    not RSVApproval(date > $date, startDate == $seasonStart, doses != null)
  then
    modify($s) { setApprovedDoses($doses), setInsuranceStatus($status); }
end

# Data Extraction methods
# identify and extract insurance approval information for each season
rule "RSV season insurance approvals"
ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
  # identify timing of each season
    RSVSeason($startDate: startDate, $endDate: endDate, $prepDate: prepDate)
  # iterate over insurance information
    $auth: PriorAuthorization(instant >= $prepDate, instant < $endDate, item matches "(?ism).*\b(?:rsv|palivizumab|synagis|resp.*syncytial.*virus)\b.*") from
    $p.getPriorAuthorizations()
  then
    RSVApproval fact = new RSVApproval();
    fact.setStartDate($startDate);
    fact.setDate($auth.getInstant());
    fact.setDoses($auth.getQuantity().intValue());
    fact.setStatus($auth.getStatus());
    insert(fact);
end

# ALLERGY related extraction methods
rule "RSV allergy"
ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
  # identify any allergy to Palivizumab or its ingredients
    Allergy($noted: noted, description matches "(?ism).*\b(?:rsv|palivizumab|synagis|resp.*syncytial.*virus)\b.*") from
    $p.getAllergies()
  then
    RSVAllergy fact = new RSVAllergy();
    # assume conservatively as noted at birth if no noted date is specified
    fact.setNotedDate($noted == null ? $p.getBirthDate() : $noted);
    insert(fact);
end

# DISEASE related extraction methods
# look for diseases of interest based on diagnosis codes
# chronic lung disease

**rule** "has chronic lung disease"  
**ruleflow-group** "rsv-extract-data"  
when  
  $p$: Patient()  
  # look for existence of a qualifying associated with this patient  
  # include chronic lung disease  
  $dx$: ArrayList(size > 0) from collect (Diagnosis(status not matches "(?ism).*\(?\:del\).*", icd9 matches "(?:770\.7|518\.8).*") from $p.getDiagnoses())  
then  
  RSVChronicLungDisease fact = new RSVChronicLungDisease();  
  fact.setDescription("chronic lung disease");  
  fact.setDiagnoses(Diagnosis.abbreviatedList($dx));  
  fact.setIcd9(Diagnosis.icd9List($dx));  
  insert(fact);  
end

**rule** "chart review has chronic lung disease"  
**ruleflow-group** "rsv-chart-review"  
when  
  # identify chronic lung disease  
  RSVChronicLungDisease($icd9: icd9)  
  # identify seasons where chronic lung disease has not been described  
  $s$: RSVSeason(chronicLungDisease == "")  
then  
  modify($s) { setChronicLungDisease("Yes" + ($icd9.equals("") ? "" : " - " + $icd9)); }  
end

**rule** "chart review has bronchodilators"  
**ruleflow-group** "rsv-chart-review"  
when  
  $p$: Patient()  
  # determine season start date  
  $s$: RSVSeason($seasonStart: startDate, bronchodilators == "")  
  # check med list for a qualifying prescription  
  $rx$: ArrayList(size > 0) from collect (Prescription(endDate == null || endDate >= ($seasonStart.minusMonths(6)), pharmClass matches "(?ism).*\b(?:antiasthmatics?)\b.*", pharmSubclass matches "(?ism).*\b(?:sympathomimetics?)\b.*" || generic matches "(?ism).*\b(?:albuterol|salmeterol)\b.*") from $p.getPrescriptions())  
then  
  modify($s) { setBronchodilators(Prescription.abbreviatedList($rx)); }  
end

**rule** "chart review has corticosteroids"  
**ruleflow-group** "rsv-chart-review"  
when  
  $p$: Patient()  
  # determine season start date  
  $s$: RSVSeason($seasonStart: startDate, corticosteroids == "")  
  # check med list for a qualifying prescription
$rx: ArrayList(size > 0) from collect (Prescription(endDate == null || endDate >=
($seasonStart.minusMonths(6)),
pharmClass matches "(?ism).*\b(?:antiasthmatics?)\b.*",
pharmSubclass matches "(?ism).*\b(?:steroids?)\b.*" ||
generic matches "(?ism).*\b(?:fluticasone|budesonide|
beclomethasone|triamcinolone)\b.*")
from $p.getPrescriptions())
then
    modify($s) { setCorticosteroids(Prescription.abbreviatedList($rx)); }
end

rule "chart review has diuretics"
ruleflow-group "rsv-chart-review"
when
  $p: Patient()
  # determine season start date
  $s: RSVSeason($seasonStart: startDate, diuretics == "")
  # check med list for a qualifying prescription
  $rx: ArrayList(size > 0) from collect (Prescription(endDate == null || endDate >=
($seasonStart.minusMonths(6)),
pharmClass matches "(?ism).*\b(?:diuretics?)\b.*")
from $p.getPrescriptions())
then
    modify($s) { setDiuretics(Prescription.abbreviatedList($rx)); }
end

rule "chart review has oxygen"
ruleflow-group "rsv-chart-review"
when
  $p: Patient()
  # determine season start date
  $s: RSVSeason($seasonStart: startDate, oxygen == "")
  # check med list for a qualifying prescription
  $rx: ArrayList(size > 0) from collect (Prescription(endDate == null || endDate >=
($seasonStart.minusMonths(6)),
generic matches "(?ism).*\b(?:oxygen?)\b.*")
from $p.getPrescriptions())
then
    modify($s) { setOxygen("Yes"); }
end

# airway abnormality
rule "has airway abnormality"
ruleflow-group "rsv-extract-data"
when
  $p: Patient()
  # look for existence of a qualifying associated with this patient
  # include: tracheostomy, pulmonary hypoplasia, CCAM, CDH, and giant omphalocoele
  $dx: ArrayList(size > 0) from collect (Diagnosis(status not matches "(?ism).*\b(resolve|del)\b.*",
description: description not matches "(?ism).*malacia.*",
icd9: icd9 matches "(?ism)(?:V44\.0|748\.[2345]|756\.6|518\.8[34]).*")
from $p.getDiagnoses())
then
    RSVAirwayNeuromuscularDisease fact = new RSVAirwayNeuromuscularDisease();
    fact.setDescription("airway or pulmonary abnormality");
    fact.setDiagnoses(Diagnosis.abbreviatedList($dx));
    fact.setIcd9(Diagnosis.icd9List($dx));
    fact.setType("airway");
    insert(fact);
end

