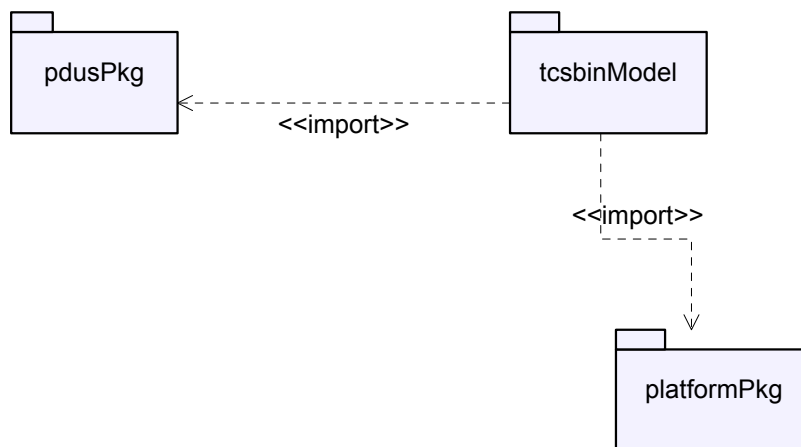
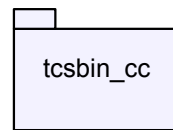
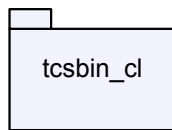
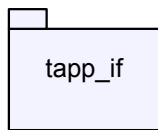
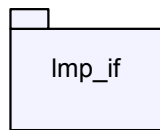


// Motorola Confidential Proprietary
Copyright 2007, Motorola Inc.



package pdusPkg

// Motorola Confidential Proprietary
Copyright 2007, Motorola Inc.



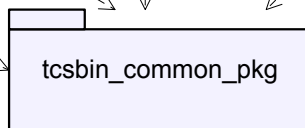
<<import>>

<<import>>

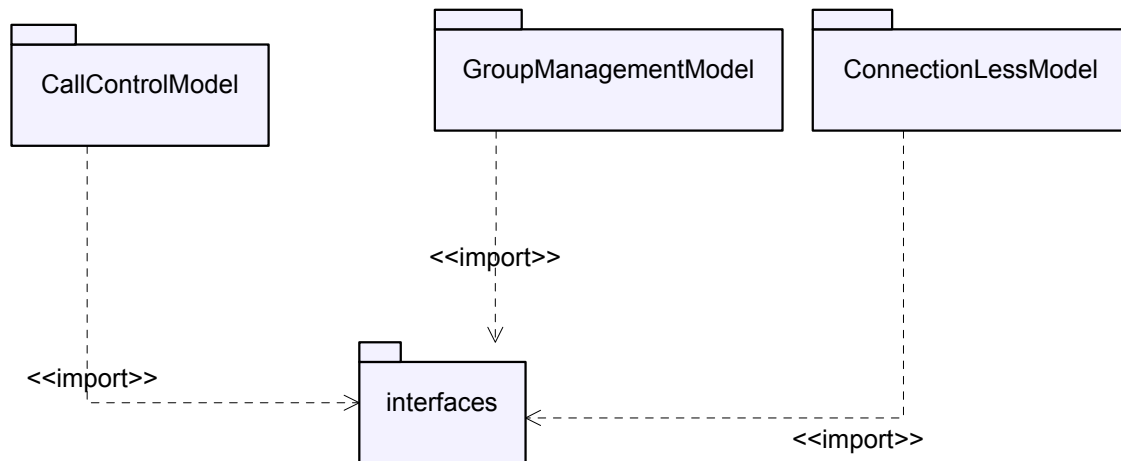
<<import>>

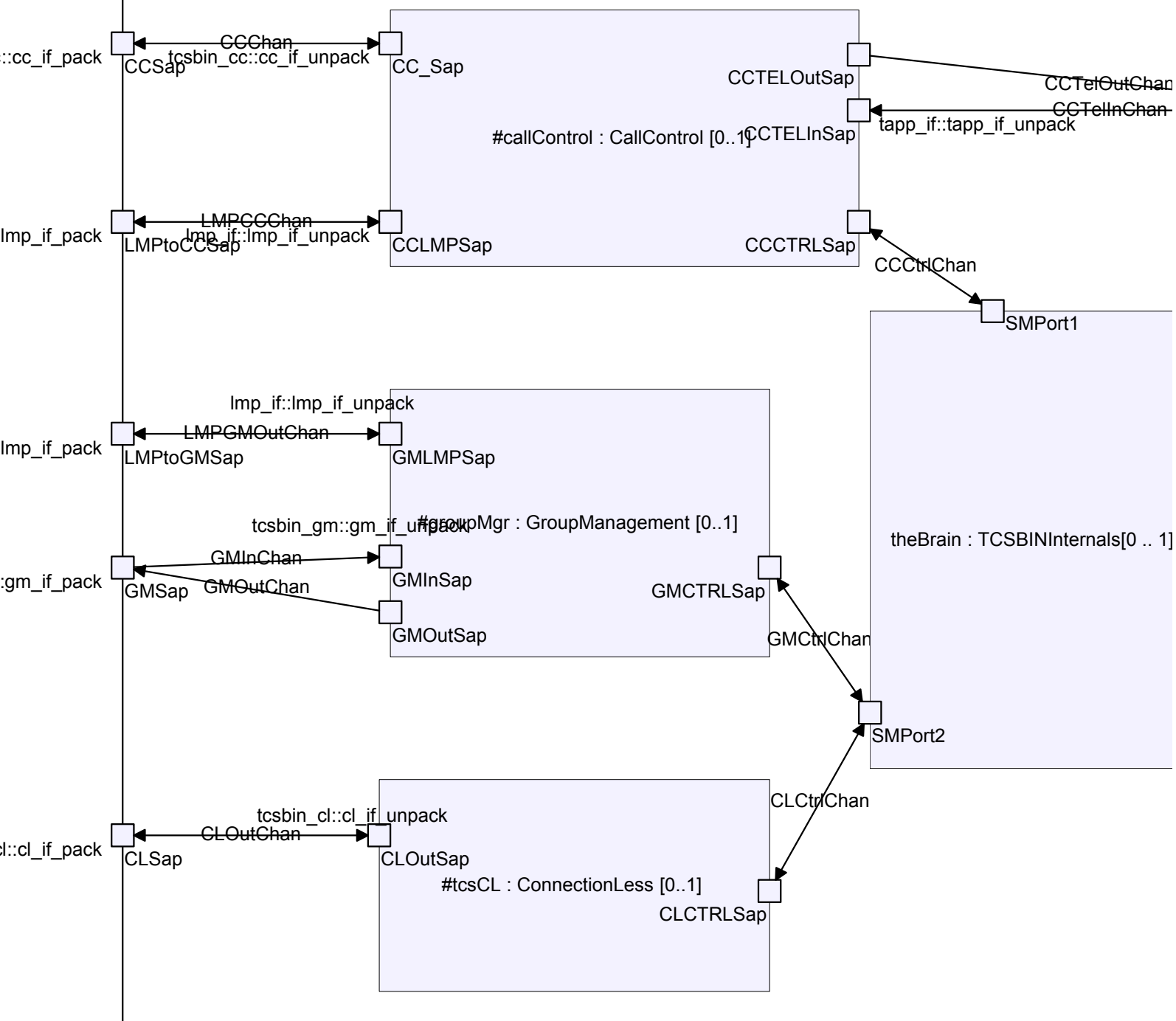
<<import>>

<<import>>



// Motorola Confidential Proprietary
Copyright 2007, Motorola Inc.





