

```
=====
subcase_nextTask.cbs
:
: This file implements the methods of Subcase required to perform
: the Next-Task Status Changes: when a subcase is completed, it informs
: its successor subcases that they may go into a "ready-to-work" state.
:
: This file contains two sections: Core Functions, those functions
: that implement the main logic of the Next-Task stuff; and Utilities,
: which are functions that may be generally useful to other developers.
:
: For more information, please see the design document:
:
:   DS_POL_003, Next Task Status Changes
:
: Future Improvements
: -----
: 1. Stop Thrashing
: The original design for this file had record information being passed
: by objid, assuming that each function would have automatic, implicit
: access to the subcase record. (I was thinking in object-oriented terms.)
: This helped keep the design clean, but has made the implementation messy:
: now, lots of different functions have to call the database and pull
: subcase records. I'd like to fix this, so that either (a) we load
: the subcase record early on, and pass it around; and/or (b) make
: most functions take Variant arguments instead of Longs, so that you
: can pass them either an objid or the record itself -- whichever is
: more efficient for the calling function.
:
: All code in this file is proprietary and confidential to
: BroadRiver Communications (http://www.radicallysimple.com).
: (c)2000 BroadRiver Communications, Inc.
: =====
: -----
: Declarations
:
: Declared as needed, to make the compiler happy. NOT a complete list
: of functions in this file.
: -----
Option Explicit

Declare Function case_getWorkflowSubcases (caseID as Long) as List
Declare Function pullCaseRec (caseID as long, caseRec as Record) as Boolean
Declare Function pullDatabaseRec (objectType as String, objectID as long, objectRec as Record) as Boolean
Declare Function pullSubcaseRec (subcaseID as long, subcaseRec as Record) as Boolean
Declare Function subcase_getOutstandingPredecessors (subcaseID as Long) as List
Declare Function subcase_getPredecessors (subcaseID as Long) as List
Declare Function subcase_getSiblings (subcaseID as Long, predOrSucc as String, status as String, statu
Declare Function subcase_getStatus (subcaseID as Long) as String
Declare Function subcase_getSuccessors (subcaseID as Long) as List
Declare Function subcase_getSuccIncompl (subcaseID as Long) as List
Declare Sub case_checkComplete (caseID as Long)
Declare Sub case_setStatus (caseID as Long, statusCode as String)
Declare Sub global_errorPrint (objectName as String, objectID as long, message as String)
Declare Sub subcase_launchSuccessors (userFriendlyID as String)
Declare Sub subcase_predecessorDone (subcaseID as Long)
Declare Sub subcase_schedule (subcaseID as Long)
Declare Sub subcase_setStatus (subcaseID as Long, statusCode as String)
Declare Function SwapStr (sourceStr as String, searchStr as String, replaceStr as String) as String

: External utilities:
```

```
Declare Function GetConstant (constName as String) as Variant
Declare Function create_act_entry (strAddnlInfo As String, lngUserObj As Long, act_objid As
Declare Function subcase_changeWFStatus (subcaseID as Long, statusCode as String, old_status As St

' FirstChoice utilities:
Declare Function change_case_status (case_id As String, _ ' required
new_status As String, _ ' optional
change_date As String, _ ' optional
notes As String, _ ' optional
user_name As String, _ ' optional
gen_time_bombs As Boolean _ ' required
) _
As Integer
```

