

# Next Task Status Changes

**Project:** BroadRiver Clarify Form Customization: Workflow: Next Task Status Changes  
**Owner:** Ron Conescu  
**Date:** Thursday, September 14, 2000

This document describes the process of scheduling a subcase — allowing it to be closed by an end-user — once its predecessor tasks have been completed. The document also describes the process of notifying a user when all the subcases of a certain case have been completed.

The document has the following sections:

<a href="#">Overview</a> .....	1
<a href="#">Object-Oriented View</a> .....	2
<a href="#">event subcase.onStatusChange ()</a> .....	3
<a href="#">private event subcase.predecessorDone ()</a> .....	3
<a href="#">private function subcase.scheduleMe ()</a> .....	3
<a href="#">protected function case.subcaseDone ()</a> .....	4
<a href="#">Procedural View</a> .....	5
<a href="#">business rule subcase launchSuccessors</a> .....	5
<a href="#">file subcase launchSuccessors.cbs (myID)</a> .....	5
<a href="#">function subcase predecessorDone (myID)</a> .....	6
<a href="#">function subcase schedule (myID)</a> .....	6
<a href="#">function case checkComplete (myID)</a> .....	7
<a href="#">business rule case subcasesCompleted</a> .....	7
<a href="#">Utilities</a> .....	7
<a href="#">function subcase getPredecessors (subcaseID)</a> .....	8
<a href="#">function subcase getOutstandingPredecessors (subcaseID)</a> .....	8
<a href="#">function subcase getSuccessors (subcaseID)</a> .....	8
<a href="#">function subcase setStatus (subcaseID, statusCode)</a> .....	9
<a href="#">function case setStatus (caseID, statusCode)</a> .....	9
<a href="#">function global generateLogEntry (objectType, objectID, message)</a> .....	9
<a href="#">Changes required elsewhere in the system</a> .....	10
<a href="#">Glossary</a> .....	10

## Overview

When a workflow begins, a number of subcases are launched. For the most part, people may begin working on those subcases as soon as the subcase shows up in a queue. However, most of those subcases cannot be *closed* until their predecessor cases, as defined by the workflow, have *also* been closed.

So here's what we'll do: when all the subcases are first distributed, we'll set them to a particular status — "Awaiting Scheduling." Again, this means that people may work on those subcases,

but they can't close them yet. When someone completes a particular subcase — one that *can* be closed — we'll look at all the subcases that follow it (the “successor” subcases). Each of those subcases may have one or more predecessor cases that *haven't* been completed yet, so we'll check the status of all those predecessors. If all the predecessors to a subcase are complete, we'll set the status of that subcase to “Ready to Work,” informing the owner that she may now work on that subcase. If the owner has already begun working on it, its status will be set to “In Progress,” and we'll leave it alone — clearly, she knows she may already begin working on it. Either way, the owner of that subcase may now complete work on it, and close it out.

If a subcase is closed, and it's the last subcase in a Case, we'll set the Case to a special status — “Workflow Complete” — and send a message to the owner of that Case.

So, in summary: when a Subcase completes, we will “launch” all its successor cases whose **other** predecessors are **also** complete. If a Subcase has no successors, and it's the last thing to be done in the workflow, we'll set the Case to a special status, and notify the owner.

We'll accomplish this as follows:

- Set up a Clarify **Business Rule** that fires when a subcase has been completed. The business rule calls...
- a piece of **ClearBasic script** (Clarify's programming language), which performs the subsequent logic, and sets the appropriate subcases (and/or the owning Case) to their new statuses. At the end, when the Case's status changes, we'll...
- call another Business Rule informing the Case owner that the workflow has been completed.

## Object-Oriented View

Here's an object-oriented view of how we can accomplish this. The OO viewpoint is really a mental trick, which allows us to check our work: it allows us to keep the data-processing code as close as possible to the data being processed, and thus helps us see whether we're missing any pieces, or whether a certain piece of code is doing too much work. It also gives us a convenient way to improve our code: we can go straight to the function responsible for a certain action, without having to do a whole lot of hunting around for that action.

In the pseudocode below, several Subcases talk to each other, and some of those talk to their Case: a subcase gets the original “completed” message, and tells each of its successor subcases that it's done. Each of the successors, in turn, checks to see whether it is safe to move into the “Ready to Work” state, and puts itself in that state if so. If a subcase discovers that it has no more successors, it tells its Case; the Case checks to see if all its subcases are done, and, if so, informs the end-user.

Our story begins when a user closes a Subcase. This changes its status, which magically invokes the `onStatusChange` function below.