tapp_if::tapp_if_pack

TELSap

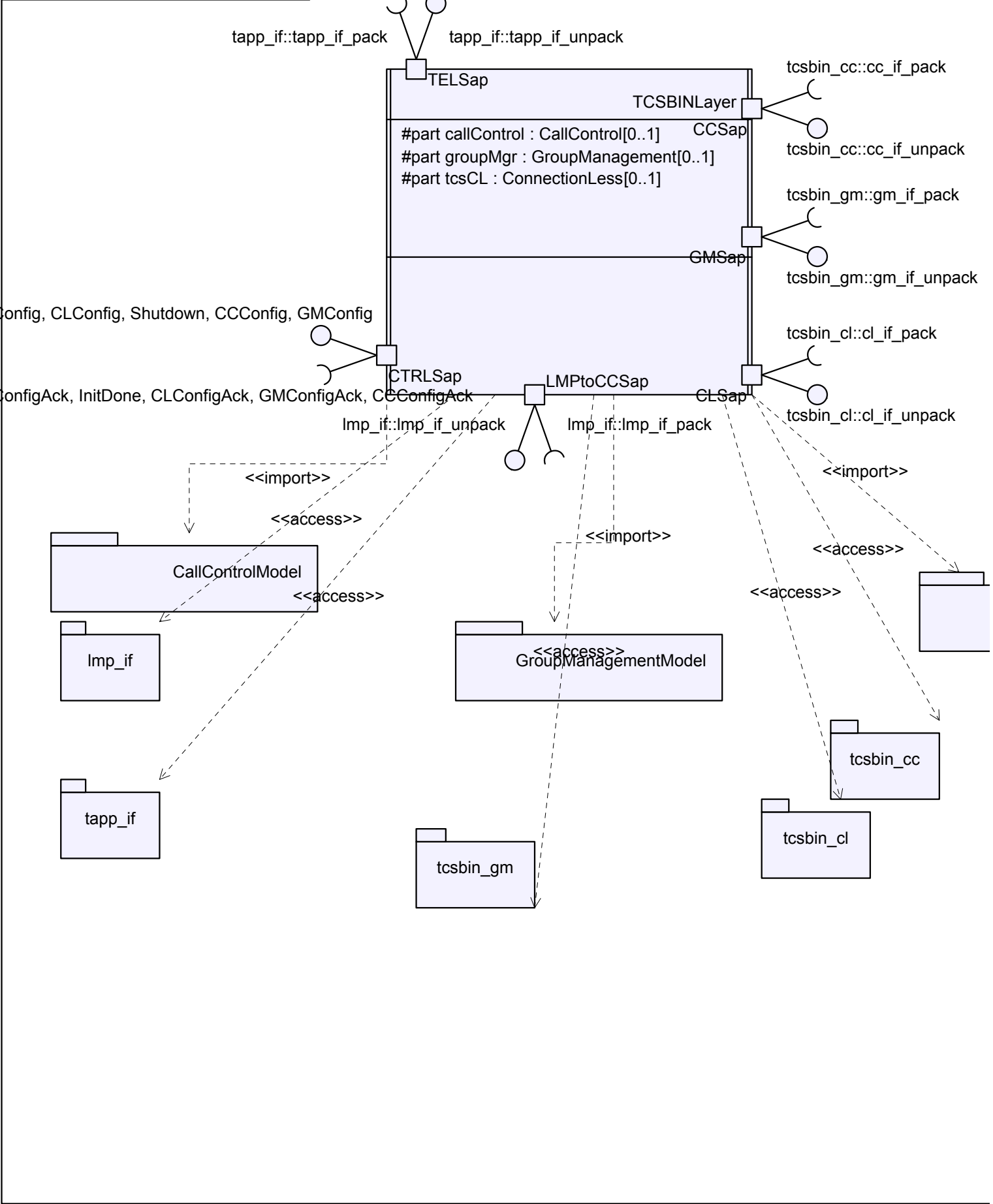
CTRL

CtrlChan

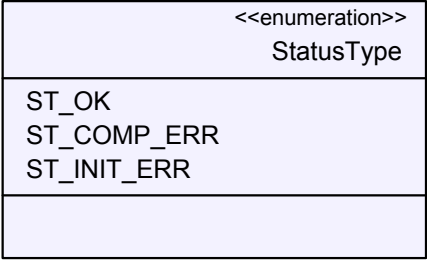
CTRLSap

/ 1

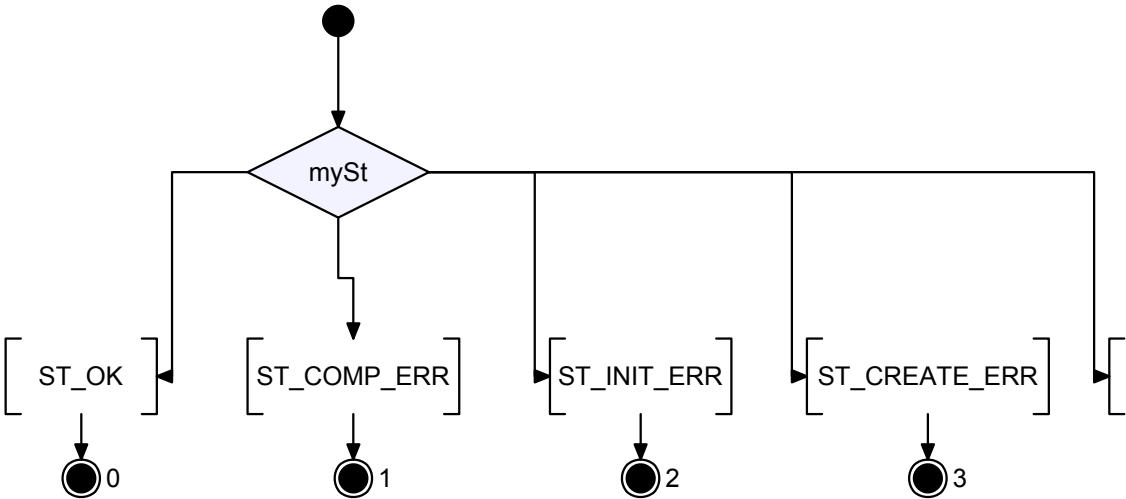
active class TCSBINLayer

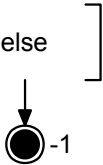


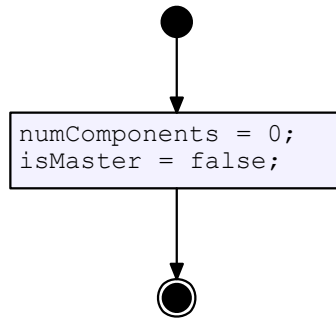
ConnectionLessModel



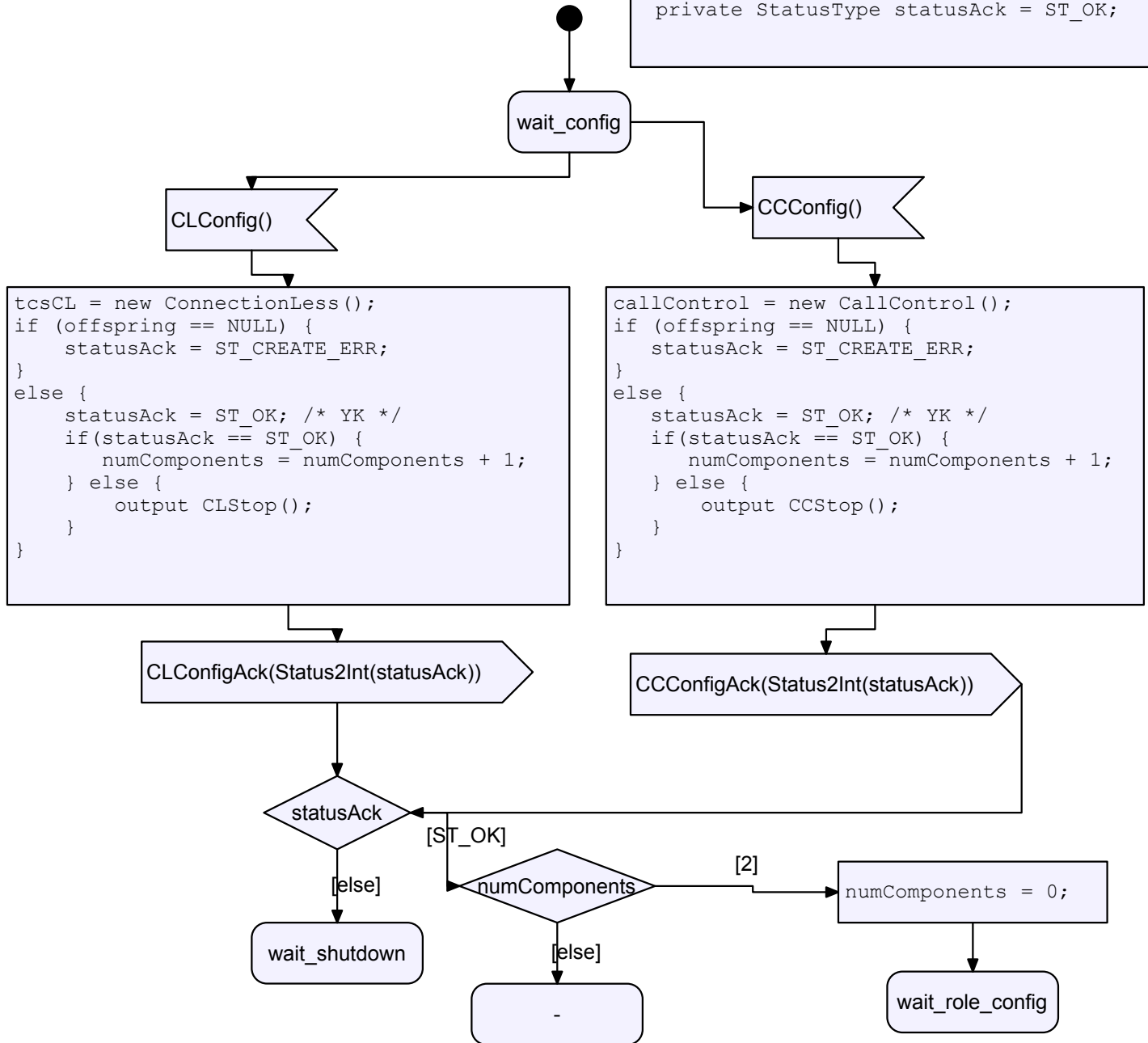
Integer Status2Int(StatusType mySt)







```
/* Temporary Variables */
private StatusType statusAck = ST_OK;
```

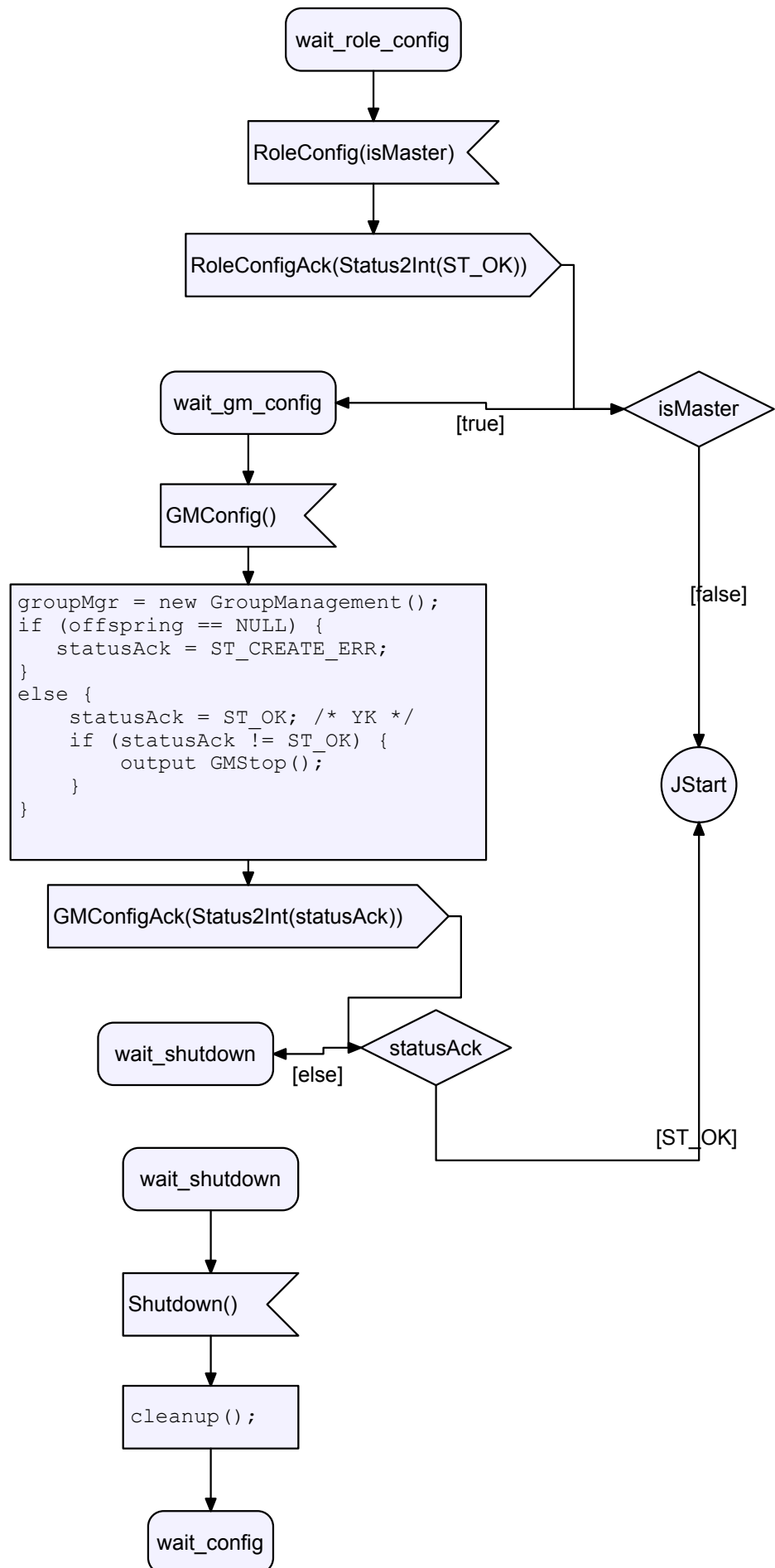


JStart

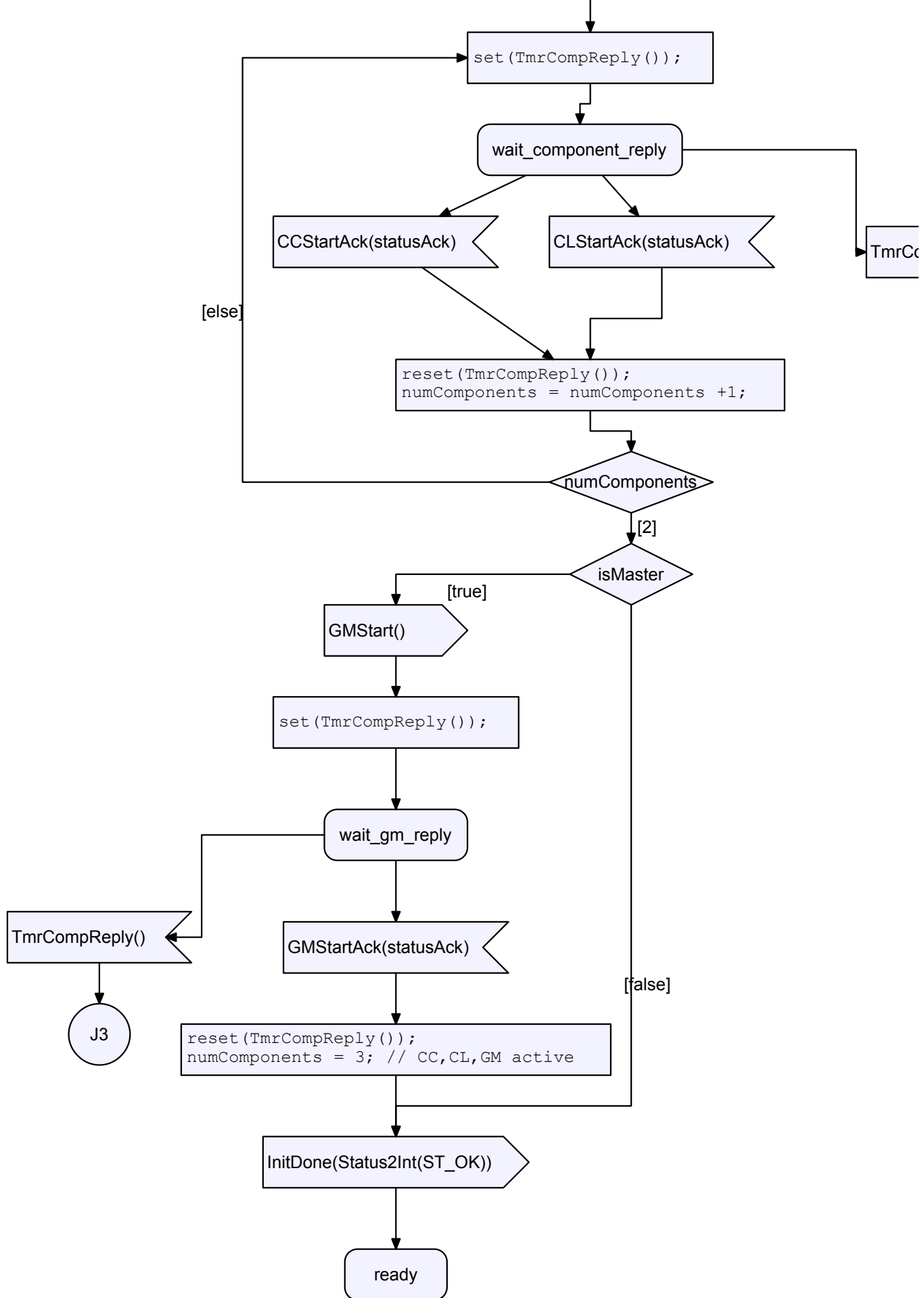
// At this point, all the modules are configured properly.