-----  
' Core Functions - Next Task Status Changes  
-----

```
subcase_launchSuccessors (subcaseID)
'
' This script is invoked when a subcase's status changes to "Complete."
' The script obtains a list of all successors of this predecessor
' subcase, and informs them that the predecessor has been completed.
'
Parameters:
' subcaseID: the "ID" of the subcase whose status has changed --
' i.e., the user-readable ID, not the object ID. That's
' because we get this value from the pop-up menu in
' the Business Rules system.
```

```
Sub subcase_launchSuccessors (userFriendlyID as String)

Debug.Print "Ron & Padmaja say 'hi' from launchSuccessors..."

' -----
#IF false Then
' -----
Dim grabber As New BulkRetrieve
grabber.SimpleQuery 0, "subcase"
grabber.AppendFilter 0, "id_number", cbEqual, userFriendlyID
grabber.RetrieveRecords

Dim resultList as List
Dim resultRec as Record
Set resultList = grabber.GetRecordList (0)
Set resultRec = resultList.ItemByIndex (0)

resultRec.SetField "title", "hello, from the subcase_launchSuccessors!"

Dim setter As New BulkSave
setter.UpdateRecord resultRec
setter.Save
```

```
-----
#ELSE
' -----
Dim subcaseID as Long ' The subcase whose
Dim subcaseRec as Record ' status has changed.
Dim subcaseList as List ' Temp variable, to hold the results
' of the query in which we determine
' subcaseID.

Dim successorList as List ' The list of all successors of subcaseID.
Dim successorIDlist as List ' The list of those successors' objids.
```

```
Dim successorRec as Record ' A specific successor, obtained as we
Dim successorID as Long ' iterate through the list of those
' successors.

Dim caseRec as Record ' The case that
Dim caseID as Long ' owns subcaseID.
Dim caseList as List ' Temp variable, to hold the results of
' the query in which we determine caseID.

Dim n as Integer ' A loop variable.
```

```
' Go get the subcase record, as well as its owning case.
```

```
Dim grabber As New BulkRetrieve
grabber.SimpleQuery 0, "subcase"
grabber.AppendFilter 0, "id_number", cbEqual, userFriendlyID
grabber.TraverseFromParent 1, "subcase2case", 0
grabber.RetrieveRecords
```

```
Set subcaseList = grabber.GetRecordList (0)
If subcaseList.count = 1 Then
```

```
' Get the subcase's objid.
Set subcaseRec = subcaseList.ItemByIndex (0)
subcaseID = subcaseRec.GetField ("objid")
```

```
' If the subcase has any successors, tell each one that one of
' its predecessors is done.
```

```
' *** DESIGN CHANGE ***: OK, make that "any OUTSTANDING
' successors" -- which allows things to get completed out of
' order. This should never happen, but it might. By modifying
' things this way, we'll end up telling the Case to check things
' out if all of a subcase's successors have somehow been
' completed BEFORE the subcase got around to telling them.
```

```
' -- Ron Conescu, 10/15/00 9:59PM
```

```
Set successorList = subcase_getSuccessors (subcaseID)
Set successorList = subcase_getSuccIncompl (subcaseID)
debug.print "subcase_launchSuccessors: "&_
"number of successors = "& successorList.Count & ">"
If successorList.Count > 0 Then
```

```
' Extract all the objids from the list of successors.
Set successorIDlist = new List ' need to do this for
successorIDlist.ItemType = "long" ' the debugger...!
successorList.ExtractList successorIDlist, "objid"
```

```
' Tell each successor that a predecessor is done.
For n = 0 to (successorIDlist.Count - 1)
debug.print "subcase_launchSuccessors: "&_
"working on successor #" & n+1 & "... "
successorID = successorIDlist.ItemByIndex (n)
subcase_predecessorDone successorID
Next n
```

```
' Otherwise, if the subcase has no successors, then it's
' entirely possible that the workflow for the parent case
' is done. Inform the case object.
Else
```

```
Set caseList = grabber.GetRelatedRecordList (subcaseRec, "subcase2case")
If caseList.Count = 1 Then
```

```
Set caseRec = caseList.ItemByIndex (0)
caseID = caseRec.GetField ("objid")

' Tell the case to see whether or not all its
' subcases are done.
case_checkComplete caseID

' If we got more than one record back for the case... uh...
' something very weird happened.
Else

    If caseList.Count = 0 Then

        global_errorPrint "subcase", subcaseID, _
            "subcase_launchSuccessors: " & _
            "Uh-oh! I couldn't find the owning case " & _
            "for the subcase ID <" & CStr (subcaseID) & ">."

    Else ' caseList.Count > 1

        global_errorPrint "subcase", subcaseID, _
            "subcase_launchSuccessors: " & _
            "Uh-oh! I got more than one owning case " & _
            "for the subcase ID <" & CStr (subcaseID) & ">."

    End If

End If

End If ' if our subcase has any successors

' If we got more than one subcaseID back, or none, we have a problem.
Else

    If subcaseList.count = 0 Then

        global_errorPrint "subcase", -1, _
            "subcase_launchSuccessors: " & _
            "Uh-oh! I couldn't find a subcase " & _
            "matching the ID <" & userFriendlyID & ">."

    Else ' subcaseList.Count > 1

        global_errorPrint "subcase", -1, _
            "subcase_launchSuccessors: " & _
            "Uh-oh! I found more than one subcase" & _
            "matching the ID <" & userFriendlyID & ">."

    End If

End If ' if subcaseIDlist.Count = 1

' -----
#END If
' -----

end Sub
```

```
' -----
' subcase_predecessorDone (subcaseID)
'
```

```
' This function is called when one of a subcase's predecessors has been
' Completed. This function obtains a list of all the predecessors of
```

```
' this successor subcase, and checks to see whether they are ALL  
' complete. If so, the successor will flip itself into a schedulable  
' state.  
'  
' Parameters:  
' subcaseID: the objid of the subcase to be scheduled.  
'  
' Design Changes  
' -----  
' I put the logging function here, instead of in subcase_schedule(),  
' because the thing we're logging really has to do with this function,  
' not that one.  
' -----
```

```
Sub subcase_predecessorDone (subcaseID as Long)  
  
Dim toDoList as List  
Set toDoList = subcase_getOutstandingPredecessors (subcaseID)  
  
If toDoList.Count = 0 Then  
    global_errorPrint "subcase", subcaseID, "Predecessors are done."  
    subcase_schedule subcaseID  
  
End If  
  
End Sub
```

```
' -----  
' subcase_schedule (subcaseID)  
'  
' This function is called when we have determined that a subcase can be  
' scheduled -- i.e., that an end-user may now close it out. In this  
' function, we change the subcase's status to "Ready to Work," so that  
' an end-user knows she can start working on it (and that such work is  
' expected). If the end-user has already begun working on the subcase,  
' we won't interfere.  
'  
' Parameters:  
' subcaseID: the objid of the subcase to be scheduled.  
'  
' Design Changes  
' -----  
' 1. Moved the logging function to predecessorDone. See  
' predecessorDone for more information.  
'  
' 2. Instead of seeing whether the subcase has been modified, I'm going  
' to see whether the subcase is in (or not in) "Awaiting Scheduling"  
' state. I think tha't easier, and truer to our core design. -- Ron  
' -----
```

```
Sub subcase_schedule (subcaseID as Long)  
  
debug.print "subcase_schedule: "&  
    "Got here!"  
  
Dim oldStatus as String  
Dim testStatus as String  
testStatus = CStr (GetConstant ("kSubcase_Status_Await_Sched"))  
  
oldStatus = subcase_getStatus (subcaseID)  
If oldStatus = testStatus Then  
    subcase_setStatus subcaseID, "kSubcase_Status_Ready2Work"  
  
' Whoops; we didn't get the subcase record. Now what?  
Else  
    global_errorPrint "subcase", subcaseID, _
```

```
"subcase_schedule: " &_  
"This subcase's status is NOT 'Awaiting Scheduling', " &_  
"so I am not permitted to Schedule it."
```

End If

End Sub

-----  
' Utilities  
' -----  
-----

subcase\_getPredecessors (subcaseID)

This function returns a list of the predecessors of a specific subcase, within a specific workflow. It also allows you to see how many predecessors that subcase has - by requesting the Count of the items in the list (information that Clarify gives you automatically).

