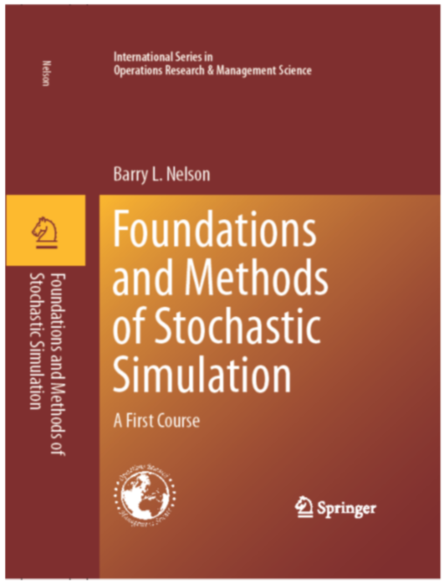

Chapter 2.3.4 & 4: VBASim

©Barry L. Nelson

Northwestern University

December 2012

Object orientation

- Discrete-event simulations often contain multiple instances of similar objects:
  - Entities: things that come and go, like customers, messages, jobs, events
  - Queues: ordered lists of entities
  - Resources: scarce quantities that entities need, like servers, computers, machines
  - Statistics: information to be recorded on all of the above
- These are more naturally treated as “objects” for which we can have many instances.

VBASim

- A small collection of VBA class modules (objects) that support simulation.
  - You can easily customize and add to these
- A module containing declarations and a few useful subs.
- VBA implementations of the random-number and random-variate generation functions from simlib by Law & Kelton.

VBA class modules

- With VBA class modules we can define the template for an object.
- A Class Module contains…
  - Properties → “attributes” in simulation terminology
  - Methods → instructions about how to do things
- The key benefit of objects is that we can create multiple instances, each uniquely identified and with its own properties and methods.

```
' Generic continuous-time statistics object
' Note that CTStat should be called AFTER the value of the variable changes
Private Area As Double
Private Tlast As Double
Private Xlast As Double
Private TClear As Double
```

Properties: each CTStat will have its own copy

```
Private Sub Class_Initialize()
' Executes when CTStat object is created to initialize variables
    Area = 0
    Tlast = 0
    TClear = 0
    Xlast = 0
End Sub
```

Automatically called when a New CTStat is created

```
Public Sub Record(X As Double)
' Update the CTStat from last time change and keep track of previous value
    Area = Area + Xlast * (Clock - Tlast)
    Tlast = Clock
    Xlast = X
End Sub
```

Called from within your simulation to update the statistic

```
Function Mean() As Double
' Return the sample mean up through current time but do not update
    Mean = 0
    If (Clock - TClear) > 0 Then
        Mean = (Area + Xlast * (Clock - Tlast)) / (Clock - TClear)
    End If
End Function
```

Called from within your simulation to report the sample mean

```
Public Sub Clear()
' Clear statistics
    Area = 0
    Tlast = Clock
    TClear = Clock
End Sub
```

Called from within the simulation to reset the CTStat

CTStat object

Anatomy of a class module

- Declarations
  - Defines the attributes (Properties) the object has
  - **Private** if only used within the object; otherwise **Public**
  - Each instance of the object will have its own unique copy

```
' Generic continuous-time statistics object
' Note that CTStat should be called AFTER the value of the variable changes
Private Area As Double
Private Tlast As Double
Private Xlast As Double
Private TClear As Double
```

Anatomy of a class module

- Methods can be Subs, Functions, Property Let and Property Get
- To the best of my ability to tell, Property Get is the same as a Function

```
Dim QueueLength As New CTStat
QueueLength.Record(Q)
Xbar = QueueLength.Mean
```

```
Public Sub Record(X As Double)
' Update the CTStat from last time change and keep track of previous value
    Area = Area + Xlast * (Clock - Tlast)
    Tlast = Clock
    Xlast = X
End Sub

Function Mean() As Double
' Return the sample mean up through current time but do not update
    Mean = 0
    If (Clock - TClear) > 0 Then
        Mean = (Area + Xlast * (Clock - Tlast)) / (Clock - TClear)
    End If
End Function
```