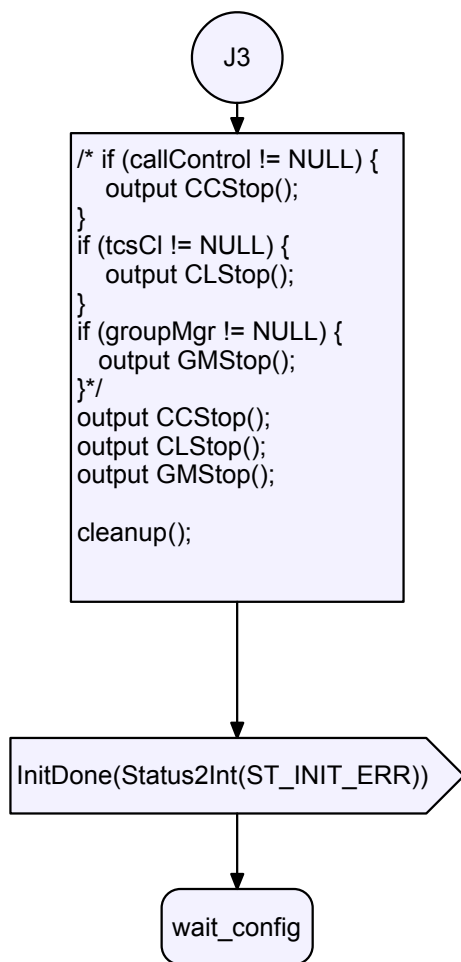
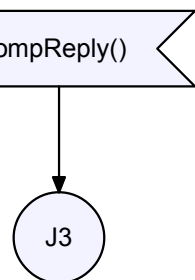
CCStart()

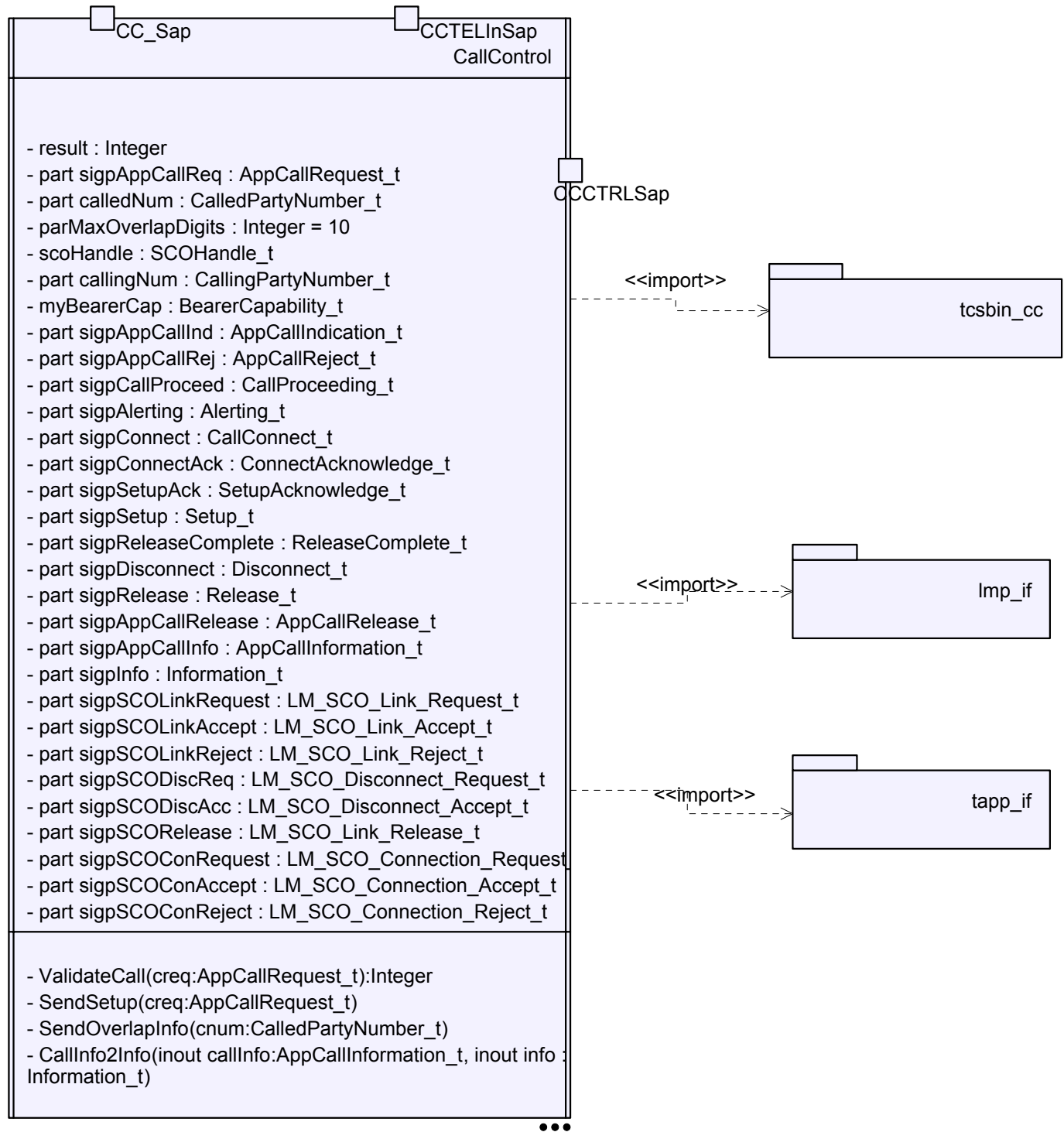
CLStart()



ed and





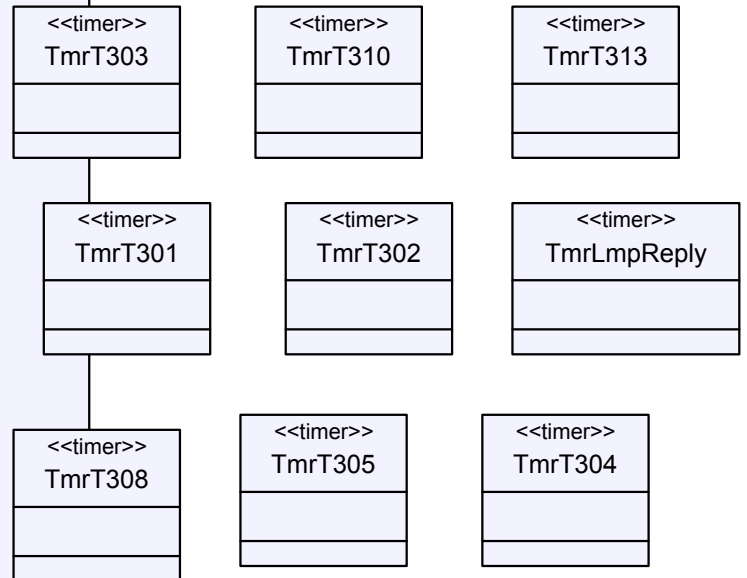


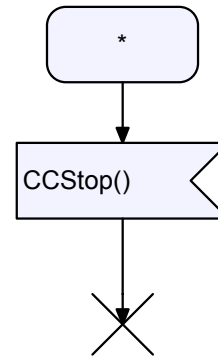
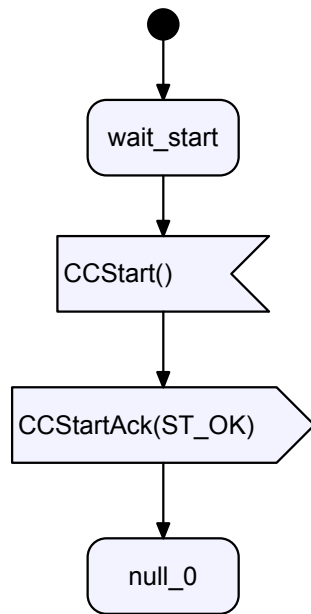
active class CallControl

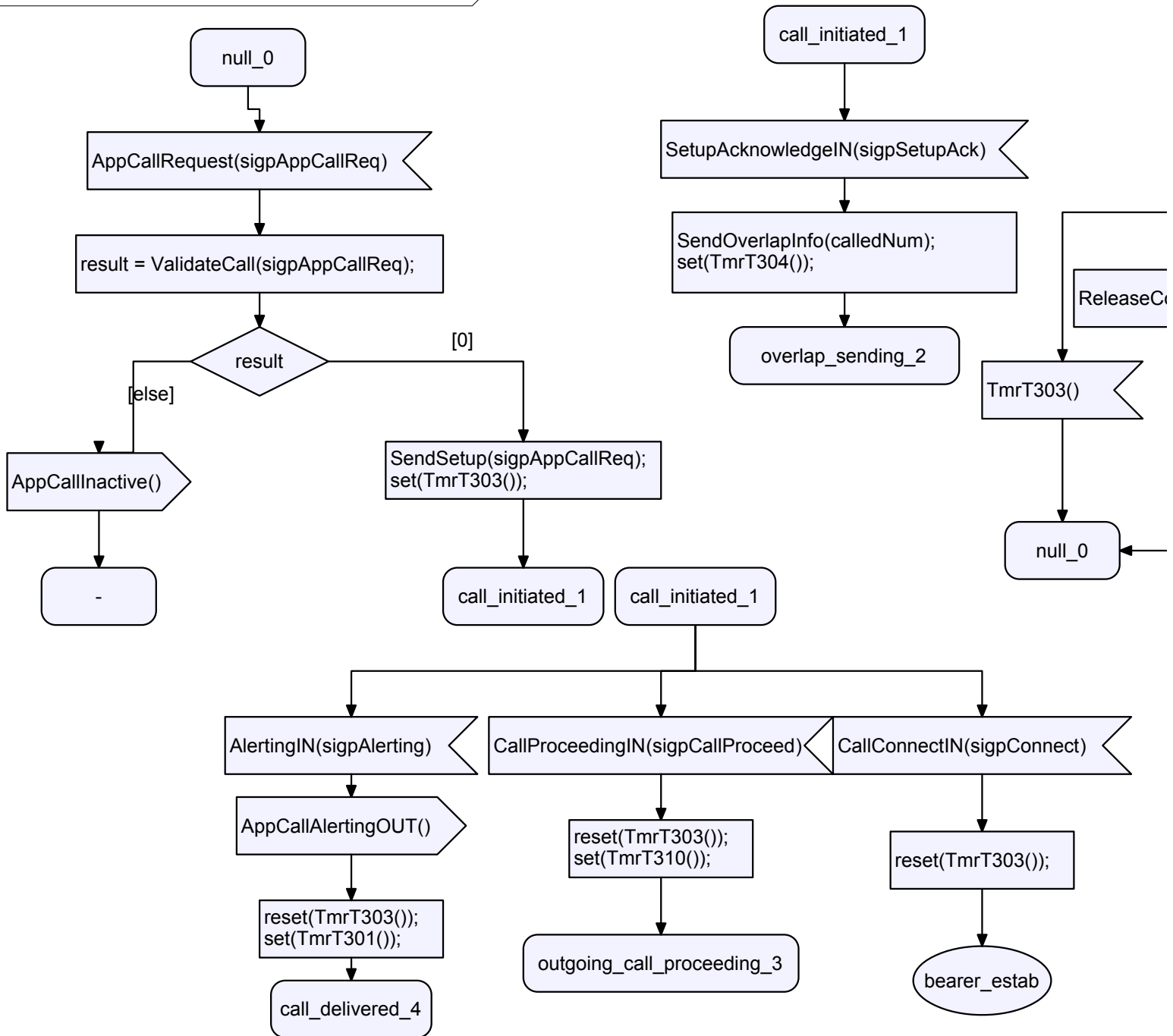
```
/* Local Variables */
private Integer result;
private CalledPartyNumber_t calledNum;
private CallingPartyNumber_t callingNum;
private SCOHandle_t scoHandle;
private BearerCapability_t myBearerCap;

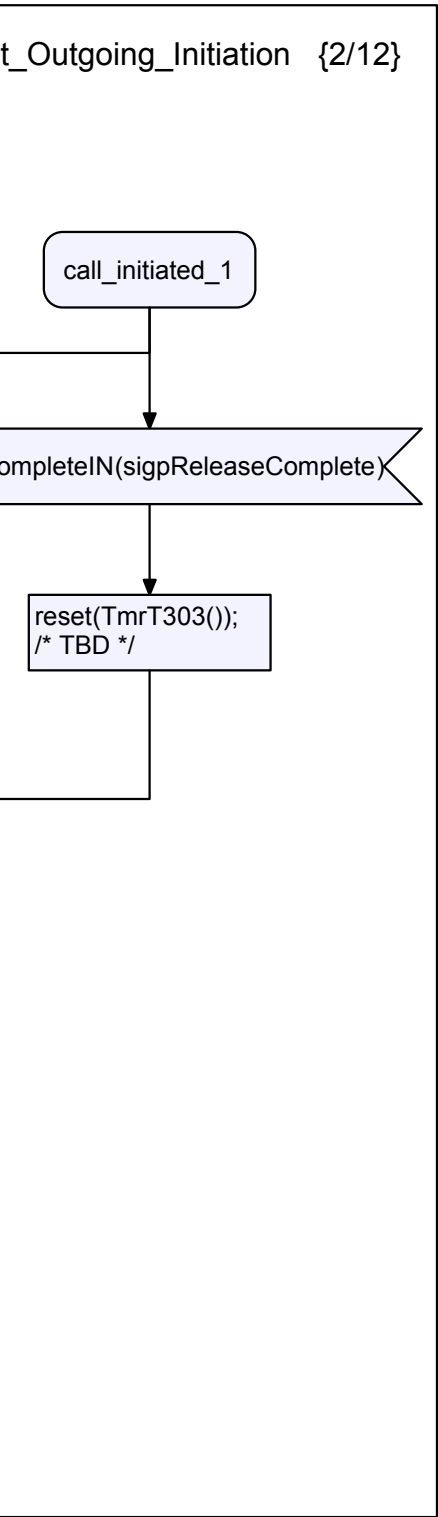
/* Initialization Parameters */
private Integer parMaxOverlapDigits = 10;

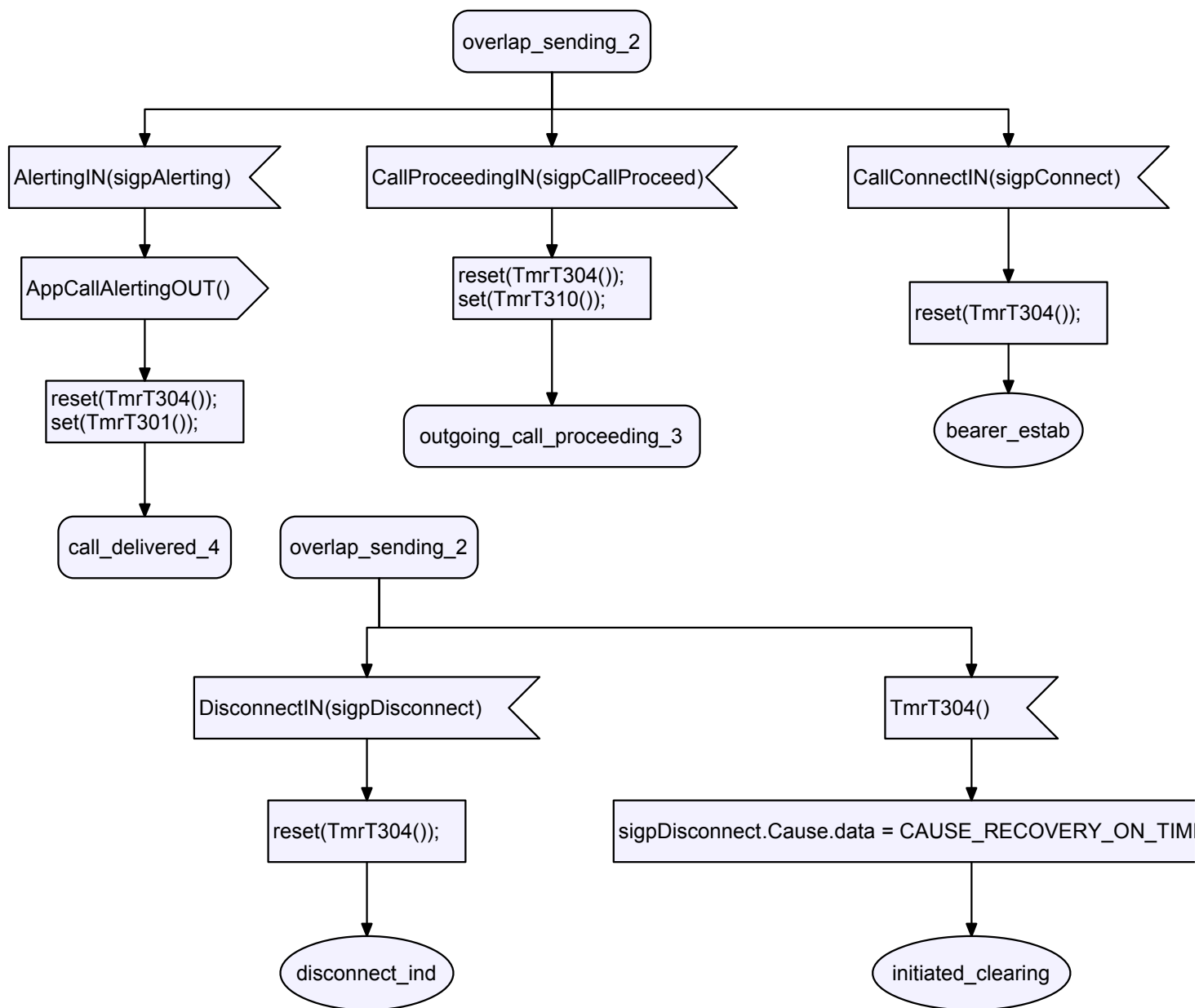
/* Signal Parameters */
private AppCallRequest_t sigpAppCallReq;
private AppCallIndication_t sigpAppCallInd;
private AppCallReject_t sigpAppCallRej;
private CallProceeding_t sigpCallProceed;
private Alerting_t sigpAlerting;
private CallConnect_t sigpConnect;
private ConnectAcknowledge_t sigpConnectAck;
private SetupAcknowledge_t sigpSetupAck;
private Setup_t sigpSetup;
private ReleaseComplete_t sigpReleaseComplete;
private Disconnect_t sigpDisconnect;
private Release_t sigpRelease;
private AppCallRelease_t sigpAppCallRelease;
private AppCallInformation_t sigpAppCallInfo;
private Information_t sigpInfo;
private LM_SCO_Link_Request_t sigpSCOLinkRequest;
private LM_SCO_Link_Accept_t sigpSCOLinkAccept;
private LM_SCO_Link_Reject_t sigpSCOLinkReject;
private LM_SCO_Disconnect_Request_t sigpSCODiscReq;
private LM_SCO_Disconnect_Accept_t sigpSCODiscAcc;
private LM_SCO_Link_Release_t sigpSCORElease;
private LM_SCO_Connection_Request_t sigpSCOConRequest;
private LM_SCO_Connection_Accept_t sigpSCOConAccept;
private LM_SCO_Connection_Reject_t sigpSCOConReject;
```