Parameters:

subcaseID: the objid of the subcase whose predecessors are needed.

Returns: a List of Subcase records. If the subcase in question has no predecessors, the list will be empty (it will contain zero elements).

-----  
Function subcase\_getPredecessors (subcaseID as Long) as List

```
Dim result as List  
Set result = subcase_getSiblings (subcaseID, "pred", "", 0)  
Set subcase_getPredecessors = result
```

End Function

-----  
' subcase\_getOutstandingPredecessors (subcaseID)  
' -----

This function returns a list of the un-Completed predecessors of a specific subcase, within a specific workflow.

Parameters:

subcaseID: the objid of the subcase whose outstanding predecessors are needed.

Returns: a List of Subcase records. If the subcase in question has no outstanding predecessors, the list will be empty (it will contain zero elements).

-----  
Function subcase\_getOutstandingPredecessors (subcaseID as Long) as List

```
Dim result as List  
Dim status as String  
status = CStr (GetConstant ("kSubcase_Status_Completed"))  
Set result = subcase_getSiblings (subcaseID, "pred", status, cbNotEqual)  
Set subcase_getOutstandingPredecessors = result
```

End Function

-----  
' subcase\_getSuccessors (subcaseID)  
' -----

This function returns a list of the successors to a specific subcase, within a specific workflow. It also allows you to see how many

```
' successors that subcase has -- by requesting the Count of the items
' in the list (information that Clarify gives you automatically).
'
' Parameters:
'   subcaseID:   the objid of the subcase whose successors are needed.
'
' Returns:   a List of Subcase records.  If the subcase in question has
' no successors, the list will be empty (it will contain zero elements).
'-----
Function subcase_getSuccessors (subcaseID as Long) as List

    Dim result as List
    Set result = subcase_getSiblings (subcaseID, "succ", "", 0)
    Set subcase_getSuccessors = result

End Function

'-----
' subcase_getSuccIncompl (subcaseID)
' "Get Incomplete Successors"
'
' This function returns a list of the successors to a specific subcase,
' within a specific workflow.  It also allows you to see how many
' successors that subcase has -- by requesting the Count of the items
' in the list (information that Clarify gives you automatically).
'
' Parameters:
'   subcaseID:   the objid of the subcase whose successors are needed.
'
' Returns:   a List of Subcase records.  If the subcase in question has
' no successors, the list will be empty (it will contain zero elements).
'-----
Function subcase_getSuccIncompl (subcaseID as Long) as List

    Dim result as List
    Dim status as String
    status = CStr (GetConstant ("kSubcase_Status_Completed"))
    Set result = subcase_getSiblings (subcaseID, "succ", status, cbNotEqual)
    Set subcase_getSuccIncompl = result

End Function

'-----
' subcase_getSiblings (subcaseID as Long, _
'                      predOrSucc as String, _
'                      status as String, _
'                      statusTestCondition as Long) as List
'
' This function returns a list of the successors or predecessors of a
' specific subcase, within its workflow.  These "siblings" will match
' the status criteria you specify in the last two parameters.
'
' Returns:   a List of Subcase records.  If there are no subcases that
' match the specified criteria, the list will be valid, but empty (it
' will contain zero elements).
'
' Parameters:
'
'   subcaseID:   The objid of the subcase whose successors or
' predecessors is needed.
'
'   predOrSucc:  Specifies whether the returned subcases are the
' predecessors or successors of the specified subcase.
```

Possible values are:  
"pred": returns the predecessors.  
"succ": returns the successors.

status: Lets you check for a certain status in the siblings' x\_wf\_status field. This parameter is simply passed through to the database, so the possible values are the same as the possible values of that status field in the user interface:  
"Completed"  
"In Progress"  
etc.

There is one additional value you may pass:  
"" (an empty string)

This will retrieve all statuses -- i.e., all siblings specified in predOrSucc. See "Examples," below, for more information.

statusTestCondition: Together with the status parameter, lets you specify the siblings you'd like to retrieve. This parameter, too, is simply passed through to the database, so the possible values are the same as the list of possible values for BulkRetrieve.AppendFilter:  
cbEqual  
cbNotEqual  
etc.  
If status is "", this parameter is ignored (although the compiler requires it). See "Examples," below, for more information.

#### Examples

1. To retrieve all successors of a subcase:  
subcase\_getSiblings (subcaseID, "succ", "", 0)
2. To retrieve all the predecessors that are in a "Hold" state:  
subcase\_getSiblings (subcaseID, "pred", "On Hold", cbEqual)
3. To retrieve the incomplete predecessors of a subcase:  
subcase\_getSiblings (subcaseID, "pred", "Completed", cbNotEqual)

The status strings in examples 2 and 3 were taken from the list of statuses available on 10/6/00; before using this function, please check the current values of that list.

