MOST

Media **O**riented **S**ystems **T**ransport

Multimedia and Control
Networking Technology

MOST Dynamic Specification

Rev 1.3
12/2006

Legal Notice

COPYRIGHT

LICENSE DISCLAIMER

Nothing on any MOST Cooperation Web Site, or in any MOST Cooperation document, shall be construed as conferring any license under any of the MOST Cooperation or its members or any third party's intellectual property rights, whether by estoppel, implication, or otherwise.

CONTENT AND LIABILITY DISCLAIMER

MOST Cooperation or its members shall not be responsible for any errors or omissions contained at any MOST Cooperation Web Site, or in any MOST Cooperation document, and reserves the right to make changes without notice. Accordingly, all MOST Cooperation and third party information is provided "AS IS". In addition, MOST Cooperation or its members are not responsible for the content of any other Web Site linked to any MOST Cooperation Web Site. Links are provided as Internet navigation tools only.

MOST COOPERATION AND ITS MEMBERS DISCLAIM ALL WARRANTIES WITH REGARD TO THE INFORMATION (INCLUDING ANY SOFTWARE) PROVIDED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

In no event shall MOST Cooperation or its members be liable for any damages whatsoever, and in particular MOST Cooperation or its members shall not be liable for special, indirect, consequential, or incidental damages, or damages for lost profits, loss of revenue, or loss of use, arising out of or related to any MOST Cooperation Web Site, any MOST Cooperation document, or the information contained in it, whether such damages arise in contract, negligence, tort, under statute, in equity, at law or otherwise.

FEEDBACK INFORMATION

Any information provided to MOST Cooperation in connection with any MOST Cooperation Web Site, or any MOST Cooperation document, shall be provided by the submitter and received by MOST Cooperation on a non-confidential basis. MOST Cooperation shall be free to use such information on an unrestricted basis.

TRADEMARKS

MOST Cooperation and its members prohibit the unauthorized use of any of their trademarks. MOST Cooperation specifically prohibits the use of the MOST Cooperation LOGO unless the use is approved by the Steering Committee of MOST Cooperation.

SUPPORT AND FURTHER INFORMATION

For more information on the MOST technology, please contact:

MOST Cooperation
Administration
Bannwaldallee 48
D-76185 Karlsruhe
Germany

Tel: (+49) (0) 721 966 50 00
Fax: (+49) (0) 721 966 50 01
E-mail: contact@mostcooperation.com
Web: www.mostcooperation.com

MOST®
COOPERATION

Contents

Document References

Number	Document	Revision
[1]	MOST Specification	2.5
[2]	MOST FunctionBlock NetworkMaster	2.5
[3]	MOST FunctionBlock ConnectionMaster	2.5
[4]	MOST FunctionBlock NetBlock	2.5

Document History

Changes MOST Dynamic Specification Rev 1V2 to MOST Dynamic Specification Rev 1V3

Change Ref.	Section	Changes
1V3_001	General	– Minor spelling and grammar corrections. – Fixed parameters to match Function Library. – Modified timer names to match the MOST Specification. – Replaced "synchronous" with "streaming" to match the terminology of the MOST Specification. – Where applicable, added note that Secondary Nodes are not supported by MOST50. – Removed elements/notes that mentioned the stored Registry. The concept of a stored Central/Decentral Registry is no longer supported by the MOST Specification. – Replaced NetOn event with Init Ready event. – Replaced MOST.NCE with USERDEF.NCE because the NCE is not contained in any function catalog. – Changed channel info from "AudioAmplifier" to "MOST" to match other MSCs that belong to the Dynamic Specification. – Replaced "all bypass" with "bypass".
1V3_002	3.1	– Updated System States diagram to include Shutdown.Start(Execute) transition.
1V3_003	3.2.2	NM_Gen_Startup – Added end node to HMSC. – Removed NetOn setting condition because the NetOn state is only reached after Init Ready is received.
1V3_004	3.2.3	NM_Gen_Init – Replaced *when NetOn* condition with reception of Init Ready event.
1V3_005	3.2.7	NM_Gen_ProcessNCE – Changed single cast NCE into a broadcast message.
1V3_006	3.3.3.1	NM_Sc_Initial_Scan_CR_Not_Stored_SystemState_NotOk – Modified and renamed to NM_Sc_Initial_Scan_NoNodeAddress_SystemState_NotOk because Central Registry is no longer stored. – Renamed section to “Initial Scan without Node Address”.
1V3_007	3.3.3.2	NM_Sc_Initial_Scan_CR_Stored_SystemState_NotOk_To_Ok – Modified and renamed to NM_Sc_Initial_Scan_SystemState_NotOk_To_Ok because Central Registry is no longer stored. – Renamed section to “Initial Scan System State NotOK to OK”
1V3_008	3.3.3.3	NM_Sc_Initial_Scan_CR_Stored_Node_Not_Responding – MSC removed because the MOST Specification does not support a stored Central Registry anymore.
1V3_009	3.3.3.4	NM_Sc_Initial_Scan_CR_Stored_Registration_Error – MSC removed because the MOST Specification does not support a stored Central Registry anymore.
1V3_010	3.4.1	NS_Gen_Startup – Added HMSC End element. – Added note that secondary nodes are not supported by MOST50. – Removed NetOn condition because Init Ready event was received yet.

Change Ref.	Section	Changes
1V3_011	3.4.2	NS_Gen_Init – Renamed NetOn_event to InitReady event. NetOn event is no longer used.
1V3_012	3.3.4.16	NM_Sc_NS_Change_Of_NodeAddress – This MSC has been removed from the collection due to lack of compliance with the MOST Specification. A network slave is not allowed to change its NodeAddress during runtime. The NetworkMaster would signal a transition to NotOk in such an error case, as soon as the inconsistency is noticed.
1V3_013	3.5.1.1	NS_Sc_StartupOk – Renamed NetOn_event to InitReady event. NetOn event is no longer used.
1V3_014	3.5.1.2	NS_Sc_StartupNotOk – Renamed NetOn_event to InitReady event. NetOn event is no longer used.
1V3_015	4.5.1.2	CM_Gen_M_CleanUp – Changed guarding condition "when (SourceType = Allocate)" to "otherwise".
1V3_016	4.10	– Added new MSC CM_Boundary_Change.
1V3_017	5.5	PM_Gen_Device_WakeUp – Modeled pre-condition as guarding condition.
1V3_018	5.6	PM_Gen_Overtemp_Shutdown: – Changed AbilityToWake to PermissionToWake. – Switching light off when theta_dead is reached is modeled as exception instead of a mere parallel action.
1V3_019	5.7	PM_Gen_Restart_After_Overtemp_Shutdown – Changed AbilityToWake to PermissionToWake. – In those cases where no corresponding events are modeled, added comments, stating that restarting the network is performed by the NetworkMaster. – Added Over-Temperature-Shutdown broadcast message from device that is still in the overtemperature state. – The device that initiated the over temperature shutdown is allowed to wake up the network. – The PowerMaster may restart the network but is not required to do so. – The network restart may be triggered by the user after $t_{WaitAfterOverTemp}$ has expired.
1V3_020	-	NM_Sc_Avoiding_InstID_Collision – This empty MSC was removed from the collection.
1V3_021	-	NM_Gen_ScanType – This outdated MSC was removed from the collection.

Changes MOST Dynamic Specification Rev 1V2 (04/2006) to MOST Dynamic Specification Rev 1V2 (06/2006)

Change Ref.	Section	Changes
1V2_06_001	3.3.4.13	Substituted unguarded OPT inline expression with additional branch in ALT inline expression to make the behavior deterministic.

Changes MOST Dynamic Specification Rev 1V1 to MOST Dynamic Specification Rev 1V2

Change Ref.	Section	Changes
1V2_001	General	Changed light to modulated signal.
1V2_002	Document References	Added Function Blocks that also affects the Dynamic Specification.
1V2_003	3.2.3	Updated MSC comments and added timer t_WaitBeforeScan.
1V2_004	3.3.2	Changed order in MSC. Added timer t_WaitBeforeScan and action.

Change Ref.	Section	Changes
1V2_005	3.3.3.2, 3.3.3.3, 3.3.3.4	Added timer t_WaitBeforeScan.
1V2_006	3.3.4.16	Corrected typo in MSC.
1V2_007	3.4.5	Changed order in MSC and added remark.
1V2_008	3.5.1	Deleted section “Startup – Timeout” due to deletion of t_CfgStatus.
1V2_009	4.4.1.1.4, 4.4.1.3, 4.4.2.1-4.4.2.4, 4.9.3,	Added remark that the source activity is optional.
1V2_010	5.3	Added timer t_SlaveShutdown.
1V2_011	4.5.2	Added Chapter and MSCs handling source and sink drop.
1V2_012	8	Updated Timers table.
1V2_013	3.3.4.4, 3.3.4.10	Deleted specific description of reasons for scan initiation.
1V2_014	3.4.2, 3.5.1.1, 3.5.1.2	Replaced $t_{CfgStatus}$ with t_{Answer}. $t_{CfgStatus}$ equivalent removed from MSCs.
1V2_015	5.4	MSC changed with respect to t_RetryShutdown timer.
1V2_016	General	Unified spelling of NetworkMaster and ConnectionMaster.
1V2_017	3.2.3, 3.3.2, 3.3.3.2, 3.3.3.3, 3.3.3.4	Condition NotOk now set before timer t_WaitBeforeScan starts.
1V2_018	3.2.3	OPT inline expression with “Wait until NCE has not occurred…” action removed - already covered by $t_{WaitBeforeScan}$ timer.
1V2_019	3.3.4.7, 3.3.4.9	Configuration.Status(invalid) now directly after Central Registry change detected, “No errors occurred during…” text block removed.
1V2_020	3.3.4.5, 3.3.4.10, 3.3.4.11, 3.3.4.13	Added Remark: “This scenario is only valid for the mechanism of parallel scanning of the system. It does not cover sequential scanning.”
1V2_021	3.4.5	Deleted Configuration.Status(NotOk) message - already contained in referencing MSCs.
1V2_022	5.2	MSC change: Removed the idle loop. Removed the alternative path that deals with devices that do not have the permission to wake the network.
1V2_023	3.3.4.3	MSC change: Configuration.Status(Invalid) is now sent immediately after a conflict occurs. At which point Configuration.Status(New) is sent now depends on whether the NetworkMaster supports “immediate notification”.
1V2_024	3.3.4.4	SystemState(NotOk) event added as trigger for this MSC. MSC change: At which point Configuration.Status(New) is sent now depends on whether the NetworkMaster supports “immediate notification”.
1V2_025	3.3.4.12	Original section 3.3.4.12 NM_Sc_NCE_SystemState_To_Ok has been split into two; NM_Sc_NCE_SystemStateNotOk_To_Ok now only deals with SystemState NotOk.
1V2_026	3.3.4.13	New section, derived from former section 3.3.4.12 NM_Sc_NCE_SystemState_To_Ok. Here, the focus is on SystemState Ok. The MSC now differentiates between NetworkMasters with or without the “immediate notification” feature.

Change Ref.	Section	Changes
1V2_027	3.3.3.3, 3.3.4.6, 3.3.4.7, 3.3.4.8, 3.3.4.9	Corrected inconsistent use of timer t_DelayCfgRequest: Timers t_DelayCfgRequest1 and t_DelayCfgRequest2 were renamed to t_DelayCfgRequest. The latter is now initialized with values t_DelayCfgRequest1 and t_DelayCfgRequest2, in accordance with 3.2.5 NM_Gen_ReceiveConfiguration. Changed guarding condition “node has been scanned less than 20 times without answering” to “ScansWithoutAnswer < 20” and added improved description as comment.

Changes MOST Dynamic Specification Rev. 1V0 to MOST Dynamic Specification Rev 1V1

Change Ref.	Section	Changes
1V1_001	General	Deleted old chapters 3.1.1 and 3.1.2.
1V1_002	3	Changed un-initialized logical node address from 0x0FFD to 0xFFFF.
1V1_003	3.3.2	Changed order in MSC.
1V1_004	3.3.4.12	MSC23 compliant with MSC5.
1V1_005	4.3	Added chapter.
1V1_006	4.4	Timeout replaced abort in connection management.
1V1_007	4.4.1.2.1	Sink changed to source.
1V1_008	4.5	Timeout replaced abort in connection management.
1V1_009	5	Added chapter.

1 Introduction

1.1 Purpose

Most Dynamic Specification is aimed to be complementary to the MOST Specification and the MOST FunctionCatalog. The behavior of controller – slave (FBlock) communication is described with Message Sequence Charts (MSC).

1.2 Scope

The scope of MSCs in this specification is to describe dynamic communication sequences between controllers and slaves. The Dynamic Specification covers the main scenarios.

2 MSC Structuring

There are many different ways to describe a MOST System. In this specification, we look at the system as consisting of slaves or functional services (FBlocks) which are managed by different controllers.

When looking at the general behavior in a MOST Network (e.g., network startup procedures), all network slaves are managed by the NetworkMaster.

The Connection Management manages all connections between slaves.

Audio and video implemented in larger systems are managed by logical entities, Audio or Video Management. The purpose of these is to control amplifiers and displays. In some systems, these are implemented as a part of the HMI but could also be implemented as separate devices. Therefore, Synchronous Management (Audio, Video, or Camera Management) is useful in order to keep complex systems well structured.

Every FBlock in a MOST System is controlled by a controller. After a startup of a MOST system, it is possible for different controllers to act simultaneously. Timers and other constraints may affect this behavior. Communication in the MOST System could, therefore, be seen as consisting of many controller–slave communication sequences which are either mandatory or optional. The perspective of looking at these communication sequences could either be a master perspective or a slave perspective. The aim of this specification is to describe communication from either a master or a slave perspective. Only the relevant perspective is shown.

2.1 General MSCs vs. Scenario MSCs

The purpose of the general MSCs is to, as complete as possible, describe the dynamic behavior of the different functional areas (e.g., NetworkMaster, Network Slave or Connection Management). All possible events and responses from the communication partner are considered. The high-level MSC of a specific functional area shows how the general MSCs are combined to describe the complete behavior of this area.

The purpose of the scenario MSCs is to extract a specific path from the general MSCs and thereby show a simple case. To reduce the total number of MSCs, there can still be alternative or optional paths inside the scenario MSCs (e.g., handle different responses to a sent message). However, the intention is to describe the different example cases as simple as possible.

For some functional areas (e.g., AudioDiskPlayer), it is not convenient to describe the complete dynamic behavior with general MSCs. Instead, different scenario MSCs are used to exemplify the usage of these functional areas.

3 Network Management

The MSCs in this section show how the NetworkMaster maintains the Central Registry by collecting configuration information from all network slaves. The NetworkMaster then distributes information about the status of the network to all network slaves. Note that the information stored in the Central Registry is not distributed; the controllers in need of this information have to request this information from the NetworkMaster.

3.1 SystemStates

The NetworkMaster distributes the SystemState of the network to the network slaves by broadcasting Configuration.Status() messages. The state diagram in Figure 3-1 shows the SystemStates and which events affect the states.

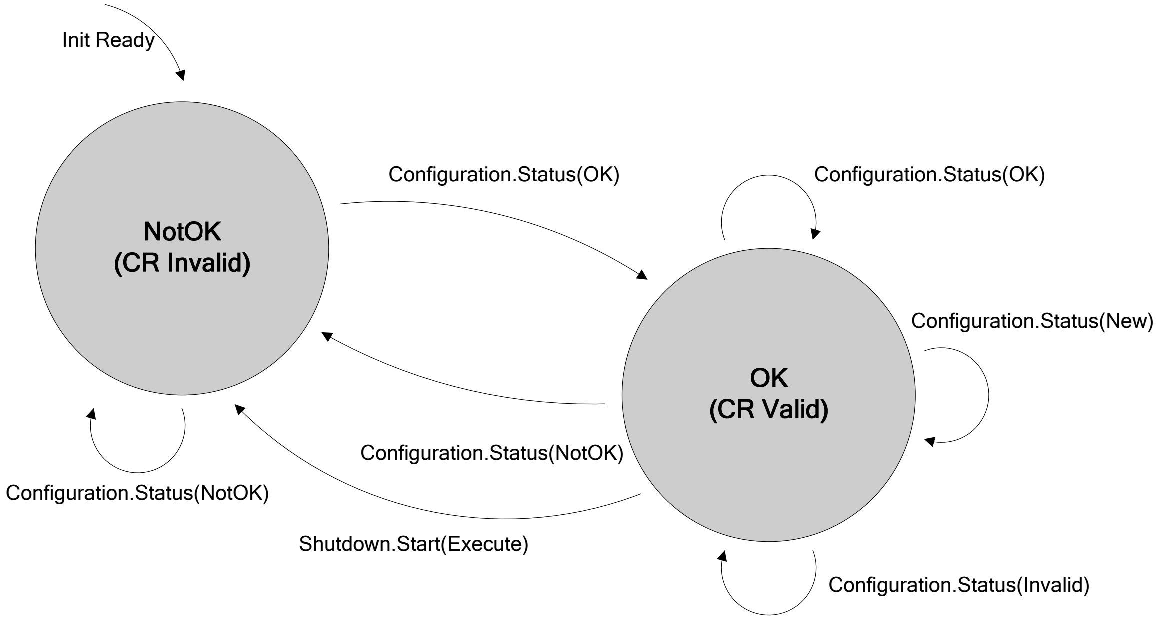


Figure 3-1: States of the network are shown, as well as the status of the Central Registry (CR).

3.2 NetworkMaster General MSCs

The general NetworkMaster MSCs are divided into two parallel processes. One process requests configurations from the network slaves when required. The other process receives the registrations when provided by the network slaves. The latter process checks the validity of the registrations. All network slaves in the network are treated individually.

3.2.1 Variables used in general NetworkMaster MSCs

The general MSCs use variables to simplify the MSCs, as well as reducing the total number of MSCs. Table 3-1 shows a list of the variables used in the general NetworkMaster MSCs. Figure 3-2 shows an example of what a Central Registry using some of these variables may look like.

Note that these variables, and the Central Registry in figure Table 3-1 and Figure 3-2 respectively, are used only to show the behavior and do not specify the actual implementation of the NetworkMaster.

Variable	Range	Explanation
numErr_nodepos[1]	0..2	The number of times that this node has caused Configuration.Status(NotOk) in succession. If the same node causes Configuration.Status(NotOk) three times in succession, then the node will be ignored until the next NCE or system restart.
request_nodepos[1]	True, False	Holds information if this node should be scanned. If request_4 is set to true, then the node at node position four should be scanned.
$t_{nodepos}$[1]	0..$t_{WaitForAnswer}$	This is a $t_{WaitForAnswer}$ for each node.
numCompScans	0..∞	The number of times the NetworkMaster has made complementary scans. The value of numCompScans affects the time between the complementary scans. If the node has been scanned less than 20 times, then the $t_{DelayCfgRequest1}$ is used, otherwise $t_{DelayCfgRequest2}$ is used instead.
doRequest	True, False	This variable tells the NetworkMaster if all nodes are to be scanned. If it is set to true, then all nodes are requested.
ConfigUpdate	Success, Error	This variable tells if the last registration was correct or not. It is used when updating the configuration status of the network and broadcasting the result of a scan or registration.
numNodes	1..64	The number of nodes to scan.

Table 3-1: Variables used in the general Network Management MSCs

Figure 3-2 shows a cleared Central Registry in a network with four nodes before the NetworkMaster starts scanning the network. The example Central Registry uses some of the variables in Table 3-1.