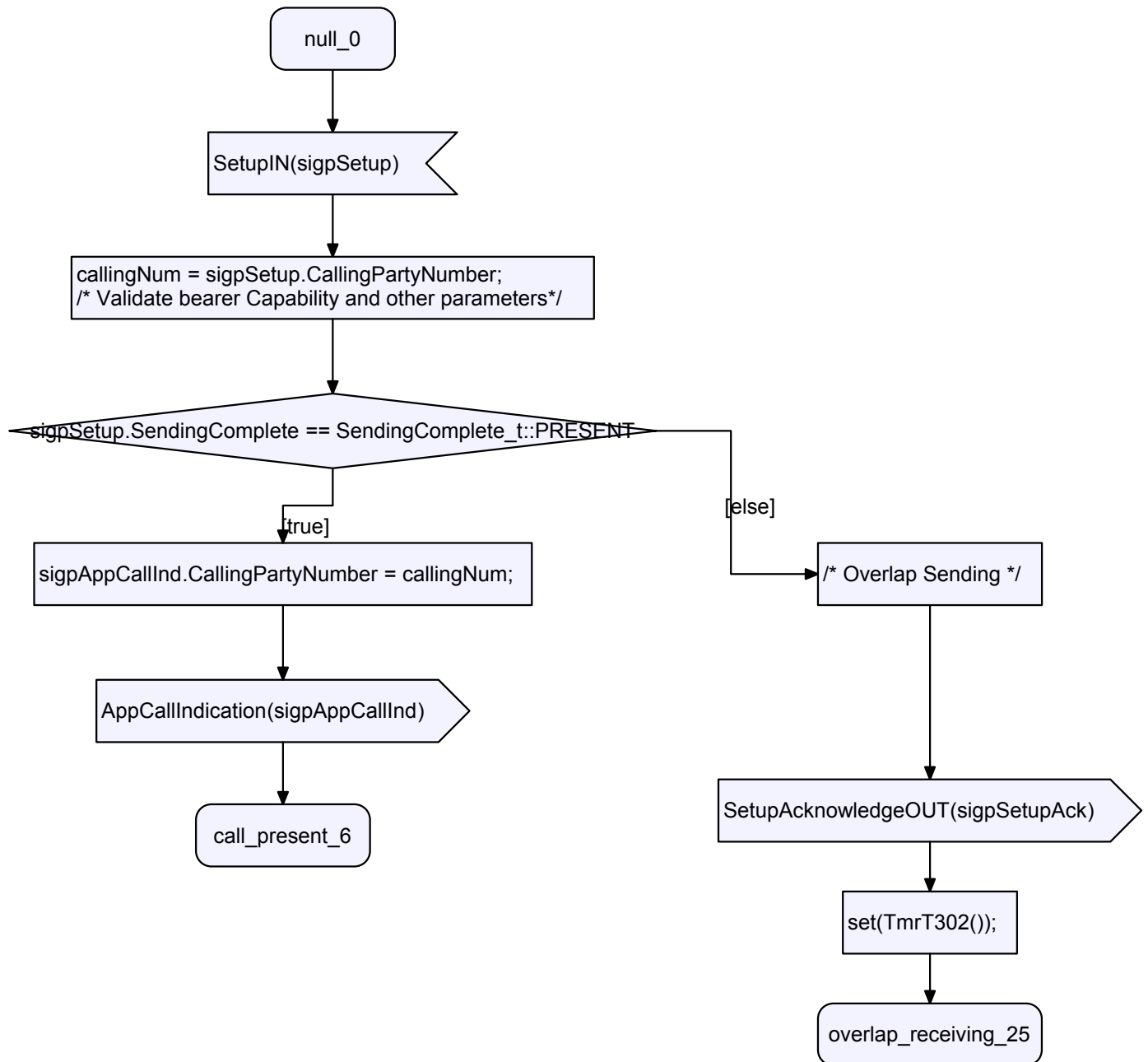


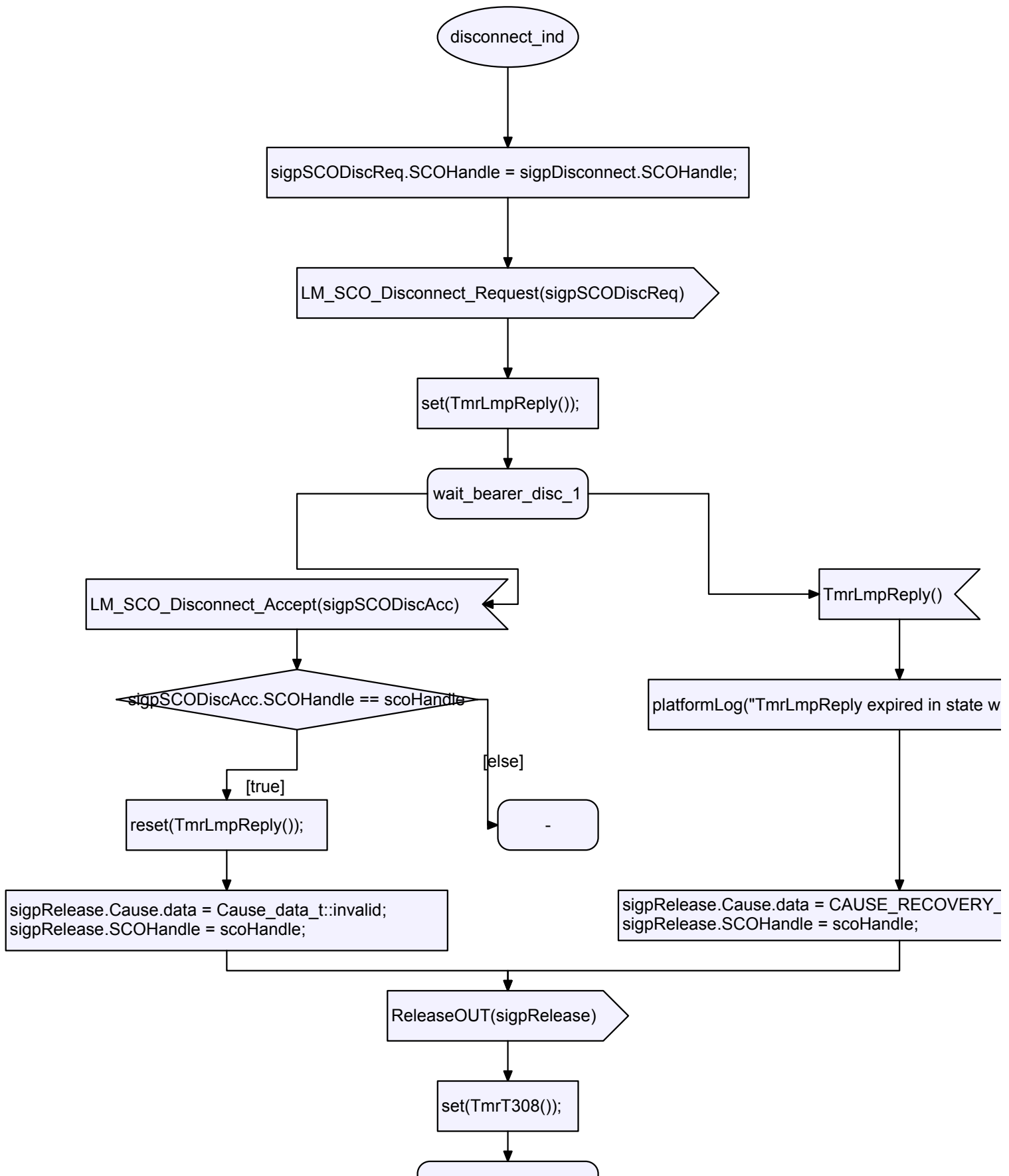






ER_EXPIRY;





wait_bearer_disc");