#### Future Improvements

I'd like to make the last two parameters optional; that would make the calls to this function much more readable. For example, the examples above would become:

1. subcase\_getSiblings (subcaseID, "succ")
2. subcase\_getSiblings (subcaseID, "pred", "On Hold")
3. subcase\_getSiblings (subcaseID, "pred", "Completed", cbNotEqual)

Specifically, the behavior would be:

1. omit both status and statusTestCondition: get all statuses
2. omit testCondition: cbEqual is implied

Anyone know how to make parameters optional?

```
' -----  
Function subcase_getSiblings (subcaseID as Long, _  
                             predOrSucc as String, _  
                             status as String, _  
                             statusTestCondition as Long) as List  
  
Dim scheduleQuery as String  
Dim siblingQuery as String  
  
Dim sourceRec as Record  
Dim sourceSet as List  
Dim scheduleRec as Record  
Dim scheduleSet as List  
Dim siblingRec as Record  
Dim siblingSet as List  
  
Dim grabber as new BulkRetrieve  
Dim n as Integer  
  
' Create a list to hold our results.  If we have no results, we'll  
' return this empty list.  
Dim resultList as List  
Set resultList = new List  
resultList.ItemType = "Record"  
  
Select Case predOrSucc  
    Case "pred"  
        scheduleQuery = "subcase2x_wfsubc"  
        siblingQuery = "x_wfsubcpred2subcase"  
    Case "succ"  
        scheduleQuery = "subcase2x_wfsubcpred"  
        siblingQuery = "x_wfsubc2subcase"  
End Select  
  
grabber.SimpleQuery 0, "subcase"  
grabber.AppendFilter 0, "objid", cbEqual, subcaseID  
grabber.TraverseFromParent 1, scheduleQuery, 0  
grabber.TraverseFromParent 2, siblingQuery, 1  
If status <> "" Then  
    grabber.AppendFilter 2, "x_wf_status", statusTestCondition, status  
End If  
grabber.RetrieveRecords  
  
Set sourceSet = grabber.GetRecordList (0)  
If sourceSet.Count = 1 Then  
  
    Set sourceRec = sourceSet.ItemByIndex (0)  
    Set scheduleSet = grabber.GetRelatedRecordList (sourceRec, scheduleQuery)  
  
    ' Go through the Schedule records, extracting the Sibling records  
    ' that matched the statusTestCondition.  
    If scheduleSet.Count <> 0 Then  
        For n = 1 to scheduleSet.Count  
  
            Set scheduleRec = scheduleSet.ItemByIndex (n-1)  
            Set siblingSet = grabber.GetRelatedRecordList (scheduleRec, siblingQuery)  
  
            ' If we found a sibling matching the test criteria,  
            ' add it to the list.  
            If siblingSet.Count = 1 Then  
                Set siblingRec = siblingSet.ItemByIndex (0)  
                resultList.AppendItem siblingRec  
  
            ' If we didn't, no worries.  
            Else
```

```
        ' Do nothing.

    End If

Next n
Else
    ' No schedule records, i.e., no siblings that match these
    ' criteria. No problem.
End If

Else
    ' Shouldn't happen.
    global_errorPrint "subcase", subcaseID, _
        "subcase_getSiblings: couldn't find a subcase record " & _
        "for this subcase ID!"

End If

Set subcase_getSiblings = resultList

End Function
```

```
-----
' pullSubcaseRec (subcaseID, subcaseRec) as Boolean
'
' This function retrieves the subcase matching the specified objid
' from the database, and puts that record in subcaseRec.
'
' Parameters:
'   subcaseID:   the objid of the subcase you wish to retrieve.
'   subcaseRec:  a Record variable, containing no data. The retrieved
'                 record will be placed into this variable. If the
'                 record is not found, this variable is untouched.
'
' Returns:  TRUE if the record was found, FALSE otherwise.
'-----
```

```
Function pullSubcaseRec (subcaseID as long, _
                        subcaseRec as Record) as Boolean
```

```
    Dim result as Boolean
    result = pullDatabaseRec ("subcase", subcaseID, subcaseRec)
    pullSubcaseRec = result
```

```
End Function
```

```
-----
' pullCaseRec (caseID, caseRec) as Boolean
'
' This function retrieves the case matching the specified objid
' from the database, and puts that record in caseRec.
'
' Parameters:
'   caseID:   the objid of the subcase you wish to retrieve.
'   caseRec:  a Record variable, containing no data. The retrieved
'                 record will be placed into this variable. If the
'                 record is not found, this variable is untouched.
'
' Returns:  TRUE if the record was found, FALSE otherwise.
'-----
```

```
Function pullCaseRec (caseID as long, _
                    caseRec as Record) as Boolean
```

```
Dim result as Boolean
result = pullDatabaseRec ("case", caseID, caseRec)
pullCaseRec = result
```