---

```
event subcase.onStatusChange ()
```

*I am a Subcase that someone has been working on. This function is called when my status changes. If I have changed to the Complete state, I'll tell each of my successors that I, as one of their predecessors, am done.*

```
myCase = the case that owns me

if my status = complete
  if I have any successors
    for all successors
      send predecessorDone() message to successor
    next successor
  else
    send subcaseDone() message to myCase
  end if
end if

end onStatusChange
```

---

```
private event subcase.predecessorDone ()
```

*I am a Subcase that is waiting for my predecessors to finish. This function is called by one of my predecessor subcases, when he's been Completed. When I get this message, I'll check to see whether all of my predecessors are Complete; if so, I'll flip myself into a schedulable state, which allows my human owner to Close me.*

*Note: I don't really care which predecessor finished; since I have to check them all, I'll end up looking at him, too.*

```
ready = true

for all predecessors
  if predecessor is not complete
    ready = false
    exit for
  end if
next predecessor

if ready
  scheduleMe ()
end if

end function
```

---

```
private function subcase.scheduleMe ()
```

*I am a Subcase who has just decided that all my predecessors are Complete. This function is called when I have determined that I can be scheduled — i.e., that an end-user may now close me out.*

*In this function, I'll change my status to "Ready to Work," so that an end-user knows she can start working on me (and that such work is expected). If the end-user has already begun working on me, I won't interfere.*

*I will also log the fact that I am now ready to be worked.*

```
Log the fact that this I am ready to be worked

if the user has NOT modified me
    set my status to "Ready to Work"
    Log the fact that I have changed my status.
else
    do nothing
end if

end function
```

---

```
protected function case.subcaseDone ()
```

*I am a Case who is waiting for my Subcases to be Completed. This function is called when a subcase in my workflow Completes, and has no successors. If that subcase is my last subcase to finish, I change my status and inform my owner. I will also log the fact that all my subcases have been completed.*

*Note: I don't really care which subcase finished; since I have to check them all, I'll end up looking at him, too.*

```
ready = true

for all subcases
    if subcase has no successors AND
        subcase is NOT complete then

        ready = false
        exit for
    end if
next subcase

if ready
    Set my status to "Workflow Complete"
    Send my owner a message

    Log the fact that all my subcases are complete
    Log the fact that I have changed my status
end if

end function
```

## Procedural View

Here's how we might implement the logic above. Each of the items below serves roughly the same purpose as the object-oriented one above; there are slight differences in names and parameters, though, to reflect Clarify's procedural view of things. Also, the whole mess is surrounded by a pair of business rules, since that's how Clarify knows to start an event: the first Business Rule kicks off the whole process, and the last Business Rule informs the user when the whole Case is complete.

---

```
business rule subcase_launchSuccessors
```

*This business rule is invoked when a subcase's status is set to "complete." It invokes a ClearBasic function which tells the subcase's successors which subcase is done.*

```
when subcase status changed to "Complete"  
    launch subcase_launchSuccessors.cbs (subcaseObjectID)  
  
end business rule
```

---

```
file subcase_launchSuccessors.cbs (myID)
```

*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:*

myID: the objid of the subcase whose status has changed.

*Variables:*

caseID: the objid of the case of whom the target is a subcase.  
successorList: the list of all successors to the target subcase.  
successorID: the objid of any given successor, obtained as we iterate through the list of those successors.

```
successorList = subcase_getSuccessors (myID)  
  
if target has any successors  
    for all successors in successorList  
        subcase_predecessorDone (successorID)  
    next successor  
else  
    case_checkComplete (caseID)  
end if  
  
end onStatusChange
```

---

```
function subcase_predecessorDone (myID)
```

*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:**

myID: the objid of the subcase to be scheduled.

**Variables:**

toDoList: the list of all my successors.

```
toDoList = subcase_getOutstandingPredessors (myID)

if toDoList is empty
  subcase_schedule (myID)
end if

end function
```

---

```
function subcase_schedule (myID)
```

*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:**

myID: the objid of the subcase to be scheduled.

```
global_generateLogEntry ("subcase", myID, "Predecessors are done.")

if the user has NOT modified the subcase
  subcase_setStatus (myID, "Ready to Work")

  global_generateLogEntry
    ("subcase", myID, "Set status to 'Ready to Work'.")
else
  do nothing
end if

end function
```

---

```
function case_checkComplete (myID)
```

*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.*

*Variables:*

```
  myID:      the objid of the Case to which the completed Subcase
              belongs.

  ready = true

  for all subcases of myID
    if subcase has no successors AND
      subcase is NOT complete then

      ready = false
      exit for
    end if
  next subcase

  if ready
    case_setStatus (myID, "Workflow Complete")
    invokes business rule case_subcasesCompleted

    global_generateLogEntry
      ("case", myID, "All subcases in workflow completed.")
    global_generateLogEntry
      ("case", myID, "Set status to 'Workflow Complete'.")

  end if

end function
```

---

```
business rule case_subcasesCompleted
```

*This business rule is invoked when we change a case’s status to “Workflow Complete.” It sends a message to the owner of that Case.*

```
  when case status changed to "Workflow Complete"
    message to case owner:
      "All Subcases of case id XXX have been completed."

end business rule
```