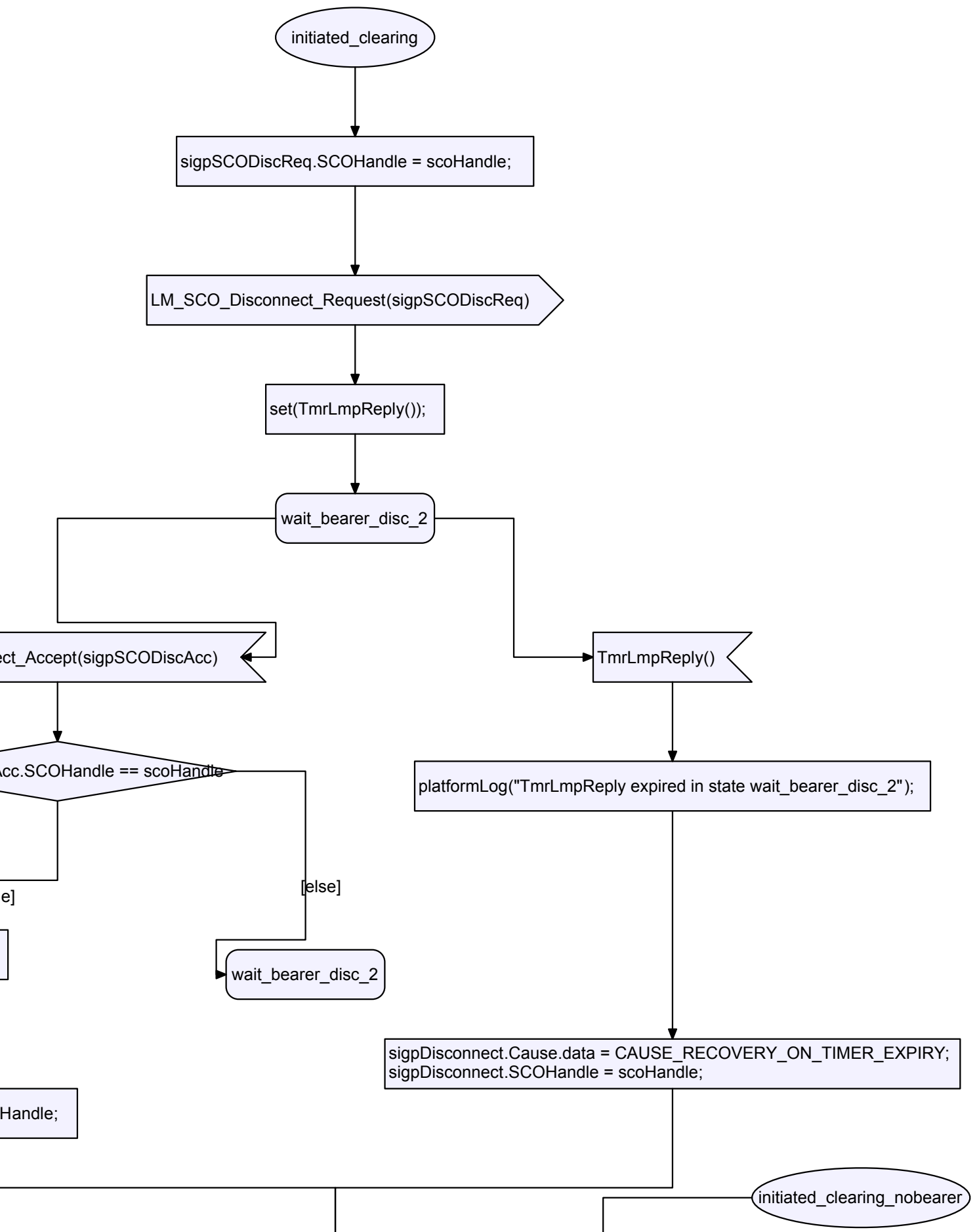
_ON_TIMER_EXPIRY;

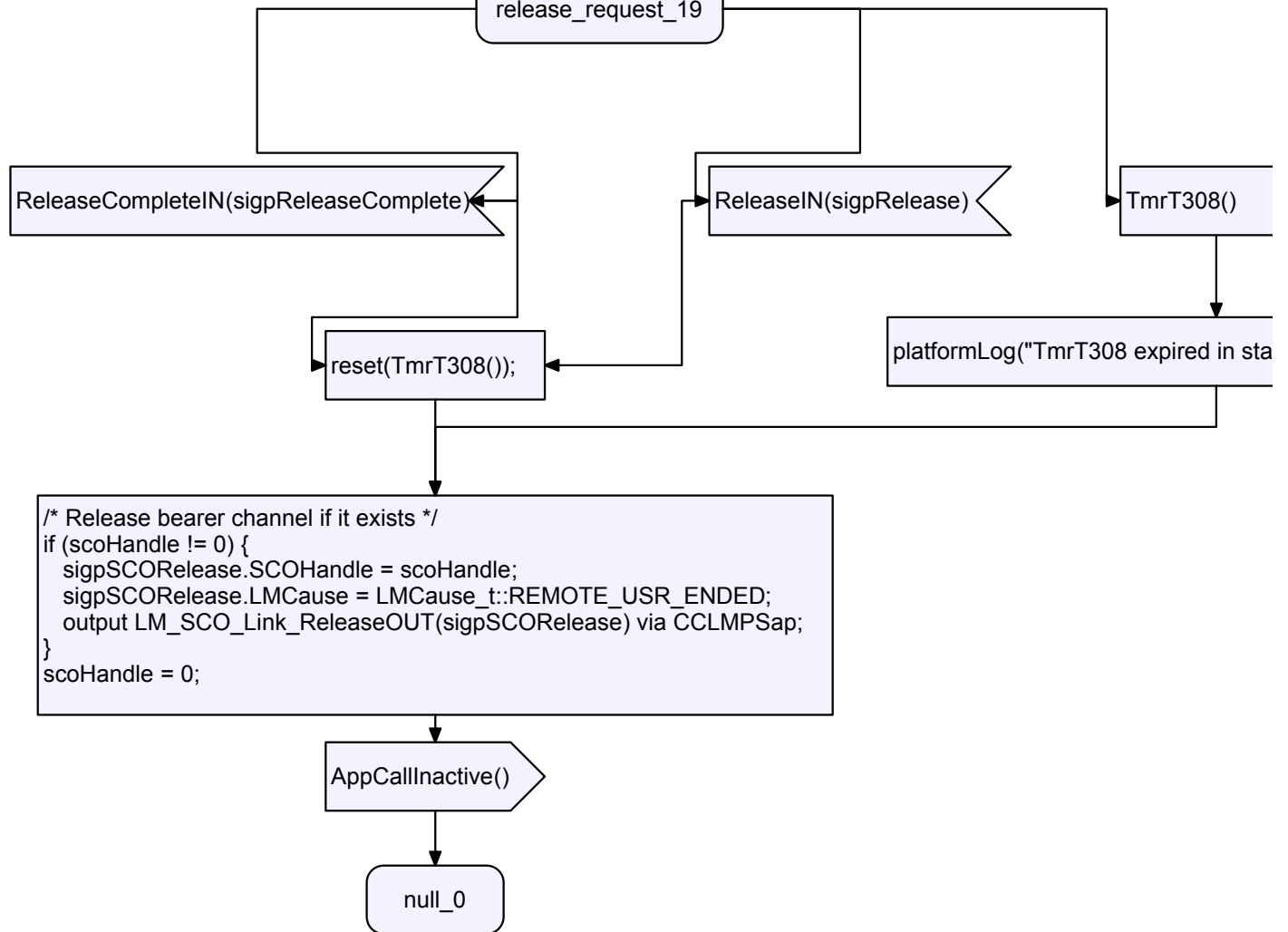
LM_SCO_Disconne

sigpSCODiscA

[tru
reset(TmrLmpReply());

sigpDisconnect.SCOHandle = sco







te release_request_19");

ReleaseIN(si

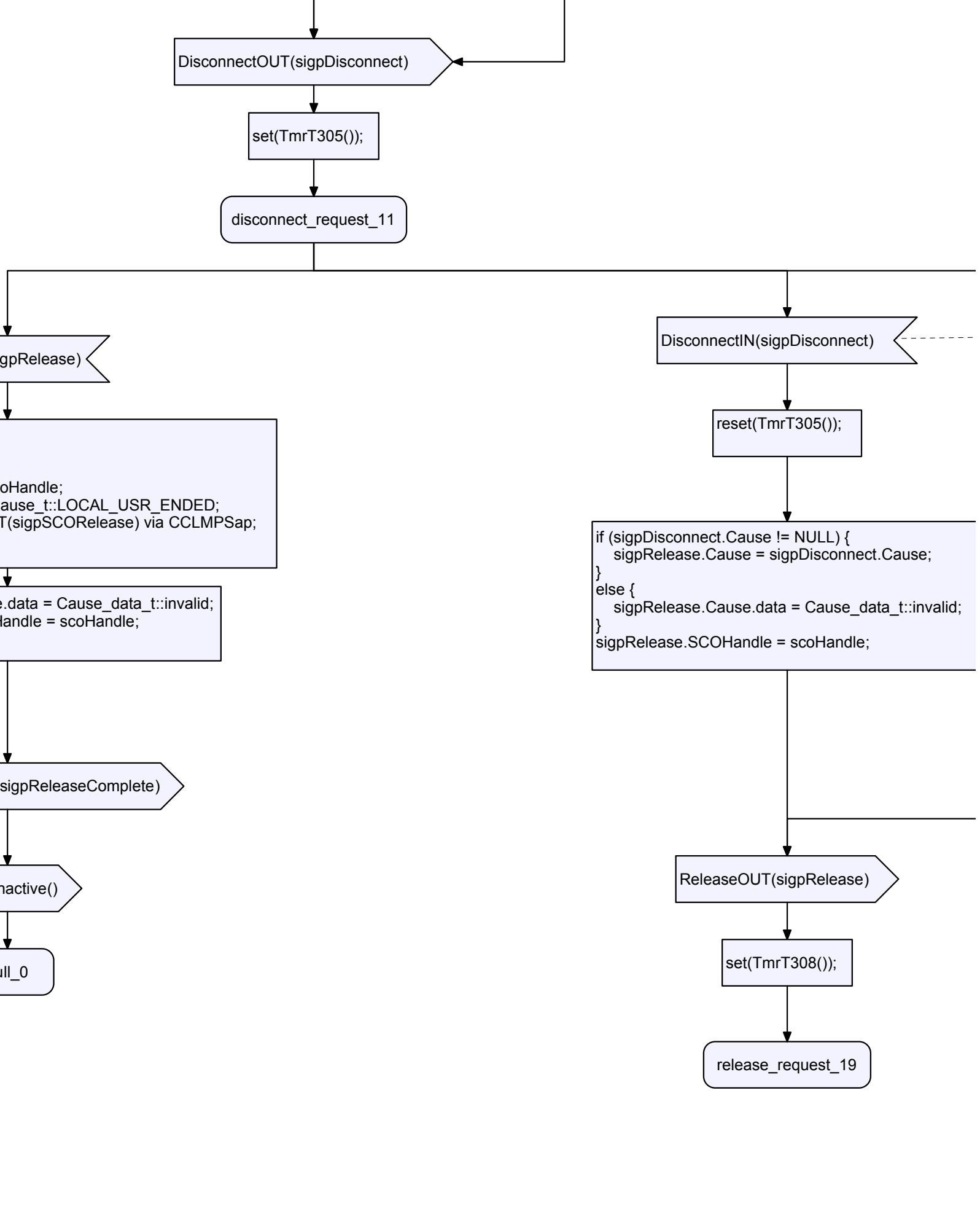
```
reset(TmrT305());  
/* Check if a bearer channel exists */  
if (scoHandle != 0) {  
    sigpSCORelease.SCOHandle = scoHandle;  
    sigpSCORelease.LMCause = LMCauseRelease;  
    output LM_SCO_Link_ReleaseOUT;  
}
```

```
sigpReleaseComplete.Cause = CauseRelease;  
sigpReleaseComplete.SCOHandle = scoHandle;  
scoHandle = 0;
```