End Function

```
-----
pullDatabaseRec (objectType, objectID, objectRec) as Boolean
```

```
This function retrieves the object matching the specified type and
objid from the database, and puts that record in objectRec.
```

```
Parameters:
```

```
objectType:  the name of the object to retrieve -- "case",
              "subcase", etc.
```

```
objectID:    the objid of the object you wish to retrieve.
```

```
objectRec:   a Record variable, containing no data.  The retrieved
              record will be placed into this variable.  If the
              record is not found, this variable is untouched.
```

```
Returns:  TRUE if the record was found, FALSE otherwise.
```

```
-----
Function pullDatabaseRec (objectType as String, _
                        objectID as long, _
                        objectRec as Record) as Boolean
```

```
Dim grabber as new BulkRetrieve
grabber.SimpleQuery 0, objectType
grabber.AppendFilter 0, "objid", cbEqual, objectID
grabber.RetrieveRecords
```

```
Dim result as Boolean
result = false
```

```
Dim resultSet as List
Set resultSet = grabber.GetRecordList (0)
If resultSet.Count = 1 Then
    Set objectRec = resultSet.ItemByIndex (0)
    result = true
```

```
Else
    global_errorPrint objectType, objectID, _
        "pullDatabaseRec: Warning: Couldn't find a <" &_
        objectType & "> with the id <" & objectID & ">."
End If
```

```
pullDatabaseRec = result
```

End Function

```
-----
subcase_setStatus (subcaseID, statusCode)
```

```
This function sets the status of the specified subcase to the specified
code.  This function allows us, elsewhere in the system, to set
statuses using simple constants; inside the body of this function,
those constants are translated to the appropriate status indicator.
```

```
Parameters:
```

```
subcaseID:  the objid of the subcase whose status you wish to set.
```

```
' statusCode: the string constant representing the desired status.
'-----
Sub subcase_setStatus (subcaseID as Long, statusCode as String)

    debug.print "subcase_setStatus: "&_
        "Got here!"

    ' invoking Danielle's version of this function...
    '
    Dim result as Boolean          ' the result of the call to changeWFStatus
    Dim statusText as String      ' the text corresponding to the statusCode
    Dim oldStatusText as String   ' the text corresponding to the previous status
    Dim subcaseRec as Record      ' dummy variable
    Dim caseRec as Record         ' dummy variable

    statusText = CStr (GetConstant (statusCode))
    oldStatusText = subcase_getStatus (subcaseID)

    debug.print "subcase_schedule: "&_
        "About to call Danielle's 'subcase_changeWFStatus' function..."

    result = subcase_changeWFStatus (subcaseID, statusText, oldStatusText, subcaseRec, caseRec)

    debug.print "subcase_schedule: "&_
        "Back from Danielle's 'subcase_changeWFStatus' function."

    If not (result) Then
        global_errorPrint "subcase", subcaseID, _
            "subcase_setStatus: " & _
            "Uh-oh! Couldn't change the status of subcase <" & _
            subcaseID & "> to <" & statusText & ">!"
    End If

'-----
#IF false Then
'-----

    Dim subcaseRec as Record      ' the subcase whose status we want to set.
    Dim saver as new BulkSave     ' an envelope for saving our modified record.

    ' If we can pull the subcase record, and the new status is not the
    ' same as the current status, then set the new status.
    If pullSubcaseRec (subcaseID, subcaseRec) Then

        subcaseRec.SetField "x_wf_status", CStr (GetConstant (statusCode))
        saver.UpdateRecord subcaseRec
        saver.Save

    Else
        global_errorPrint "subcase", subcaseID, _
            "subcase_setStatus: " & _
            "Uh-oh! Couldn't find subcase ID <" & subcaseID & "> !"
    End If

'-----
#END If
'-----

End Sub
```

```
'-----
' subcase_getStatus (subcaseID) as String
'
' This function returns the status of the specified subcase.
'
```

```
' Parameters:
'   subcaseID:   the objid of the subcase whose status you wish to get.
'
' Returns:   a string, the name of the constant corresponding to the
' the status of the subcase in question.  If the subcase cannot be
' found, the empty string ("") is returned.
'-----
Function subcase_getStatus (subcaseID as Long) as String

    Dim subcaseRec as Record          ' the subcase whose status we want.
    Dim result as String              ' temp variable.

    If (pullSubcaseRec (subcaseID, subcaseRec)) Then
        result = subcaseRec.GetField ("x_wf_status")
    Else
        result = ""
    End If

    subcase_getStatus = result

End Function
```

```
'-----
' Case functions
'-----
'-----
' case_checkComplete (caseID)
'
' This function is called when a subcase in a workflow has finished, and
' has no successors.  If that subcase is the final subcase to finish --
' of all the subcases in the Case's workflow - we'll send a message to
' the human owner.
'
' Parameters:
'   caseID:     the objid of the Case to which the completed Subcase
'               belongs.
'
' Design Changes
'-----
' In the inner "if" statement, I don't check to see whether a particular
' subcase has successors; I just check to see if it's complete or not.
' (I'm not sure why I put that in the design in the first place; hold-
' over from an earlier version of the function, maybe?)  -- ron
'-----
Sub case_checkComplete (caseID as Long)
```

```
    Dim workflowSet as List
    Dim ready as Boolean
    Dim n as Integer          ' loop variable.
    Dim subcaseRec as Record
    Dim subcaseStatus as String
    Dim desiredStatus as String

    ready = true
    desiredStatus = CStr (GetConstant ("kSubcase_Status_Completed"))

    ' Walk through all the workflow subcases.  If all of them are
    ' complete, the Case is complete.
    Set workflowSet = case_getWorkflowSubcases (caseID)
    For n = 1 to workflowSet.Count
        Set subcaseRec = workflowSet.ItemByIndex (n-1)
        subcaseStatus = subcaseRec.GetField ("x_wf_status")
```

```
    If subcaseStatus <> desiredStatus Then
        ready = false
        exit for
    End If
Next n

' If we're done, then change the status.  THIS KICKS OFF
' A BUSINESS RULE, which informs the owner that the workflow
' is done.
If ready Then

    global_errorPrint _
        "case", caseID, "All subcases in workflow completed."

    case_setStatus caseID, "kStatus_Case_WorkflowDone"

End If

End Sub
```

```
-----
' case_getWorkflowSubcases (caseID as Long) as List
' -----
Function case_getWorkflowSubcases (caseID as Long) as List

    Dim stepConstant as String
    stepConstant = CStr (GetConstant ("kSubcase_Type_Step"))

    Dim grabber as new BulkRetrieve
    grabber.SetRootById "case", caseID
    grabber.TraverseFromRoot 0, "case2subcase"
    grabber.AppendFilter 0, "x_wf_type", cbEqual, stepConstant
    grabber.RetrieveRecords

    Dim resultSet as List
    Set resultSet = grabber.GetRecordList (0)
    Set case_getWorkflowSubcases = resultSet

End Function
```

```
-----
' case_setStatus (caseID, statusCode)
' -----
' Notes
' -----
' We have a lot to do here.  A case's current status is indicated by
' the global string record (gbst_elm, or global string element) that it
' is related to.  To CHANGE the case status, though, we have to update
' a bunch of other relationships:
'
' - Relate the case to the new string record -- i.e., the text
'   representing the new status, which shows up in the Status field
'   in the Case window.
' - Create a new Status Change record, which holds the user's notes
'   about this status change; date-stamp it; and relate it to the
'   case, the user, the new string record, and the previous string
'   record.
' - Create a new activity log entry, saying that we've changed the
'   status, and relate it to the case, the status change record, the
'   person making the change, and the string record representing the
'   type of activity we're performing.
'
' In our case, the "person making the change" will typically be the
' business rule/batch file that fires this script, instead of a human.
```