## Utilities

Throughout all of the processing we just saw, there were several tasks that had to be done repeatedly: determine how many predecessors have yet to be completed, say, or acquire a list of a subcase’s predecessors or successors. We will likely split those off into individual pieces of code that can be called independently; this saves lots of effort, prevents typos, and allows us to fix bugs faster. We’ll refer to these pieces of code as “utility functions”; here they are:

---

```
function 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.

Look through the Workflow Schedule table for predecessors of subcaseID

Return the resulting list of subcases

```
end function
```

---

```
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.

```
myList = subcase_getPredecessors (subcaseID)
```

From myList, select those predecessors that have not yet been completed.

Return that list of selected predecessors -- even if the list is empty.

```
end function
```

---

```
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.

Look through the Subcase Schedule table for subcases to whom

subcaseID is a predecessor

Return the resulting list of subcases

end function

---

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 constant representing the desired status.

Look up statusCode in a list of status codes  
Select the matching status  
Set the status of subcaseID to that status

end function

---

function **case\_setStatus** (caseID, statusCode)

*This function sets the status of the specified case 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:*

caseID: the objid of the case whose status you wish to set.  
statusCode: the constant representing the desired status.

Look up statusCode in a list of status codes  
Select the matching status  
Set the status of caseID to that status

end function

---

function **global\_generateLogEntry** (objectType, objectID, message)

*This function generates a Clarify-standard log entry about the specified object.*

*Parameters:*

objectType: the type of the object about which to log this  
message -- case, subcase, etc.  
objectID: the objid of the object in question.  
message: the log entry itself.

Add message to the log for the specified object.

end function

## Changes required elsewhere in the system

- On the Edit Subcase screen:
  - if current status = “Awaiting Scheduling,” do *not* permit the end-user to Close the subcase.  
[Subsumed into: if any subcase has any predecessors outstanding, do not permit the end-user to Close the subcase.]
  - when the subcase is Closed, set its status to Complete.
  - Display the meaning of each status next to the pop-up menu of status codes.

## Glossary

**Case**..... A task to be completed, or a problem to be solved. For the purposes of this document, cases are attached to [workflows](#), so that the case will not be completed until a number of component [subcases](#) are also completed.

**dependency**..... A relationship between two subcases, defined as follows: if a subcase *B* depends on another subcase *A*, then someone may begin working on *B*, but *B* itself cannot be completed until *A* is also complete. We also say that *A* is a [predecessor](#) to *B*, and that *B* is a [successor](#) to *A*. As a side effect, any subcases that depend on *B* also cannot complete until *A* has been completed.

**duration**..... (1) The amount of time taken to complete a [subcase](#). (2) The amount of time that we expect a subcase to take.

**owner** ..... The person who is responsible for completing a particular [case](#) or [subcase](#).

**predecessor** ..... A subcase *A* on which another subcase *B* [depends](#); *A* must be completed before *B* can be completed.

**status** ..... The current state of a particular case or subcase in a [workflow](#). The end-user may change the status to reflect the current state of work; the status may also change behind the scenes, to inform the user of the next phase of work expected to be completed in the workflow. [Subcases](#) have these statuses:

**Awaiting Scheduling**..... Work has not yet begun on a case, and its predecessors are not yet completed. The owner may begin working on the subcase, but will not be able to complete it until all its [predecessors](#) are complete.

**Complete** ..... The user has closed the subcase.

- Error** ..... The user has determined that there is a problem with the subcase.
- In Progress** ..... The owner has begun working on this subcase. The owner sets this status herself.
- On Hold** ..... The user has determined that she cannot complete the subcase for some unusual reason.
- Pending** ..... [Are we still using this?]
- Ready to Work** ..... The predecessors to this subcase are completed; work may now begin on this subcase. Furthermore, it is expected that work will begin promptly.

And [Cases](#) have at least this one special status:

- Workflow Complete** ..... The status of a Case indicating that all its subcases have been completed. When this status is set, the owner will receive a message saying so.

Please note that Clarify maintains internal status codes, different from these. The statuses defined here are stored separately from Clarify's internal statuses, have different meanings, and affect the system in different ways.

- Subcase** ..... A task to be completed, representing a component of a larger problem — the [Case](#) that owns this subcase. For the purposes of this document, subcases are typically a part of a [workflow](#): a set of dependencies that determine when, and under what circumstances, the owning Case will be complete.
- successor** ..... A subcase *B* which [depends](#) on a subcase *A*; the successor cannot be completed until all of its predecessors are complete.
- workflow** ..... A set of [Subcases](#) that must be completed before the [Case](#) that owns them may be completed. The workflow defines two attributes of these subcases: their [dependencies](#) on each other, and the expected [duration](#) of each subcase.