# giant omphalocele
rule "has omphalocele"
  ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
    # look for existence of a qualifying associated with this patient
    # include: tracheostomy, pulmonary hypoplasia, CCAM, CDH, and giant omphalocele
    $dx: ArrayList(size > 0) from collect (Diagnosis(status not matches "(?ism).*(?:resolve|del).*", $description: description matches "(?ism).*omphaloco?le?le.*", $icd9: icd9 matches "(?ism)(?:756\.79).*") from $p.getDiagnoses())
  then
    RSVAirwayNeuromuscularDisease fact = new RSVAirwayNeuromuscularDisease();
    fact.setDescription("omphalocele");
    fact.setDiagnoses(Diagnosis.abbreviatedList($dx));
    fact.setIcd9(Diagnosis.icd9List($dx));
    fact.setType("airway");
    insert(fact);
end

# stridor
rule "has stridor"
  ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
    # patients with stridor may be eligible if there are significant airway issues present
    $dx: ArrayList(size > 0) from collect (Diagnosis(status not matches "(?ism).*(?:resolve|del).*", $description: description not matches "(?ism).*malacia.*", $icd9: icd9 matches "(?ism)(?:786\.1).*") from $p.getDiagnoses())
  then
    RSVStridor fact = new RSVStridor();
    fact.setDescription("stridor");
    fact.setDiagnoses(Diagnosis.abbreviatedList($dx));
    fact.setIcd9(Diagnosis.icd9List($dx));
    fact.setType("airway");
    insert(fact);
end

rule "chart review has airway abnormality"
  ruleflow-group "rsv-chart-review"
when
  # identify airway abnormalities
  RSVAirwayNeuromuscularDisease(type == "airway", $icd9: icd9)
  # identify seasons where airway abnormalities have not been described
  $s: RSVSeason(airwayAbnormalities == "")
  then
    modify($s) { setAirwayAbnormalities("Yes" + ($icd9.equals("") ? "" : " - " + $icd9)); }
end

# neuromuscular disease
rule "has neuromuscular disease"
  ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
    # look for existence of a qualifying associated with this patient
    # include spinal muscle atrophy and myopathies
    $dx: ArrayList(size > 0) from collect (Diagnosis(status not matches "(?ism).*(?:resolve|del).*", $description: description, $icd9: icd9 matches "(?:33[456]|359|742.2).*") from $p.getDiagnoses())
    then
      RSVAirwayNeuromuscularDisease fact = new RSVAirwayNeuromuscularDisease();
      fact.setDescription("neuromuscular disease");
      fact.setDiagnoses(Diagnosis.abbreviatedList($dx));
      fact.setIcd9(Diagnosis.icd9List($dx));
      fact.setType("neuromuscular");
      insert(fact);
  end

rule "chart review has neuromuscular disease"
  ruleflow-group "rsv-chart-review"
  when
    # identify neuromuscular abnormalities
    RSVAirwayNeuromuscularDisease(type == "neuromuscular", $icd9: icd9)
    # identify seasons where neuromuscular abnormalities have not been described
    $s: RSVSeason(neuroMuscularDisease == "")
    then
      modify($s) { setNeuroMuscularDisease("Yes" + ($icd9.equals("") ? "" : " - " + $icd9)); }
  end

# congenital heart disease
rule "has congenital heart disease"
  ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
    # look for existence of a qualifying associated with this patient
    # include cardiac and "pulmonary heart" disease categories such as pulmonary hypertension
    $dx: ArrayList(size > 0) from collect (Diagnosis(status not matches "(?ism).*(?:resolve|del).*", $description: description, $icd9: icd9 matches "(?:41[567]|42[458]|74[567]|759\.3).*") from $p.getDiagnoses())
    then
      RSVCongenitalHeartDisease fact = new RSVCongenitalHeartDisease();
fact.setDescription("congenital heart disease");
fact.setDiagnoses(Diagnosis.abbreviatedList($dx));
fact.setIcd9(Diagnosis.icd9List($dx));
insert(fact);
end

rule "chart review has congenital heart disease"
ruleflow-group "rsv-chart-review"
when
    # identify congenital heart disease
    RSVCongenitalHeartDisease($icd9: icd9)
    # identify seasons where congenital heart disease has not been described
    $s: RSVSeason(cardiacDisease == "")
then
    modify($s) { setCardiacDisease("Yes" + ($icd9.equals("") ? "" : " - " + $icd9)); }
end

rule "chart review has pulmonary hypertension"
ruleflow-group "rsv-chart-review"
when
    $s: RSVSeason(pulmonaryHypertension == "")
    $p: Patient()
    # look for existence of a qualifying associated with this patient
    $dx: ArrayList(size > 0) from collect (Diagnosis(status not matches "(?ism).*?(resolve|del).*", description matches "(?ism).*\b(?:pulm.*(?:htn|hypertension))\b.*") from $p.getDiagnoses())
then
    modify($s) { setPulmonaryHypertension("Yes - " + Diagnosis.icd9List($dx)); }
end

rule "chart review hypoxemia"
ruleflow-group "rsv-chart-review"
when
    $s: RSVSeason($startSeason: startDate, minPulseOx == null)
    # verify existence of congenital heart disease
    RSVCongenitalHeartDisease()
    # verify presence of hypoxemia within 6 months before season
    exists RSVHypoxemia(seasonStart == $startSeason)
    Number($pulseOxMin: intValue) from accumulate (RSVHypoxemia($pulseOx: pulseOx, seasonStart == $startSeason), min($pulseOx))
then
    modify($s) { setMinPulseOx($pulseOxMin), setCyanoticCardiacDisease("Yes") }
end

# recent cardiac medications
rule "has recent cardiac medications"
ruleflow-group "rsv-extract-data"
when
    $p: Patient()
    # determine season start date
    $s: RSVSeason($seasonStart: startDate, cardiacMedications == "")
# only check if there is congenital heart disease
RSVCongenitalHeartDisease()
# check med list for a qualifying prescription (diuretics, afterload reducing agents, inotropes, salicylates)
$rx: ArrayList(size > 0) from collect (Prescription(endDate == null || endDate >= ($seasonStart.minusMonths(6)),
  pharmClass matches "(?ism).*\b(?:diuretics|cardiotonics|cardiovascular|antihypertensives?)\b.*" ||
generic matches "(?ism).*\b(?:acetyl\s*salicyl\w*)\b.*") from $p.getPrescriptions())
then
  RSVCardiacMedications fact = new RSVCardiacMedications();
  fact.setDescription("medications: " + Prescription.abbreviatedList($rx));
  fact.setSeasonStart($seasonStart);
  insert(fact);
  modify($s) { setCardiacMedications(Prescription.abbreviatedList($rx)); }
end