Anatomy of a class module

- There are special methods that will be executed when an object is created or destroyed.
- To release the object name
  Set QueueLength = **Nothing**

```
Private Sub Class_Initialize()
' Executes when CTStat object is created to initialize variables
    Area = 0
    Tlast = 0
    TClear = 0
    Xlast = 0
End Sub


Private Sub Class_Terminate()
' Termination code goes here…
End Sub
```

VBA Collections

- A Collection is a VBA generalization of an array; it can store objects of the same class.
  - VBA itself uses lots of collections
  - Worksheets(“Sheet2”) or Worksheets(2)
- We use these in the event calendar management and queue management.

Collection syntax

`Dim` `Queue` **`As New Collection`**

`Queue.`**`Item(i)`** ← the object in position i

`Queue.`**`Count`** ← number of objects currently in the Queue collection

`Queue.`**`Remove(j)`** ← remove the object in position j of the Queue collection

`Queue.`**`Add`** `customer,` **`Before:=`**k ← insert object customer before object currently in position k (also has option **after**)

VBASim EventCalendar Class Module

```
' This class module creates an Event Calendar object

Private ThisCalendar As New Collection

Public Sub Schedule(addedEvent As EventNotice)
' Add EventNotice in EventTime order
    Dim i As Integer
    If ThisCalendar.Count = 0 Then 'no events in calendar
        ThisCalendar.Add addedEvent
    ElseIf ThisCalendar(ThisCalendar.Count).EventTime <= addedEvent.EventTime Then
                                'added event after last event in calendar
        ThisCalendar.Add addedEvent, After:=ThisCalendar.Count
    Else                        'search for the correct place to insert the event
        For i = 1 To ThisCalendar.Count
            If ThisCalendar(i).EventTime > addedEvent.EventTime Then
                Exit For
            End If
        Next i
        ThisCalendar.Add addedEvent, before:=i
    End If
End Sub

Public Function Remove() As EventNotice
' Remove next event and return the EventNotice object
    If ThisCalendar.Count > 0 Then
        Set Remove = ThisCalendar.Item(1)
        ThisCalendar.Remove (1)
    End If
End Function

Function N() As Integer
' Return current number of events on the event calendar
    N = ThisCalendar.Count
End Function
```

Note: EventNotice is also a Class Module with two attributes: EventTime and EventType

M/G/1 Queue in VBASim

```
' Example illustrating use of VBASim for
' simulation of M/G/1 Queue.

' See VBASim module for generic declarations
' See Class Modules for the supporting VBASim classes

' Parameters we may want to change
Public MeanTBA As Double          ' mean time between arrivals
Public MeanST As Double           ' mean service time
Public Phases As Integer          ' number of phases in service distribution
Public RunLength As Double        ' run length
Public WarmUp As Double           ' "warm-up" time

' Global objects needed for simulation
' These will usually be queues and statistics

Dim Queue As New FIFOQueue   'customer queue
Dim Wait As New DTStat       'discrete-time statistics on customer waiting
Dim Server As New Resource   'server resource
```

```
Public Sub MG1()
    Dim Reps As Integer
    Dim NextEvent As EventNotice

    Call MyInit ' special initializations for this simulation
    For Reps = 1 To 10
        Call VBASimInit 'initialize VBASim for each replication
        Call Schedule("Arrival", Expon(MeanTBA, 1))
        Call Schedule("EndSimulation", RunLength)
        Call Schedule("ClearIt", WarmUp)
        Do
            Set NextEvent = Calendar.Remove
            Clock = NextEvent.EventTime
            Select Case NextEvent.EventType
            Case "Arrival"
                Call Arrival
            Case "EndOfService"
                Call EndOfService
            Case "ClearIt"
                Call ClearStats
            End Select
        Loop Until NextEvent.EventType = "EndSimulation"

' Write output report for each replication
        Call Report(Wait.Mean, "MG1", Reps + 1, 1)
        Call Report(Queue.Mean, "MG1", Reps + 1, 2)
        Call Report(Queue.NumQueue, "MG1", Reps + 1, 3)
        Call Report(Server.Mean, "MG1", Reps + 1, 4)
    Next Reps
    End ' ends execution, closes files, etc.
End Sub
```

```
Public Sub MyInit()

' Initialize the simulation
    Call InitializeRNSeed
    Server.SetUnits (1) ' set the number of servers to 1
    MeanTBA = 1
    MeanST = 0.8
    Phases = 3
    RunLength = 55000
    WarmUp = 5000

' Add queues, resources and statistics that need to be
' initialized between replications to the global collections

    TheDTStats.Add Wait
    TheQueues.Add Queue
    TheResources.Add Server

' Write headings for the output reports

    Call Report("Average Wait", "MG1", 1, 1)
    Call Report("Average Number in Queue", "MG1", 1, 2)
    Call Report("Number Remaining in Queue", "MG1", 1, 3)
    Call Report("Server Utilization", "MG1", 1, 4)

End Sub
```