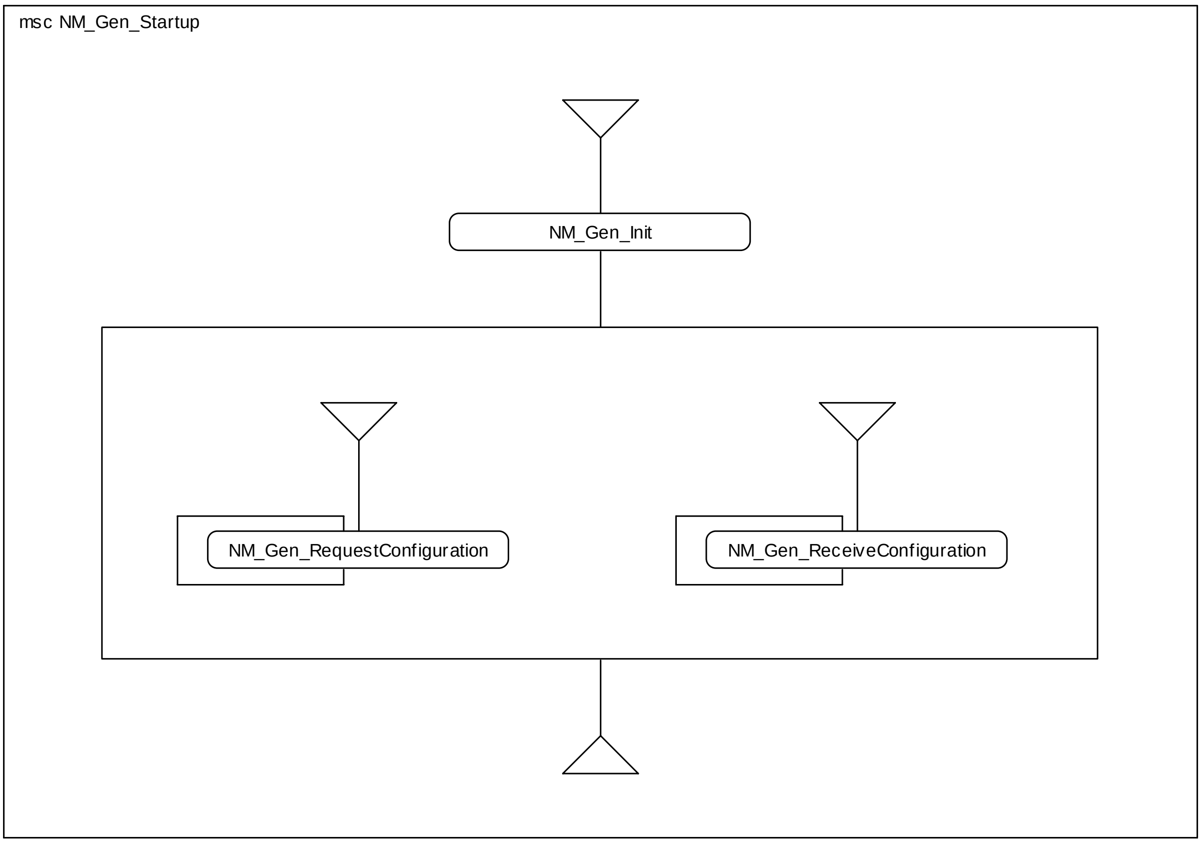
Node position	NodeAddress	FBlockID	InstID	request_nodepos	numErr_nodepos	$t_{nodepos}$
0	-	-	-	request_0 = True	numErr_0 = 0	t_0
1	-	-	-	request_1 = True	numErr_1 = 0	t_1
2	-	-	-	request_2 = True	numErr_2 = 0	t_2
3	-	-	-	request_3 = True	numErr_3 = 0	t_3

Figure 3-2: An example of what a Central Registry may look like.

[1] "nodepos" is replaced by the node's actual node position.

3.2.2 High-level NetworkMaster MSC

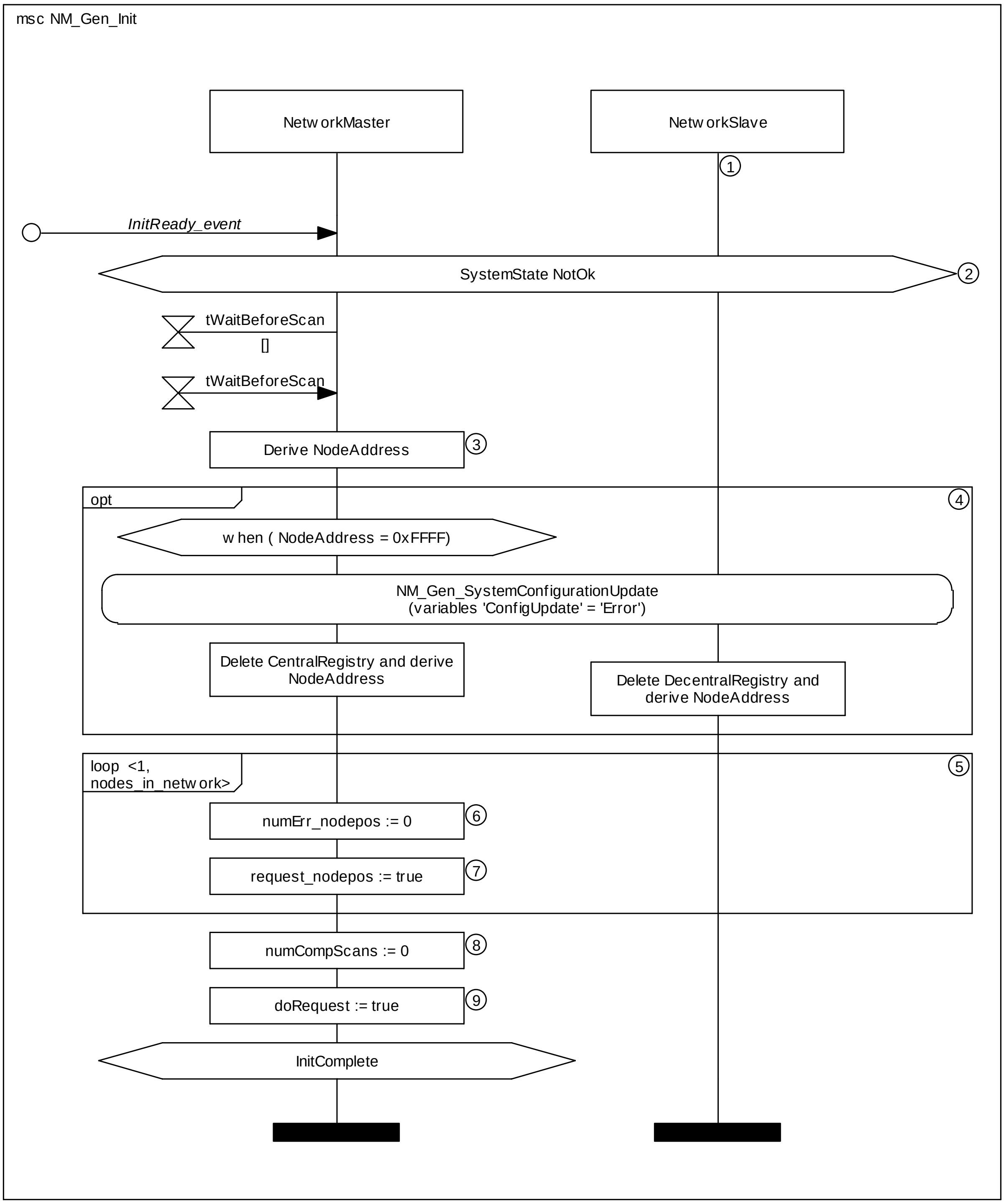
General MSC:	NM_Gen_Startup
Description:	High-level MSC of NetworkMaster network configuration process. After detecting the Init Ready event, the NetworkMaster initializes itself, this is shown in NM_Gen_Init. When the NM_Gen_Init has completed, two parallel processes are started. One process (NM_Gen_RequestConfiguration) asks nodes for their configuration and one process (NM_Gen_ReceiveConfiguration) handles the reception of registration messages. These two processes run in parallel until shutdown.
Prior Condition:	
Initiator:	
Communication Partners:	All NetBlocks
Events	Init Ready
Timers/Timing constraints	
Remarks:	



MSC 1: NM_Gen_Startup

3.2.3 Initializing the NetworkMaster

General MSC:	NM_Gen_Init
Description:	The NetworkMaster initializes its NodeAddress and resets all variables used during scanning. It also sets “request_nodepos” for all nodes; this leads to all nodes being scanned, as well as setting “doRequest” which triggers the scanning process.
Prior Condition:	
Initiator:	
Communication Partners:	All NetBlocks
Events	Init Ready
Timers/Timing constraints	- $t_{WaitBeforeScan}$
Remarks:	

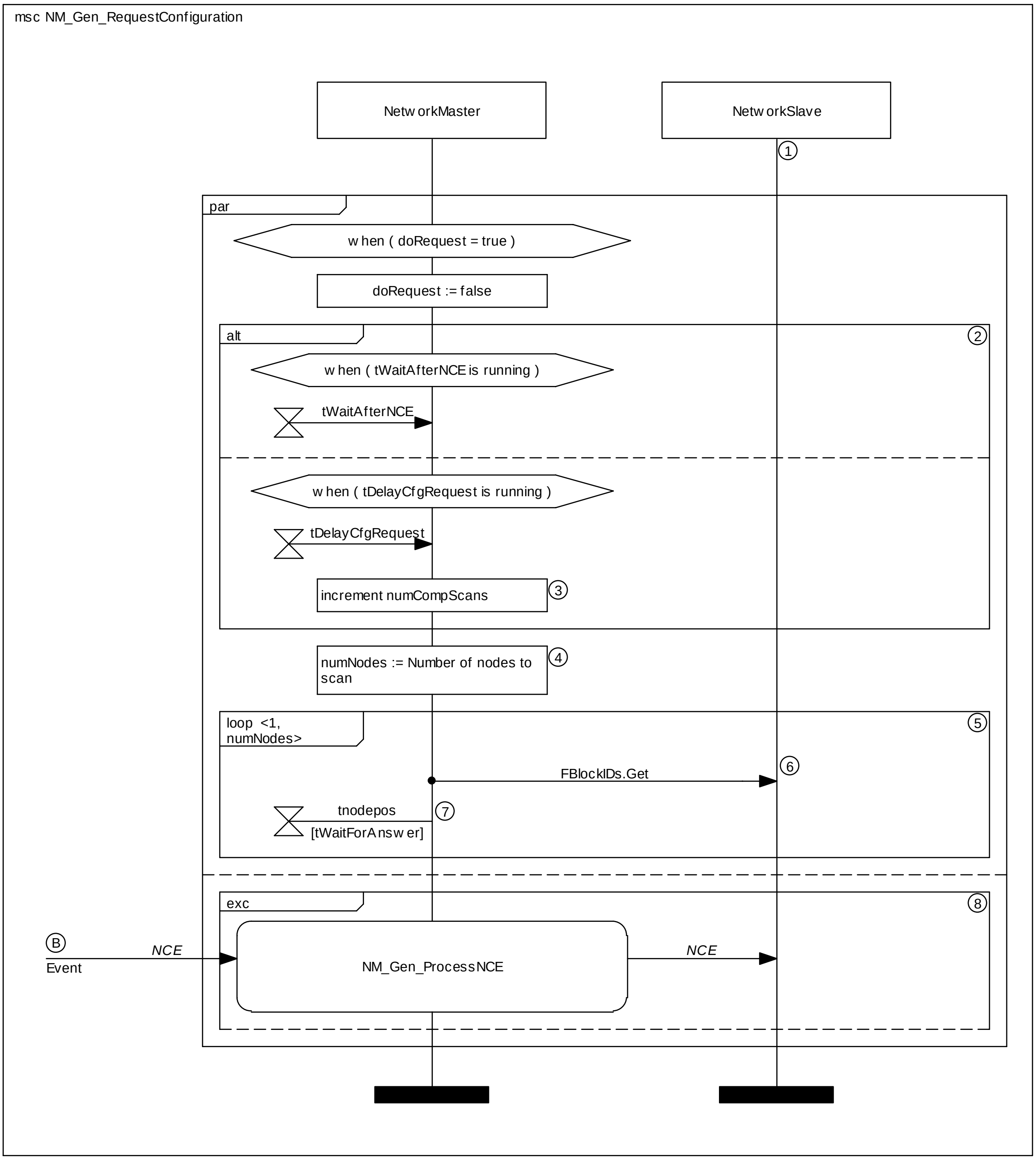


MSC 2: NM_Gen_Init

1. All nodes in the network, treated individually.
2. The system state is always NotOk following the Init Ready event.
3. The address should be static, stored, or calculated.
4. All NodeAddresses have to be recalculated and all Decentral Registries have to be cleared if the Central Registry is not stored.
5. Nodepos is incremented for each node.
6. Number of times a node has caused a Configuration.Status(NotOk).
7. All nodes will be requested.
8. Number of Complementary scans performed since startup.
9. Run configuration request of nodes.

3.2.4 Requesting Configuration

General MSC:	NM_Gen_RequestConfiguration
Description:	This process requests the configuration from nodes that have their “request_nodepos” set.
Prior Condition:	doRequest = True
Initiator:	NetworkMaster whenever doRequest = True
Communication Partners:	All NetBlocks that have their respective “request_nodepos” set. Please refer to section 3.2.1.
Events	NCE
Timers/Timing constraints	- $t_{DelayCfgRequest}$ - $t_{WaitAfterNCE}$ - $t_{WaitForAnswer}$
Remarks:	- An NCE interrupts this process. - Note that $t_{DelayCfgRequest}$ and $t_{WaitAfterNCE}$ never run simultaneously. Please refer to section 3.2.7.