# hypoxemia within 6 months before season start
rule "has hypoxemia"
ruleflow-group "rsv-extract-data"
when
  $p: Patient()
  # determine season start date
  $s: RSVSeason($seasonStart: startDate, $seasonEnd: endDate)
  # only check if there is congenital heart disease
  RSVCongenitalHeartDisease()
  # iterate over encounters on or after 6 months before season start to find documented hypoxemia
  $e: Encounter(instant >= ($seasonStart.minusMonths(6)), instant < $seasonEnd) from $p.getEncounters()
  # report any documented pulse ox below 90% as significant
  VitalSign(type == VitalSign.PULSE_OX, $pulseOx: value < 90.0, value >= 50.0) from $e.getVitalSigns()
then
  RSVHypoxemia fact = new RSVHypoxemia();
  fact.setPulseOx($pulseOx);
  fact.setSeasonStart($seasonStart);
  insert(fact);
end

# immune deficiency
rule "has immune deficiency"
ruleflow-group "rsv-extract-data"
when
  $p: Patient()
  # look for existence of a qualifying associated with this patient
  # very few immune deficiencies meet eligibility criteria
  # include heterotaxy and DiGeorge, which often has cardiac anomalies combined with immunologic issues
  $dx: ArrayList(size > 0) from collect (Diagnosis(status not matches "(?ism).*\b(?:resolve|del).*", $description: description, $icd9: icd9,
    icd9 matches "(?:753\.|321279\.|111279\.|2).*" || description matches "(?ism).*\b(?:heterotaxy)\b.*") from $p.getDiagnoses())
then
    RSVImmuneDeficiency fact = new RSVImmuneDeficiency();
    fact.setDescription("immune deficiency");
    fact.setDiagnoses(Diagnosis.abbreviatedList($dx));
    fact.setIcd9(Diagnosis.icd9List($dx));
    insert(fact);
end

rule "chart review has immune deficiency"
    ruleflow-group "rsv-chart-review"
    when
        # identify immune deficiencies
        RSVImmuneDeficiency($icd9: icd9)
        # identify seasons where immune deficiency has not been described
        $s: RSVSeason(immuneDeficiency == ")"
    then
        modify($s) { setImmuneDeficiency("Yes" + ($icd9.equals("") ? "" : " - " + $icd9)); }
end

# cystic fibrosis
rule "has cystic fibrosis"
    ruleflow-group "rsv-extract-data"
    when
        $p: Patient()
        # look for existence of a qualifying associated with this patient
        # include cystic fibrosis
        $dx: ArrayList(size > 0) from collect (Diagnosis(status not matches "(?ism).*(?:resolve|del).*", $description: description, $icd9: icd9 matches "277\ |.0.*") from $p.getDiagnoses())
    then
        RSVCysticFibrosis fact = new RSVCysticFibrosis();
        fact.setDescription("cystic fibrosis");
        fact.setDiagnoses(Diagnosis.abbreviatedList($dx));
        fact.setIcd9(Diagnosis.icd9List($dx));
        insert(fact);
end

# the infant attends child care, defined as a home or facility in which care
# is provided for any number of infants or toddlers in the child care facility
rule "attends child care"
    ruleflow-group "rsv-chart-review"
    when
        # identify patients with a documented social history
        Patient($socialHistory: socialHistory != "")
        # construct a matcher object to classify multiple birth
        $matcher: Matcher() from RSVSeason.CHILDRENGARE_REGEX.matcher($socialHistory.getNarrative())
    then
        eval($matcher.matches())
        # see dependent rules
end
rule "child care present" extends "attends child care"
ruleflow-group "rsv-chart-review"
  when
    # verify that the documentation is not "no"
    eval(!$matcher.group(1).matches("(?ism)no.*"))
  then
    RSVExposureRisk fact = new RSVExposureRisk();
    fact.setReason("attends child care");
    insert(fact);
  end

rule "chart review attends child care" extends "attends child care"
ruleflow-group "rsv-chart-review"
  when
    # identify seasons where child care has not been specified
    $s: RSVSeason(childCare == ")
  then
    modify($s) { setChildCare($matcher.group(1)); }
  end

# 1 or more siblings or other children younger than 5 years live
# permanently in the same household
rule "sibling under 5 years"
ruleflow-group "rsv-extract-data"
  when
    # identify patients with a documented social history
    Patient($socialHistory: socialHistory != "")
    # construct a matcher object to classify multiple birth
    $matcher: Matcher() from
    RSVSeason.SIBLINGS_UNDER5_REGEX.matcher($socialHistory.getNarrative())
    # determine if the pattern matches
    eval($matcher.matches())
  then
    # see dependent rules
  end

rule "sibling under 5 present" extends "sibling under 5 years"
ruleflow-group "rsv-chart-review"
  when
    # verify that the documentation is not "no"
    eval(!$matcher.group(1).matches("(?ism)no.*"))
  then
    RSVExposureRisk fact = new RSVExposureRisk();
    fact.setReason("has sibling under age 5 years");
    insert(fact);
  end

rule "chart review sibling under 5 years" extends "sibling under 5 years"
ruleflow-group "rsv-chart-review"
  when
    # identify seasons where siblings have not been specified
    $s: RSVSeason(siblings == ")
  then
    modify($s) { setSiblings($matcher.group(1)); }
# multiple birth implies 1 or more siblings or other children younger than 5 years live
# permanently in the same household

rule "multiple birth"
  ruleflow-group "rsv-chart-review"
  when
    # identify patients with a documented social history
    Patient($socialHistory: socialHistory != "")
    # construct a matcher object to classify multiple birth
    $matcher: Matcher() from
    RSVSeason.MULTIPLE_BIRTH_REGEX.matcher($socialHistory.getNarrative())
    # determine if the pattern matches
    eval($matcher.matches())
  then
    # see dependent rules
  end

rule "multiple birth present" extends "multiple birth"
  ruleflow-group "rsv-chart-review"
  when
    # verify that the multiple birth documentation is not "no" or "singleton"
    eval(!$matcher.group(1).matches("(?ism)(?:no|singl).*"))
  then
    RSVExposureRisk fact = new RSVExposureRisk();
    fact.setReason($matcher.group(1) + " birth");
    insert(fact);
  end

rule "chart review multiple birth" extends "multiple birth"
  ruleflow-group "rsv-chart-review"
  when
    # identify seasons where multiple birth has not been specified
    $s: RSVSeason(multipleBirth == "")
  then
    modify($s) { setMultipleBirth($matcher.group(1)); }
# pollutants
rule "chart review pollutants"
  ruleflow-group "rsv-chart-review"
  when
    # identify patients with a documented social history
    Patient($socialHistory: socialHistory != "")
    # construct a matcher object to classify multiple birth
    $matcher: Matcher() from RSVSeason.POLLUTANTS_REGEX.matcher($socialHistory.getNarrative())
    # determine if the pattern matches
    eval($matcher.matches())
    # identify seasons where pollutant exposure has not been specified
    $s: RSVSeason(pollutantExposure == "")
  then
    modify($s) { setPollutantExposure($matcher.group(1)); }
  end