ReleaseCompleteOUT(

AppCallIn

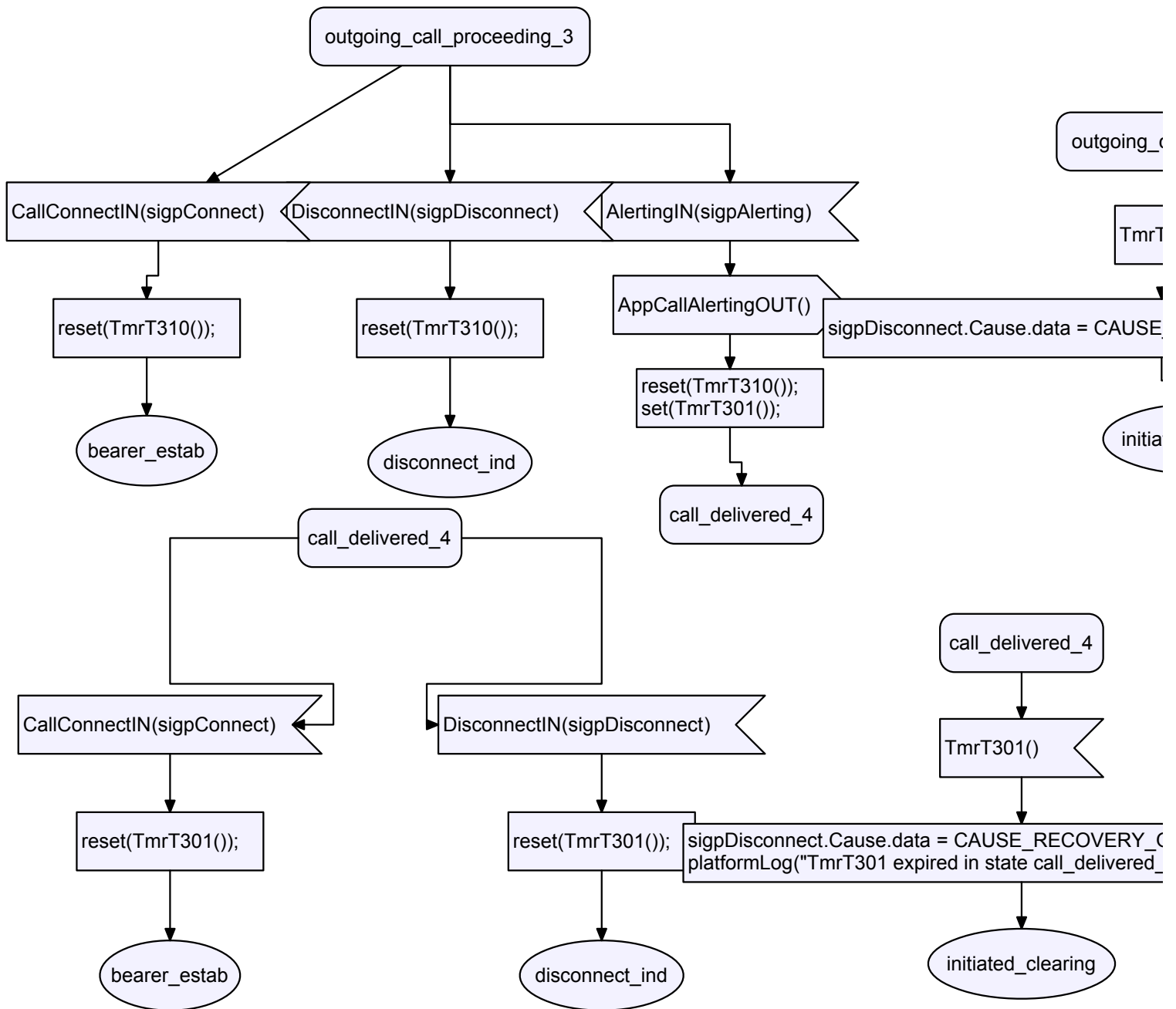
nu



// Disconnect Collision

TmrT305()

```
platformLog("TmrT305 expired in state disconnect_request_11");  
sigpRelease.Cause.data = CAUSE_RECOVERY_ON_TIMER_EXPIRY;  
sigpRelease.SCOHandle = scoHandle;
```

Establishment_Outgoing_ {6/12}
delivered

call_proceeding_3

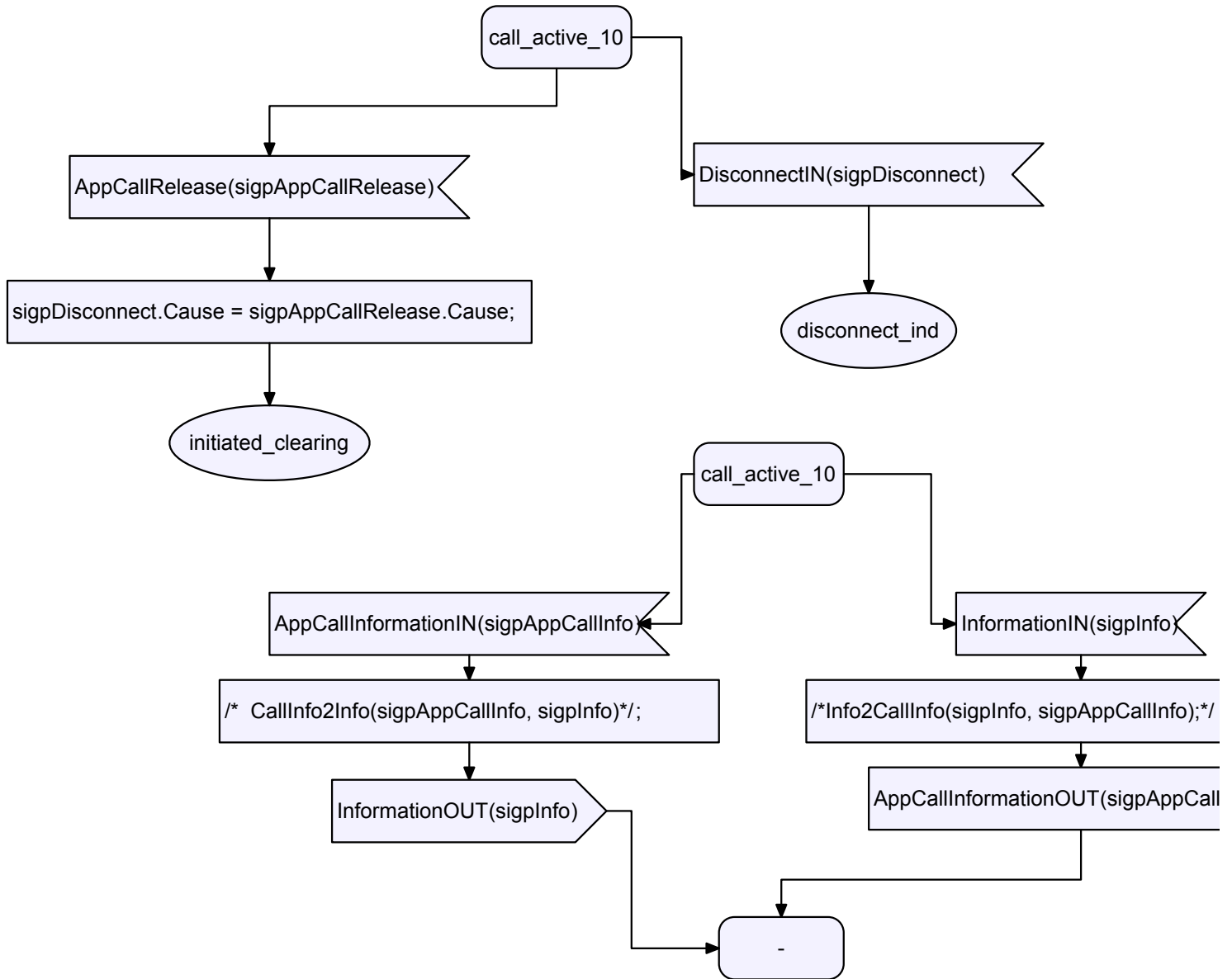
310()

_RECOVERY_ON_TIMER_EXPIRY;

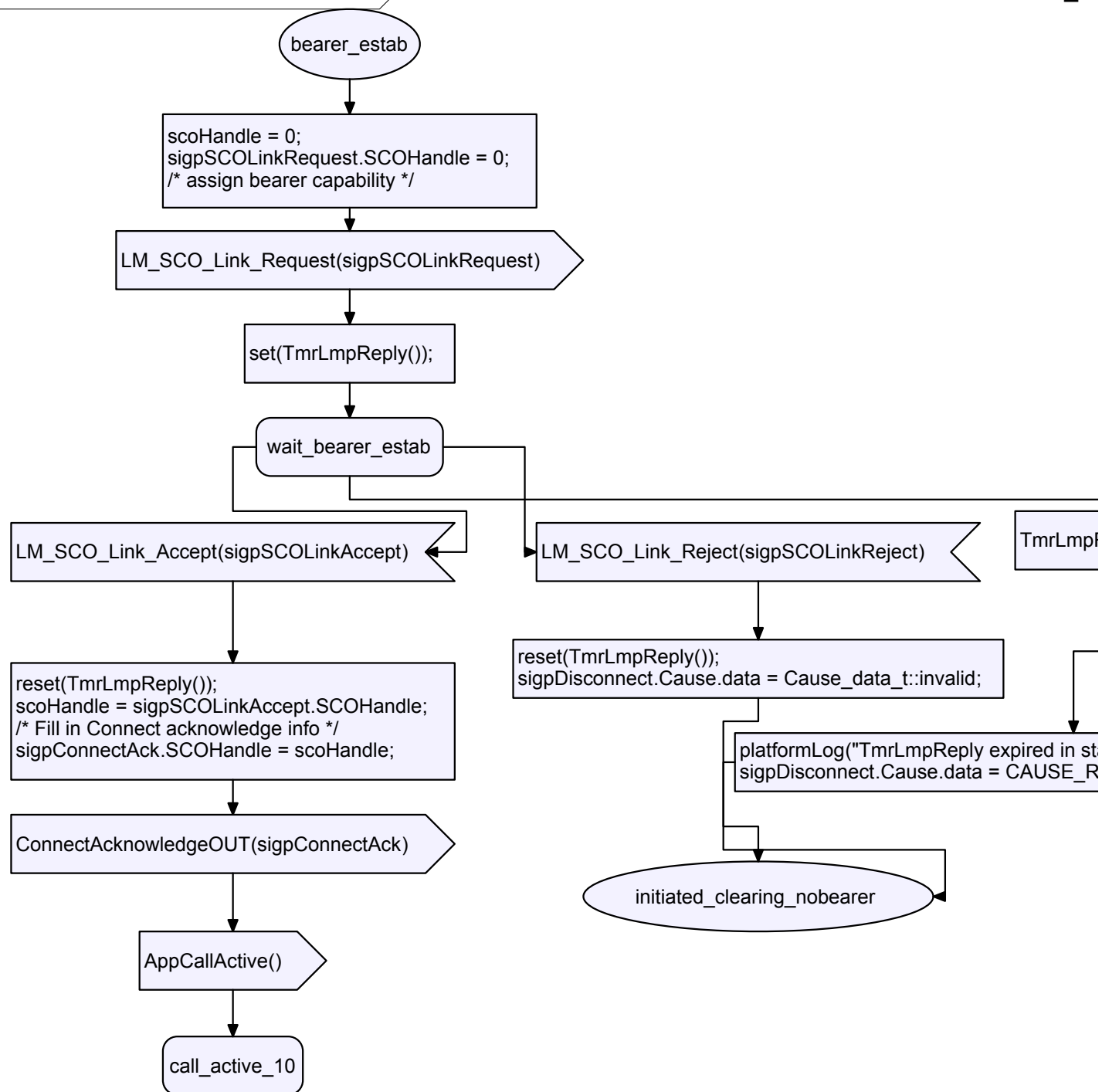
ted_clearing

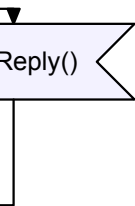
ON_TIMER_EXPIRY;
_4");