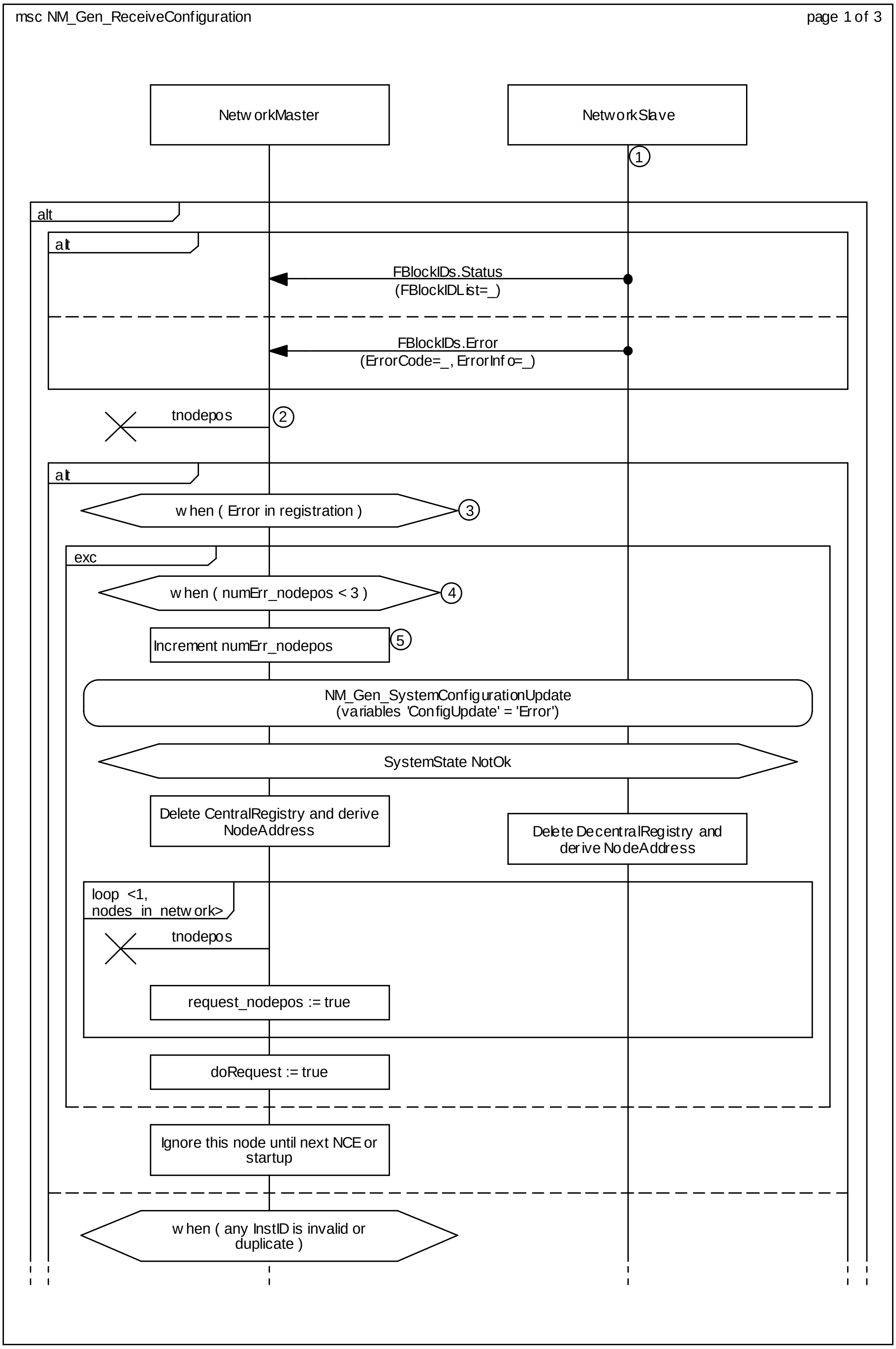


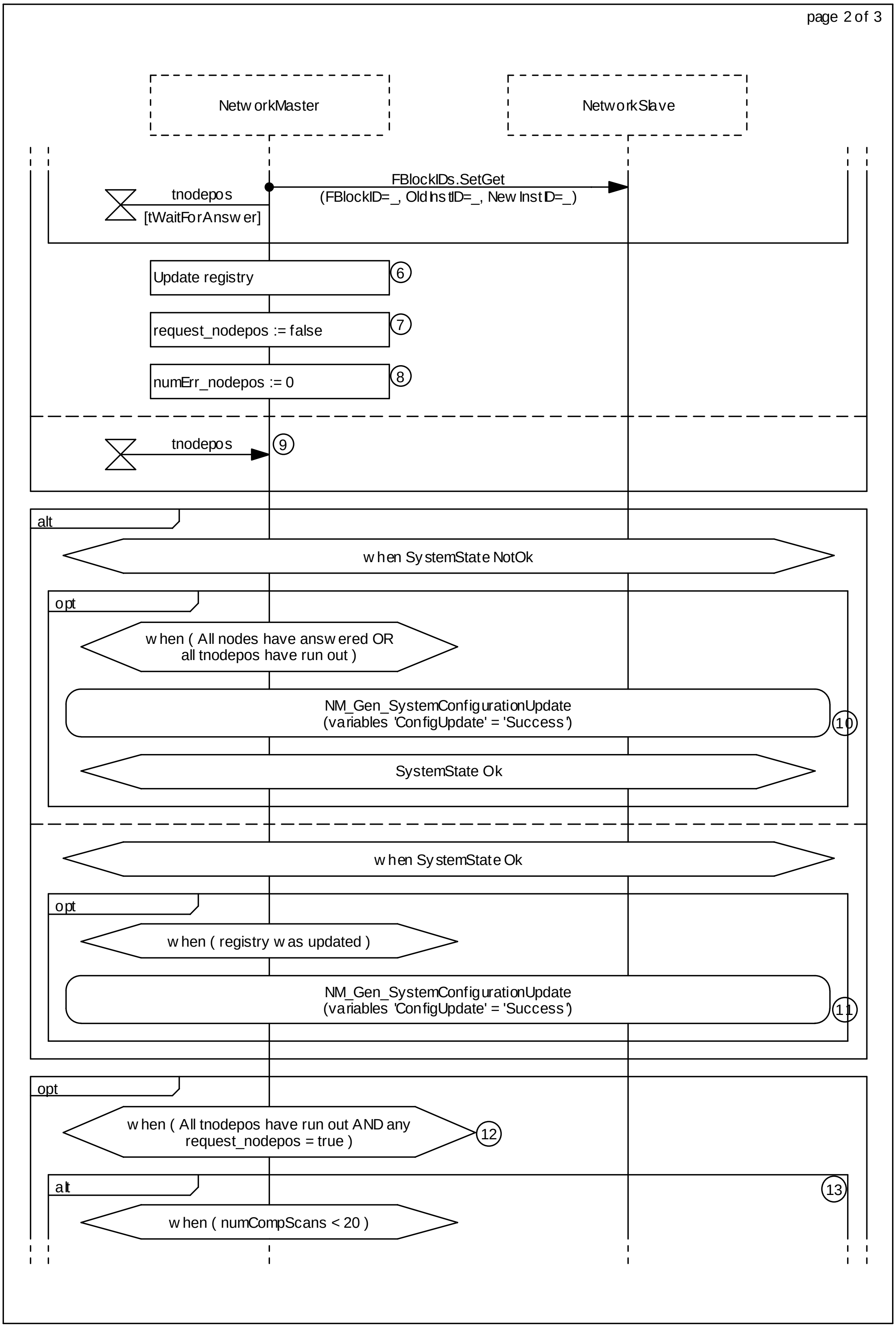
MSC 3: NM_Gen_RequestConfiguration

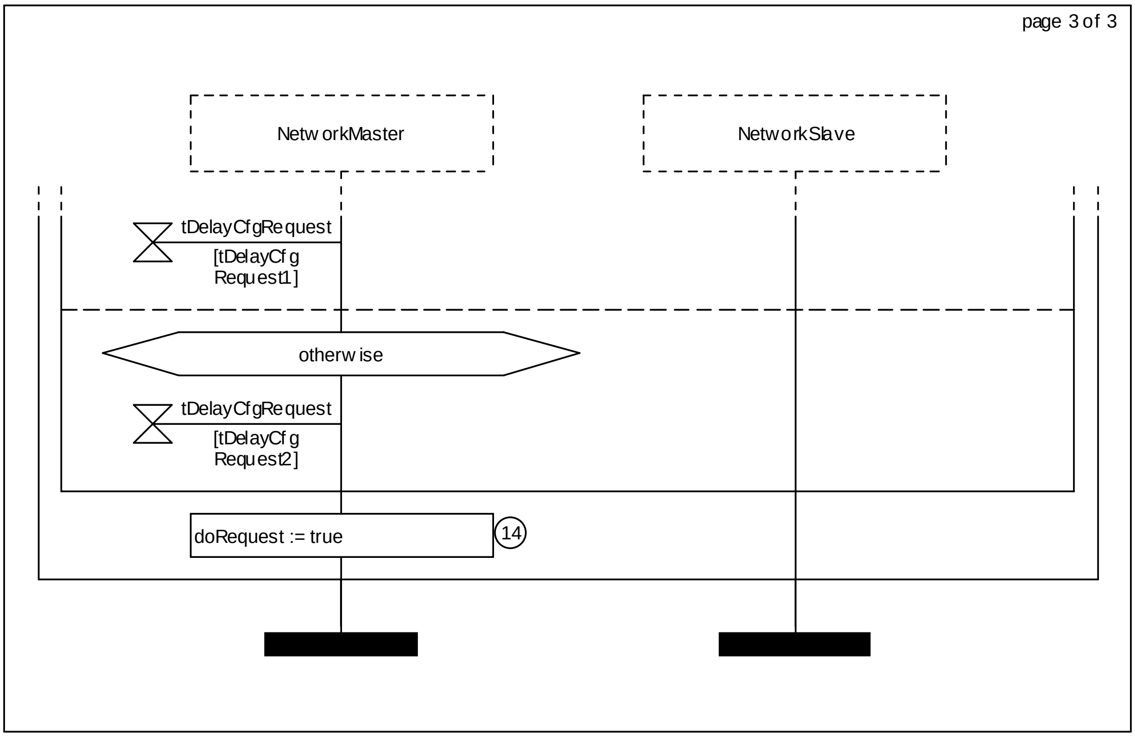
1. All nodes in the network, treated individually.
2. Only wait if an appropriate timer is running.
3. Number of scans determines the timer value.
4. Nodes that has request_nodepos = true.
5. Request is sent by physical addressing. An appropriate delay can be used between each single request.
6. To each nodepos that should be requested.
7. nodepos is replaced by currently requested nodeposition.
8. The whole MSC will be aborted on a NCE.

3.2.5 Receiving Registrations

General MSC:	NM_Gen_ReceiveConfiguration
Description:	This process handles the reception of registration messages from the network slaves. If a node registration is correct, it will always be entered into the Central Registry no matter the state of the network.
Prior Condition:	
Initiator:	Any NetworkSlave
Communication Partners:	All NetBlocks that send registrations
Events	
Timers/Timing constraints	- $t_{WaitForAnswer}$ - $t_{DelayCfgRequest}$
Remarks:	This process is not affected by an NCE.


msc NM_Gen_ReceiveConfiguration
page 1 of 3
NetworkMaster
NetworkSlave
1
alt
alt
FBlockIDs.Status
(FBlockIDList=_)
FBlockIDs.Error
(ErrorCode=_, ErrorInfo=_)
tnodepos
2
alt
when (Error in registration)
3
exc
when (numErr_nodepos < 3)
4
Increment numErr_nodepos
5
NM_Gen_SystemConfigurationUpdate
(variables 'ConfigUpdate' = 'Error')
SystemState NotOk
Delete CentralRegistry and derive NodeAddress
Delete DecentralRegistry and derive NodeAddress
loop <1, nodes in network>
tnodepos
request_nodepos := true
doRequest := true
Ignore this node until next NCE or startup
when (any InstID is invalid or duplicate)


page 2 of 3
NetworkMaster
NetworkSlave
FBlockIDs.SetGet
(FBlockID=_, OldInstID=_, NewInstID=_)
tnodepos
[tWaitForAnswer]
Update registry
6
request_nodepos := false
7
numErr_nodepos := 0
8
tnodepos
9
alt
when SystemState NotOk
opt
when (All nodes have answered OR
all tnodepos have run out)
NM_Gen_SystemConfigurationUpdate
(variables 'ConfigUpdate' = 'Success')
10
SystemState Ok
when SystemState Ok
opt
when (registry was updated)
NM_Gen_SystemConfigurationUpdate
(variables 'ConfigUpdate' = 'Success')
11
opt
when (All tnodepos have run out AND any
request_nodepos = true)
12
alt
13
when (numCompScans < 20)

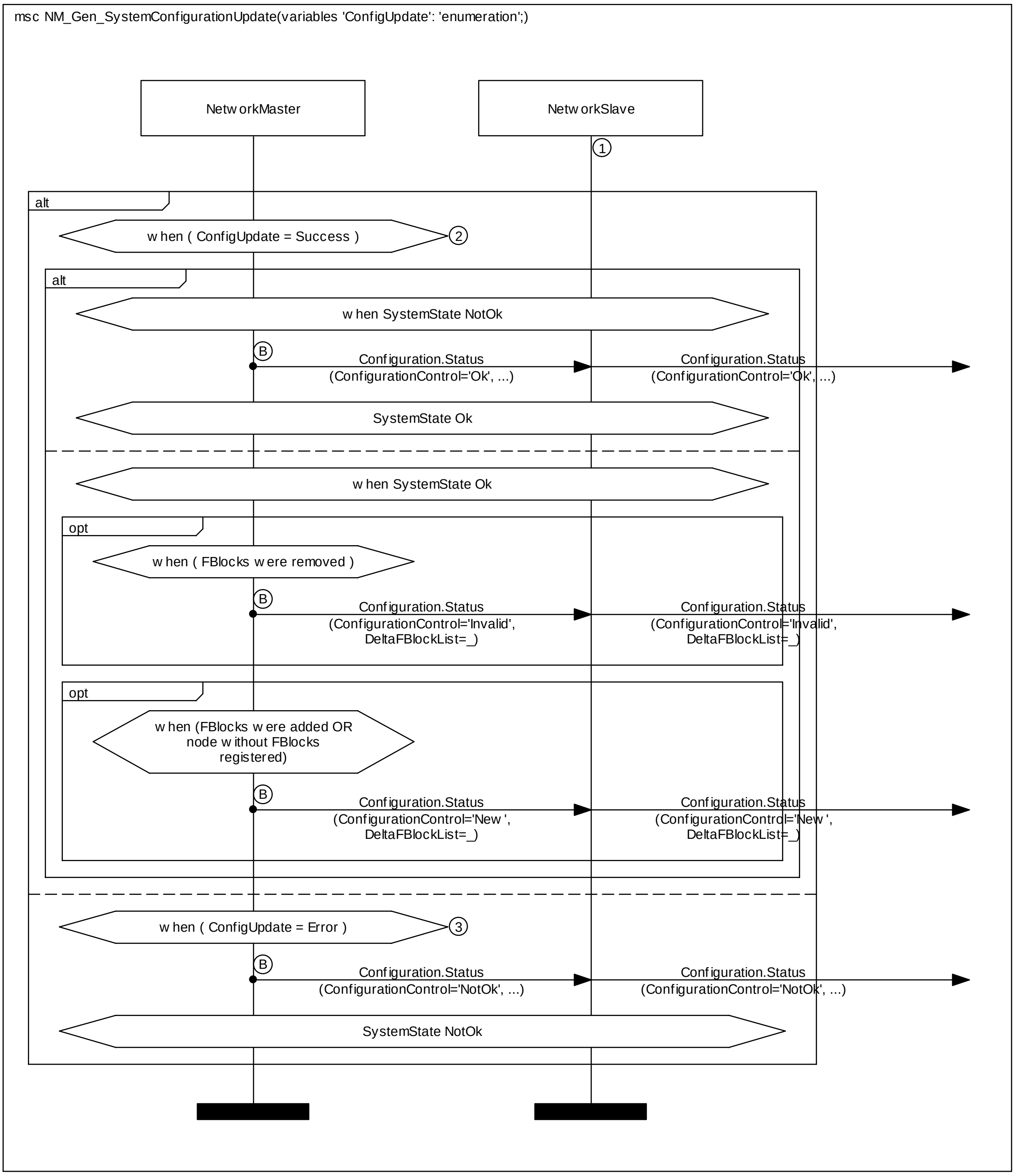


MSC 4: NM_Gen_ReceiveConfiguration

1. A node in the network.
2. nodepos is position of the node that sent the message.
3. Duplicate or invalid NodeAddress.
4. Since last NCE or startup.
5. Reset on NCE, startup, or accepted answer.
6. If any new useable information was obtained.
7. This node needs not to be requested in the next request run.
8. This is cleared since it only counts successive errors that generate a Configuration.Status(NotOk).
9. The node will be requested again in the next complementary scan.
10. Go to SystemState Ok.
11. Broadcast new information.
12. There is at least one node that has not answered.
13. Set timeout for the complementary scan.
14. Complementary scan is started when tDelayCfgRequest has run out.

3.2.6 Updating SystemState

General MSC:	NM_Gen_SystemConfigurationUpdate
Description:	This MSC shows how the NetworkMaster determines the value of the ConfigurationControl parameter of the Configuration.Status() message. The value of the ConfigurationControl parameter depends on the current SystemState and the value of ConfigUpdate.
Prior Condition:	
Initiator:	
Communication Partners:	All NetBlocks
Events	
Timers/Timing constraints	
Remarks:	

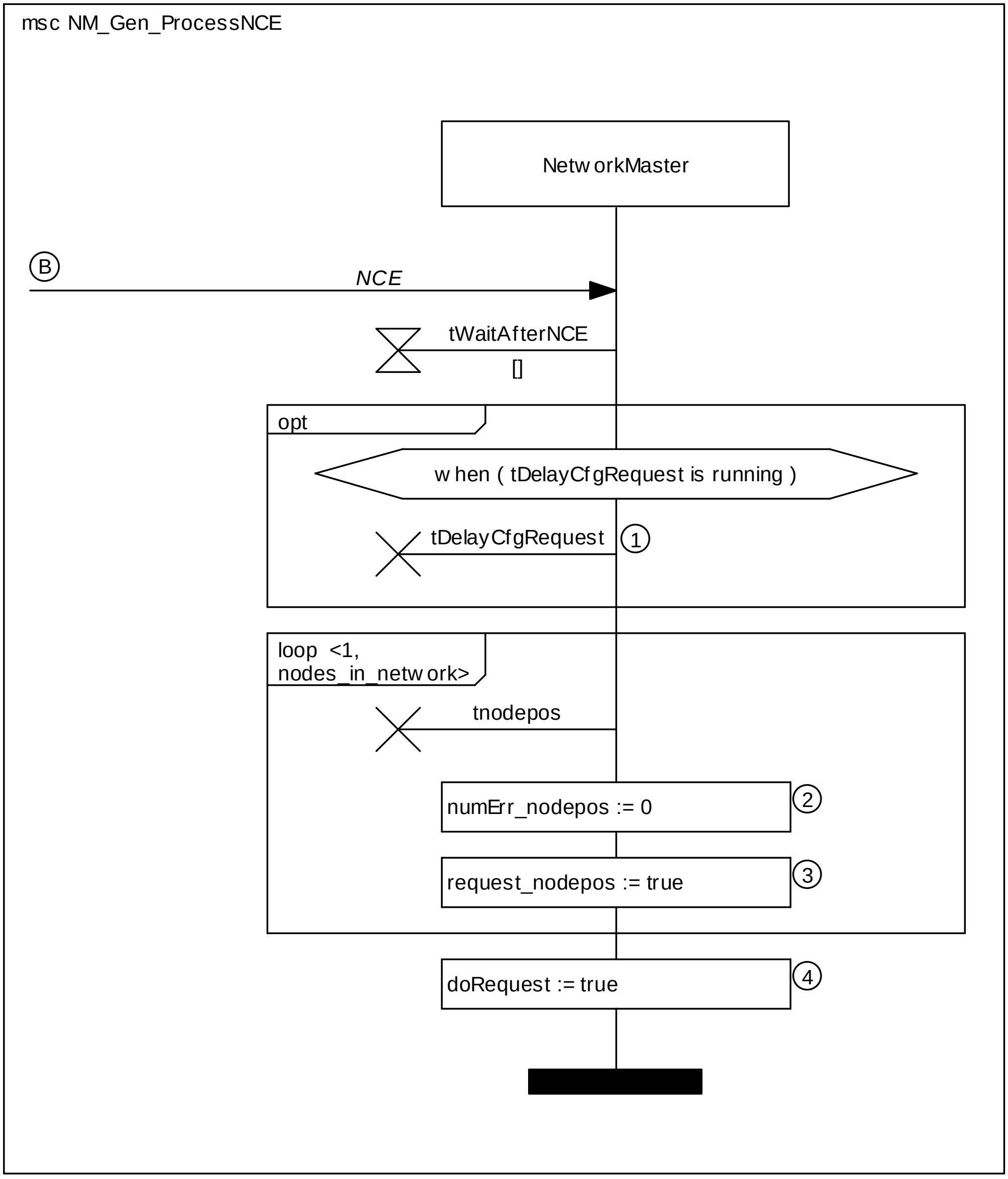

msc NM_Gen_SystemConfigurationUpdate(variables 'ConfigUpdate': 'enumeration';)
NetworkMaster
NetworkSlave
1
alt
when (ConfigUpdate = Success)
2
alt
when SystemState NotOk
B
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
SystemState Ok
when SystemState Ok
opt
when (FBlocks were removed)
B
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
opt
when (FBlocks were added OR node without FBlocks registered)
B
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
when (ConfigUpdate = Error)
3
B
Configuration.Status (ConfigurationControl='NotOk', ...)
Configuration.Status (ConfigurationControl='NotOk', ...)
SystemState NotOk

MSC 5: NM_Gen_SystemConfigurationUpdate

1. All nodes in the network, treated individually.
2. Scan or single request generated no error.
3. Scan generated a fatal error.

3.2.7 Processing NCEs

General MSC:	NM_Gen_ProcessNCE
Description:	When a NCE is detected, the whole network will be rescanned. This MSC shows how the NetworkMaster resets and sets the relevant properties.
Prior Condition:	
Initiator:	Any node switching its bypass.
Communication Partners:	
Events	NCE
Timers/Timing constraints	- $t_{DelayCfgRequest}$ - $t_{WaitAfterNCE}$ - $t_{WaitForAnswer}$ (t_nodepos)
Remarks:	



MSC 6: NM_Gen_ProcessNCE

1. Not necessary to wait for after a NCE.
2. Reset since a node may have changed position.
3. Rescan all nodes.
4. Restart requesting.

3.3 NetworkMaster Scenario MSCs

This section contains a selection of scenarios that describe the basic behavior of the NetworkMaster.

3.3.1 Scan Types

There are two types of scans used in the scenarios:

- Initial Scan - An Initial Scan follows an Init Ready event and continues until the transmission of the first Configuration.Status() message.

- Regular Scan - Regular Scans refer to all scans following a Configuration.Status() message. That is, a Regular Scan does not directly follow an Init Ready event.

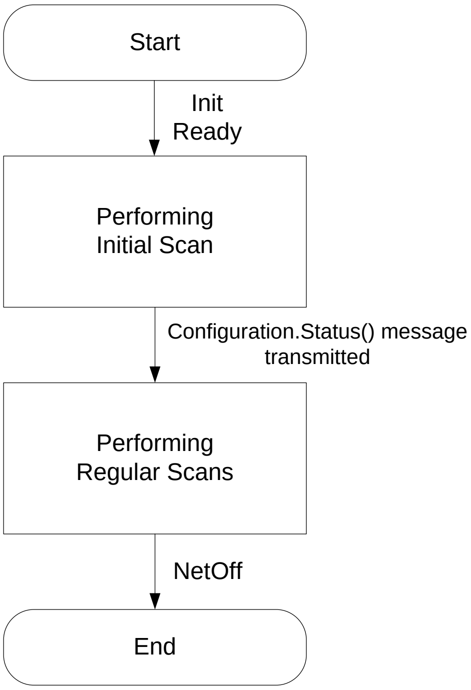


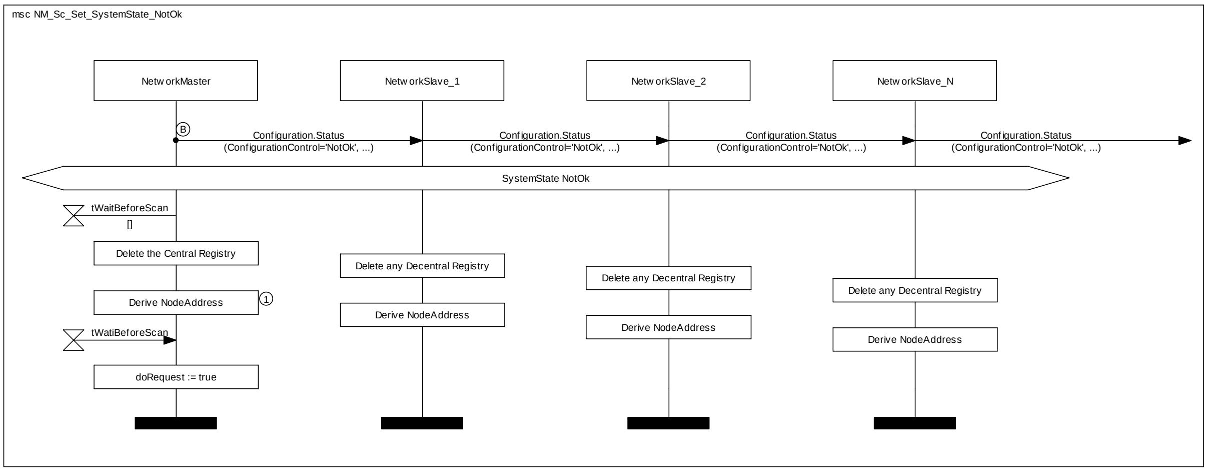
Figure 3-3: Difference between an Initial Scan and Regular Scans.