# discharge date
rule "chart review valid discharge date"
  ruleflow-group "rsv-chart-review"
  when
    # identify patients with a valid discharge date
    Patient($birthDate: birthDate, $birthHistory: birthHistory, eval(birthHistory.getDischargeDate() != null))
    # extract discharge date for convenience
    $dischDate: DateTime() from $birthHistory.getDischargeDate()
    # determine if discharge date is meaningful
    eval($dischDate.isAfter($birthDate))
  then
    # see dependent rules
  end

rule "chart review nicu stay" extends "chart review valid discharge date"
  ruleflow-group "rsv-chart-review"
  when
    $s: RSVSeason(nicuStay == "")
  then
    modify($s) { setNicuStay("" + Days.daysBetween($birthDate, $dischDate).getDays() + " days (d/c " + FuzzyDate.monthDayYearBrief($dischDate) + ")"); }
  end

rule "chart review nicu palivizumab" extends "chart review valid discharge date"
  ruleflow-group "rsv-chart-review"
  when
    # check to see if immunization given on or before discharge date
    RSVImmunization($given: date <= $dischDate)
    # find seasons where NICU palivizumab not specified
    $s: RSVSeason(nicuPalivizumab == "")
  then
    modify($s) { setNicuPalivizumab("Given " + FuzzyDate.monthDayYearBrief($given)); }
  end

# extract doses of RSV given
rule "doses given this season"
  ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
    # loop over immunizations to find RSV immunization products
    $imm: Immunization($given: given, product matches "(?ism).*\b(?:rsv|palivizumab|synagis|resp.*syncytial.*virus)\b.*") from $p.getImmunizations()
    # identify season that contains this immunization
    RSVSeason($seasonStart: startDate <= $given, endGracePeriod > $given)
    # suppress duplicates
    not RSVImmunization(date == $given)
  then
    RSVImmunization fact = new RSVImmunization();
    fact.setDate($given);
    fact.setSeasonStart($seasonStart);
    insert(fact);
    # publish this as a fact for use by other rules
    insert(new DateFact($given, DateFact.RSV_IMMUNIZATION));
  end

rule "rsv current encounter department"
  ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
    # find ID of current encounter
    Identifier(type == Identifier.ENCOUNTER_ID, $enc_id: id) from
    $p.getIdentifiers()
    # iterate over encounters
    $e: Encounter() from $p.getEncounters()
    # verify that this matches the current encounter id
    Identifier(type == Identifier.ENCOUNTER_ID, id == $enc_id) from
    $e.getIdentifiers()
    # find the department ID of this encounter
    Identifier(type == Identifier.DEPARTMENT_ID, $dept_id: id) from
    $e.getIdentifiers()
    then
      # describe this department publicly
      insert(new DocFact(DocFact.RSV_ENC_DEPARTMENT_ID, $dept_id));
  end

rule "rsv login department"
  ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
    # find ID of current encounter
    Identifier(type == Identifier.DEPARTMENT_ID, $dept_id: id) from
    $p.getIdentifiers()
    then
      # describe this department publicly
      insert(new DocFact(DocFact.RSV_DEPARTMENT_ID, $dept_id));
  end

rule "appointments in current department"
  ruleflow-group "rsv-extract-data"
  when
    $p: Patient()
    # find ID of current encounter
    Identifier(type == Identifier.DEPARTMENT_ID, $dept_id: id) from
    $p.getIdentifiers()
    then
      # describe this department publicly
      insert(new DocFact(DocFact.RSV_ENC_DEPARTMENT_ID, $dept_id));
  end
when $p: Patient()  
    # find current department  
    DocFact(attribute == DocFact.RSV_DEPARTMENT_ID, $dept_id: value)  
    # iterate over encounters  
    $e: Encounter() from $p.getEncounters()  
    # verify this encounter is in the current department  
    Identifier(type == Identifier.DEPARTMENT_ID, id == $dept_id) from  
    $e.getIdentifiers()  
    # verify that this encounter is of a suitable type  
    Identifier(type == Identifier.ENCOUNTER_TYPE, $status: id matches "(?  
    ism).*\b(?:appointment|scheduled?)\b.*") from $e.getIdentifiers()  
    # verify that this encounter is not cancelled or no show  
    not (Identifier(type == Identifier.APPOINTMENT_STATUS, id matches "(?  
    ism).*\b(?:no.?show|cancel\w*)\b.*") from $e.getIdentifiers())  
    then  
    RSVAppointment fact = new RSVAppointment();  
    fact.setStatus($status);  
    fact.setDate($e.getInstant());  
    insert(fact);  
end  