VBASim will reinitialize any objects in these collections between replications; there is also a The CTStats collection.

```
Public Sub Arrival()
' Arrival event

' Schedule next arrival
    Call Schedule("Arrival", Expon(MeanTBA, 1))

' Process the newly arriving customer

    Dim Customer As New Entity
    Queue.Add Customer
    Set Customer = Nothing
' If server is not busy, start service by seizing the server

    If Server.Busy = 0 Then
        Server.Seize (1)
        Call Schedule("EndOfService", Erlang(Phases, MeanST, 2))
    End If

End Sub
```

Note that we dim a NEW Entity; "New" means not only declare, but also create

Seize is VBASim for "make busy this many units of the resource"

Without “New” this is only a declaration

```
Public Sub EndOfService()
' End of service event

' Remove departing customer from queue and record wait time

    Dim DepartingCustomer As Entity
    Set DepartingCustomer = Queue.Remove
    Wait.Record (Clock - DepartingCustomer.CreateTime)
    Set DepartingCustomer = Nothing    'be sure to free up memory

' Check to see if there is another customer; if yes start service
' otherwise free the server

    If Queue.NumQueue > 0 Then
        Call Schedule("EndOfService", Erlang(Phases, MeanST, 2))
    Else
        Server.Free (1)
    End If

End Sub
```

How did this get set?

Free is VBASim for “make idle this many units of the resource”

Using VBASim

- VBASim Module
  - Declarations
  - Subs: VBASimInit, Schedule, SchedulePlus, Report
- VBASim Class Modules
  - CTStat, DTStat
  - Entity
  - EventCalendar, EventNotice
  - FIFOQueue
  - Resource
- Changing and adding to VBASim

VBASim module: Delcarations

```
Public Clock As Double                        'simulation global clock
Public Calendar As New EventCalendar          'event calendar

' Set up Collections to be reinitialized between replications
Public TheCTStats As New Collection    ' continuous-time statistics
Public TheDTStats As New Collection    ' discrete-time statistics
Public TheQueues As New Collection     ' queues
Public TheResources As New Collection ' resources
```

- Everything in VBASim is “Public” so that it can be used from any module in the Workbook.
- The [...] are collections of VBASim objects that will be reinitialized whenever `VBASimInit` is called.

VBASimInit

- Usage: `Call VBASimInit`
- Typically placed **inside** the replication loop
- Resets the `Clock, Calendar`, and all of `The[…]` collections

Schedule, SchedulePlus & Report

```
Public Sub Schedule(EventType As String, EventTime As Double)

Public Sub SchedulePlus(EventType As String, EventTime As
Double, TheObject as Object)

Public Sub Report(Output As Variant, WhichSheet As String, Row
As Integer, Column As Integer)
```

- Usage:

```
Call Schedule("Arrival", Expon(2,1))

Dim Customer as New Entity
Call SchedulePlus("Arrival", Expon(2,1), Customer)

Call Report(Queue.Mean, "Sheet2", 3, 5)
```

- Notice that EventTime is how far into the future the event is to occur, not the absolute time.
- The "Plus" version allows another object (usually an Entity) to be attached to the EventNotice.

