OpenDSS COM Documentation

CtrlQueue Interface

Sept 2012
Updated Mar 2015

The CtrlQueue interface contains properties and methods for interactions between internal DSS control objects and control algorithms written in programs driving the OpenDSS program through its interface.

DSS Control Queue

The OpenDSS program has an internal control dispatching process implemented as a queue of action requests at specified times. This control queue is primarily for managing *discrete* controls such as tap-changing voltage regulators and switched capacitors that can have delayed actions. For example, when a voltage regulator goes out of band, it pushes a message on the Control Queue with a requested time for a tap-changing action. This simulates starting the timer in the control. When the requested time is reached, the message is sent back to the regulator, which decides at that time whether or not it really needs to change taps. This simulates the timer expiring and putting a signal on an AND gate with the out-of-band signal, which is the way some regulator controls work.

Continuous controls generally act directly on variables within the program and any delay is accomplished through time integration. Continuous controls are primarily used in dynamics simulations while discrete controls are mainly for sequential power flow simulations. They are often implemented by user-written DLLs.

The Control Queue gets populated after a converged solution is achieved. All of the control objects currently enabled in the circuit are polled. If they determine that they need to make a delayed change, they will push a control action onto the Control Queue using codes that are unique to each class of control object. When it comes time to execute the action, the OpenDSS pops the appropriate control actions off the Control Queue and dispatches each of them to the appropriate control handler (a virtual function called *DoPendingAction* in the module that pushed the action code onto the control queue).

The *CtrlQueue* COM interface instantiates a proxy that provides a *DoPendingAction* function to the OpenDSS and, thus imitates an internal OpenDSS control object.

COM Interface Control Proxy

The COM interface contains a proxy for any control objects that might be implemented in some program external to the OpenDSS and driving the OpenDSS through the COM interface. The COM Control Proxy contains an Action List that contains a list of actions that have been popped off the OpenDSS Control Queue (see Figure 1) and sent back to the module that pushed the message onto the control queue. User-written control algorithms should check this List and execute the requested action, if it is still appropriate to do so.

The Action List dispatches control requests using a Device Handle and an Action Code. Controls implemented in external languages via the COM interface should be associated with a particular user-assigned Device Handle for dispatching by the COM Control Proxy object. At this time, Device Handles and Action Codes are defined in the external user-written code, but a future version may also employ handles and codes managed by the simulator.

Typically, an external control would step through the solution one step at a time. After achieving a converged circuit solution, the control would execute the functions that sample the quantities of interest. It could then *Push* any control actions onto the OpenDSS Control Queue and rely on the OpenDSS to dispatch the control. The OpenDSS would know the action request came from the COM Control Proxy and dispatch any actions back to the Proxy where they are accumulated in the Action Queue. The user-written control would then be responsible for checking the Action List, setting the Active Action (directly or by popping off the list in sequence), and then dispatching the action code to the proper control algorithm.

All discrete external controls with time-delayed actions should employ the OpenDSS Control Queue for proper synchronization with other active controllers in the circuit. When the time comes to execute an action, external controls may act directly upon circuit objects through the COM interface. For example, if a user-written relay control determines that a device terminal needs to be opened, it can do so through either the text interface or the CktElement interface.

Note that accessing most of these properties and methods without an active circuit defined will result in no action.