# number doses in season and determine timeliness  
rule "number doses given in season"  
    ruleflow-group "rsv-schedule"  
    when  
        # identify an un-numbered dose in season  
        $imm: RSVImmunization($givenDate: date, $seasonStart: seasonStart, doseNum ==  
        null)  
        # verify no prior immunizations this season which have not been numbered  
        not RSVImmunization(date < $givenDate, seasonStart == $seasonStart, doseNum ==  
        null)  
        then  
            # see dependent rules  
    end  

rule "first dose given in season" extends "number doses given in season"  
    ruleflow-group "rsv-schedule"  
    when  
        # verify no prior immunizations this season that is numbered  
        not RSVImmunization(date < $givenDate, seasonStart == $seasonStart, doseNum !=  
        null)  
        then  
            modify($imm) { setDoseNum(1), setStatus("first") }  
    end  

rule "subsequent dose given in season" extends "number doses given in season"  
    ruleflow-group "rsv-schedule"  
    when  
        # find prior immunizations this season that has been numbered  
        RSVImmunization($priorDate: date < $givenDate, seasonStart == $seasonStart,  
        $priorNum: doseNum != null)  
        # verify there are no more recent numbered doses  
        not RSVImmunization(date > $priorDate, date < $givenDate, seasonStart ==  
        null)  
        then  
            modify($prior) { setDoseNum($priorNum + 1), setStatus("subsequent") }  
    end
$seasonStart, doseNum != null)
    # calculate days since preceding immunization
    $daysBetween: Integer() from Days.daysBetween($priorDate, $givenDate).getDays()
    then
        # see dependent rules
end

rule "subsequent dose given early" extends "subsequent dose given in season"
ruleflow-group "rsv-schedule"
when
    eval($daysBetween < 25)
then
    modify($imm) { setDoseNum($priorNum + 1), setStatus("early") }
end

rule "subsequent dose given on time" extends "subsequent dose given in season"
ruleflow-group "rsv-schedule"
when
    eval($daysBetween >= 25 && ($priorNum > 1 && $daysBetween <= 35 || $priorNum == 1
    && $daysBetween <= 30))
then
    modify($imm) { setDoseNum($priorNum + 1), setStatus("given") }
end

rule "subsequent dose given late" extends "subsequent dose given in season"
ruleflow-group "rsv-schedule"
when
    eval($priorNum > 1 && $daysBetween > 35 || $priorNum == 1 && $daysBetween > 30)
then
    modify($imm) { setDoseNum($priorNum + 1), setStatus("late") }
end

# describe the expected schedule this season based on both eligibility and approval
rule "doses expected this season"
ruleflow-group "rsv-schedule"
when
    $p: Patient()
    # permit up to 5 doses per season regardless of eligibility
    # this permits some decision uspport even for patients who
    # are not eligible in the event that they are nevertheless
    # approved to receive synagis
    RSVSeason($seasonStart: startDate)
then
    for(int i = 0; i < 5; i++) {
        RSVActualSchedule fact = new RSVActualSchedule();
        fact.setDoseNum(i+1);
        fact.setSeasonStart($seasonStart);
        insert(fact);
    }
end

# IDEAL schedule for analysis purposes
# describe an ideal schedule for this season derived from a reasonable start date
# and based on the eligibility
rule "ideal schedule for this season"
  ruleflow-group "rsv-annotate-schedule"
  when
    RSVActualSchedule($seasonStart: seasonStart, doseNum == 1, $interval: interval != null)
  # determine number of doses expected
    RSVEligibleFinal(startDate == $seasonStart, $totalDoses: doses > 0)
  then
    DateTime idealStart = $interval.getStart();
    for(int i = 0; i < $totalDoses; i++) {
      DateTime idealDate = idealStart.plusDays(i*30);
      insert(new Schedule(Schedule.RSV_IMMUNIZATION, idealDate, new Interval(idealDate, idealDate.plusDays(30)), Period.days(25)));
    }
  end

# determine interval for first dose
rule "interval first dose"
  ruleflow-group "rsv-annotate-schedule"
  when
    $p: Patient($birthDate: birthDate)
  # identify schedule items that have not had an interval assigned for dose number 1
    $fact: RSVActualSchedule($seasonStart: seasonStart, doseNum == 1, $interval: interval == null)
  then
    # do not recommend giving first dose before birth date, or before gestational age 35
    # weeks as a reasonable start. Also check to make sure it is not before the
    # discharge date (if that date is known)
    DateTime gest35 = $p.getDueDate().minusWeeks(5);
    DateTime dischargeDate = $p.getBirthHistory().getDischargeDate();
    DateTime effectiveStart = ($seasonStart.isBefore($birthDate) ? $birthDate : $seasonStart);
    effectiveStart = (effectiveStart.isBefore(gest35) ? gest35 : effectiveStart);
    effectiveStart = (dischargeDate != null && effectiveStart.isBefore(dischargeDate) ? dischargeDate : effectiveStart);
    modify($fact) {
      # ideally dose should be given within 10 days of season start
      setInterval(new Interval(effectiveStart, effectiveStart.plusDays(10)))
    }
  end

rule "annotate dose interval"
  ruleflow-group "rsv-annotate-schedule"
  when
    # identify schedule items that have not had a date assigned
    $fact: RSVActualSchedule($seasonStart: seasonStart, $doseNum: doseNum > 1, interval == null)
  # identify date of prior dose, do not calculate intervals for more than 1 dose
  beyond end of official season
    RSVActualSchedule(seasonStart == $seasonStart, doseNum == ($doseNum - 1), $priorDate: date < ($seasonStart.plusMonths(5)))
then
    # see dependent rules
end

rule "interval second dose" extends "annotate dose interval"
ruleflow-group "rsv-annotate-schedule"
when
eval($doseNum == 2)
then
    modify($fact) {
        # ideally dose #2 should be given within 25-30 days of dose #1
        setInterval(new Interval($priorDate.plusDays(25), $priorDate.plusDays(31)))
    }
end

rule "interval subsequent dose" extends "annotate dose interval"
ruleflow-group "rsv-annotate-schedule"
when
eval($doseNum > 2)
then
    modify($fact) {
        # ideally dose #3-5 should be given within 25-35 days after prior dose, end
date is excluded
        setInterval(new Interval($priorDate.plusDays(25), $priorDate.plusDays(36)))
    }
end

# annotate schedule for doses that were given
rule "annotate schedule given doses"
ruleflow-group "rsv-annotate-schedule"
when
    # identify schedule items that have not had a date assigned
    $fact: RSVActualSchedule($seasonStart: seasonStart, $doseNum: doseNum, date ==
null)
    # identify corresponding administered dose
    RSVImmunization($givenDate: date, seasonStart == $seasonStart, doseNum ==
$doseNum)
then
    modify($fact) {
        setDate($givenDate), setStatus("Given")
    }
end