Entity class module

- Usage
  ```
  Dim Customer as New Entity

  Delay = Clock – Customer.CreateTime
  ```
- You can add as many additional attributes as you need the entities to have to the Entity Class Module.

```
' This is a generic entity that has a single attribute CreateTime

Public CreateTime As Double

' Add additional problem specific attributes here


Private Sub Class_Initialize()
' Executes when Entity object is created to initialize variables
       CreateTime = Clock
End Sub
```

EventNotice class module

- Usage

```
Dim NextEvent as EventNotice

Set NextEvent = Calendar.Remove

Clock = NextEvent.EventTime
Select Case NextEvent.EventType

     Call EOS(NextEvent.WhichObject)
```

- The EventNotices are usually created by Schedule or SchedulePlus; you use them when advancing to the next event.

```
' This is a generic EventNotice object with EventTime, EventType
‘ and WhichObject attributes

Public EventTime As Double
Public EventType As String
Public WhichObject As Object

' Add additional problem specific attributes here
```

About the other class modules

- You are unlikely to modify the other class modules (although you may create your own variations using them as templates).
- The most important thing is to know how to use them.
- Remember:
  When you `Dim X as New Object`, a **pointer** is created to that (perhaps very complex) object. That pointer needs to be retained, either in a specific name (e.g., `TicketQueueStatistic`) or stored in a collection or else the object is lost.

CTStat

- Collects continuous-time statistics
- Methods: Record, Mean and Clear
- Usage

```
Dim TotalCustomerStats as New CTStat

TotalCustomerStats.Record(NumCust)

Call Report(TotalCustomerStats.Mean,“Sheet1”, 1,2)

TotalCustomerStats.Clear
```

Call AFTER the value has changed

DTStat

- Collects discrete-time statistics
- Methods: Record, Mean, StdDev, N and Clear
- Usage

```
Dim Wait as New DTStat

Wait.Record(Clock - Customer.CreateTime)

Call Report(Wait.Mean,“Sheet1”, 1,2)
Call Report(Wait.StdDev,“Sheet1”, 1,3)
Call Report(Wait.N, “Sheet1”, 1,4)

Wait.Clear
```

Resource

- Models resources and also keeps a CTStat on average number in use
- Properties: Busy [current number in use]
- Methods: SetUnits, Seize, Free, Mean
- Usage

```
Dim Server as New Resource

Server.SetUnits(5)        ‘ resource has capacity 5
Server.Seize(1)           ‘ make busy 1 unit of resource
Server.Free(1)            ‘ make idle 1 unit of resource

If Server.Busy = 5 Then …

Call Report(Server.Mean, “Sheet1”, 5,4)
```

FIFOQueue

- Models first-in-first-out queue, and also keeps a CTStat on average number in queue
- Methods: NumQueue, Add, Remove, Mean
- Usage

```
Dim Line as New FIFOQueue

Dim Shopper as New Entity
Line.Add Shopper

Dim DepartingShopper as Entity
Set DepartingShopper = Line.Remove

If Line.NumQueue = 0 Then…

Call Report(Line.Mean, “Sheet1”, 5,10)
```

Some notes…

- The CTStat’s created by `FIFOQueue` and `Resource` are automatically added to `TheCTStats` collection, so they are reinitialized by `VBASimInit`.
- The most common change you will make is to add attributes to the Entity class.
- VBASim currently does not have a lot of error checking.

A note on creating new objects

- Consider the following code

```
Dim Queue as New FIFOQueue
Dim Customer as New Entity
Customer.SomeAttribute = 10
Queue.Add Customer
Dim Customer as New Entity
Customer.SomeAttribute = 11
Queue.Add Customer
```

- Surprisingly, this code puts 2 of the same entity (both with `SomeAttribute = 11`) in the Queue.
- This is because `Dim`…New only creates a new object if the target pointer variable (Customer here) is currently unassigned.