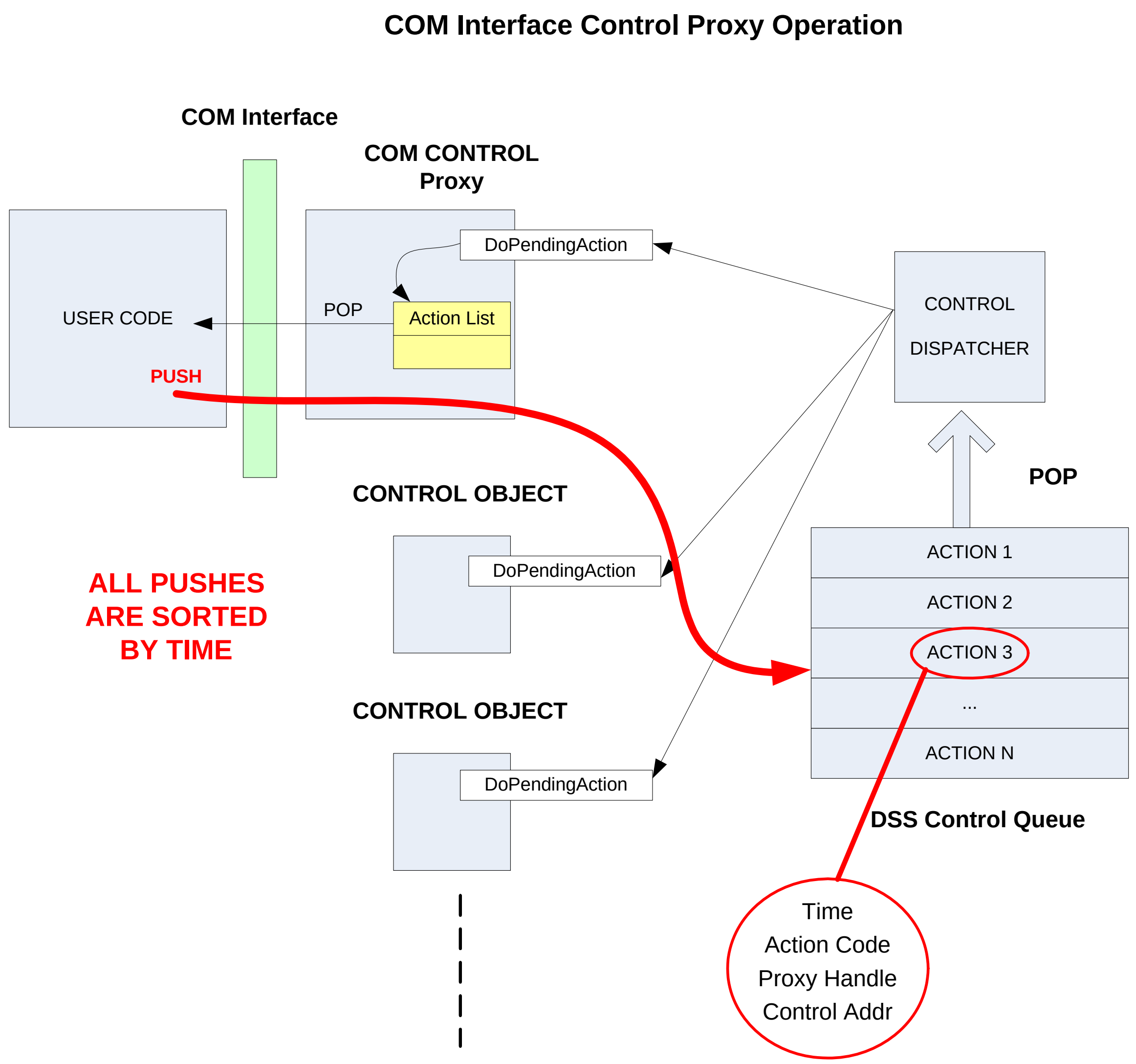

Figure 1. Illustration of how the COM Control Proxy interacts with the DSS Control Queue

Properties

NumActions:Integer (Read only)

Returns the number of actions in the COM Control Proxy Action List. Check this value to see if the OpenDSS control queue has popped any actions off the list that are destined for the user-written control.

Action(Index: Integer): (Write Only)

One way to set which of the Actions in the Action List is the *Active Action*. Sets it by index value after checking to make sure that Index is within the valid range (based on NumActions). This is a **zero-based** index - the first element in the Action List is index 0.

ActionCode: Integer: (Read Only)

Action Code for the Active Action in the COM Control Proxy Action List. Use this to determine what the user-defined controls are supposed to do. Can be any long (32-bit) integer of the programmer's choosing and is the same value that the control pushed onto the control queue earlier.

DeviceHandle:Integer (Read only)

Returns a handle to the user's control device from the Active Action. The user-written code driving the COM interface may support more than one control element as necessary to perform the simulation. This handle is an index returned to the user program that lets the program know which control is to perform the active action.

PopAction: Integer (Read Only)

Sets the Active Action by popping the next action off the COM Control Proxy action list. Returns zero when there are no more actions to pop.