There is a need to make this differentiation because the Initial Scan has no tolerance for differences to the Central Registry (missing nodes are tolerated). If a Network Slave makes a registration that does not exactly match the Central Registry, a Configuration.Status(NotOk) will be broadcast and a Regular Scan is started. When performing a Regular Scan, the NetworkMaster has the ability to correct errors and mismatches in Network Slave registrations.

3.3.2 Setting the SystemState to NotOk

This scenario is used in the other scenarios whenever the system is reset from a network point of view.

Scenario MSC:	NM_Sc_Set_SystemState_NotOk
Description:	This scenario shows what happens in the network when the NetworkMaster broadcasts Configuration.Status(NotOk). - The Central Registry is cleared. - All Decentral Registries are cleared. - All nodes recalculate their NodeAddress. - The SystemState is set to NotOk.
Prior Condition:	
Initiator:	NetworkMaster after a scan error
Communication Partners:	All NetworkSlaves
Events	
Timers/Timing constraints	$t_{WaitBeforeScan}$
Remarks:	This scenario is valid for all SystemStates.


msc NM_Sc_Set_SystemState_NotOk
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
B
Configuration.Status (ConfigurationControl='NotOk', ...)
Configuration.Status (ConfigurationControl='NotOk', ...)
Configuration.Status (ConfigurationControl='NotOk', ...)
Configuration.Status (ConfigurationControl='NotOk', ...)
SystemState NotOk
tWaitBeforeScan
[]
Delete the Central Registry
Derive NodeAddress
1
tWatiBeforeScan
doRequest := true
Delete any Decentral Registry
Derive NodeAddress
Delete any Decentral Registry
Derive NodeAddress
Delete any Decentral Registry
Derive NodeAddress

MSC 7: NM_Sc_Set_SystemState_NotOk

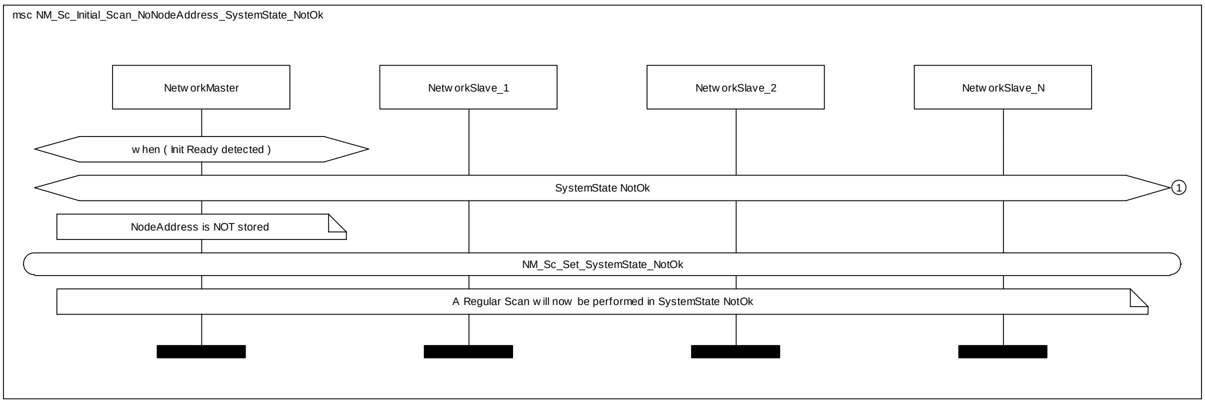
1. The address should be static, stored, or calculated.

3.3.3 Initial Scan

Initial scans follow directly after an Init Ready event until a Configuration.Status() message is transmitted.

3.3.3.1 Initial Scan without Node Address

Scenario MSC:	NM_Sc_Initial_Scan_NoNodeAddress_SystemState_NotOk
Description:	This scenario is started by the Init Ready event. When the Init Ready event is detected, the NetworkMaster checks if it has a NodeAddress stored from last run. In this scenario, it does not have a NodeAddress stored so it broadcasts Configuration.Status(NotOk) to clear any Decentral Registries in the network. The NetworkMaster then starts a Regular scan.
Prior Condition:	
Initiator:	NetworkMaster following an Init Ready event
Communication Partners:	All NetworkSlaves
Events	Init Ready
Timers/Timing constraints	
Remarks:	

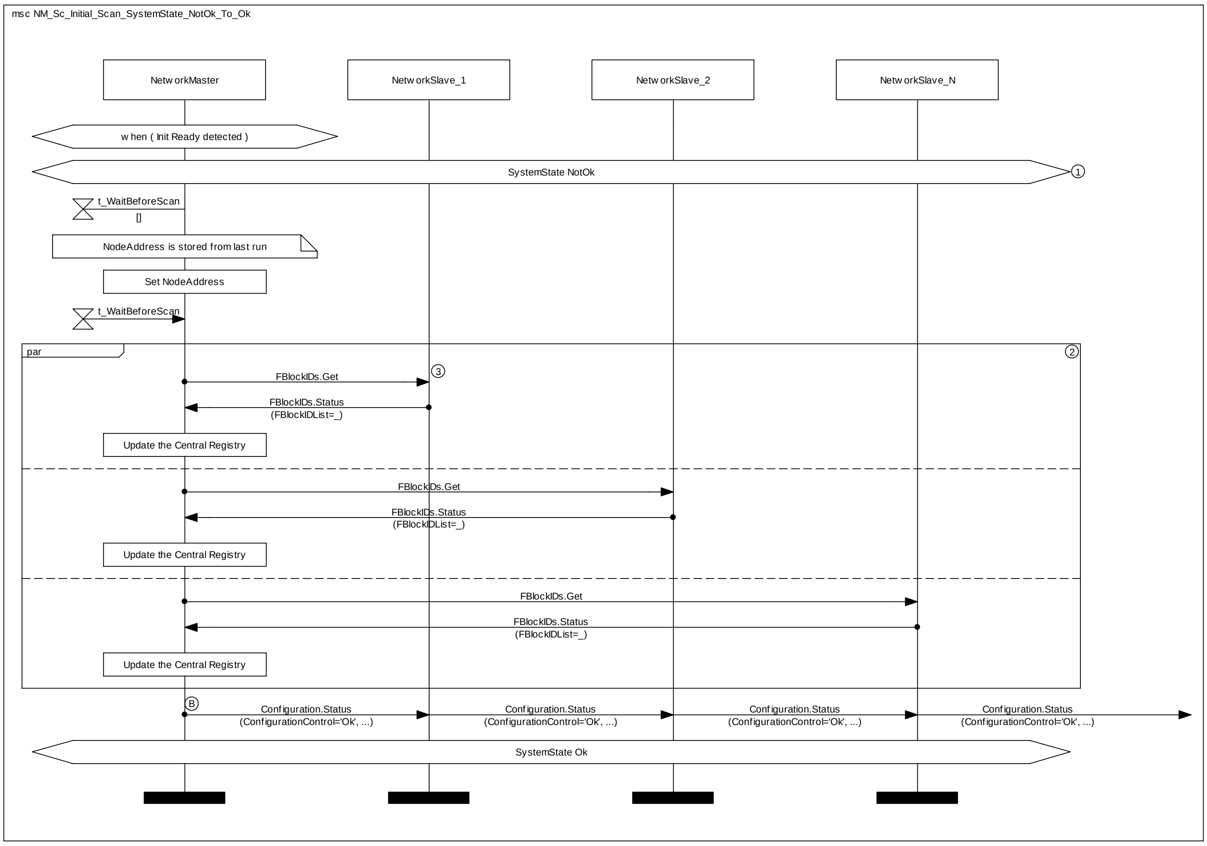

msc NM_Sc_Initial_Scan_NoNodeAddress_SystemState_NotOk
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
when (Init Ready detected)
SystemState NotOk
1
NodeAddress is NOT stored
NM_Sc_Set_SystemState_NotOk
A Regular Scan will now be performed in SystemState NotOk

MSC 8: NM_Sc_Initial_Scan_NoNodeAddress_SystemState_NotOk

1. The SystemState is always NotOk directly after the Init Ready event.

3.3.3.2 Initial Scan System State NotOK to OK

Scenario MSC:	NM_Sc_Initial_Scan_SystemState_NotOk_To_Ok
Description:	This scenario is started by the Init Ready event. NetworkSlave_2 has a Decentral Registry stored from last run. When the Init Ready event is detected, the NetworkMaster checks if it has a NodeAddress stored from last run. In this scenario, it has a NodeAddress stored so it starts to scan the network. All nodes register correctly and the NetworkMaster broadcasts Configuration.Status(Ok).
Prior Condition:	
Initiator:	NetworkMaster after Init Ready
Communication Partners:	All NetworkSlaves
Events	Init Ready
Timers/Timing constraints	- $t_{WaitBeforeScan}$
Remarks:	


msc NM_Sc_Initial_Scan_SystemState_NotOk_To_Ok
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
when (Init Ready detected)
SystemState NotOk
1
t_WaitBeforeScan
[]
NodeAddress is stored from last run
Set NodeAddress
t_WaitBeforeScan
par
2
FBlockIDs.Get
3
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
B
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
SystemState Ok

MSC 9: NM_Sc_Initial_Scan_SystemState_NotOk_To_Ok

1. The SystemState is always NotOk directly after the Init Ready event.
2. This parallel part can also be done sequentially or a mixture of both.
3. Addressing is done by using NodePositionAddress.

3.3.3.3 Initial Scan with Node Not Responding

This MSC has become obsolete because the notion of a stored Central Registry no longer exists.

Scenario MSC:	NM_Sc_Initial_Scan_CR_Stored_Node_Not_Responding

3.3.3.4 Error during Initial Scan with a stored Central Registry

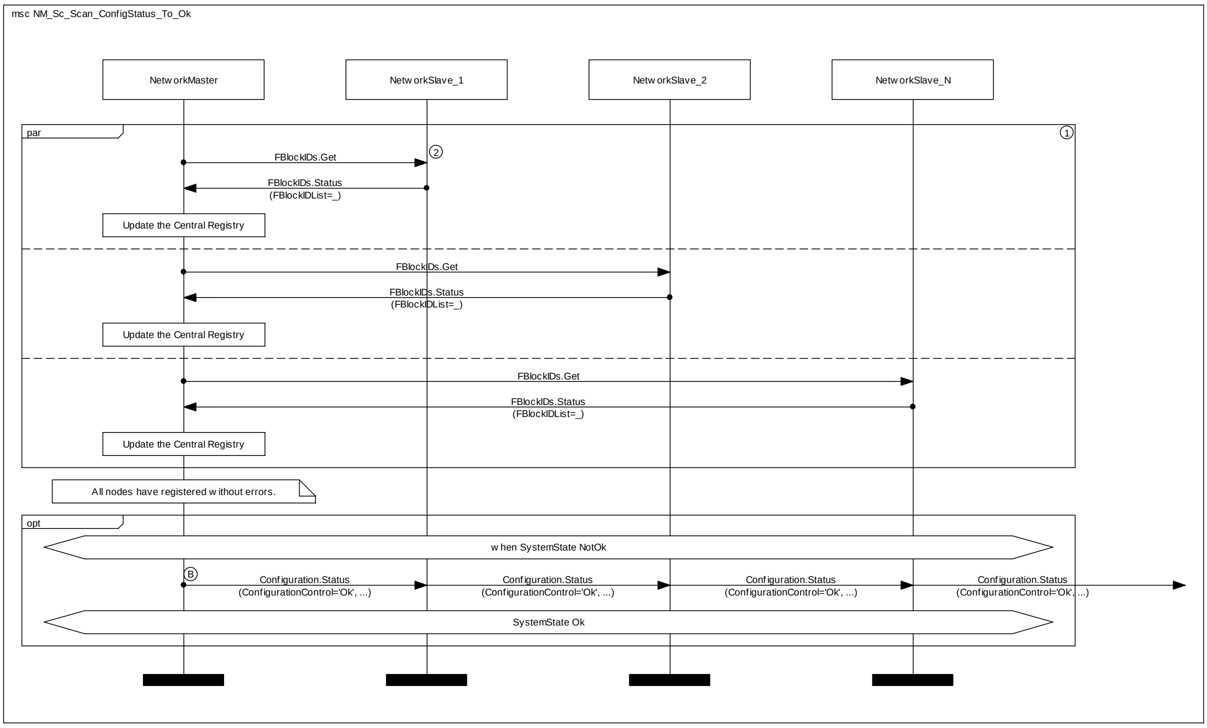
This MSC has become obsolete because the notion of a stored Central Registry no longer exists.

Scenario MSC:	NM_Sc_Initial_Scan_CR_Stored_Registration_Error

3.3.4 Regular Scan

3.3.4.1 Normal Scan without implications

Scenario MSC:	NM_Sc_Scan_ConfigStatus_To_Ok
Description:	The NetworkMaster initiates a scan.
Prior Condition:	
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events	**Application request (optional) or SystemState(NotOK)**
Timers/Timing constraints	
Remarks:	- This scenario is valid for all SystemStates. - All nodes respond correctly and on time.

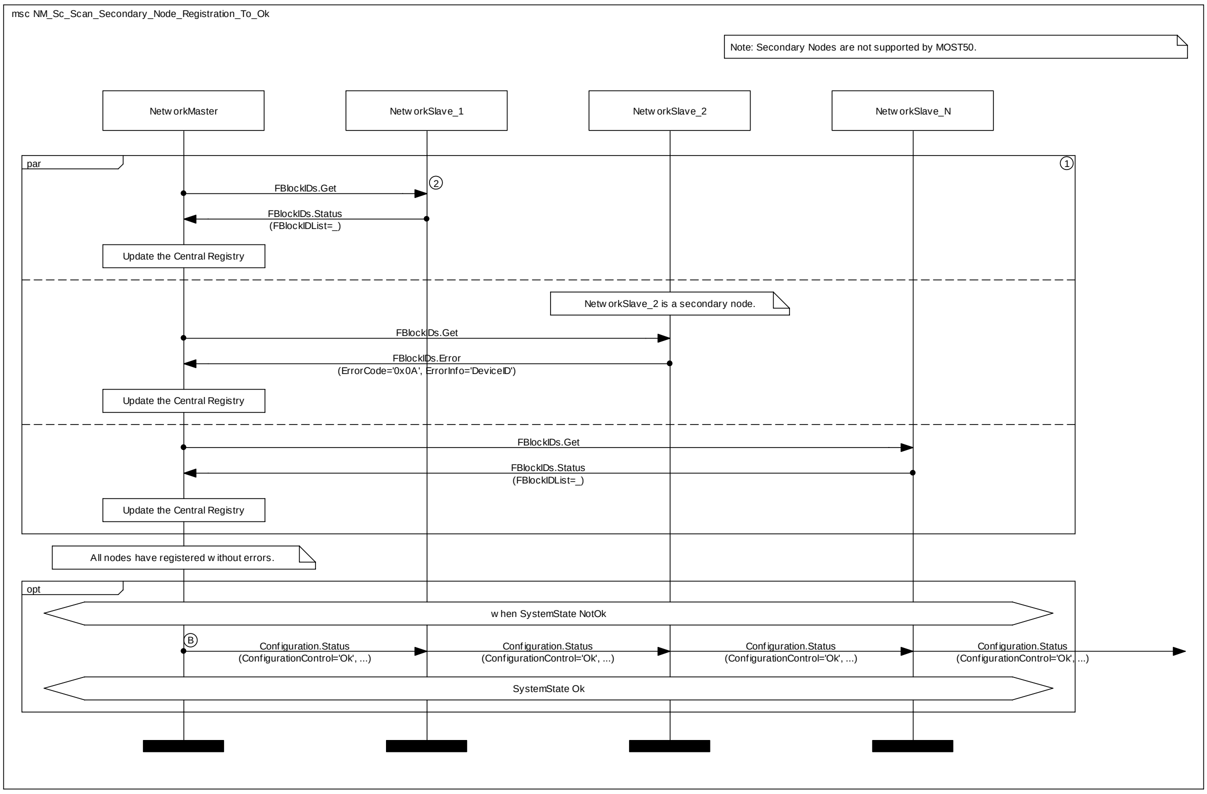


MSC 10: NM_Sc_Scan_ConfigStatus_To_Ok

1. This parallel part can also be done sequentially or a mixture of both.
2. Addressing is done by using NodePositionAddress.

3.3.4.2 Normal Scan with Secondary Node

Scenario MSC:	NM_Sc_Scan_Secondary_Node_Registration_To_Ok
Description:	The NetworkMaster scans a system where NetworkSlave_2 is a secondary node that registers correctly.
Prior Condition:	Init Ready
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events	**Application request (optional) or SystemState(NotOK)**
Timers/Timing constraints	
Remarks:	This scenario is valid for all SystemStates.

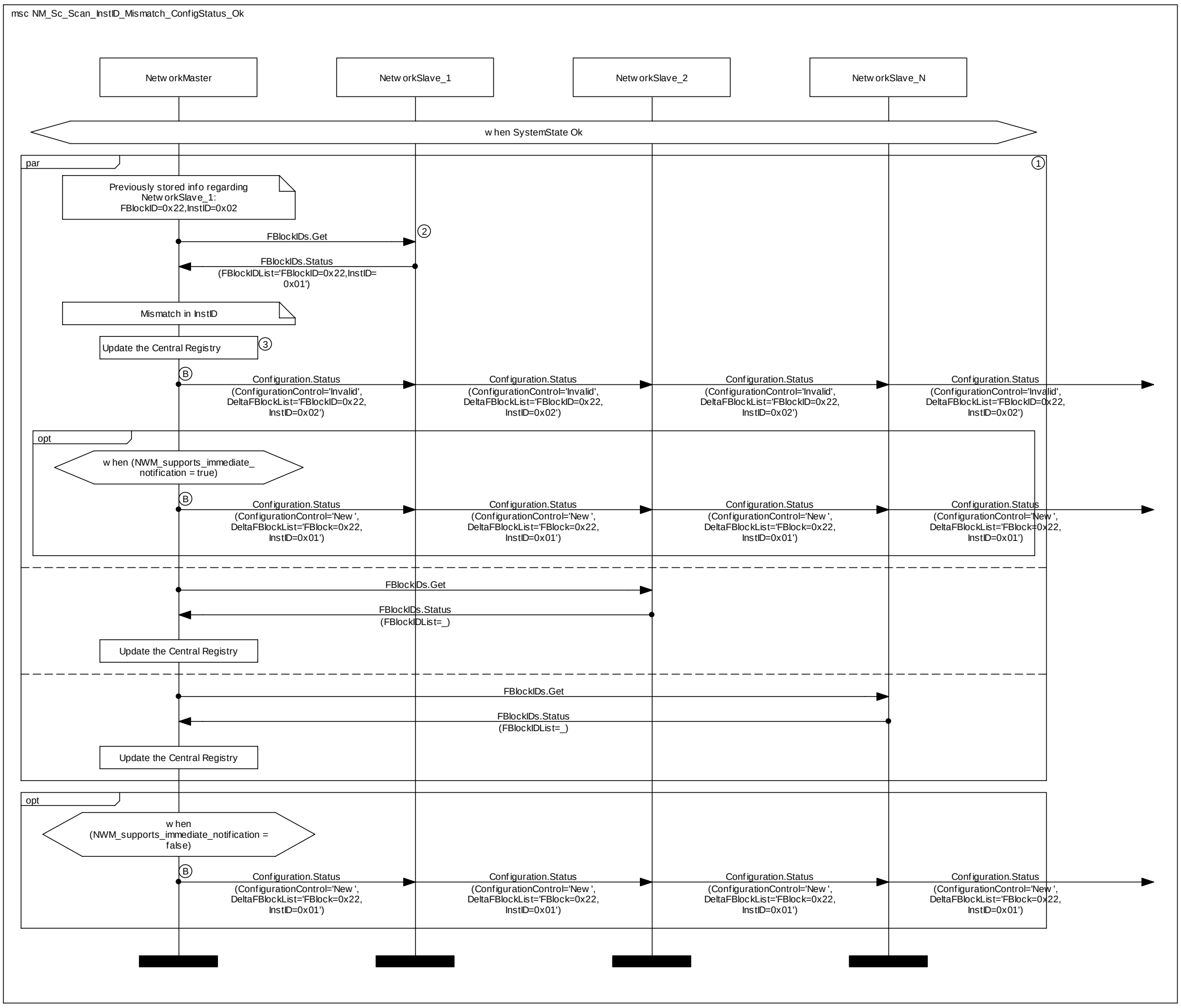

msc NM_Sc_Scan_Secondary_Node_Registration_To_Ok
Note: Secondary Nodes are not supported by MOST50.
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
par
1
FBlockIDs.Get
2
FBlockIDs.Status
(FBlockIDList=_)
Update the Central Registry
NetworkSlave_2 is a secondary node.
FBlockIDs.Get
FBlockIDs.Error
(ErrorCode='0x0A', ErrorInfo='DeviceID')
Update the Central Registry
FBlockIDs.Get
FBlockIDs.Status
(FBlockIDList=_)
Update the Central Registry
All nodes have registered without errors.
opt
when SystemState NotOk
B
Configuration.Status
(ConfigurationControl='Ok', ...)
Configuration.Status
(ConfigurationControl='Ok', ...)
Configuration.Status
(ConfigurationControl='Ok', ...)
Configuration.Status
(ConfigurationControl='Ok', ...)
SystemState Ok

MSC 11: NM_Sc_Scan_Secondary_Node_Registration_To_Ok

1. This parallel part can also be done sequentially or a mixture of both.
2. Addressing is done by using NodePositionAddress.