statemachine CallControl :: CallControl

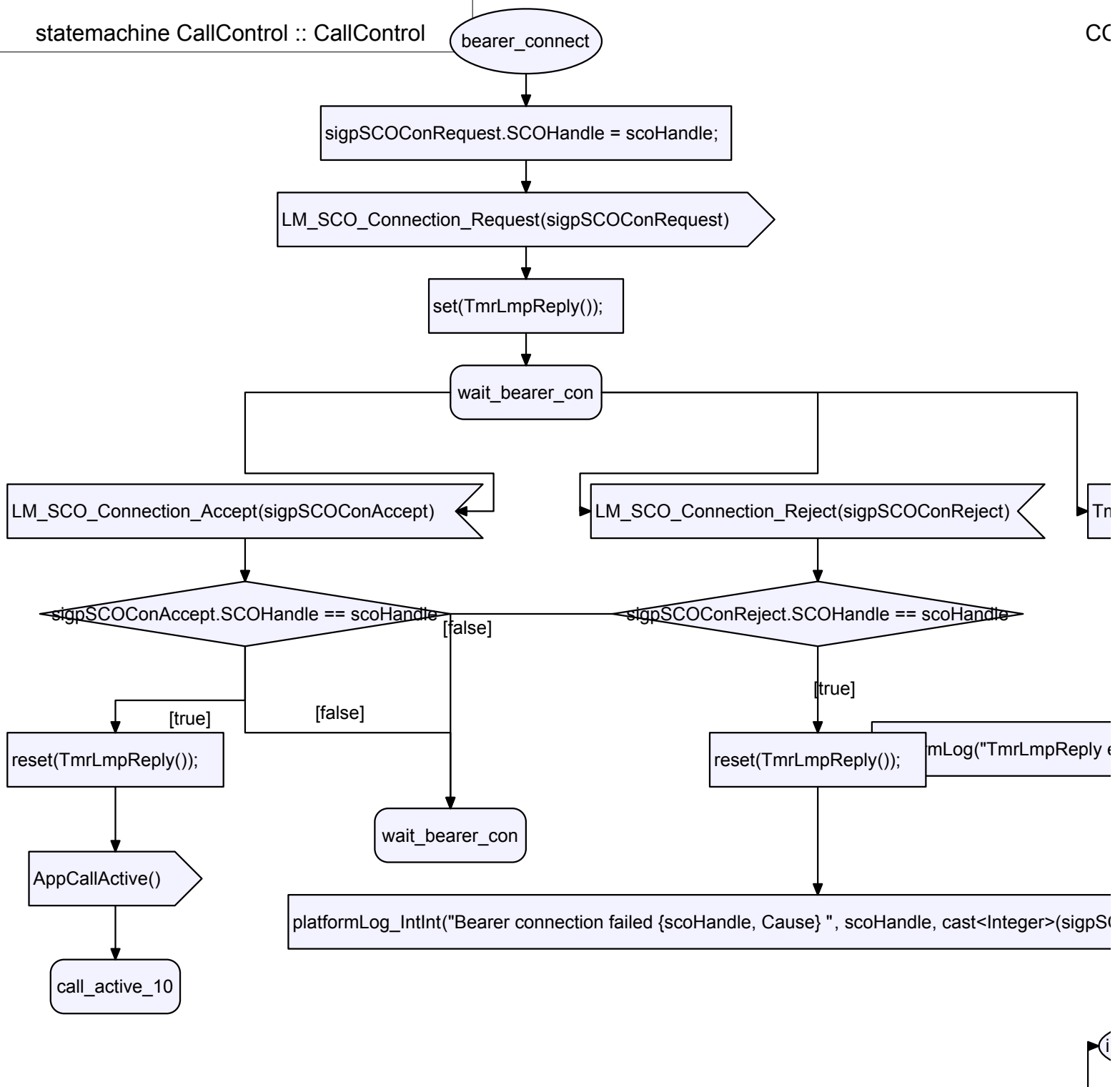


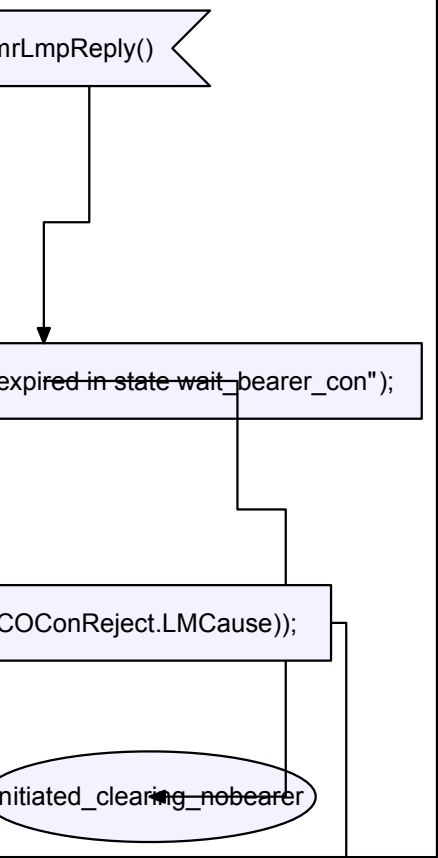




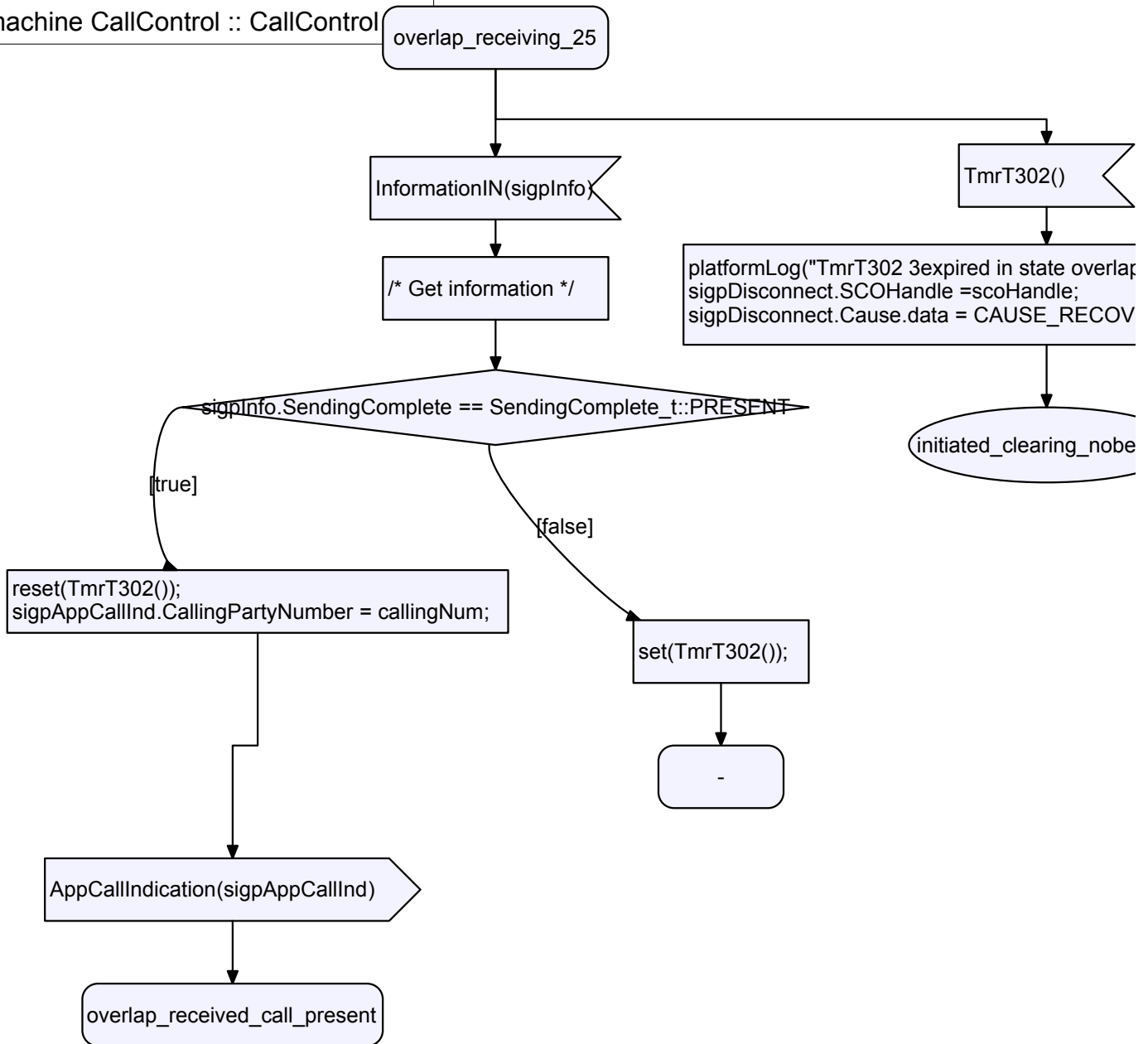


```
ate wait_bearer_estab");  
RECOVERY_ON_TIMER_EXPIRY;
```

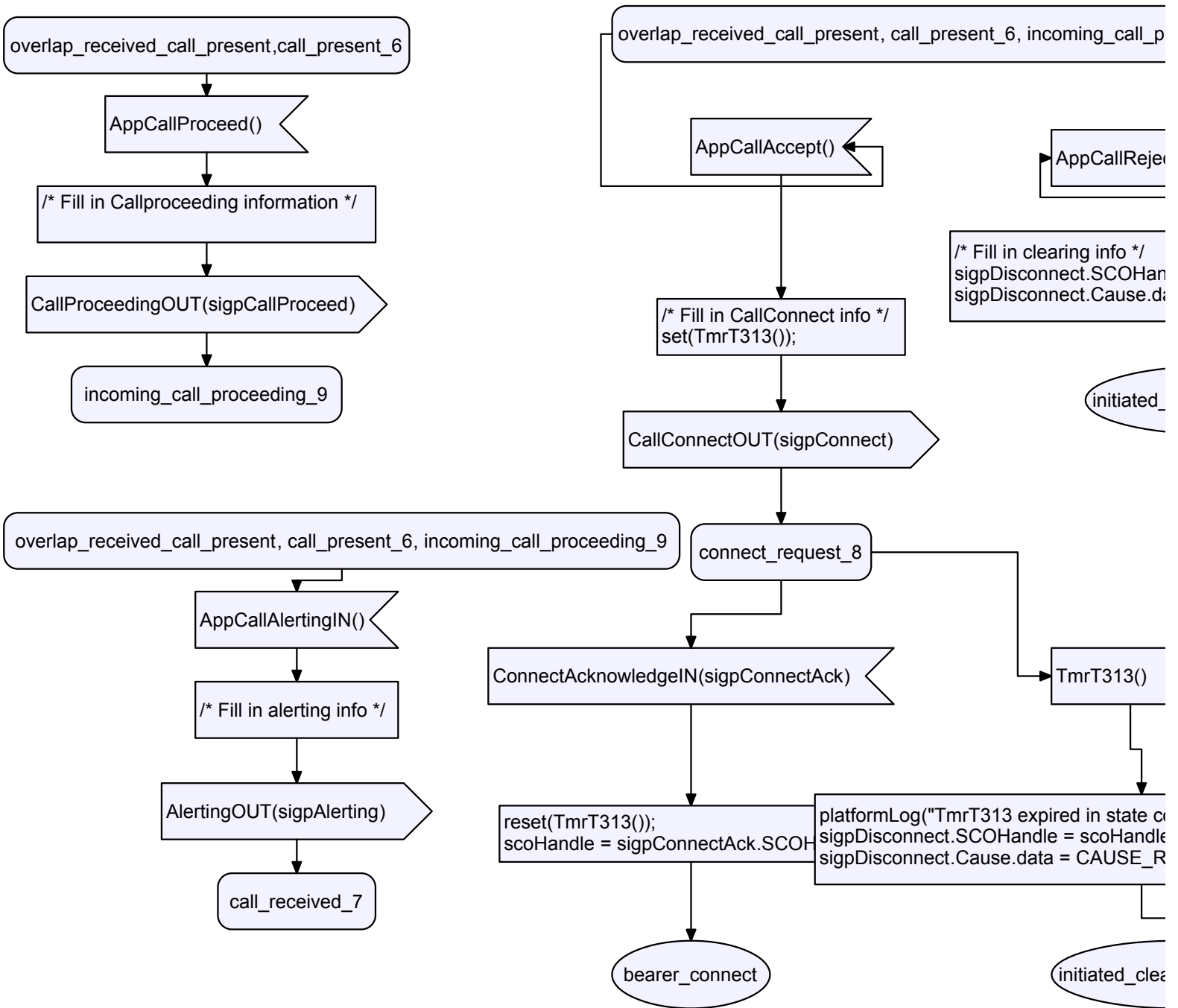


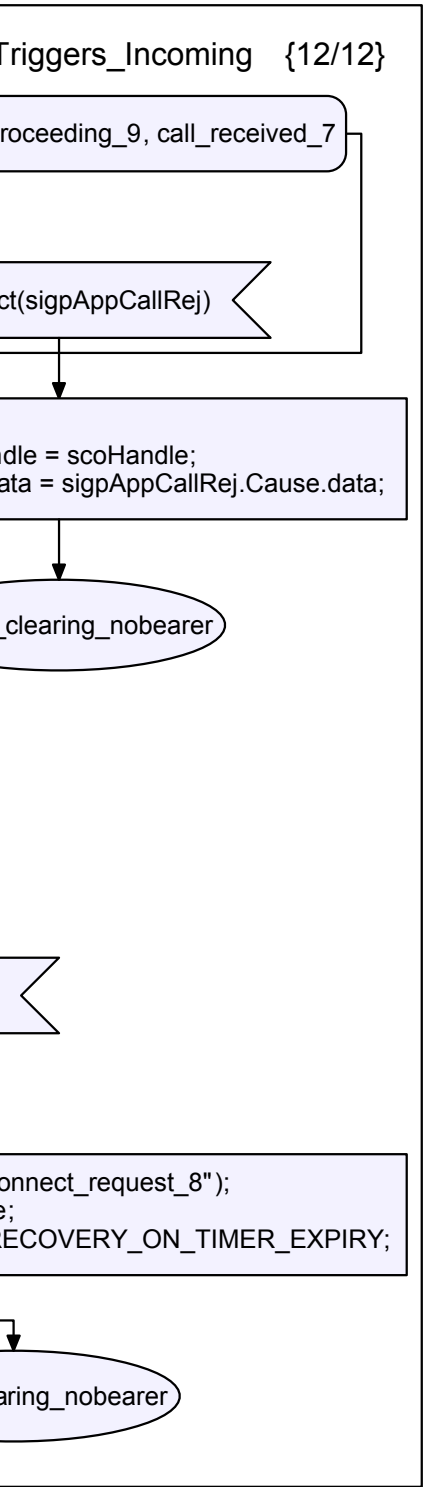
statemachine CallControl :: CallControl



```
p_receiving_25\n");  
ERY_ON_TIMER_EXPIRY;
```

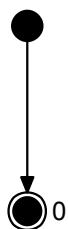
arer





private Integer ValidateCall(
AppCallRequest_t creq)

StatechartDiagram1 {1/1}



private void SendSetup(AppCallRequest_t creq)

```
/* Locals */  
Setup_t msgSetup;  
Integer numDigits = 0;
```

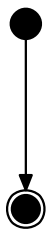
```
msgSetup.CallClass = creq.CallClass;  
msgSetup.CallingPartyNumber = creq.CallingPartyNumber;  
msgSetup.CalledPartyNumber.NType = creq.CalledPartyNumber.NType;  
msgSetup.CalledPartyNumber.NumberingPlanID = creq.CalledPartyNumber.NumberingPlanID;  
numDigits = creq.CalledPartyNumber.NumberDigits.length();  
if (numDigits > parMaxOverlapDigits) {  
    // Get only a portion of the called number and don't set SendingComplete in the message'  
    msgSetup.CalledPartyNumber.NumberDigits = substring(creq.CalledPartyNumber.NumberDigits, 1, parMaxOverlapDigits)  
}  
else {  
    msgSetup.CalledPartyNumber.NumberDigits = creq.CalledPartyNumber.NumberDigits;  
    msgSetup.SendingComplete = SendingComplete_t::PRESENT;  
}
```