# annotate schedule for doses that were NOT given
rule "annotate schedule dose not given"
ruleflow-group "rsv-annotate-schedule"
when
    $p: Patient($birthDate: birthDate)
    # identify schedule items that have not had a date assigned
    $fact: RSVActualSchedule($seasonStart: seasonStart, $doseNum: doseNum, $date: date ==
null, $interval: interval != null)
    # verify absence of corresponding administered dose
    not RSVImmunization(seasonStart == $seasonStart, doseNum == $doseNum)
then
# see dependent rules

rule "scheduled date for dose not given" extends "annotate schedule dose not given"
ruleflow-group "rsv-annotate-schedule"
when
  # Look for first RSVAppointment when vaccine can be given in interval
  RSVAppointment($status: status, $scheduledDate: date, eval($interval.contains(date)))
  # verify no earlier RSVAppointment when vaccine can be given on time
  not RSVAppointment(date < $scheduledDate, eval($interval.contains(date)))
then
  modify($fact) { setDate($scheduledDate), setStatus("Scheduled") }
  # indicate to growth module that we would like a predicted weight
  insert(new DateFact($scheduledDate, DateFact.PREDICT_MEASUREMENT, VitalSign.WEIGHT_KG));
end

rule "no scheduled date for dose not given" extends "annotate schedule dose not given"
ruleflow-group "rsv-annotate-schedule"
when
  # verify no RSVAppointment when vaccine can be given on time
  not RSVAppointment($scheduledDate: date, eval($interval.contains(date)))
then
  DateTime idealDate = $interval.getStart().plusDays(5);
  modify($fact) {
    # will target 5 days into the interval as "ideal date"
    setDate(idealDate), setStatus("No appointment")
  }
  # indicate to growth module that we would like a predicted weight
  insert(new DateFact(idealDate, DateFact.PREDICT_MEASUREMENT, VitalSign.WEIGHT_KG));
end

rule "has late scheduled appointment"
ruleflow-group "rsv-annotate-schedule"
when
  $fact: RSVActualSchedule(status == "No appointment", $interval: interval != null)
  # define a broader window when a "late" appointment may exist -- look
  # 20 days beyond end of acceptable interval
  $lateInterval: Interval() from $interval.withEnd($interval.getEnd().plusDays(20))
  # verify presence of an RSVAppointment when vaccine can be given late
  RSVAppointment($status: status, $scheduledDate: date, eval($lateInterval.contains(date)))
  # verify no earlier "late appt"
  not RSVAppointment(date < $scheduledDate, eval($lateInterval.contains(date)))
then
  modify($fact) {
    # will still target 5 days into the interval as "ideal date," but report the
    scheduled date
    setStatus("None in range")
    #setStatus("Out of range: " + FuzzyDate.monthDayYearBrief($scheduledDate))
  }
end
rule "describe date range"
ruleflow-group "rsv-annotate-schedule"
when
  $fact: RSVActualSchedule($interval: interval != null, dateRange == null)
then
  modify($fact) {
    # describe date in m/d/yy format, recall that end date is exclusive in the
    internal representation
    setDateRange(FuzzyDate.monthDayYearBrief($interval.getStart()) + " - "+
                 FuzzyDate.monthDayYearBrief($interval.getEnd().minusDays(1)));
  }
end

rule "describe date for given doses or appointments"
ruleflow-group "rsv-annotate-schedule"
when
  $fact: RSVActualSchedule($date: date != null, dateActual == null, status ==
  "Given" || status == "Appointment" || status == "Scheduled")
then
  modify($fact) {
    # describe date in m/d/yy format
    setDateActual(FuzzyDate.monthDayYearBrief($date));
  }
end

# final recommendations -- execute after all other rules have fired
rule "RSV final recommendations"
ruleflow-group "rsv-recommendation"
when

  # include season information in final results
  $seasons: ArrayList() from collect (RSVSeason())
  # include any special considerations that affect eligibility or schedule
  $imperatives: ArrayList() from collect (RSVImperative())
  # collect ideal schedule information
  $schedule: ArrayList() from collect (Schedule(event == Schedule.RSV_IMMUNIZATION))
  # include actual RSV administration information
  $scheduleActual: ArrayList() from collect (RSVActualSchedule())
  # include all RSV immunization schedule information, both given and recommended
  $given: ArrayList() from collect (RSVImmunization())
then
  RSVRecommendation fact = new RSVRecommendation();
  fact.setSeasons($seasons);
  fact.setImperatives($imperatives);
  fact.setSchedule($schedule);
  fact.setScheduleActual($scheduleActual);
  fact.setGivenDoses($given);
  fact.setSummaryStatement(""");
  fact.setEligibleReasonThisSeason(""");
  fact.setEligibleReasonBriefThisSeason(""");
  fact.setChartReviewQuery(""");
  fact.setIsOnList(false);
  fact setShowRsvOnly(false);
  insert(fact);
# final recommendations -- count doses given this season
rule "RSV summarize season count given doses"
ruleflow-group "rsv-recommendation"
when
   # identify the seasons where number of doses haven't been counted
   $s: RSVSeason($interval: interval, givenDoses == null)
   # count doses given in the season
   ArrayList($numGiven: size) from collect
   (RSVImmunization(eval($interval.contains(date))))
then
   modify($s) { setGivenDoses($numGiven); }
end

# final recommendations -- summarize status this season if allergic
rule "RSV summarize season for allergy"
ruleflow-group "rsv-recommendation"
when
   # identify RSV recommendations that do not have a summary
   $fact: RSVRecommendation(summaryStatement == "")
   # identify evaluation date
   $p: Patient($evalDate: evalDate)
   # identify the correct season
   $s: RSVSeason(prepDate <= $evalDate, $seasonEnd: endDate > $evalDate)
   # verify that there is no documented allergy before the end of this season
   # note that we only suppress eligibility if allergy was noted BEFORE the season,
   # during the season we will suppress any recommendations to administer further
   doses
      exists RSVAllergy(notedDate < $seasonEnd)
then
   modify($fact) { setSummaryStatement("Documented allergy") }
end

# final recommendations -- summarize status this season
rule "RSV summarize season"
ruleflow-group "rsv-recommendation"
when
   # identify RSV recommendations that do not have a summary
   $fact: RSVRecommendation()
   # identify evaluation date
   $p: Patient($evalDate: evalDate)
   # identify the correct season
   $s: RSVSeason(prepDate <= $evalDate, $seasonEnd: endDate > $evalDate,
      $seasonStart: startDate, $eligibleReason: eligibleReason, $eligibleReasonBrief: eligibleReasonBrief,
      $eligibleDoses: eligibleDoses != null, $approvedDoses: approvedDoses != null,
      $givenDoses: givenDoses != null)
   # verify that there is no documented allergy before the end of this season
   # note that we only suppress eligibility if allergy was noted BEFORE the season,
   # during the season we will suppress any recommendations to administer further
   doses
      not RSVAllergy(notedDate < $seasonEnd)
then
# see dependent rules
end