3.3.4.3 Mismatch in InstID in SystemState Ok

Scenario MSC:	NM_Sc_Scan_InstID_Mismatch_ConfigStatus_Ok
Description:	NetworkSlave_1 submits a registration with an InstID mismatch from a previous registration. The NetworkMaster will accept the new registration and broadcast Configuration.Status(Invalid) and Configuration.Status(New). Other nodes register with an FBlockIDList identical to the previous scan.
Prior Condition:	SystemState Ok
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events	**NCE or an application request (optional)**
Timers/Timing constraints	
Remarks:	- This scenario assumes that the InstID of NetworkSlave_2 does not collide with another FBlock. - This scenario shows the behavior during a scan but the behavior is also applicable on a single node making a registration in SystemState Ok.

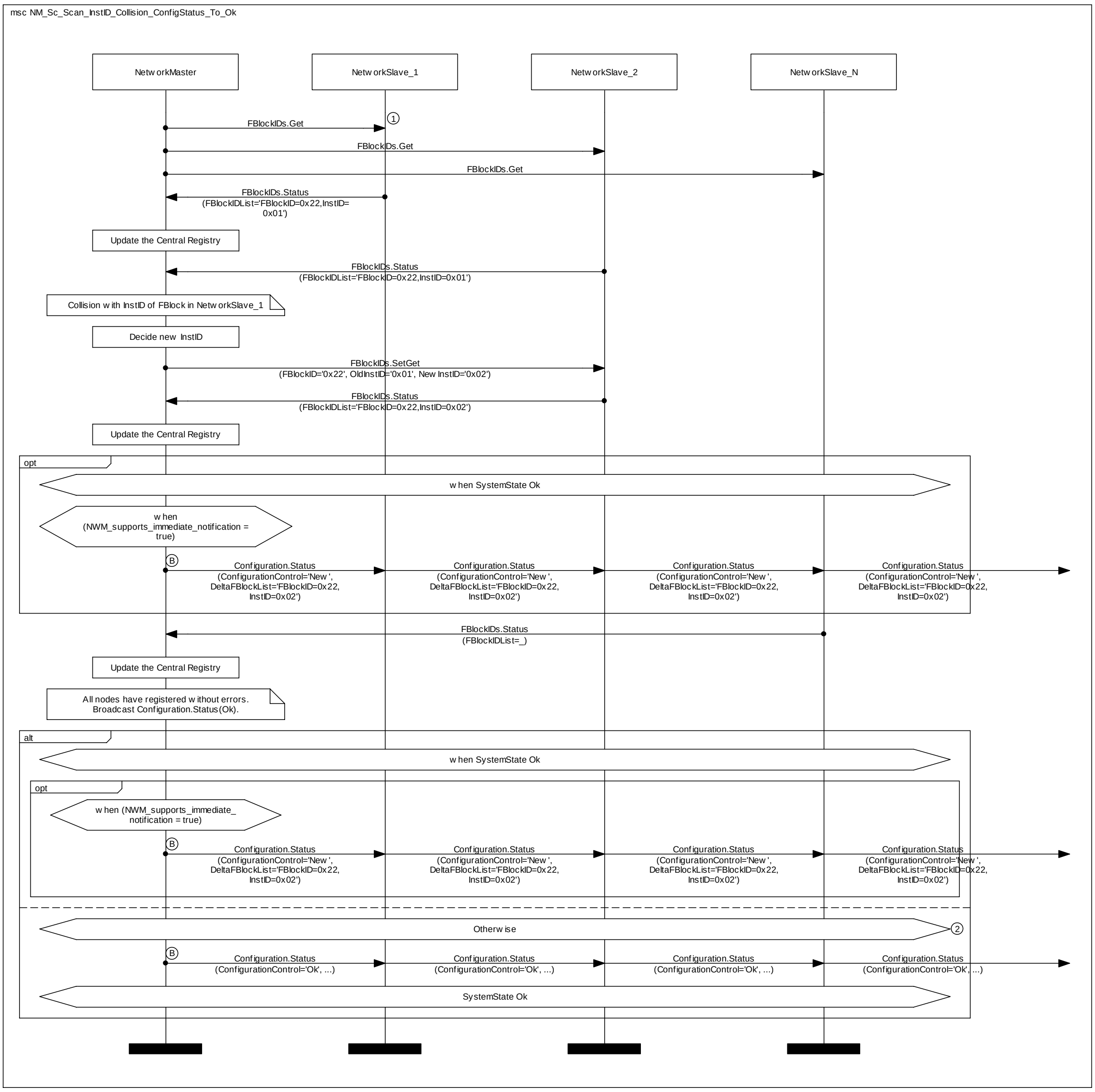

msc NM_Sc_Scan_InstID_Mismatch_ConfigStatus_Ok
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
when SystemState Ok
par
1
Previously stored info regarding NetworkSlave_1: FBlockID=0x22,InstID=0x02
FBlockIDs.Get
2
FBlockIDs.Status (FBlockIDList='FBlockID=0x22,InstID=0x01')
Mismatch in InstID
Update the Central Registry
3
B
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList='FBlockID=0x22, InstID=0x02')
opt
when (NWM_supports_immediate_notification = true)
B
Configuration.Status (ConfigurationControl='New', DeltaFBlockList='FBlock=0x22, InstID=0x01')
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
opt
when (NWM_supports_immediate_notification = false)
B
Configuration.Status (ConfigurationControl='New', DeltaFBlockList='FBlock=0x22, InstID=0x01')

MSC 12: NM_Sc_Scan_InstID_Mismatch_ConfigStatus_Ok

1. This parallel part can also be done sequentially or a mixture of both.
2. Addressing is done by using NodePositionAddress.
3. If the new InstID collides with a previously registered FBlock, the NetworkMaster may resolve this by assigning a new InstID. Please refer to section 3.3.4.4.

3.3.4.4 Collision between InstIDs

Scenario MSC:	NM_Sc_Scan_InstID_Collision_ConfigStatus_To_Ok
Description:	The NetworkMaster scans the network and NetworkSlave_2 registers an FBlock with an InstID that collides with an FBlock instance in NetworkSlave_1. In case there is a collision between two InstIDs, the NetworkMaster can set a new InstID for the colliding FBlock. The NetworkMaster reports the changes to the network differently depending on the current SystemState.
Prior Condition:	
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events:	**NCE, application request (optional), or SystemState(NotOK)**
Timers/Timing constraints:	
Remarks:	This scenario is valid for all SystemStates.

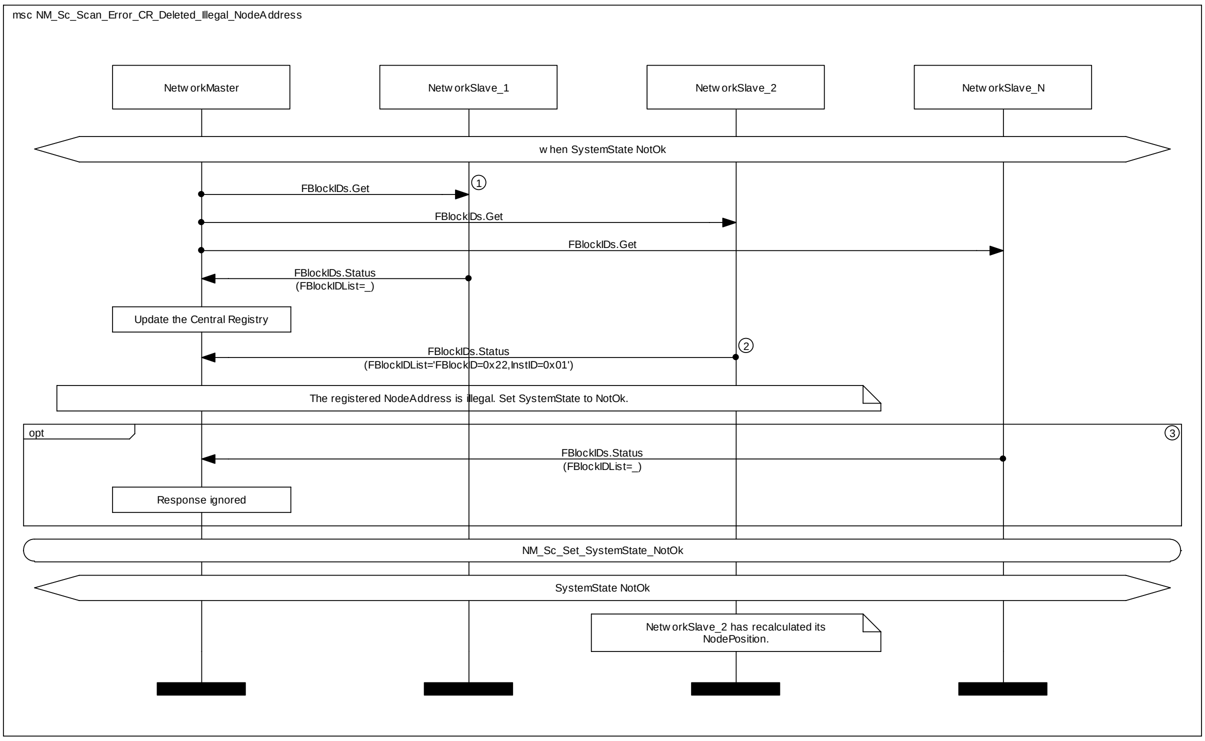

msc NM_Sc_Scan_InstID_Collision_ConfigStatus_To_Ok
Netw orkMaster
Netw orkSlave_1
Netw orkSlave_2
Netw orkSlave_N
1
FBlockIDs.Get
FBlockIDs.Get
FBlockIDs.Get
FBlockIDs.Status
(FBlockIDList='FBlockID=0x22,InstID=
0x01')
Update the Central Registry
FBlockIDs.Status
(FBlockIDList='FBlockID=0x22,InstID=0x01')
Collision w ith InstID of FBlock in Netw orkSlave_1
Decide new InstID
FBlockIDs.SetGet
(FBlockID='0x22', OldInstID='0x01', New InstID='0x02')
FBlockIDs.Status
(FBlockIDList='FBlockID=0x22,InstID=0x02')
Update the Central Registry
opt
w hen SystemState Ok
w hen
(NWM_supports_immediate_notification =
true)
B
Configuration.Status
(ConfigurationControl='New ',
DeltaFBlockList='FBlockID=0x22,
InstID=0x02')
Configuration.Status
(ConfigurationControl='New ',
DeltaFBlockList='FBlockID=0x22,
InstID=0x02')
Configuration.Status
(ConfigurationControl='New ',
DeltaFBlockList='FBlockID=0x22,
InstID=0x02')
Configuration.Status
(ConfigurationControl='New ',
DeltaFBlockList='FBlockID=0x22,
InstID=0x02')
FBlockIDs.Status
(FBlockIDList=_)
Update the Central Registry
All nodes have registered w ithout errors.
Broadcast Configuration.Status(Ok).
alt
w hen SystemState Ok
opt
w hen (NWM_supports_immediate_
notification = true)
B
Configuration.Status
(ConfigurationControl='New ',
DeltaFBlockList='FBlockID=0x22,
InstID=0x02')
Configuration.Status
(ConfigurationControl='New ',
DeltaFBlockList='FBlockID=0x22,
InstID=0x02')
Configuration.Status
(ConfigurationControl='New ',
DeltaFBlockList='FBlockID=0x22,
InstID=0x02')
Configuration.Status
(ConfigurationControl='New ',
DeltaFBlockList='FBlockID=0x22,
InstID=0x02')
Otherw ise
2
B
Configuration.Status
(ConfigurationControl='Ok', ...)
Configuration.Status
(ConfigurationControl='Ok', ...)
Configuration.Status
(ConfigurationControl='Ok', ...)
Configuration.Status
(ConfigurationControl='Ok', ...)
SystemState Ok

MSC 13: NM_Sc_Scan_InstID_Collision_ConfigStatus_To_Ok

1. Addressing is done by using NodePositionAddress.
2. SystemState NotOk

3.3.4.5 Error when Node Registers an Invalid NodeAddress

Scenario MSC:	NM_Sc_Scan_Error_CR_Deleted_Illegal_NodeAddress
Description:	In this scenario, NetworkSlave_2 makes a registration with an invalid NodeAddress.
Prior Condition:	SystemState NotOk
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events	
Timers/Timing constraints	
Remarks:	This scenario is only valid for the mechanism of parallel scanning of the system. It does not cover sequential scanning.

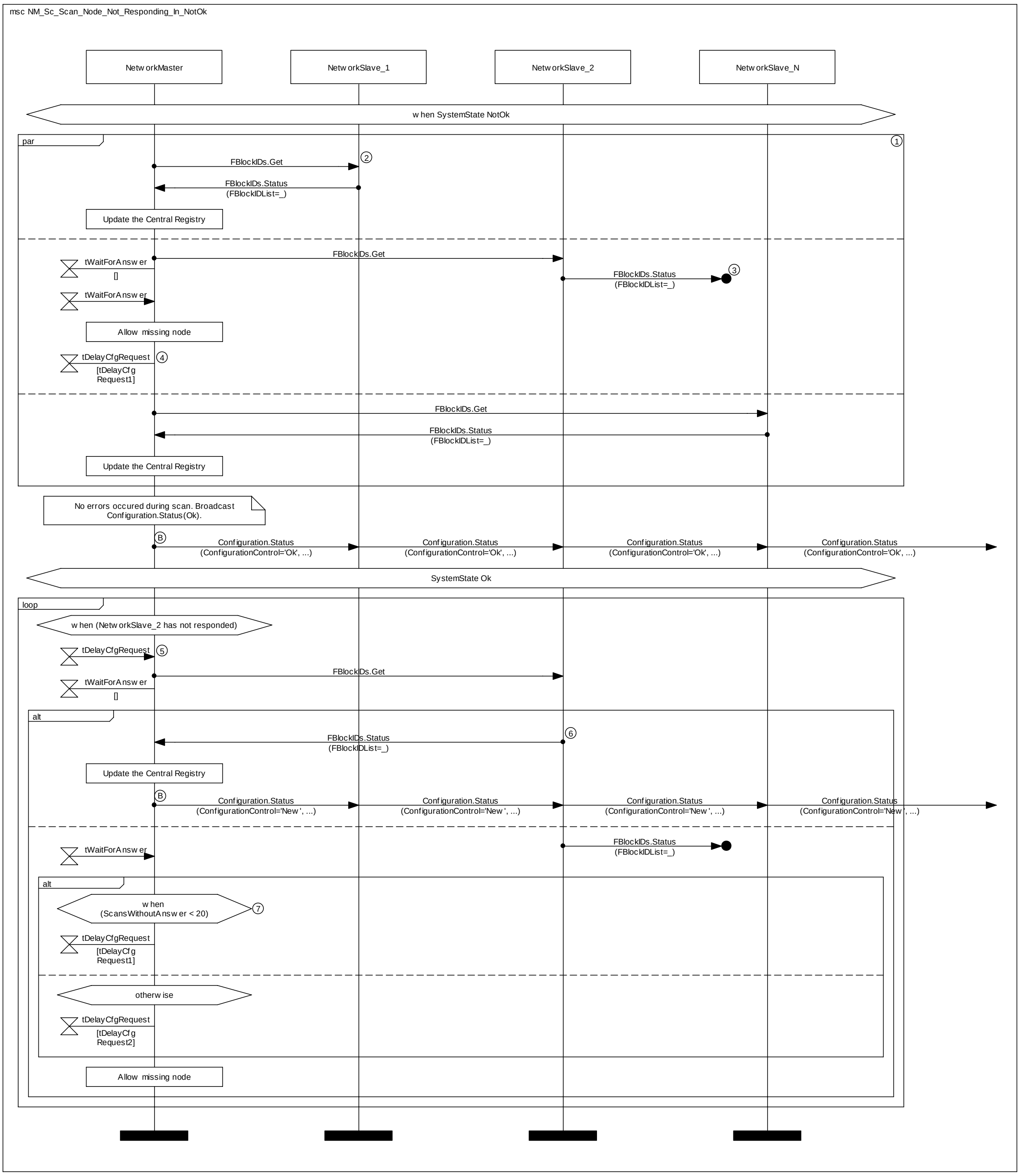

msc NM_Sc_Scan_Error_CR_Deleted_Illegal_NodeAddress
Netw orkMaster
Netw orkSlave_1
Netw orkSlave_2
Netw orkSlave_N
w hen SystemState NotOk
FBlockIDs.Get
1
FBlockIDs.Get
FBlockIDs.Get
FBlockIDs.Status
(FBlockIDList=_)
Update the Central Registry
2
FBlockIDs.Status
(FBlockIDList='FBlockID=0x22,InstID=0x01')
The registered NodeAddress is illegal. Set SystemState to NotOk.
opt
3
FBlockIDs.Status
(FBlockIDList=_)
Response ignored
NM_Sc_Set_SystemState_NotOk
SystemState NotOk
Netw orkSlave_2 has recalculated its NodePosition.

MSC 14: NM_Sc_Scan_Error_CR_Deleted_Illegal_NodeAddress

1. Addressing is done by using NodePositionAddress.
2. NodeAddress = 0xFFFF
3. A registration of a NetworkSlave can be ignored if it is received after the detection of an error.

3.3.4.6 Node Not Responding in SystemState NotOk

Scenario MSC:	NM_Sc_Scan_Node_Not_Responding_In_NotOk
Description:	The NetworkMaster scans the system in SystemState NotOk. NetworkSlave_2 does not answer the request in time. The NetworkMaster will allow this node and continue to request its configuration.
Prior Condition:	SystemState NotOk
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events	
Timers/Timing constraints	- $t_{WaitForAnswer}$ - $t_{DelayCfgRequest}$
Remarks:	- Compare to a similar scenario during an Initial Scan in section 3.3.3.3. - See also scenario is section 3.3.4.7.