SetupOUT(msgSetup)

);

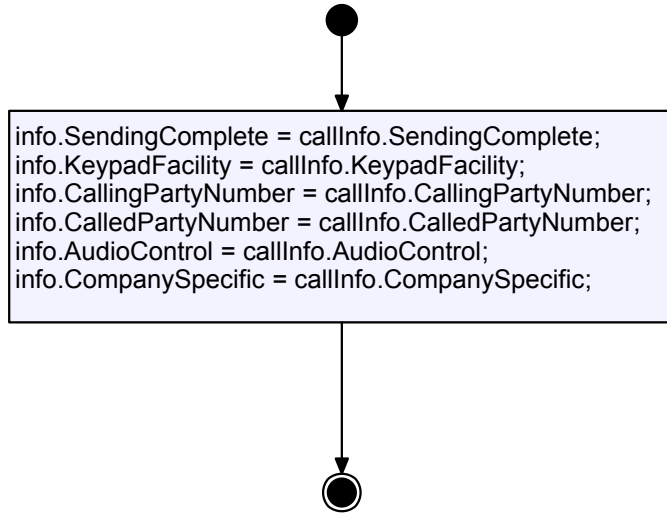
private void SendOverlapInfo(
CalledPartyNumber_t cnum)

StatechartDiagram1 {1/1}



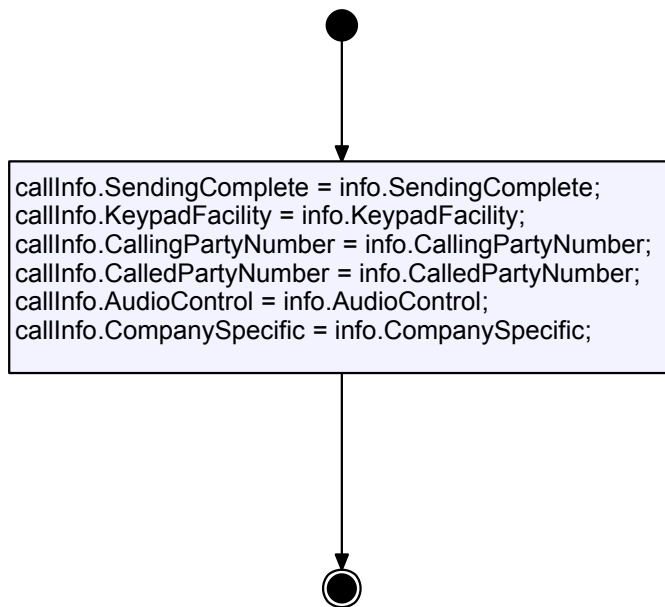
```
private void CallInfo2Info(inout
AppCallInformation_t callInfo, inout
Information_t info)
```

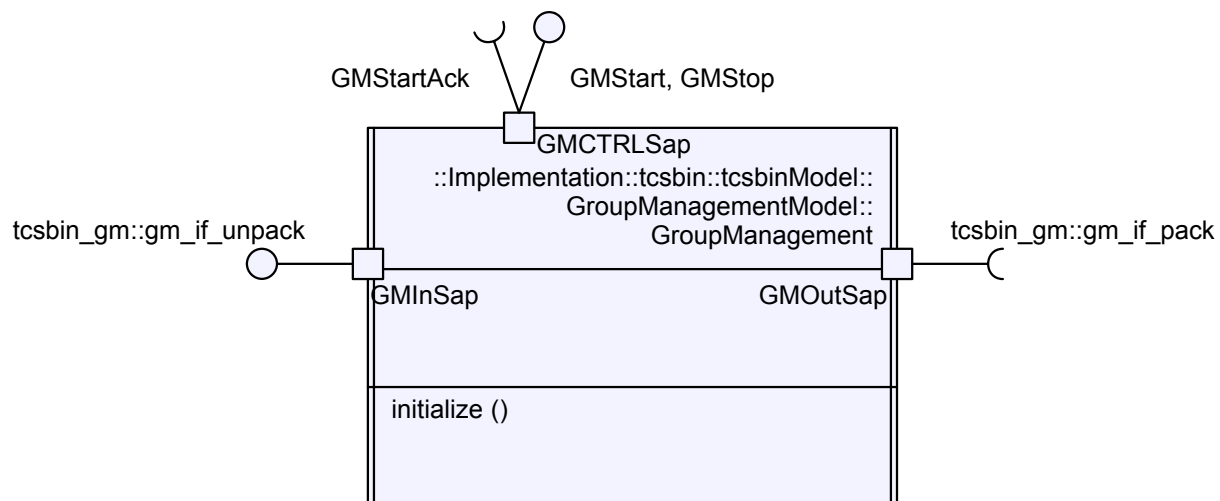
CallInfo2Info_StatechartDiagram {1/1}

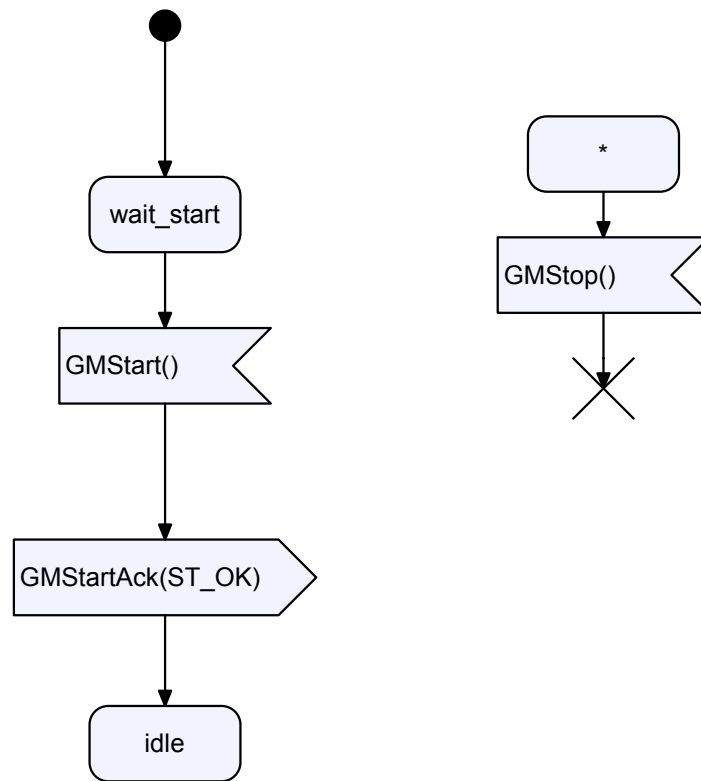


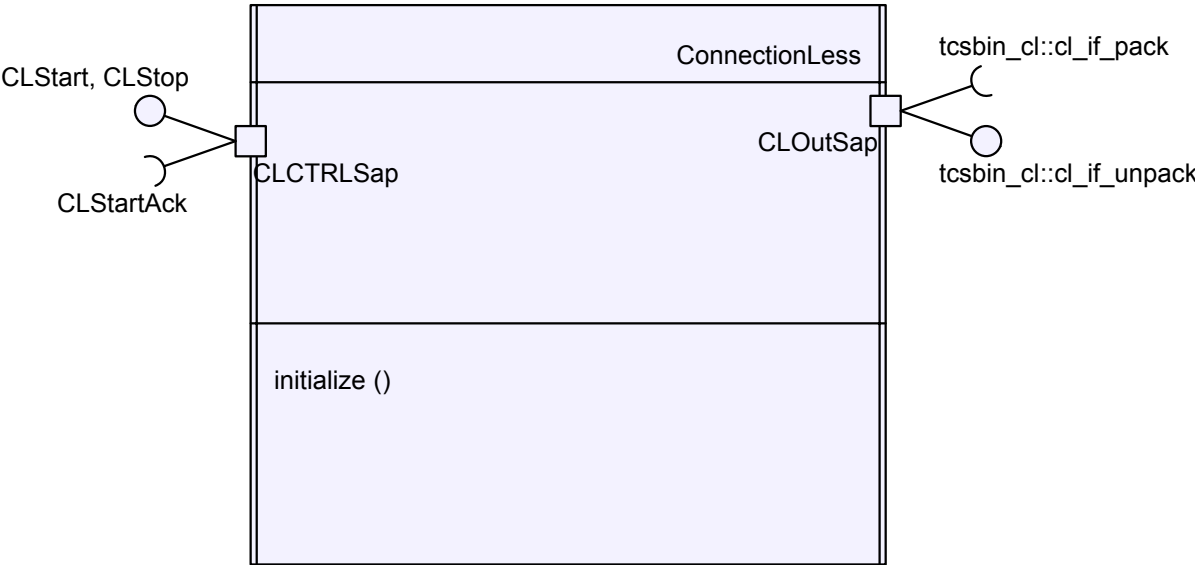
```
private void Info2CallInfo(inout  
    Information_t info, inout  
    AppCallInformation_t callInfo)
```

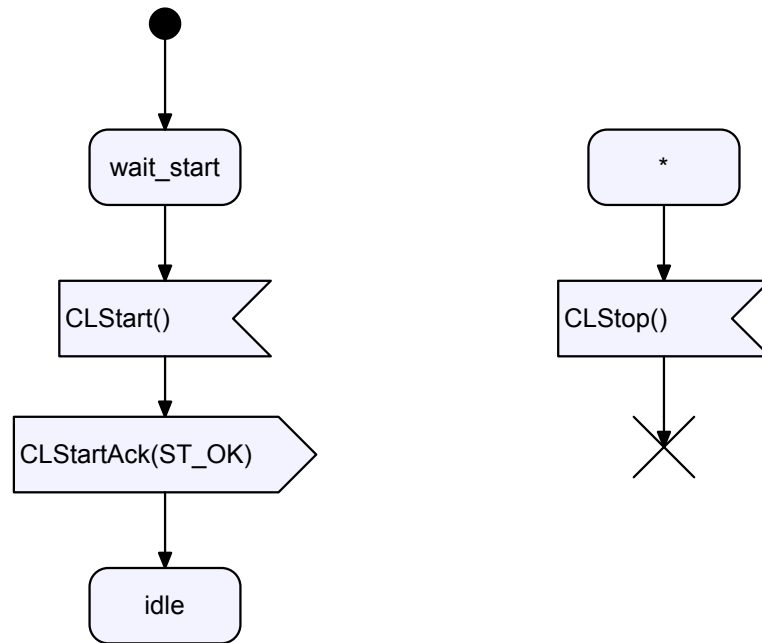
Info2Call_StatechartDiagram {1/1}

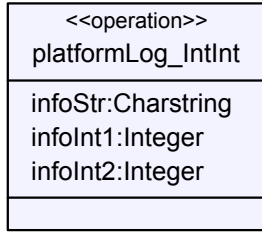
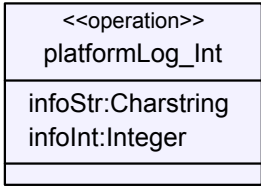
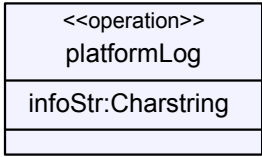






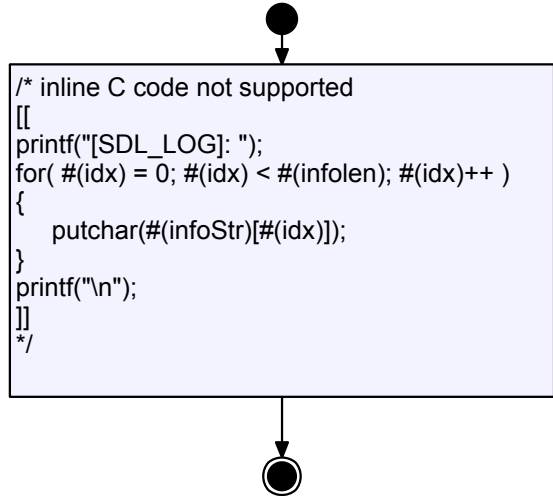






```
Integer idx;  
Integer infolen = infoStr.length();
```

```
/* inline C code not supported  
[[  
printf("[SDL_LOG]: ");  
for( #(idx) = 0; #(idx) < #(infolen); #(idx)++ )  
{  
    putchar(#(infoStr)[#(idx)]);  
}  
printf("\n");  
]]  
*/
```



void platformLog_Int(Charstring
infoStr, Integer infoInt)

StatechartDiagram1 {1/1}

Integer idx;
Integer infoLen = infoStr.length();

```
/* inline C code not supported
[[
printf("[SDL_LOG]: ");
for( #(idx) = 0; #(idx) < #(infoLen); #(idx)++ )
{
    putchar(#(infoStr)[#(idx)]);
}
printf(" = {%d}\n", #(infoInt));
]]
*/
```

void platformLog_IntInt(Charstring
infoStr, Integer infoInt1, Integer

StatechartDiagram1 {1/1}

Integer idx;
Integer infolen = infoStr.length();

```
/* inline C code not supported
[[
printf("[SDL_LOG]: ");
for( #(idx) = 0; #(idx) < #(infolen); #(idx)++ )
{
    putchar(#(infoStr)[#(idx)]);
}
printf(" = {%d, %d}\n", #(infoInt1), #(infoInt2));
]]
*/
```

interaction AppCallRequest

// Sequence diagram trace
generated for
TCSBINLayer