There are some interesting things happening here:

- The string record table (gbst\_elm) contains a list of strings used for various purposes in Clarify. Those purposes don't necessarily have anything to do with each other. In our change-status situation, we are using two such unrelated features: the gbst\_elm list holds strings representing system activities, as well as strings representing case statuses.
- We have, implicitly and explicitly, created a history of the status changes a case undergoes. Once we've finished the above, we know what state we're in, as well as all status changes up to that point, the order in which they happened, and both user and system notes about each change. We can get to any of this information from almost anywhere else -- case, user, the status string, a status change record, or the activity log. Not bad.

\*\*\* Design Decision \*\*\*: this was never decided in design: if the Case is already in the requested status, do we go ahead and change its status anyway? I decided to do so, just in case we want to force a new set of notes to go along with the current status. Maybe, in the future, we'll make it more efficient, and NOT do this; but for now, I figured this was safer.

'Nuff talking. Here we go.

-----  
Sub case\_setStatus (caseID as Long, statusCode as String)

```
Dim resultSet as List
Dim caseRec as Record
Dim caseIDNum as String      ' user-readable ID
Dim oldStatusRec as Record
Dim oldStatusText as String
Dim newStatusText as String
Dim notes as String
Dim history as String      ' necessary?
```

```
Dim grabber as new BulkRetrieve
Dim result as Integer
```

```
' Pull some stuff from the database
grabber.SimpleQuery 0, "case"
grabber.AppendFilter 0, "objid", cbEqual, caseID
grabber.TraverseFromParent 1, "casests2gbst_elm", 0
grabber.RetrieveRecords
```

```
Set resultSet = grabber.GetRecordList (0)
```

```
If resultSet.Count = 1 Then
    Set caseRec = resultSet.ItemByIndex (0)
    caseIDNum = caseRec.GetField ("id_number")
```

```
    Set resultSet = grabber.GetRelatedRecordList (caseRec, "casests2gbst_elm")
    Set oldStatusRec = resultSet.ItemByIndex (0)
    oldStatusText = oldStatusRec.GetField ("title")
    newStatusText = CStr (GetConstant (statusCode))
```

```
    notes = CStr (GetConstant (statusCode & "_Note"))
    notes = SwapStr (notes, "<oldStatus>", oldStatusText)
    notes = SwapStr (notes, "<newStatus>", newStatusText)
```

```
    result = change_case_status (caseIDNum, newStatusText, "", notes, "", true)
```

```
If result <> 0 Then
    global_ErrorPrint "case", caseID, "case_setStatus: the change_case_status function return
End If
```

```
Else  
    ' uh-oh; couldn't find case!
```

```
End If
```

```
End Sub
```

```
Sub case_setStatus_old (caseID as Long, statusCode as String)
```

```
' -----  
' Variable declarations.  
' -----
```

```
' The objects we're going to relate to each other:  their objids,  
' their records, and the result sets for the queries we use to  
' acquire these things.
```

```
' The "-Set" suffix stands for "result set":  a list of records,  
' or sometimes a list of lists, returned from the database.
```

```
Dim caseRec as Record          ' The Case record we're relating  
                              ' everything to.
```

```
Dim oldStatusID as Long       ' The current status record in the  
Dim oldStatusRec as Record    ' Strings table (gbst_elm).  
Dim oldStatusText as String
```

```
Dim newStatusID as Long       ' The new status record in the  
Dim newStatusRec as Record    ' Strings table (gbst_elm).  
Dim newStatusText as String
```

```
Dim statusListID as Long      ' The record containing the list  
Dim statusListRec as Record   ' (from gbst_lst) of Case Status  
Dim statusListName as String  ' strings.
```

```
Dim statusChangeRec as Record ' The status-change record (table_  
                              ' status_chg) we're going to create.
```

```
Dim activityID as Long        ' The activity entry we're going to  
                              ' create.
```

```
Dim activityListText as String ' \ The string-list record (gbst_lst)  
Dim activityListRec as Record  ' / containing the list of activities.  
Dim activityRank as Long       ' \ The Rank of the String record  
Dim activityRankRec as Record  ' | (gbst_elm) indicating a Status  
                              ' / Change activity.
```

```
Dim userID as Long            ' The employee making this change.  
Dim userRec as Record
```

```
Dim resultSet as List         ' Temp variable, to hold the results  
                              ' of a given database query.
```

```
Dim grabber as BulkRetrieve   ' To go get stuff from the database.  
Dim saver as BulkSave         ' To save stuff back to the database.
```

```
' Check for errors.  We have lots of errors to check for, so we'll  
' use if (stillOK) to wrap each piece of logic; this makes the code  
' easier to read than having lots and lots of cascading 'if'  
' statements.
```

```
Dim stillOK as Boolean  
Dim errorMsg as String  
Dim notes as String  
Dim tempResult as Integer
```

```
' -----  
' Initialize our variables.  
' -----  
Set grabber = new BulkRetrieve  
Set saver = new BulkSave  
stillOK = true  
errorMsg = ""  
  
' -----  
' Get all our records we need from the database:  case, current user,  
' current status, and new status.  
' -----  
  
' The case.  
grabber.SimpleQuery 0, "case"  
grabber.AppendFilter 0, "objid", cbEqual, caseID  
  
' The current status.  
grabber.TraverseFromParent 1, "casests2gbst_elm", 0  
  
' The status we're changing to.  
statusListName = CStr (GetConstant ("kStatus_Case_List"))  
newStatusText = CStr (GetConstant (statusCode))  
grabber.SimpleQuery 2, "gbst_lst"  
grabber.AppendFilter 2, "title", cbEqual, statusListName  
grabber.TraverseFromParent 3, "gbst_lst2gbst_elm", 2  
grabber.AppendFilter 3, "title", cbEqual, newStatusText  
  
' The user who's currently logged in.  