# final recommendations -- identify the current season
rule "current season start" extends "RSV summarize season"
  ruleflow-group "rsv-recommendation"
  when
    # identify RSV recommendations that do not have a season start specified
eval($fact.getStartThisSeason() == null && $seasonStart != null)
  then
    modify($fact) { setStartThisSeason($seasonStart); }
  end

# final recommendations -- see if patient is on an RSV list for this season
rule "on rsv list this season" extends "RSV summarize season"
  ruleflow-group "rsv-recommendation"
  when
    # identify RSV recommendations that do not have a next dose attribute
eval($fact.getIsOnList() != true && $seasonStart != null)
    # find year for matching purposes
    $year: Integer() from $seasonStart.getYear()
    # find a list identifier related to RSV
    Identifier(type == Identifier.LIST_ID, $listid: id matches "(?ism).*\b(?:rsv|synagis|palivizumab)\b.*", id matches (.\b + $year.toString() + \b.*)) from $p.getIdentifiers()
  then
    modify($fact) { setIsOnList(true); }
  end

rule "show rsv only" extends "RSV summarize season"
  ruleflow-group "rsv-recommendation"
  when
    # identify RSV recommendations that are not flagged as "show RSV only"
eval($fact.getIsOnList() == true && $fact.getShowRsvOnly() != true)
    # find ID of current encounter
    Identifier(type == Identifier.ENCOUNTER_ID, $enc_id: id) from $p.getIdentifiers()
    # iterate over encounters
    $e: Encounter() from $p.getEncounters()
    # verify that this matches the current encounter id
    Identifier(type == Identifier.ENCOUNTER_ID, id == $enc_id) from $e.getIdentifiers()
    # determine if this is an orders only or care coordination type of encounter
    Identifier(type == Identifier.ENCOUNTER_TYPE, id matches "(?ism).*\b(?:orders\?|coordination)\b.*") from $e.getIdentifiers()
  then
    modify($fact) { setShowRsvOnly(true); }
    insert(new DocFact(DocFact.RSV_VISIBLE_ONLY));
  end

# final recommendations -- add next dose information
rule "next dose num" extends "RSV summarize season"
  ruleflow-group "rsv-recommendation"
when # identify RSV recommendations that do not have a next dose attribute eval($fact getNextDoseNum() == null && $givenDoses < Math.max($eligibleDoses, $approvedDoses))
then
  modify($fact) { setNextDoseNum($givenDoses + 1); }
end

rule "schedule this season" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when
eval($fact.getScheduleActualThisSeason() == null)
# collect actual schedule information for this season
$scheduleActual: ArrayList() from collect (RSVActualSchedule(seasonStart == $seasonStart))
then
  RSVActualSchedule scheduleThisSeason[] = new RSVActualSchedule[5];
  # organize the actual schedule for this season in order for convenience
  for (RSVActualSchedule schedule : (ArrayList<RSVActualSchedule>)$scheduleActual) {
    if(schedule.getDoseNum() <= 5) {
      scheduleThisSeason[schedule.getDoseNum() - 1] = schedule;
    }
  }
  modify($fact) { setScheduleActualThisSeason(new ArrayList(Arrays.asList(scheduleThisSeason))) }
end

# describe eligibility
rule "summarize eligible doses and reason this season" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when
eval($fact.getEligibleReasonThisSeason().equals("") && ! $eligibleReason.equals(""))
then
  modify($fact) {
    setEligibleDosesThisSeason($eligibleDoses),
    setEligibleReasonThisSeason($eligibleReason),
    setEligibleReasonBriefThisSeason($eligibleReasonBrief)
  }
end

# summarize season if not eligible and not approved
rule "summarize not eligible" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when
eval($eligibleDoses == 0 && $approvedDoses == 0 && $fact.getSummaryStatement().equals("") && !$eligibleReason.equals(""))
then
  modify($fact) { setSummaryStatement($eligibleReason); }
end

# summarize season if not eligible and not approved
rule "summarize not eligible but on list" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when
    eval($eligibleDoses == 0 && $approvedDoses == 0 &&
        !$fact.getSummaryStatement().equals("") && $fact.getIsOnList() == true &&
        !$fact.getSummaryStatement().contains("added to list"))
then
    modify($fact) { setSummaryStatement($fact.getSummaryStatement() + "<br />
        Manually added to list"); }
end

# summarize season if eligible but not approved
rule "summarize eligibility pre-season" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when
    eval($eligibleDoses > 0 && $approvedDoses == 0 && $evalDate.isBefore($seasonStart) &&
        $fact.getSummaryStatement().equals(""))
then
    modify($fact) {
        setSummaryStatement("Eligible for "+ $eligibleDoses + " doses of Palivizumab: " + $eligibleReasonBrief),
        setFlag(true)
    }
end

# summarize season if approved
rule "summarize approval pre-season" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when
    eval($approvedDoses > 0 && $evalDate.isBefore($seasonStart) &&
        $fact.getSummaryStatement().equals(""))
then
    modify($fact) { setSummaryStatement("Approved for "+ $approvedDoses + " doses of Palivizumab"); }
end

# summarize season if eligible but not approved
rule "summarize eligibility in-season give dose today" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when
    # verify that more doses can be given
    eval($givenDoses < $eligibleDoses && $approvedDoses == 0 &&
        $evalDate.isBefore($seasonStart) &&
        $fact.getSummaryStatement().equals(""))
    # check to see if next dose could be given today
    RSVActualSchedule(seasonStart == $seasonStart, doseNum == ($givenDoses + 1),
        eval(!interval.getStart().isAfter($evalDate)))
then
    modify($fact) {
        setSummaryStatement("Give Palivizumab dose #" + ($givenDoses + 1) + " today.
            Eligible for " + $eligibleDoses + " doses: " + $eligibleReasonBrief),
            setFlag(true)
    }
end

rule "summarize eligibility in-season give dose in future" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when
    # verify that more doses can be given
    eval($givenDoses < $eligibleDoses && $approvedDoses == 0 && !
    $evalDate.isBefore($seasonStart) && $fact.getSummaryStatement().equals(""))
    # check to see if next dose should be given in the future
    RSVActualSchedule(seasonStart == $seasonStart, doseNum == ($givenDoses + 1),
    $idealDate: date, eval(interval.getStart().isAfter($evalDate)))
    then
        modify($fact) {
            setSummaryStatement("Give Palivizumab dose #" + ($givenDoses + 1) + " " +
            FuzzyDate.fuzzyDate($idealDate, $evalDate) + ". Eligible for " + $eligibleDoses + " doses:"
            + $eligibleReasonBrief)
        }
    end