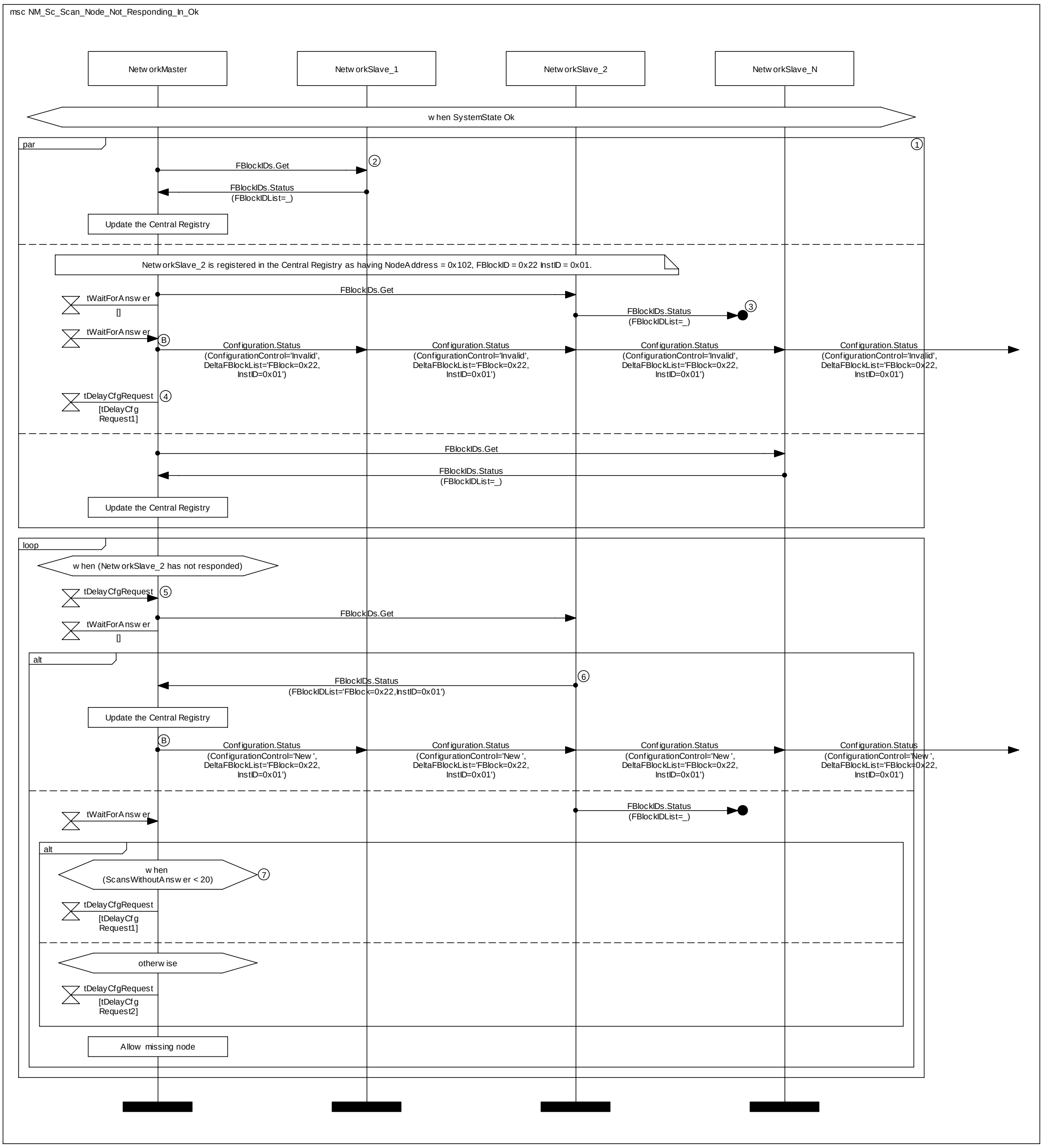


MSC 15: NM_Sc_Scan_Node_Not_Responding_In_NotOk

1. This parallel part can also be done sequentially or a mixture of both.
2. Addressing is done by using NodePositionAddress.
3. Message lost or not sent.
4. Use tDelayCfgRequest1 since it is the first time this scan that the node is scanned.
5. Wait for tDelayCfgRequest to expire.
6. NetworkSlave_2 registers new FBlocks correctly.
7. Node has not answered during 20 system scans since the network entered the state NormalOperation.

3.3.4.7 Node Not Responding in SystemState Ok

Scenario MSC:	NM_Sc_Scan_Node_Not_Responding_In_Ok
Description:	The NetworkMaster scans the system in SystemState Ok. NetworkSlave_2 does not answer the request in time. The NetworkMaster will allow this node and continue to request its configuration but it will inform the network of the invalid FBlocks that were previously registered in NetworkSlave_2.
Prior Condition:	SystemState Ok
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events	**NCE or an application request (optional)**
Timers/Timing constraints	- $t_{WaitForAnswer}$ - $t_{DelayCfgRequest}$
Remarks:	- Compare to a similar scenario during an Initial Scan in section 3.3.3.3. - See also scenario is section 3.3.4.6.

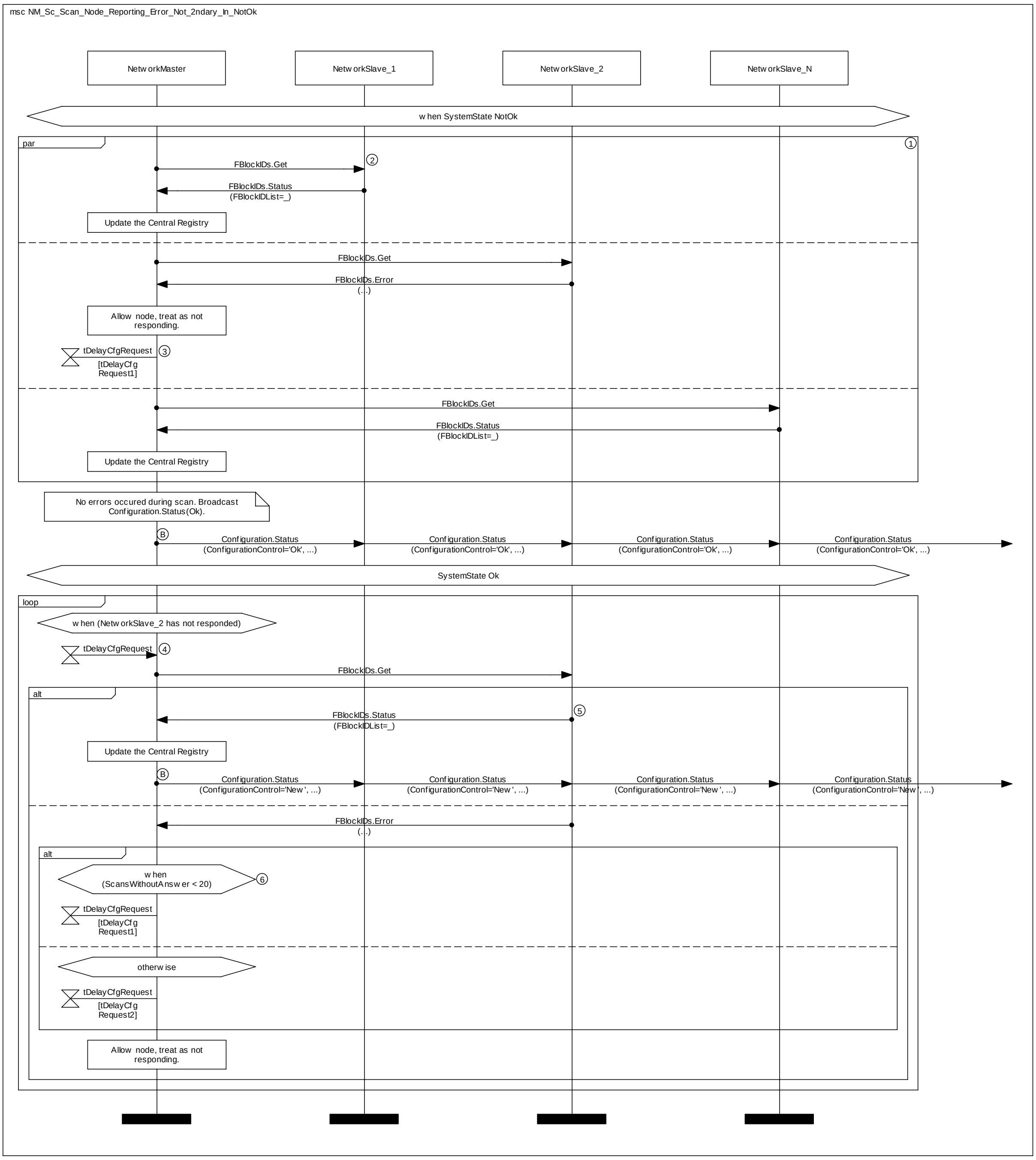


MSC 16: NM_Sc_Scan_Node_Not_Responding_In_Ok

1. This parallel part can also be done sequentially or a mixture of both.
2. Addressing is done by using NodePositionAddress.
3. Message lost or not sent.
4. Use tDelayCfgRequest1 since it is the first scan the node disappeared.
5. Wait for tDelayCfgRequest to expire.
6. NetworkSlave_2 registers new FBlocks correctly.
7. Node has not answered during 20 system scans since the network entered the state NormalOperation.

3.3.4.8 Node Reporting Error (not 2ndary) in SystemState NotOk

Scenario MSC:	NM_Sc_Scan_Node_Reporting_Error_Not_2ndary_In_NotOk
Description:	The NetworkMaster scans the system in SystemState NotOk. NetworkSlave_2 reports Error when the NetworkMaster requests its configuration. The NetworkMaster will treat this node as a non-responding node, and continue requesting FBlockIDs from this node. After 20 retries timer $t_{DelayCfgRequest}$ will change value.
Prior Condition:	SystemState NotOk
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events	
Timers/Timing constraints	$t_{DelayCfgRequest}$
Remarks:	

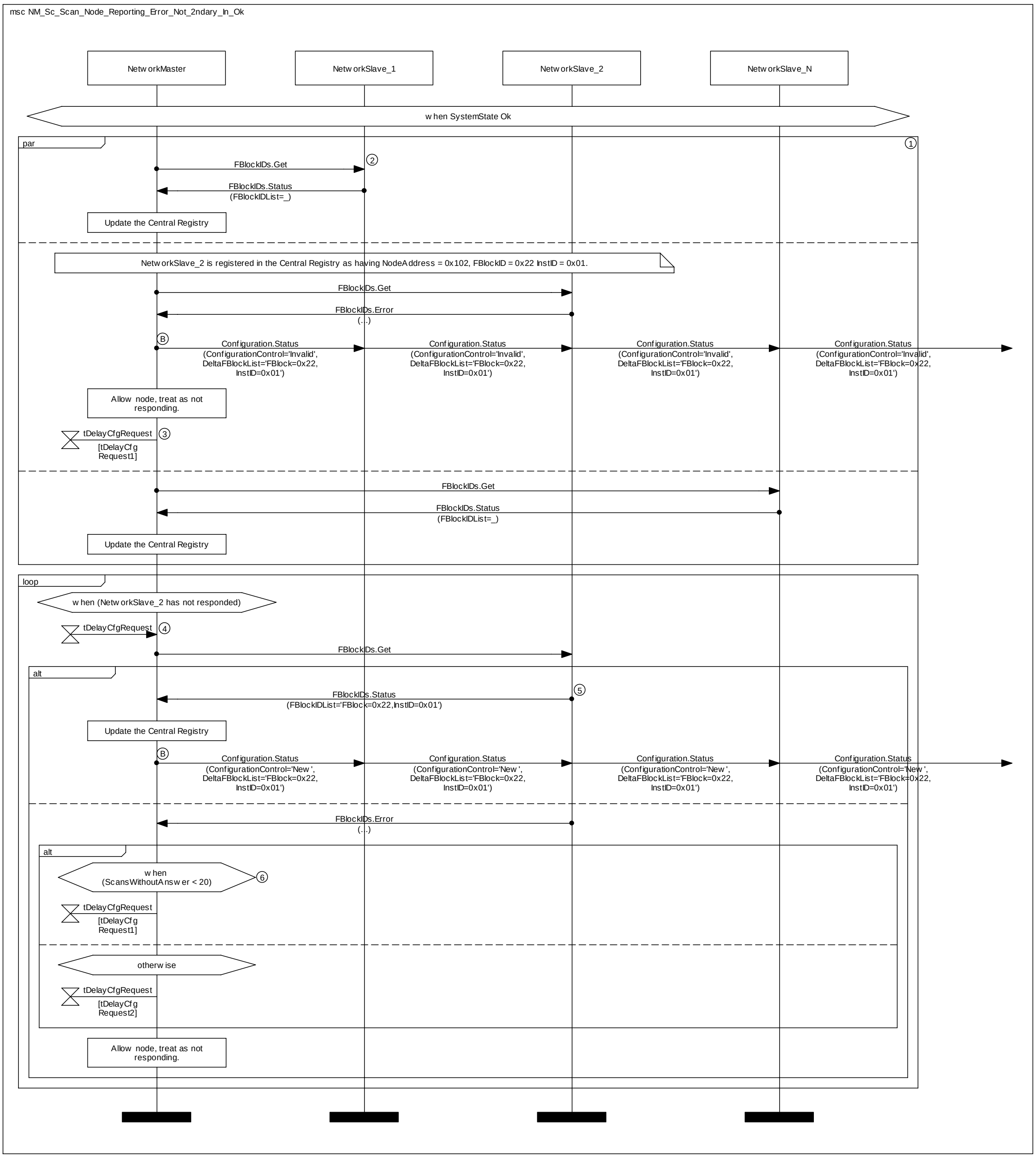

msc NM_Sc_Scan_Node_Reporting_Error_Not_2ndary_In_NotOk
Netw orkMaster
Netw orkSlave_1
Netw orkSlave_2
Netw orkSlave_N
w hen SystemState NotOk
par
1
FBlockIDs.Get
2
FBlockIDs.Status
(FBlockIDList=_)
Update the Central Registry
FBlockIDs.Get
FBlockIDs.Error
(...)
Allow node, treat as not responding.
tDelayCfgRequest
3
[tDelayCfg Request1]
FBlockIDs.Get
FBlockIDs.Status
(FBlockIDList=_)
Update the Central Registry
No errors occured during scan. Broadcast Configuration.Status(Ok).
B
Configuration.Status
(ConfigurationControl='Ok', ...)
Configuration.Status
(ConfigurationControl='Ok', ...)
Configuration.Status
(ConfigurationControl='Ok', ...)
Configuration.Status
(ConfigurationControl='Ok', ...)
SystemState Ok
loop
w hen (Netw orkSlave_2 has not responded)
tDelayCfgRequest
4
FBlockIDs.Get
alt
FBlockIDs.Status
(FBlockIDList=_)
5
Update the Central Registry
B
Configuration.Status
(ConfigurationControl='New ', ...)
Configuration.Status
(ConfigurationControl='New ', ...)
Configuration.Status
(ConfigurationControl='New ', ...)
Configuration.Status
(ConfigurationControl='New ', ...)
FBlockIDs.Error
(...)
alt
w hen
(ScansWithoutAnsw er < 20)
6
tDelayCfgRequest
[tDelayCfg Request1]
otherw ise
tDelayCfgRequest
[tDelayCfg Request2]
Allow node, treat as not responding.

MSC 17: NM_Sc_Scan_Node_Reporting_Error_Not_2ndary_In_NotOk

1. This parallel part can also be done sequentially or a mixture of both.
2. Addressing is done by using NodePositionAddress.
3. Use tDelayCfgRequest1 since it is the first time this scan that the node is scanned.
4. Wait for tDelayCfgRequest to expire.
5. NetworkSlave_2 registers new FBlocks correctly.
6. Node has not answered during 20 system scans since the network entered the state NormalOperation.

3.3.4.9 Node Reporting Error (not 2ndary) in SystemState Ok

Scenario MSC:	NM_Sc_Scan_Node_Reporting_Error_Not_2ndary_In_Ok
Description:	The NetworkMaster scans the system in SystemState Ok. NetworkSlave_2 reports FBlockIDs.Error() (not ErrorCode 0xA0) when the NetworkMaster requests its configuration. Since NetworkSlave_2 is registered in the Central Registry, the NetworkMaster has to inform the other NetworkSlaves that the FBlock in NetworkSlave_2 is invalid. The NetworkMaster will treat this node as a non-responding node, and continue requesting FBlockIDs from this node. After 20 retries timer $t_{DelayCfgRequest}$ will change value.
Prior Condition:	SystemState Ok
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events	**NCE or an application request (optional)**
Timers/Timing constraints	$t_{DelayCfgRequest}$
Remarks:	

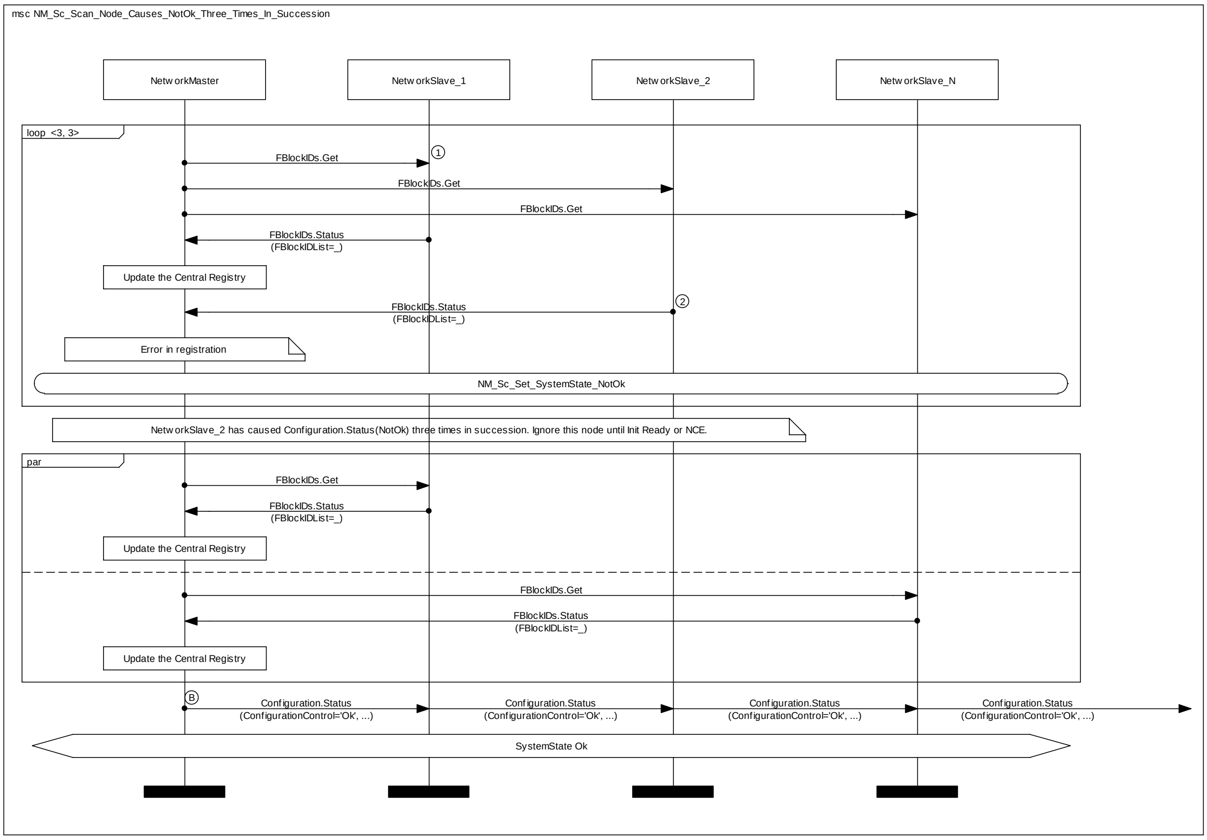

msc NM_Sc_Scan_Node_Reporting_Error_Not_2ndary_In_Ok
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
when SystemState Ok
par
1
FBlockIDs.Get
2
FBlockIDs.Status
(FBlockIDList=_)
Update the Central Registry
NetworkSlave_2 is registered in the Central Registry as having NodeAddress = 0x102, FBlockID = 0x22 InstID = 0x01.
FBlockIDs.Get
FBlockIDs.Error
(...)
B
Configuration.Status
(ConfigurationControl='Invalid',
DeltaFBlockList='FBlock=0x22,
InstID=0x01')
Allow node, treat as not responding.
tDelayCfgRequest
3
[tDelayCfg Request1]
FBlockIDs.Get
FBlockIDs.Status
(FBlockIDList=_)
Update the Central Registry
loop
when (NetworkSlave_2 has not responded)
tDelayCfgRequest
4
FBlockIDs.Get
alt
FBlockIDs.Status
(FBlockIDList='FBlock=0x22,InstID=0x01')
5
Update the Central Registry
B
Configuration.Status
(ConfigurationControl='New ',
DeltaFBlockList='FBlock=0x22,
InstID=0x01')
FBlockIDs.Error
(...)
alt
when
(ScansWithoutAnswer < 20)
6
tDelayCfgRequest
[tDelayCfg Request1]
otherwise
tDelayCfgRequest
[tDelayCfg Request2]
Allow node, treat as not responding.