Methods

Function ClearQueue: HResult;

This clears all actions from the DSS Control Queue. The return value is always zero.

Function Delete (ActionHandle: Integer): HResult

Deletes an Action from the DSS Control Queue by the handle that is returned when the action is added. (The Push function returns the handle.) The return value is always zero.

Function Push (Hour: Integer; Seconds: Double; ActionCode: Integer; DeviceHandle: Integer): HResult;

Pushes an Action onto the DSS Control Queue. The return value is the handle to the action on the queue. Specify the time of the requested action in Hour, Seconds. Push an ActionCode that is meaningful to the user-written control object. *DeviceHandle* is a user-defined handle (a 32-bit Integer) to the control object that will be used to dispatch the control action to the proper control device managed by the user-written code when the Action is popped off the Control Queue. The DSS will automatically return this handle to the control proxy in the COM interface when the action is popped.

The actual internal DSS Control Queue *Push* function passes one more argument not shown in this COM interface method. For internal control objects, it is the *Self* variable; for any control model coming through the COM interface, it is the address of the *COMControlProxyObj* variable. When an action destined for the user-written code is popped off the DSS Control Queue, the *COMControlProxyObj.DoPendingAction* function is called. This dispatches the action message to the Action List in the user-written code. Then the user-written code must decipher it.

Function Show: HResult;

This function executes the procedure inside the OpenDSS to show a text file in CSV form containing the present contents of the DSS Control Queue. The return value is always zero.

When this command is complete the Result property of the Text interface contains the name of the file, which you may use for further processing. (The Result property must be accessed before doing anything else or it will disappear.)

Function ClearActions: HResult;

This clears all actions from the Control Proxy's Action List (they are popped off the list). The return value is always zero.

Example 1

The following MATLAB snippet demonstrates how to execute a single solution by performing the solution and control sampling separately. After polling all the control elements (SampleControlDevices) the contents of the CtrlQueue are displayed. This example uses the IEEE 123 bus test feeder.

```
[DSSStartOK, DSSObj, DSSText] = DSSStartup;
if DSSStartOK
    DSSText.command='Compile (C:\opendss\IEEETestCases\123Bus\IEEE123Master.dss)';

    % Set up the interface variables
    DSSCircuit=DSSObj.ActiveCircuit;
    DSSSolution=DSSCircuit.Solution;

    DSSText.Command='Set MaxControlIter=30';

    DSSSolution.SolveNoControl; % solve power flow with no control sampling or actions
    disp(['Result='  DSSText.Result])
    if DSSSolution.Converged
       a = 'Solution Converged';
       disp(a)
    else
       a = 'Solution did not Converge';
       disp(a)
    end

    DSSText.Command='Export Voltages';
    disp(DSSText.Result)

    DSSSolution.SampleControlDevices;
    DSSCircuit.CtrlQueue.Show; % show the contents of the control queue
    disp(DSSText.Result)
    DSSSolution.DoControlActions; % execute the control actions
    DSSCircuit.CtrlQueue.Show;  % see what is left on the queue

else
    a = 'DSS Did Not Start'
    disp(a)
end
```

Example 2 (requires 7.6.4.42 or later)