- Correct approach:
```
Dim Queue as New FIFOQueue
Dim Customer as New Entity
Customer.SomeAttribute = 10
Queue.Add Customer
Set Customer = Nothing
Dim Customer as New Entity
Customer.SomeAttribute = 11
Queue.Add Customer
Set Customer = Nothing
```
- Note that the `Customer` is not lost, because it has been placed in the `Queue` (a collection); that is, its reference is being maintained in another way.
- When `Dim`…`New` encounters an unassigned pointer variable it creates a new object.

Using RNG

- `Call InitializeRNSeed()`
  - Call **once** at the beginning of the simulation to initialize the pseudorandom-number generator
- `lcgrand(Stream)`
  - Pseudorandom-number generator
  - Streams 1-100
- `Expon, Erlang, Random_integer, Normal, Lognormal, Triangular`
  - Arguments are distribution parameters first, with last argument being the stream number 1-100
  - Ex: `Expon(15.2, 7)`

M/G/5 Queue

- What would we have to change to make this a single waiting line, but multiple server queue?
- Let’s modify the M/G/1 code…

Changes in Sub MG1

```
Do
      Set NextEvent = Calendar.Remove
      Clock = NextEvent.EventTime
      Select Case NextEvent.EventType
      Case "Arrival"
          Call Arrival
      Case "EndOfService"
          Call EndOfService(NextEvent.WhichObject)
      Case "ClearIt"
          Call ClearStats
      End Select
Loop Until NextEvent.EventType = "EndSimulation"
```

The key difference is that the Queue will now only contain the customers waiting for service, but not those in service. The ones in service will be passed along with the Event Notice.

Changes in Sub Arrival

```
Sub Arrival()
' Arrival event
' Schedule next arrival
    Call Schedule("Arrival", Expon(MeanTBA, 1))
' Process the newly arriving customer
    Dim Customer As New Entity
' If server is not busy, start service by seizing the server
    If Server.Busy < NumServers Then
        Server.Seize (1)
        Call SchedulePlus("EndOfService", _
                    Erlang(Phases, MeanST, 2), Customer)
    Else
        Queue.Add Customer
    End If
    Set Customer = Nothing
End Sub
```

Changes in Sub EndOfService

```
Sub EndOfService(DepartingCustomer As Entity)
' End of service event
' record wait time of departing customer
    Wait.Record (Clock - DepartingCustomer.CreateTime)
    Set DepartingCustomer = Nothing
' Check to see if there is another customer;
' if yes start service otherwise free the server
    If Queue.NumQueue > 0 Then
        Dim NextCustomer As Entity
        Set NextCustomer = Queue.Remove
        Call SchedulePlus("EndOfService", _
              Erlang(Phases, MeanST, 2), NextCustomer)
        Set NextCustomer = Nothing
    Else
        Server.Free (1)
    End If

End Sub
```

Changes in Sub MyInit

```
NumServers = 5
Server.SetUnits (NumServers) ' set the number of servers
```

Written in this way, the simulation can look at any number of servers simply by changing one line of code.