grabber.SimpleQuery 4, "user"  
grabber.AppendFilter 4, "objid", cbEqual, App.UserObjID  
  
' The String record indicating a Status Change activity.  
activityListText = CStr (GetConstant ("kActivity"))  
activityRank = CLng (GetConstant ("kActivity_StsChg"))  
grabber.SimpleQuery 5, "gbst_lst"  
grabber.AppendFilter 5, "title", cbEqual, activityListText  
grabber.TraverseFromParent 6, "gbst_lst2gbst_elm", 5  
grabber.AppendFilter 6, "rank", cbEqual, activityRank  
  
' Go get 'em.  
grabber.RetrieveRecords  
  
' Extract the case.  
If (stillOK) Then  
    Set resultSet = grabber.GetRecordList (0)  
    If resultSet.Count = 1 Then  
        Set caseRec = resultSet.ItemByIndex (0)  
    Else  
        stillOK = false  
        errorMsg = "Couldn't get Case record"  
    End If  
End If  
  
' Extract the current status.  
If (stillOK) Then  
    Set resultSet = grabber.GetRelatedRecordList _  
        (caseRec, "casests2gbst_elm")  
  
    If resultSet.Count = 1 Then  
        Set oldStatusRec = resultSet.ItemByIndex (0)  
        oldStatusID = oldStatusRec.GetField ("objid")  
    Else  
        stillOK = false
```

```
        errorMsg = "Couldn't get current status record"
    End If
End If

' Get the user who's currently logged in.
If (stillOK) Then
    Set resultSet = grabber.GetRecordList (4)
    If resultSet.Count = 1 Then
        Set userRec = resultSet.ItemByIndex (0)
        userID = userRec.GetField ("objid")
    Else
        stillOK = false
        errorMsg = "Couldn't determine current user."
    End If
End If

' Get the new status, part 1:  the List of Case Statuses.
If (stillOK) Then
    Set resultSet = grabber.GetRecordList (2)
    If resultSet.Count = 1 Then
        Set statusListRec = resultSet.ItemByIndex (0)
    Else
        stillOK = false
        errorMsg = "Couldn't get list of case statuses"
    End If
End If

' Get the new status, part 2:  the new status itself, from
' the List of Case Statuses.
If (stillOK) Then
    Set resultSet = grabber.GetRelatedRecordList _
        (statusListRec, "gbst_lst2gbst_elm")

    If resultSet.Count = 1 Then
        Set newStatusRec = resultSet.ItemByIndex (0)
        newStatusID = newStatusRec.GetField ("objid")
    Else
        stillOK = false
        errorMsg = "Couldn't get new status record."
    End If
End If

' Get the activity, part 1:  the list of Activities.
If stillOK Then
    Set resultSet = grabber.GetRecordList (5)
    If resultSet.Count = 1 Then
        Set activityListRec = resultSet.ItemByIndex (0)
    Else
        stillOK = false
        errorMsg = "Couldn't get name of this activity."
    End If
End If

' Get the activity, part 2:  the activity itself ("Change
' Status").
If stillOK Then
    Set resultSet = grabber.GetRecordList (6)
    If resultSet.Count = 1 Then
        Set activityRankRec = resultSet.ItemByIndex (0)
    Else
        stillOK = false
        errorMsg = "Couldn't get activity rank."
    End If
End If

' - - - - -
' Set the new case history.
```

```
' -----  
If stillOK Then  
    Dim carriageReturn as String  
    Dim history as String  
  
    carriageReturn = CStr (GetConstant ("crlf"))  
    history = CStr (GetConstant (statusCode & "_History"))  
  
    ' Replace each of the variables in the history string with the  
    ' appropriate value.  
    history = SwapStr (history, "<oldStatus>", oldStatusRec.GetField ("title"))  
    history = SwapStr (history, "<newStatus>", newStatusRec.GetField ("title"))  
    history = SwapStr (history, "<userName>", userRec.GetField ("login_name"))  
    history = SwapStr (history, "<timeStamp>", CStr (Now ()))  
    history = SwapStr (history, "<CRLF>", carriageReturn)  
  
    ' Set the new history.  
    history = caseRec.GetField ("case_history") & history  
    caseRec.SetField "case_history", history  
  
End If
```

```
' -----  
' Create an activity entry, including our notes on this status  
' change. Link the new activity entry to both our user and the  
' String record indicating a "Status Change" activity.  
' -----  
If stillOK Then  
    notes = CStr (GetConstant (statusCode & "_Note"))  
    notes = SwapStr (notes, "<oldStatus>", oldStatusRec.GetField ("title"))  
    notes = SwapStr (notes, "<newStatus>", newStatusRec.GetField ("title"))  
  
    ' Create the activity entry. We will receive the objid of the  
    ' new activity record in the activityID parameter.  
    tempResult = create_act_entry _  
        (notes, userID, activityID, activityRank)  
  
    If tempResult <> 0 Then  
        stillOK = false  
        errorMsg = "Couldn't create activity entry."  
    End If  
End If
```

```
' -----  
' Fill in the status-change record, using our notes from  
' the activity log entry.  
' -----  
If stillOK Then  
    Set statusChangeRec = new Record  
    statusChangeRec.RecordType = "status_chg"  
    statusChangeRec.SetField "creation_time", App.CurrentDate  
    statusChangeRec.SetField "notes", notes  
End If
```

```
' -----  
' Relate all our records to each other.  
' -----  
If stillOK Then  
  
    ' Relate the case to its new status and the activity entry.  
    saver.UpdateRecord caseRec  
    saver.UpdateRecord newStatusRec
```

```
saver.RelateRecords caseRec, newStatusRec, "casesets2gbst_elm"  
saver.RelateRecordsToID _  
    caseRec, "act_entry", activityID, "case_act2act_entry"
```

```
' Relate the Status Change record to the case, the old status, and  
' the new status, and the activity entry.  
saver.InsertRecord statusChangeRec  
saver.UpdateRecord oldStatusRec  
saver.UpdateRecord userRec  
saver.RelateRecords statusChangeRec, caseRec, "case_status_chg2case"  
saver.RelateRecords statusChangeRec, oldStatusRec, "p_status_chg2gbst_elm"  
saver.RelateRecords statusChangeRec, newStatusRec, "c_status_chg2gbst_elm"  
saver.RelateRecordsToID _  
    statusChangeRec, "act_entry", activityID, "status_chg2act_entry"  
saver.RelateRecords statusChangeRec, userRec, "status_chger2user"
```