# describe the situation where more doses are indicated or approved, but no schedule 
information exists (e.g. if a 6th dose has been approved)
rule "summarize eligibility in-season future doses but no schedule" extends "RSV summarize season"
    ruleflow-group "rsv-recommendation"
    when
        # verify that more doses can be given based on either eligibility or approval
        eval($givenDoses < $eligibleDoses && $approvedDoses == 0 && !
        $evalDate.isBefore($seasonStart) && $fact.getSummaryStatement().equals(""))
        # verify that no schedule exists to recommend this extra dose
        not RSVActualSchedule(seasonStart == $seasonStart, doseNum == ($givenDoses + 1))
        then
            modify($fact) {
                setSummaryStatement("Eligible for " + $eligibleDoses + " doses of Palivizumab."
                Has received " + $givenDoses + " doses.")
            }
        end

# summarize season if eligible but not approved
rule "summarize approval in-season give dose today" extends "RSV summarize season"
    ruleflow-group "rsv-recommendation"
    when
        # verify that more doses can be given
        eval($givenDoses < $approvedDoses && !$evalDate.isBefore($seasonStart) &&
        $fact.getSummaryStatement().equals(""))
        # check to see if next dose could be given today
        RSVActualSchedule(seasonStart == $seasonStart, doseNum == ($givenDoses + 1),
        eval(!interval.getStart().isAfter($evalDate)))
        then
            modify($fact) {
                setSummaryStatement("Give Palivizumab dose #" + ($givenDoses + 1) + " today."
                Approved for " + $approvedDoses + " doses.")
                setFlag(true)
            }
        end

rule "summarize approval in-season give dose in future" extends "RSV summarize season"
    ruleflow-group "rsv-recommendation"
    when
# verify that more doses can be given

\[
\text{eval}($\text{givenDoses} < \text{approvedDoses} \&\& \lnot \text{evalDate.isBefore($seasonStart)} \&\& $\text{fact.getSummaryStatement}().\text{equals("")})
\]

# check to see if next dose should be given in the future

\[
\text{RSVActualSchedule}(\text{seasonStart} == $\text{seasonStart}, \text{doseNum} == ($\text{givenDoses} + 1), \text{idealDate}: \text{date}, \text{eval(interval.getStart().isAfter($evalDate)))}
\]

then

modify($\text{fact}) {
    \text{setSummaryStatement("Give Palivizumab dose "} + ($\text{givenDoses} + 1) + "). Approved for "} + $\text{approvedDoses + " doses.")}
}
end

# describe the situation where more doses are indicated or approved, but no schedule information exists (e.g. if a 6th dose has been approved)
rule "summarize approval in-season future doses but no schedule" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when

# verify that more doses can be given based on either eligibility or approval
\[
\text{eval}($\text{givenDoses} < \text{approvedDoses} \&\& \lnot \text{evalDate.isBefore($seasonStart)} \&\& $\text{fact.getSummaryStatement}().\text{equals("")})
\]

# verify that no schedule exists to recommend this extra dose
not RSVActualSchedule($\text{seasonStart} == $\text{seasonStart}, \text{doseNum} == ($\text{givenDoses} + 1))
then

modify($\text{fact}) {
    \text{setSummaryStatement("Approved for "} + $\text{approvedDoses + " doses of Palivizumab. Has received "} + $\text{givenDoses + " doses.")}
}
end

rule "summarize in-season no further doses" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when

# verify that no more doses are indicated
\[
\text{eval}((($\text{givenDoses} >= $\text{eligibleDoses} \&\& $\text{eligibleDoses} > 0 \&\& $\text{approvedDoses} == 0 \lor (\text{givenDoses} >= $\text{approvedDoses} \&\& $\text{approvedDoses} > 0) \&\& $\text{evalDate.isBefore($seasonStart)} \&\& $\text{fact.getSummaryStatement}().\text{equals("")})
\]

then

modify($\text{fact}) {
    \text{setSummaryStatement("No further doses of Palivizumab indicated this season.")}
}
end

# provide chart review details
rule "chart review details" extends "RSV summarize season"
ruleflow-group "rsv-recommendation"
when

eval($\text{fact.getChartReviewQuery}().\text{equals("")})
then

modify($\text{fact}) { $\text{setChartReviewQuery($s.queryString($p));}}
end
rule "estimate weight for future doses this season"
  ruleflow-group "final-pass"
  when
    # identify available predicted weights
    DateFact(attribute == DateFact.PREDICT_MEASUREMENT, value == VitalSign.WEIGHT_KG, $weightDate: date, $weightKG: dblValue != null)
    # identify scheduled doses this season that need a predicted weight
    $fact: RSVActualSchedule(status != "Given", date == $weightDate, projectedWeightKG == null)
    then
      modify($fact) {
        setProjectedWeightKG($weightKG),
        setProjectedWeight(String.format("%.3f", $weightKG) + " kg")
      }
  end
rule "estimate weight for next dose"
  ruleflow-group "final-pass"
  when
    $fact: RSVRecommendation($seasonStart: startThisSeason, $q: chartReviewQuery != ", chartReviewQuery not matches ".*p_weight.*", $nextDoseNum: nextDoseNum != null)
    # identify projected weight
    RSVActualSchedule($seasonStart == $seasonStart, $nextDate: date, doseNum == $nextDoseNum, $projectedWeight: projectedWeight != null)
    then
      modify($fact) { setChartReviewQuery($q + "+p_weight=" + $projectedWeight + " (" + FuzzyDate.monthDayYearBrief($nextDate) + ")")}; }
  end
rule "chart review add current weight"
  ruleflow-group "final-pass"
  when
    $fact: RSVRecommendation($q: chartReviewQuery != ", chartReviewQuery not matches ".*l_weight.*")
    # determine last weight
    DateFact(attribute == DocFact.CURRENT_WEIGHT, $weightDate: date != null, $weightKG: dblValue != null)
    then
      modify($fact) { setChartReviewQuery($q + "&l_weight=" + String.format("%.3f", $weightKG) + " kg (" + FuzzyDate.monthDayYearBrief($weightDate) + ")")}; }
  end
query "Recommendations"
  recommendation: RSVRecommendation();
end