Fax Center Simulation

```
' Parameters we may want to change

Dim MeanRegular As Double       ' mean entry time regular faxes
Dim VarRegular As Double        ' variance entry time regular faxes
Dim MeanSpecial As Double       ' mean entry time special faxes
Dim VarSpecial As Double        ' variance entry time special faxes
Dim RunLength As Double         ' length of the working day
Dim NumAgents As Integer        ' number of regular agents
Dim NumSpecialists As Integer   ' number of special agents
Dim NumAgentsPM As Integer      ' number of regular agents after noon
Dim NumSpecialistsPM As Integer ' number of special agents after noon


' Global objects needed for simulation

Dim RegularQ As New FIFOQueue       ' queue for all faxes
Dim SpecialQ As New FIFOQueue       ' queue for special faxes
Dim RegularWait As New DTStat       ' discrete-time statistics on fax waiting
Dim SpecialWait As New DTStat       ' discrete-time statistics on special fax waiting
Dim Regular10 As New DTStat         ' discrete-time statistics on < 10 minutes threshold
Dim Special10 As New DTStat         ' discrete-time statistics on < 10 minutes threshold
Dim Agents As New Resource          ' entry agents resource
Dim Specialists As New Resource     ' specialists resource
Dim ARate(1 To 8) As Double         ' arrival rates
Dim MaxRate As Double               ' maximum arrival rate
Dim Period As Double                ' period for which arrival rate stays constant
Dim NPeriods As Integer             ' number of periods in a "day"
```

```
Public Sub FaxCenterSim()
    Dim Reps As Integer
    Dim NextEvent As EventNotice
    ' Read in staffing policy
    NumAgents = Worksheets("Fax").Cells(25, 5)
    NumAgentsPM = Worksheets("Fax").Cells(25, 6)
    NumSpecialists = Worksheets("Fax").Cells(26, 5)
    NumSpecialistsPM = Worksheets("Fax").Cells(26, 6)
    Call MyInit

    For Reps = 1 To 10
        Call VBASimInit
        Agents.SetUnits (NumAgents)
        Specialists.SetUnits (NumSpecialists)
        Call Schedule("Arrival", NSPP_Fax(ARate, MaxRate, NPeriods, Period, 1))
        Call Schedule("ChangeStaff", 4 * 60)
        Do
            Set NextEvent = Calendar.Remove
            Clock = NextEvent.EventTime
            Select Case NextEvent.EventType
            Case "Arrival"
                Call Arrival
            Case "EndOfEntry"
                Call EndOfEntry(NextEvent.WhichObject)
            Case "EndOfEntrySpecial"
                Call EndOfEntrySpecial(NextEvent.WhichObject)
            Case "ChangeStaff"
                Agents.SetUnits (NumAgentsPM)
                Specialists.SetUnits (NumSpecialistsPM)
            End Select
        Loop Until Calendar.N = 0 ' stop when event calendar empty
```

```
' Write output report for each replication

        Call Report(RegularWait.Mean, "Fax", Reps + 1, 1)
        Call Report(RegularQ.Mean, "Fax", Reps + 1, 2)
        Call Report(Agents.Mean, "Fax", Reps + 1, 3)
        Call Report(SpecialWait.Mean, "Fax", Reps + 1, 4)
        Call Report(SpecialQ.Mean, "Fax", Reps + 1, 5)
        Call Report(Specialists.Mean, "Fax", Reps + 1, 6)
        Call Report(Regular10.Mean, "Fax", Reps + 1, 7)
        Call Report(Special10.Mean, "Fax", Reps + 1, 8)
        Call Report(Clock, "Fax", Reps + 1, 9)
    Next Reps
    End
End Sub
```