Here is a VBA macro for Excel that implements a capacitor control based on voltage. It uses the DSS script below that describes a simple circuit with a 4-step capacitor. It starts off turning all steps off and then increments load (using the Loadmult option) until all steps turn on because the voltage goes low. Then it reverses direction on the load multiplier until all steps turn off because the voltage goes too high. In Excel VBA, the output will show up in the "Immediate Window" (ctrl-G will open it).

```
Option Explicit

' VBA code for Excel

Public DSSobj As OpenDSSengine.DSS
Public DSSText As OpenDSSengine.Text
Public DSSCircuit As OpenDSSengine.Circuit
Public DSSSolution As OpenDSSengine.Solution
Public DSSControlQueue As OpenDSSengine.CtrlQueue
Public DSSCktElement As OpenDSSengine.CktElement
Public DSSPDElement As OpenDSSengine.PDElements
Public DSSMeters As OpenDSSengine.Meters
Public DSSBus As OpenDSSengine.Bus
Public DSSCmath As OpenDSSengine.CmathLib
Public DSSParser As OpenDSSengine.Parser
Public DSSIsources As OpenDSSengine.ISources
Public DSSMonitors As OpenDSSengine.Monitors

' Execute this first
Public Sub StartDSS()

' Create a new instance of the DSS
    Set DSSobj = New OpenDSSengine.DSS

' Start the DSS
    If Not DSSobj.Start(0) Then
        MsgBox "DSS Failed to Start"
    Else
        ' MsgBox "DSS Started successfully"
        ' Assign a variable to each of the  interfaces for easier access
        Set DSSText = DSSobj.Text
        Set DSSCircuit = DSSobj.ActiveCircuit
        Set DSSSolution = DSSCircuit.Solution
        Set DSSControlQueue = DSSCircuit.CtrlQueue
        Set DSSCktElement = DSSCircuit.ActiveCktElement
        Set DSSPDElement = DSSCircuit.PDElements
        Set DSSMeters = DSSCircuit.Meters
        Set DSSBus = DSSCircuit.ActiveBus
        Set DSSCmath = DSSobj.CmathLib
        Set DSSParser = DSSobj.Parser
        Set DSSIsources = DSSCircuit.ISources
        Set DSSMonitors = DSSCircuit.Monitors

        Range("DSSVersion").Value = "Version:  " + DSSobj.Version
        Beep
    End If


End Sub


Public Sub TestCtrlQueue()

' Example of implementing a simple voltage control for Capacitors via the COM interface

    Dim hour As Long, secDelay As Double

    ' Run simple capacitor interface test and execute local cap control that emulates
CapControl
    ' with these settings:
    ' PT=125.09 Type=voltage onsetting=118.8 offsetting=121.2

    Dim DSSCapacitors As OpenDSSengine.Capacitors
    Set DSSCapacitors = DSSCircuit.Capacitors  ' set a variable to the Capacitors
interface


    ' this test case has a four-step capacitor bank named "cap" and can be found in the
Test Folder

    DSSText.Command = "Compile
(C:\Users\prdu001\OpenDSS\Test\Master_TestCapInterface.DSS)"

    ' Set all capacitor steps open for first capacitor
    Dim i As Long, iCap As Long, iStates(1 To 10) As Variant
    Dim strValue As String

    iCap = DSSCapacitors.First   ' should check iCap for >0
    For i = 1 To DSSCapacitors.NumSteps
        iStates(i) = 0
    Next i
    DSSCapacitors.States = iStates  ' push over the interface to OpenDSS

' check to make sure it worked
    DSSText.Command = "? Capacitor.Cap.States"
    strValue = "Starting Capacitor Step States=" + DSSText.Result  ' should be [0 0 0 0]
    Debug.Print strValue

' Base solution
    DSSSolution.Solve

    ' Each message we push onto the queue will get a 5 s delay
    hour = 0
    secDelay = 5  ' delay

    Dim V As Variant   ' for getting bus voltages
    Dim PTratio As Double, Vreg As Double
    Dim ONsetting As Double, OFFsetting As Double
    Dim ActionCodeAdd As Long, DeviceHandle As Long, ActionCodeSub As Long

    PTratio = 125.09   ' for 26 kV system
    ONsetting = 118.8
    OFFsetting = 121.2
    ActionCodeAdd = 201  ' just an arbitrary action code
    ActionCodeSub = 202  ' just another arbitrary action code
    DeviceHandle = 123  ' arbitrary handle that signifies this control

    ' now, we'll crank the load up in 10% steps, checking the voltage at each step
    ' until all cap steps are on (no more available)

    i = 0
    Do While DSSCapacitors.AvailableSteps > 0
```

```
        i = i + 1
        DSSSolution.LoadMult = 1# + i * 0.1  ' 10% more each time
        DSSSolution.InitSnap
        DSSSolution.SolveNoControl
        DSSSolution.SampleControlDevices ' sample all other controls

        ' Emulate the cap control Sample Routine and get the bus voltage
        DSSCircuit.SetActiveBus "feedbus"
        V = DSSBus.VMagAngle
        ' check the first phase magnitude
        Vreg = V(0) / PTratio
        Debug.Print "Step "; i; " Voltage="; Vreg; " LoadMult="; DSSSolution.LoadMult
        If Vreg < ONsetting Then ' push a message to bump up the number of steps
            DSSControlQueue.Push hour, secDelay, ActionCodeAdd, DeviceHandle
        End If

        DSSSolution.DoControlActions   ' this sends actions to the local action list

        If DSSControlQueue.NumActions > 0 Then

           Do While DSSControlQueue.PopAction > 0

               Select Case DSSControlQueue.DeviceHandle
               Case 123
                    iCap = DSSCapacitors.First   ' Sets designated capacitor active
               End Select

               Select Case DSSControlQueue.ActionCode

               Case 201
                    DSSCapacitors.AddStep
               Case 202
                    DSSCapacitors.SubtractStep

               End Select

               ' Print result
               DSSText.Command = "? Capacitor." + DSSCapacitors.Name + ".States"
               Debug.Print "Capacitor " + DSSCapacitors.Name + " States=" +
DSSText.Result

           Loop
        End If

    Loop

    ' Now let's reverse Direction and start removing steps

    Do While DSSCapacitors.AvailableSteps < DSSCapacitors.NumSteps
        i = i - 1
        DSSSolution.LoadMult = 1# + i * 0.1  ' 10% more each time
        DSSSolution.InitSnap
        DSSSolution.SolveNoControl
        DSSSolution.SampleControlDevices ' sample all other controls

        ' Emulate the cap control Sample Routine and get the bus voltage
        DSSCircuit.SetActiveBus "feedbus"
        V = DSSBus.VMagAngle
        ' check the first phase magnitude
        Vreg = V(0) / PTratio
        Debug.Print "Step "; i; " Voltage="; Vreg; " LoadMult="; DSSSolution.LoadMult
        If Vreg > OFFsetting Then ' push a message to bump down the number of steps
            DSSControlQueue.Push hour, secDelay, ActionCodeSub, DeviceHandle
        End If

        DSSSolution.DoControlActions   ' this send actions to the local action list

        If DSSControlQueue.NumActions > 0 Then

           Do While DSSControlQueue.PopAction > 0
               Select Case DSSControlQueue.DeviceHandle
               Case 123
                    iCap = DSSCapacitors.First   ' Sets designated capacitor active
               End Select

               Select Case DSSControlQueue.ActionCode

               Case 201
                    DSSCapacitors.AddStep
               Case 202
                    DSSCapacitors.SubtractStep

               End Select

               ' Print result
               DSSText.Command = "? Capacitor." + DSSCapacitors.Name + ".States"
               Debug.Print "Capacitor " + DSSCapacitors.Name + " States=" +
DSSText.Result

           Loop
        End If

    Loop

End Sub
```