MSC 18: NM_Sc_Scan_Node_Reporting_Error_Not_2ndary_In_Ok

1. This parallel part can also be done sequentially or a mixture of both.
2. Addressing is done by using NodePositionAddress.
3. Use tDelayCfgRequest1 since it is the first time this scan that the node has reported error this time around.
4. Wait for tDelayCfgRequest to expire.
5. NetworkSlave_2 registers new FBlocks correctly.
6. Node has not answered during 20 system scans since the network entered the state NormalOperation.

3.3.4.10 Node causing NotOk three times in succession

Scenario:	NM_Sc_Scan_Node_Causes_NotOk_Three_Times_In_Succession
Description:	NetworkSlave_2 causes SystemState NotOk three times in succession. The node will be ignored until the next system start or NCE.
Prior Condition:	
Initiator:	NetworkMaster
Communication Partners:	All NetworkSlaves
Events	NCE
Timers/Timing constraints	
Remarks:	This scenario is valid for all SystemStates. This scenario is only valid for the mechanism of parallel scanning of the system. It does not cover sequential scanning.

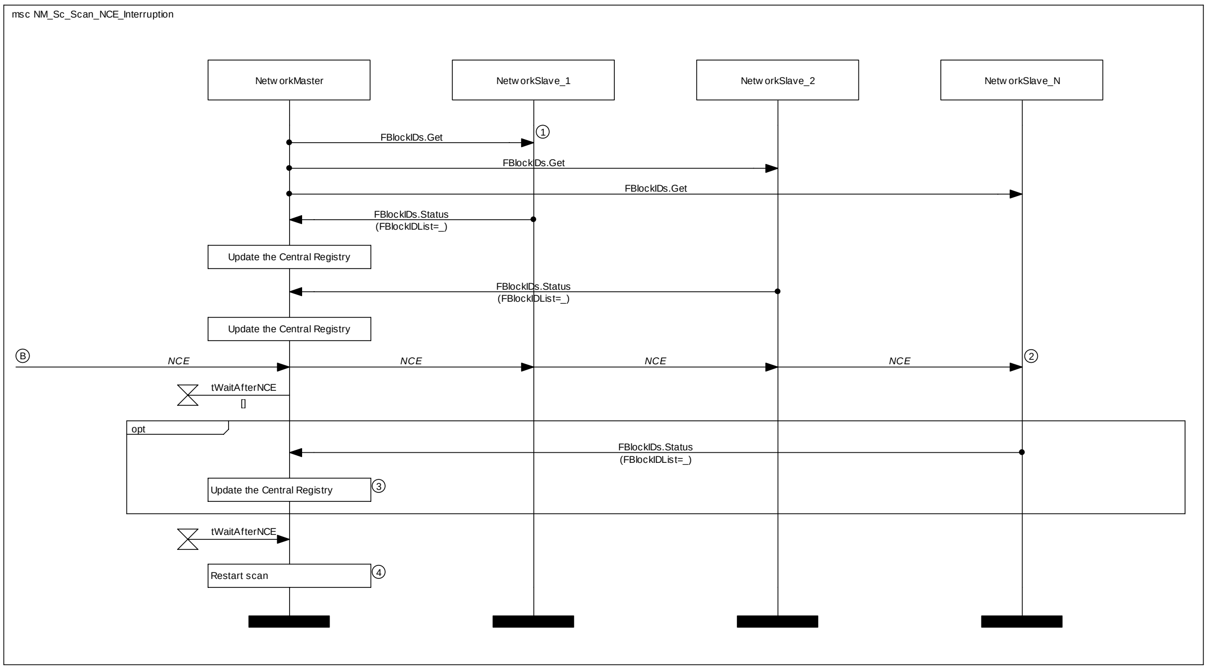

msc NM_Sc_Scan_Node_Causes_NotOk_Three_Times_In_Succession
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
loop <3, 3>
FBlockIDs.Get
1
FBlockIDs.Get
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
2
FBlockIDs.Status (FBlockIDList=_)
Error in registration
NM_Sc_Set_SystemState_NotOk
NetworkSlave_2 has caused Configuration.Status(NotOk) three times in succession. Ignore this node until Init Ready or NCE.
par
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
B
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
SystemState Ok

MSC 19: NM_Sc_Scan_Node_Causes_NotOk_Three_Times_In_Succession

1. Addressing is done by using NodePositionAddress.
2. NodeAddress = 0xFFFF

3.3.4.11 Scan Interrupted by NCE

Scenario MSC:	NM_Sc_Scan_NCE_Interruption
Description:	A scan is interrupted by a NCE. Any current scan is restarted when it is interrupted by a NCE regardless of SystemState.
Prior Condition:	
Initiator:	Any node switching its All-Bypass
Communication Partners:	All NetworkSlaves
Events	NCE
Timers/Timing constraints	$t_{WaitAfterNCE}$
Remarks:	This scenario is valid for all SystemStates This scenario is only valid for the mechanism of parallel scanning of the system. It does not cover sequential scanning.

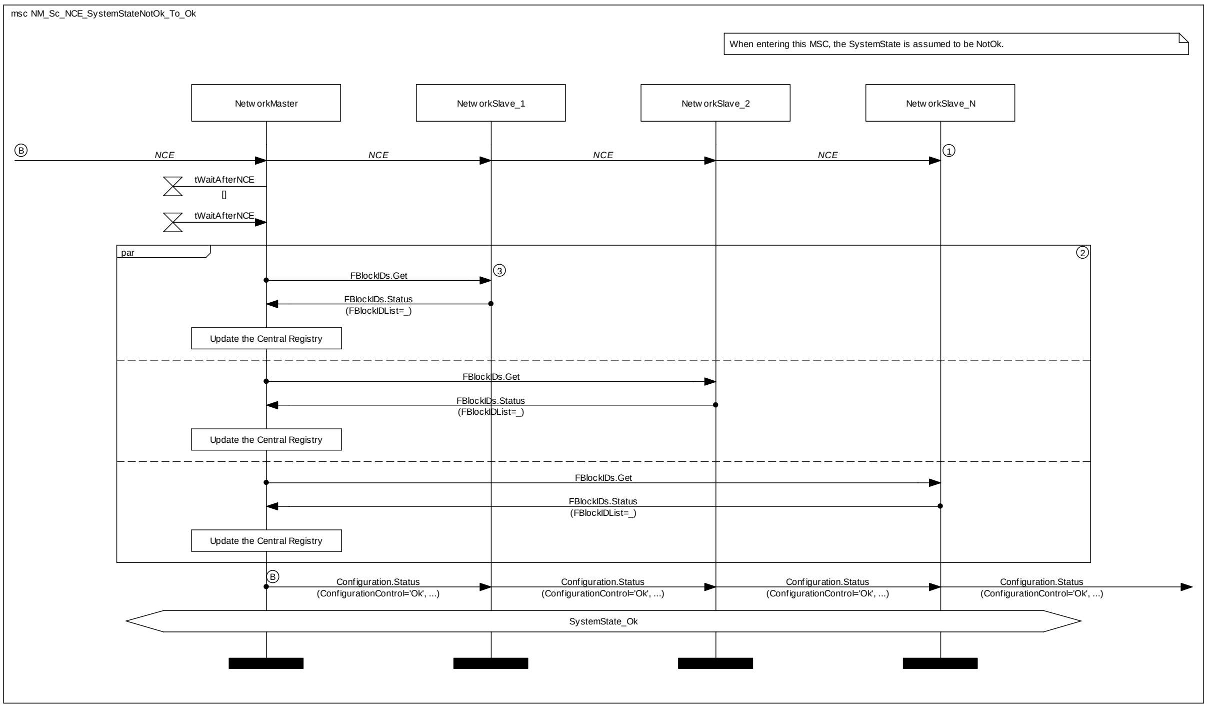


MSC 20: NM_Sc_Scan_NCE_Interruption

1. Addressing is done by using NodePositionAddress.
2. The NCE occurs during the scanning process.
3. The Central Registry is still maintained.
4. Note that the SystemState is not affected by the NCE.

3.3.4.12 NCE in SystemState NotOk Resulting in SystemState Ok

Scenario MSC:	NM_Sc_NCE_SystemStateNotOk_To_Ok
Description:	When an NCE occurs, the NetworkMaster has to scan the network (after $t_{WaitAfterNCE}$ has expired). In this scenario, all nodes respond correctly.
Prior Condition:	SystemState NotOk
Initiator:	Any node switching its All-Bypass
Communication Partners:	All NetworkSlaves
Events	NCE
Timers/Timing constraints	$t_{WaitAfterNCE}$
Remarks:	

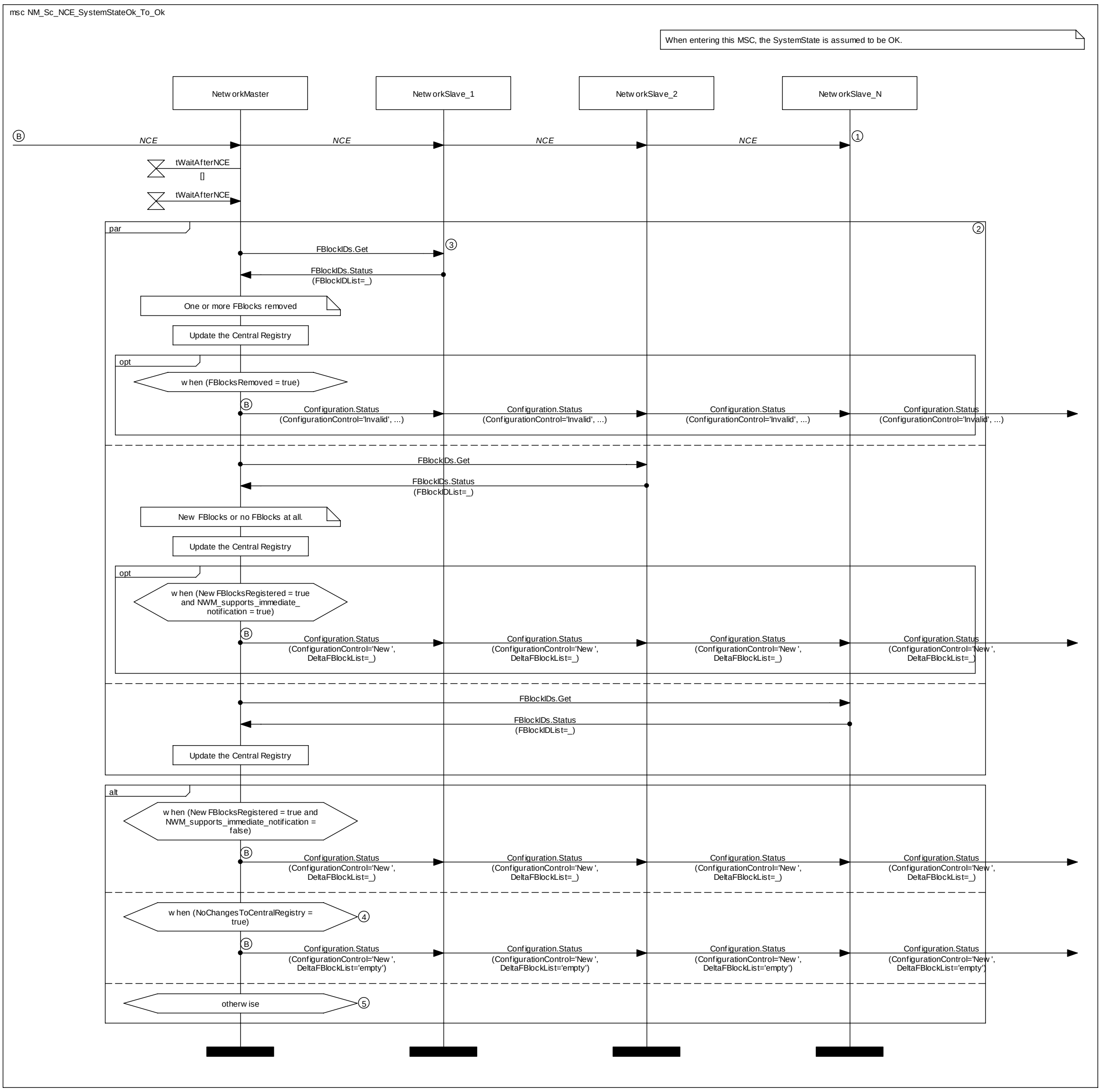


MSC 21: NM_Sc_NCE_SystemStateNotOk_To_Ok

1. When a NCE is detected, the network has to be scanned.
2. This parallel part can also be done sequentially or a mixture of both.
3. Addressing is done by using NodePositionAddress.

3.3.4.13 NCE in SystemState Ok Resulting in SystemState Ok

Scenario MSC:	NM_Sc_NCE_SystemStateOk_To_Ok
Description:	When an NCE occurs, the NetworkMaster has to scan the network (after $t_{WaitAfterNCE}$ has expired). In this scenario, all nodes respond correctly.
Prior Condition:	SystemState Ok
Initiator:	Any node switching its All-Bypass
Communication Partners:	All NetworkSlaves
Events	NCE
Timers/Timing constraints	$t_{WaitAfterNCE}$
Remarks:	


msc NM_Sc_NCE_SystemStateOk_To_Ok
When entering this MSC, the SystemState is assumed to be OK.
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
B
NCE
NCE
NCE
NCE
1
tWaitAfterNCE
[]
tWaitAfterNCE
par
2
FBlockIDs.Get
3
FBlockIDs.Status (FBlockIDList=_)
One or more FBlocks removed
Update the Central Registry
opt
when (FBlocksRemoved = true)
B
Configuration.Status (ConfigurationControl='Invalid', ...)
Configuration.Status (ConfigurationControl='Invalid', ...)
Configuration.Status (ConfigurationControl='Invalid', ...)
Configuration.Status (ConfigurationControl='Invalid', ...)
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
New FBlocks or no FBlocks at all.
Update the Central Registry
opt
when (New FBlocksRegistered = true and NWM_supports_immediate_notification = true)
B
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
FBlockIDs.Get
FBlockIDs.Status (FBlockIDList=_)
Update the Central Registry
alt
when (New FBlocksRegistered = true and NWM_supports_immediate_notification = false)
B
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
when (NoChangesToCentralRegistry = true)
4
B
Configuration.Status (ConfigurationControl='New', DeltaFBlockList='empty')
Configuration.Status (ConfigurationControl='New', DeltaFBlockList='empty')
Configuration.Status (ConfigurationControl='New', DeltaFBlockList='empty')
Configuration.Status (ConfigurationControl='New', DeltaFBlockList='empty')
otherwise
5

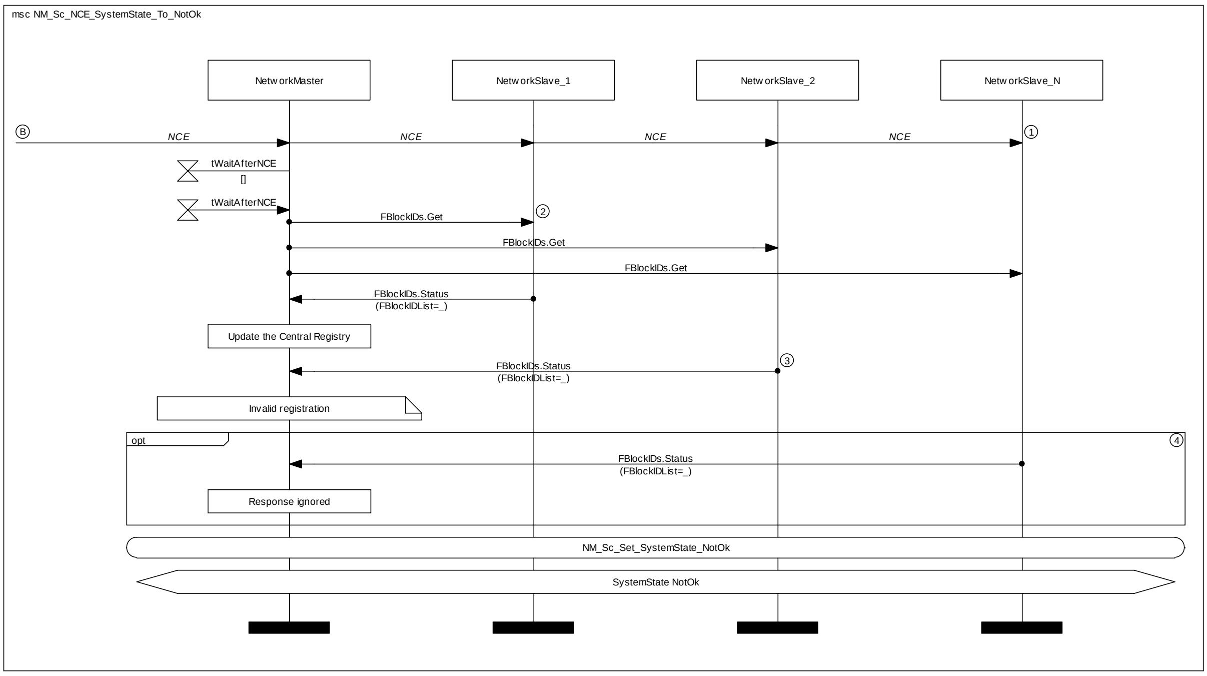
MSC 22: NM_Sc_NCE_SystemStateOk_To_Ok

1. When an NCE is detected, the network has to be scanned.
2. This parallel part can also be done sequentially or a mixture of both.
3. Addressing is done by using NodePositionAddress.

4. The reason could be, for example, a new node registration without FBlocks.
5. Due to immediate notification, Configuration.Status(New) was sent straightaway after the update to the Central Registry. Thus, no message is sent after the System Scan is complete.

3.3.4.14 NCE Resulting in SystemState NotOk

Scenario:	NM_Sc_NCE_SystemState_To_NotOk
Description:	When an NCE occurs, the NetworkMaster has to scan the network (after $t_{WaitAfterNCE}$ has expired). In this scenario, NetworkSlave_2 makes an invalid registration.
Prior Condition:	
Initiator:	Any node switching its All-Bypass
Communication Partners:	All NetworkSlaves
Events	NCE
Timers/Timing constraints	$t_{WaitAfterNCE}$
Remarks:	This scenario is valid for all SystemStates. This scenario is only valid for the mechanism of parallel scanning of the system. It does not cover sequential scanning.