Private Sub Arrival()

```
' Schedule next fax arrival if < 4 PM
    If Clock < RunLength Then
        Call Schedule("Arrival", NSPP_Fax(ARate, MaxRate, NPeriods, Period, 1))
    Else
        Exit Sub
    End If
' Process the newly arriving Fax
    Dim Fax As New Entity
    If Agents.Busy < Agents.NumberOfUnits Then
        Agents.Seize (1)
        Call SchedulePlus("EndOfEntry", Normal(MeanRegular, VarRegular, 2), Fax)
    Else
        RegularQ.Add Fax
    End If
    Set Fax = Nothing
End Sub
```

```
Private Sub EndOfEntry(DepartingFax As Entity)
    Dim Wait As Double

' Record wait time if regular; move on if special

    If Uniform(0, 1, 3) < 0.2 Then
        Call SpecialArrival(DepartingFax)
    Else
        Wait = Clock - DepartingFax.CreateTime
        RegularWait.Record (Wait)
        If Wait < 10 Then
            Regular10.Record (1)
        Else
            Regular10.Record (0)
        End If
    End If
    Set DepartingFax = Nothing

' Check to see if there is another Fax; if yes start entry
' otherwise free the agent

    If RegularQ.NumQueue > 0 And Agents.NumberOfUnits >= Agents.Busy Then
        Set DepartingFax = RegularQ.Remove
        Call SchedulePlus("EndOfEntry", Normal(MeanRegular, VarRegular, 2), DepartingFax)
        Set DepartingFax = Nothing
    Else
        Agents.Free (1)
    End If

End Sub
```

```
Private Sub SpecialArrival(SpecialFax As Entity)

' If special agent available, start entry by seizing the special agent

    If Specialists.Busy < Specialists.NumberOfUnits Then
        Specialists.Seize (1)
        Call SchedulePlus("EndOfEntrySpecial", Normal(MeanSpecial, VarSpecial, 4), SpecialFax)
    Else
        SpecialQ.Add SpecialFax
    End If
    Set SpecialFax = Nothing

End Sub
```

```
Private Sub EndOfEntrySpecial(DepartingFax As Entity)
    Dim Wait As Double

' Record wait time and indicator if < 10 minutes

    Wait = Clock - DepartingFax.CreateTime
    SpecialWait.Record (Wait)
    If Wait < 10 Then
        Special10.Record (1)
    Else
        Special10.Record (0)
    End If
    Set DepartingFax = Nothing

' Check to see if there is another Fax; if yes start entry
' otherwise free the specialist

    If SpecialQ.NumQueue > 0 And Specialists.NumberOfUnits >= Specialists.Busy Then
        Set DepartingFax = SpecialQ.Remove
        Call SchedulePlus("EndOfEntrySpecial", Normal(MeanSpecial, VarSpecial, 4), DepartingFax)
        Set DepartingFax = Nothing
    Else
        Specialists.Free (1)
    End If

End Sub
```

```
Private Sub MyInit()
' Initialize the simulation
    Call InitializeRNSeed
    MeanRegular = 2.5
    VarRegular = 1#
    MeanSpecial = 4
    VarSpecial = 1#
    RunLength = 480
' Add queues, resources and statistics that need to be
' initialized between replications to the global collections
    TheDTStats.Add RegularWait
    TheDTStats.Add SpecialWait
    TheDTStats.Add Regular10
    TheDTStats.Add Special10
    TheQueues.Add RegularQ
    TheQueues.Add SpecialQ
    TheResources.Add Agents
    TheResources.Add Specialists
' Write headings for the output reports
    Call Report("Ave Reg Wait", "Fax", 1, 1)
    Call Report("Ave Num Reg Q", "Fax", 1, 2)
    Call Report("Agents Busy", "Fax", 1, 3)
    Call Report("Ave Spec Wait", "Fax", 1, 4)
    Call Report("Ave Num Spec Q", "Fax", 1, 5)
    Call Report("Specialists Busy", "Fax", 1, 6)
    Call Report("Reg < 10", "Fax", 1, 7)
    Call Report("Spec < 10", "Fax", 1, 8)
    Call Report("End Time", "Fax", 1, 9)
' Arrival process data
    NPeriods = 8
    Period = 60
    MaxRate = 6.24
    ARate(1) = 4.37
    ARate(2) = 6.24
    ARate(3) = 5.29
    ARate(4) = 2.97
    ARate(5) = 2.03
    ARate(6) = 2.79
    ARate(7) = 2.36
    ARate(8) = 1.04
End Sub
```

```
Private Function NSPP_Fax(ARate() As Double, MaxRate As Double, NPeriods As Integer, _
                Period As Double, Stream As Integer) As Double
' This function generates interarrival times from a NSPP with piecewise constant
' arrival rate over a fixed time of Period*NPeriod time units

' ARate = array of arrival rates over a common length Period
' MaxRate = maximum value of ARate
' Period = time units between (possible) changes in arrival rate
' NPeriods = number of time periods in ARate

Dim i As Integer
Dim PossibleArrival As Double

PossibleArrival = Clock + Expon(1 / MaxRate, Stream)
i = WorksheetFunction.Min(NPeriods, WorksheetFunction.Ceiling(PossibleArrival / Period, 1))

Do Until Uniform(0, 1, Stream) < ARate(i) / MaxRate
    PossibleArrival = PossibleArrival + Expon(1 / MaxRate, Stream)
    i = WorksheetFunction.Min(NPeriods, WorksheetFunction.Ceiling(PossibleArrival / Period, 1))
Loop

NSPP_Fax = PossibleArrival - Clock

End Function
```