The DSS script for the circuit (from the Test folder).

```
Clear

New Circuit.CapInterface
~ basekv=161 pu=1.05 angle=0.0 freq=60.0 phases=3

!************** Begin Transformers **************!
new transformer.TR1 phases=3 windings=2 buses=(genbus,feedbus) conns=(delta,wye)
~ kvs=(161,26) kvas=(42000,42000) XHL=9.049 %r=0.2961
new transformer.TR2 phases=3 windings=2 buses=(genbus,feedbus) conns=(delta,wye)
~ kvs=(161,26) kvas=(42000,42000) XHL=8.26 %r=0.2877
!************** End Transformers **************!

!************** Begin Lines **************!
! R and X are real values Base V=161kV
! Assumed zero sequence R and X is 3X positive sequence
NEW line.line1 bus1=sourcebus bus2=genbus R1=0.9357 X1=6.9987 R0=2.8071 X0=20.9961
!************** End Lines **************!

!************** Begin Capacitors **************!
New Capacitor.Cap Bus1=feedbus phases=3 kv=26 numsteps=4 kvar=[5400 5400 5400 5400]
// New CapControl.CapCtrl element=transformer.TR2 terminal=2 capacitor=Cap
// ~ PT=125.09 Type=voltage onsetting=118.8 offsetting=121.2
!************** End Capacitors **************!

!************** Begin Loads **************!
!2006 Med. Summer Extreme Pk Forecast, updated PF to match PSS/E data
New Load.Load_TH_Summer bus1=feedbus phases=3 kv=26 kw=57000 pf=0.95 model=1
!************** End Loads **************!

Set voltagebases=[161  26]
Calcv
```