End If

```
' -----  
' Save the modified records into the database.  
' -----
```

```
If stillOK Then  
    tempResult = saver.Save  
    If tempResult <> 0 Then  
        stillOK = false  
        errorMsg = "Couldn't save records!"  
    End If  
End If
```

```
' Any errors?  
If not (stillOK) Then  
    global_errorPrint _  
        "case", caseID, "case_setStatus: " & errorMsg  
End If
```

End Sub

```
' -----  
' Global Utilities  
' -----
```

```
' -----  
' global_errorPrint  
' -----
```

```
Sub global_errorPrint (objectName as String, _  
    objectID as Long, _  
    message as String)  
  
    Debug.Print "(global_errorPrint) " &_  
        "Message from " & objectName & " " & CStr (objectID) &_  
        ": " & message
```

End Sub

```
' -----  
' SwapStr (sourceStr, searchStr, replaceStr) as String  
'  
' Searches through sourceString for ALL occurrences of searchString,  
' replaces them with replaceString, and returns the new string. If  
' searchString is not found, SwapStr returns sourceString.  
' -----
```

```
' The search is case-insensitive (i.e., you don't have to worry about
' capitalization).
'
' Parameters:
'   sourceStr:  the string containing the text you'd like to replace.
'   searchStr:  the substring to be replaced.
'   replaceStr: the string that will replace searchStr.
'-----
Function SwapStr (sourceStr as String, _
                 searchStr as String, _
                 replaceStr as String) as String

Dim strPos as Integer
Dim leftLen as Integer
Dim rightLen as Integer
Dim result as String

' Do a case-INsensitive search for searchStr in sourceStr.
result = sourceStr
strPos = InStr (1, result, searchStr, 1)      ' startPos, sourceStr, searchStr, case-sensitivity

' If we found it, then substitute replaceStr for searchStr.
While strPos > 0

    leftLen = strPos - 1
    rightLen = Len (result) - (strPos - 1) - Len (searchStr)

    result = Left (result, leftLen) &_
             replaceStr &_
             Right (result, rightLen)

    ' Repeat the search.
    strPos = InStr (1, result, searchStr, 1)

Wend

' Done!  If we didn't find searchStr, then we just return sourceStr.
SwapStr = result

End Function
```