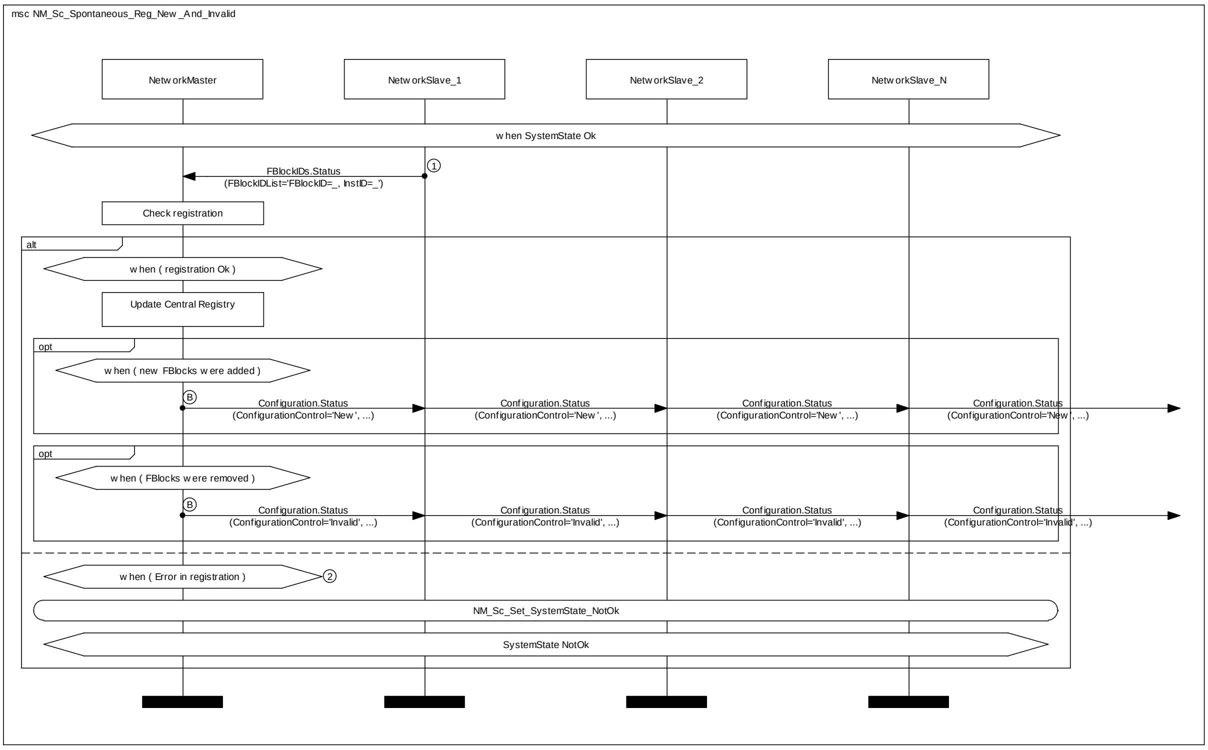


MSC 23: NM_Sc_NCE_SystemState_To_NotOk

1. When an NCE is detected, the network has to be scanned.
2. Addressing is done by using NodePositionAddress.
3. NodeAddress=0x0FFD
4. A registration of a NetworkSlave can be ignored if it is received after the detection of an error.

3.3.4.15 Spontaneous Registration of Node

Scenario MSC:	NM_Sc_Spontaneous_Reg_New_And_Invalid
Description:	NetworkSlave_1 makes a spontaneous registration. If new FBlocks are added, a Configuration.Status(New) will be broadcast. If FBlocks are removed, a Configuration.Status(Invalid) will be broadcast. If the registration is invalid, the system will change state to NotOk.
Prior Condition:	SystemState Ok
Initiator:	NetworkSlave_1
Communication Partners:	All NetworkSlaves
Events	
Timers/Timing constraints	
Remarks:	


msc NM_Sc_Spontaneous_Reg_New_And_Invalid
NetworkMaster
NetworkSlave_1
NetworkSlave_2
NetworkSlave_N
when SystemState Ok
FBlockIDs.Status
(FBlockIDList='FBlockID=_, InstID=_')
1
Check registration
alt
when (registration Ok)
Update Central Registry
opt
when (new FBlocks were added)
B
Configuration.Status
(ConfigurationControl='New', ...)
Configuration.Status
(ConfigurationControl='New', ...)
Configuration.Status
(ConfigurationControl='New', ...)
Configuration.Status
(ConfigurationControl='New', ...)
opt
when (FBlocks were removed)
B
Configuration.Status
(ConfigurationControl='Invalid', ...)
Configuration.Status
(ConfigurationControl='Invalid', ...)
Configuration.Status
(ConfigurationControl='Invalid', ...)
Configuration.Status
(ConfigurationControl='Invalid', ...)
when (Error in registration)
2
NM_Sc_Set_SystemState_NotOk
SystemState NotOk

MSC 24: NM_Sc_Spontaneous_Reg_New_And_Invalid

1. A registration of a node after the network is in SystemState Ok.
2. Note that a mismatch in InstIDs of FBlocks is not considered an error. Please refer to section 3.3.4.3.

3.3.4.16 Network Slave changes NodeAddress in SystemState OK

This MSC has been removed from the collection due to lack of compliance with the MOST Specification. A network slave is not allowed to change its NodeAddress during runtime. The NetworkMaster would signal a transition to NotOK in such an error case, as soon as the inconsistency is noticed.

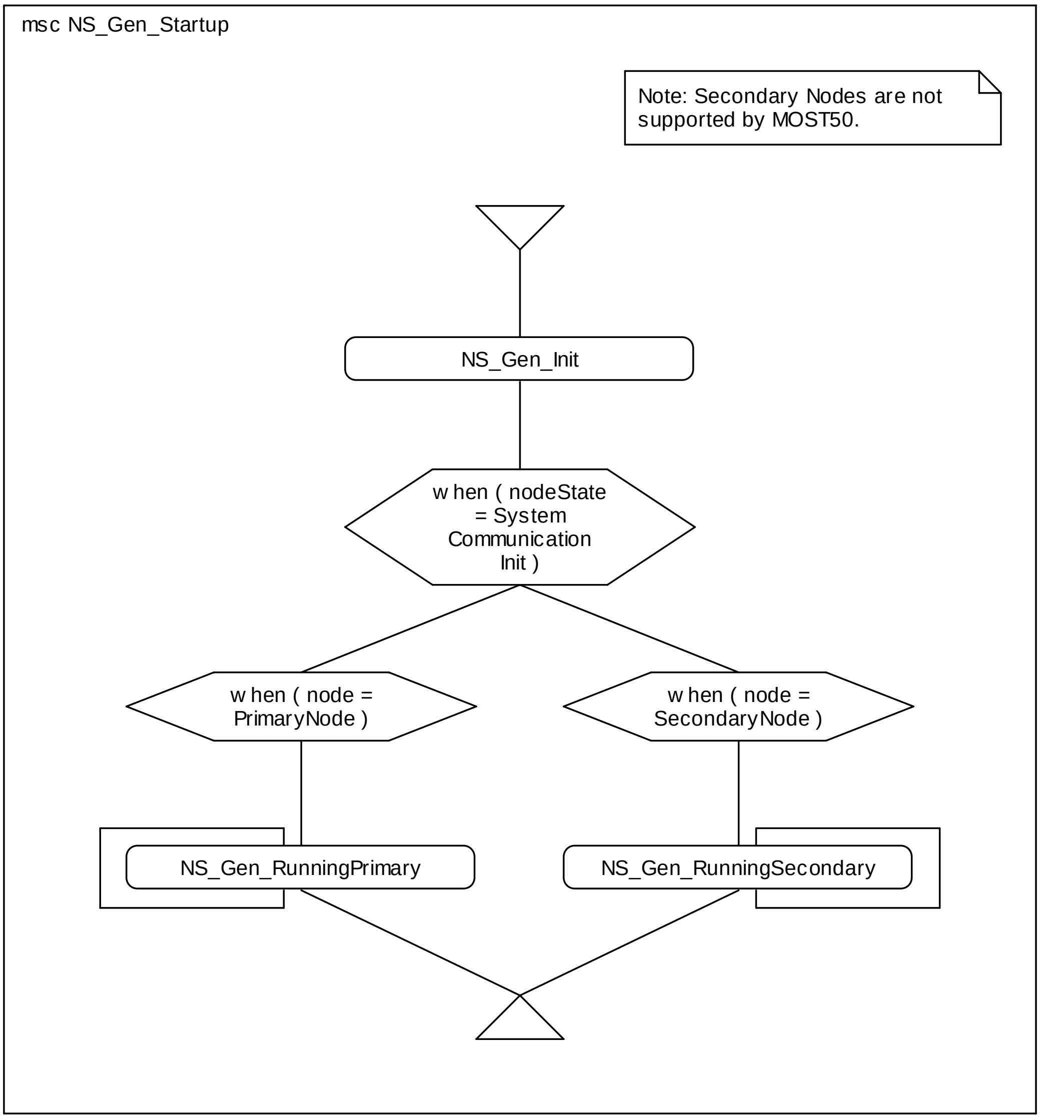
Scenario MSC:	NM_Sc_NS_Change_Of_NodeAddress

3.4 Network Slave General MSCs

The general MSCs in this section describe the complete startup sequence, from initialization to normal operation, from the perspective of a Network Slave. The high-level MSC shows how the general MSCs are combined to describe the complete flow from startup to normal (running) operation in a Network Slave.

3.4.1 High-level Network Slave MSC

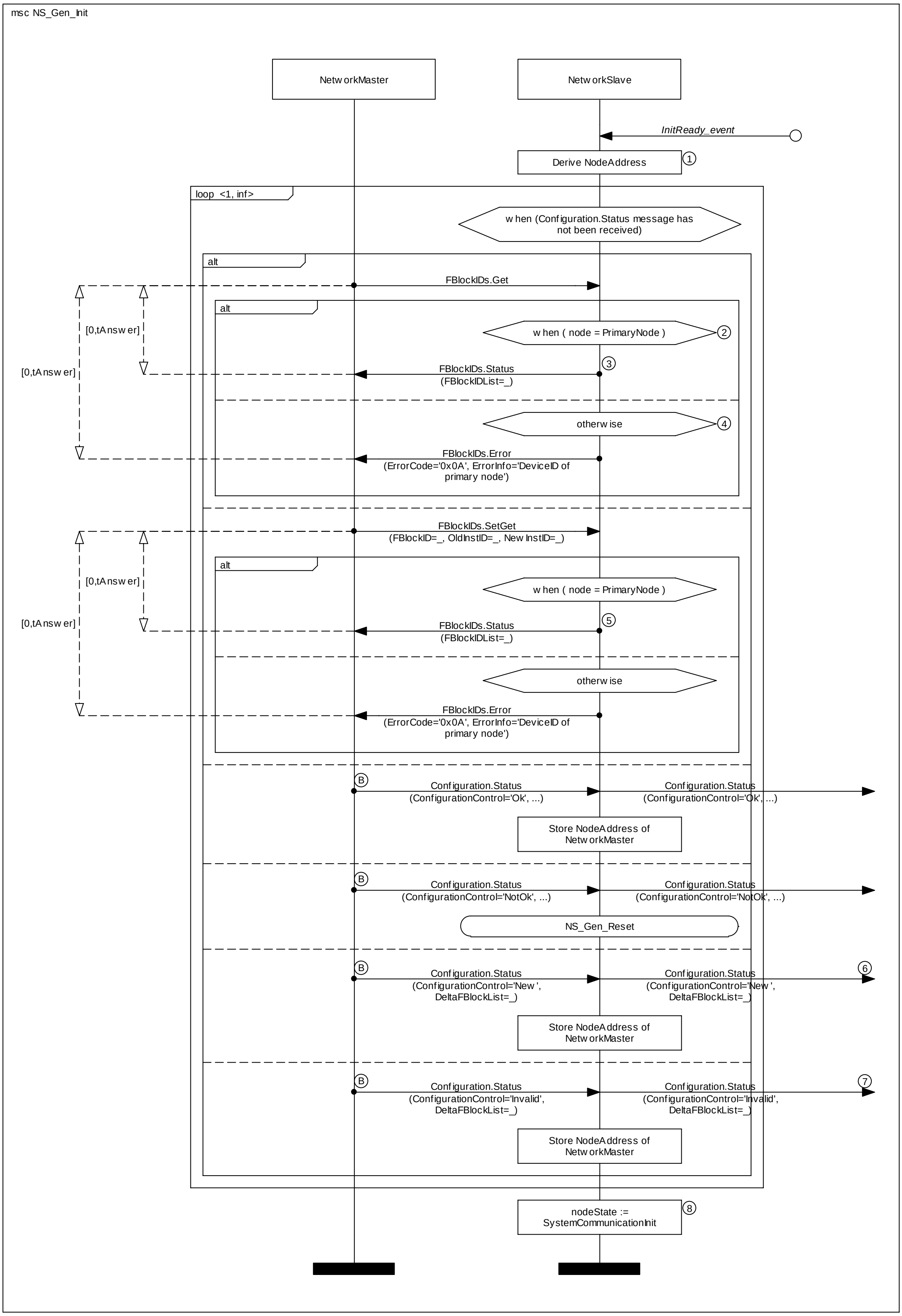
General MSC:	NS_Gen_Startup
Description:	High-level MSC of Network Slave startup process. The startup sequence for both “Primary Node” and “Secondary Node” is described in this high-level MSC.
Prior Condition:	-
Initiator:	-
Communication Partners:	-
Events:	-Init Ready
Timers/Timing constraints:	-
Remarks:	The term “Primary Node” is used regardless of the occurrence of a “Secondary Node” in the system.


msc NS_Gen_Startup
Note: Secondary Nodes are not supported by MOST50.
NS_Gen_Init
w hen (nodeState = System Communication Init)
w hen (node = PrimaryNode)
w hen (node = SecondaryNode)
NS_Gen_RunningPrimary
NS_Gen_RunningSecondary

MSC 25: NS_Gen_Startup

3.4.2 Initializing the Network Slave

General MSC:	NS_Gen_Init
Description:	Describes the initialization and startup sequence of a Network Slave after Init Ready. Both “Primary Node” and “Secondary Node” is described in this MSC.
Prior Condition:	-
Initiator:	-
Communication Partners:	- NetworkMaster
Events:	-Init Ready
Timers/Timing constraints:	- t_{Answer}
Remarks:	The term “Primary Node” is used regardless of the occurrence of a “Secondary Node” in the system.

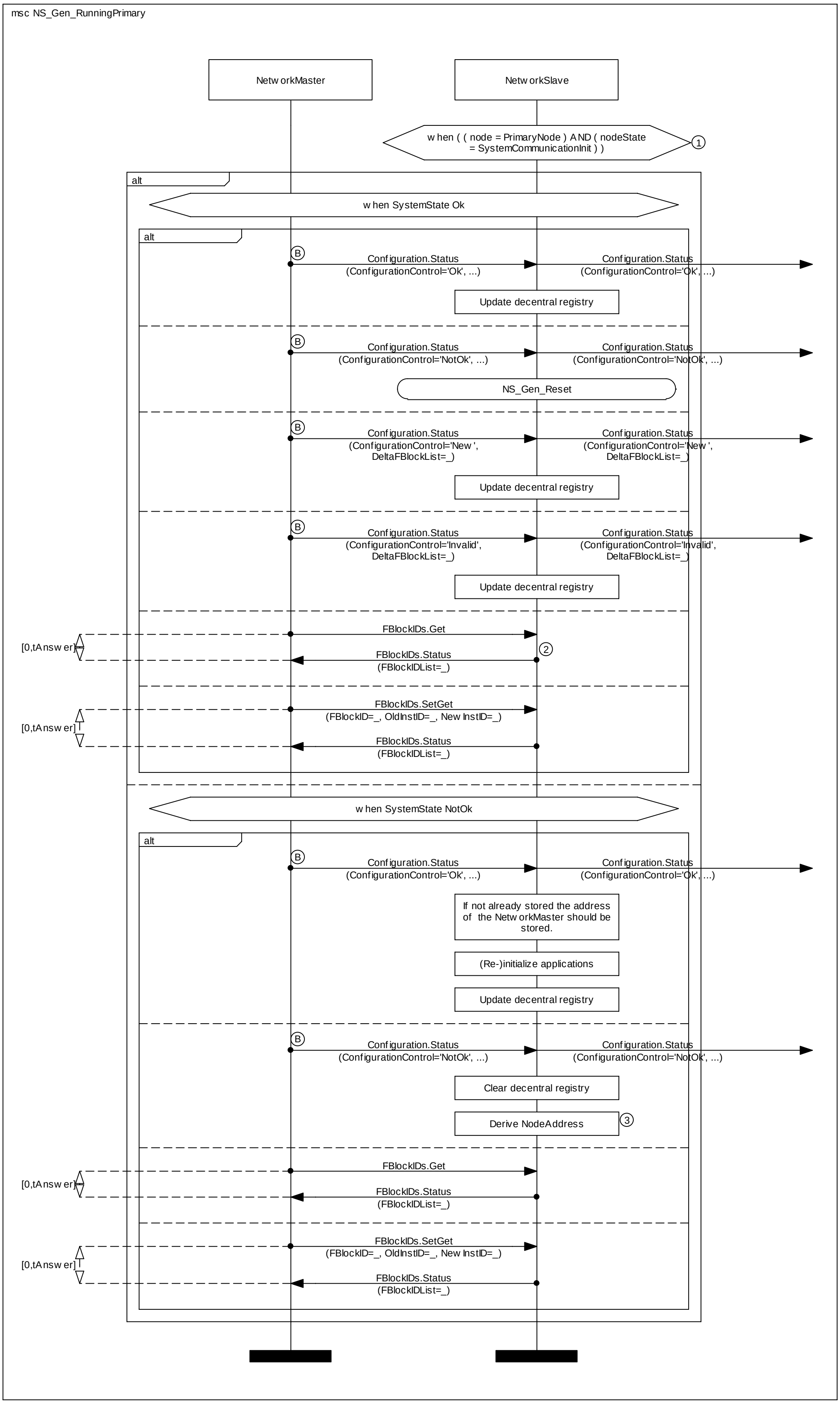

msc NS_Gen_Init
NetworkMaster
NetworkSlave
InitReady_event
Derive NodeAddress
1
loop <1, inf>
when (Configuration.Status message has not been received)
alt
FBlockIDs.Get
alt
when (node = PrimaryNode)
2
3
[0,tAnswer]
[0,tAnswer]
FBlockIDs.Status (FBlockIDList=_)
otherwise
4
FBlockIDs.Error (ErrorCode='0x0A', ErrorInfo='DeviceID of primary node')
FBlockIDs.SetGet (FBlockID=_, OldInstID=_, NewInstID=_)
alt
[0,tAnswer]
[0,tAnswer]
when (node = PrimaryNode)
5
FBlockIDs.Status (FBlockIDList=_)
otherwise
FBlockIDs.Error (ErrorCode='0x0A', ErrorInfo='DeviceID of primary node')
B
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
Store NodeAddress of NetworkMaster
B
Configuration.Status (ConfigurationControl='NotOk', ...)
Configuration.Status (ConfigurationControl='NotOk', ...)
NS_Gen_Reset
B
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
6
Store NodeAddress of NetworkMaster
B
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
7
Store NodeAddress of NetworkMaster
nodeState := SystemCommunicationInit
8

MSC 26: NS_Gen_Init

1. The address should be static or stored. If no address can be obtained in the required time, 0xFFFF should be used.
2. The term "PrimaryNode" is used regardless of an occurrence of a "SecondaryNode" in the system.
3. NetworkSlave must reply within t_Answer after the request has been received.
4. The node is a Secondary Node.
5. NetworkSlave must reply within t_Answer after the request has been received.
6. SystemState is Ok when Configuration.Status(New) is received.
7. SystemState is Ok when Configuration.Status(Invalid) is received.
8. Initialization of the application with respect to communication has completed.

3.4.3 Running Operation - Primary Node

General MSC:	NS_Gen_RunningPrimary
Description:	Describes the running (normal) operation of a Network Slave (Primary Node) after the startup sequence has been completed.
Prior Condition:	- SystemCommunicationInit (Startup sequence) has completed.
Initiator:	-
Communication Partners:	- NetworkMaster
Events:	-
Timers/Timing constraints:	- t_{Answer}
Remarks:	- The term “Primary Node” is used regardless of the occurrence of a “Secondary Node” in the system.

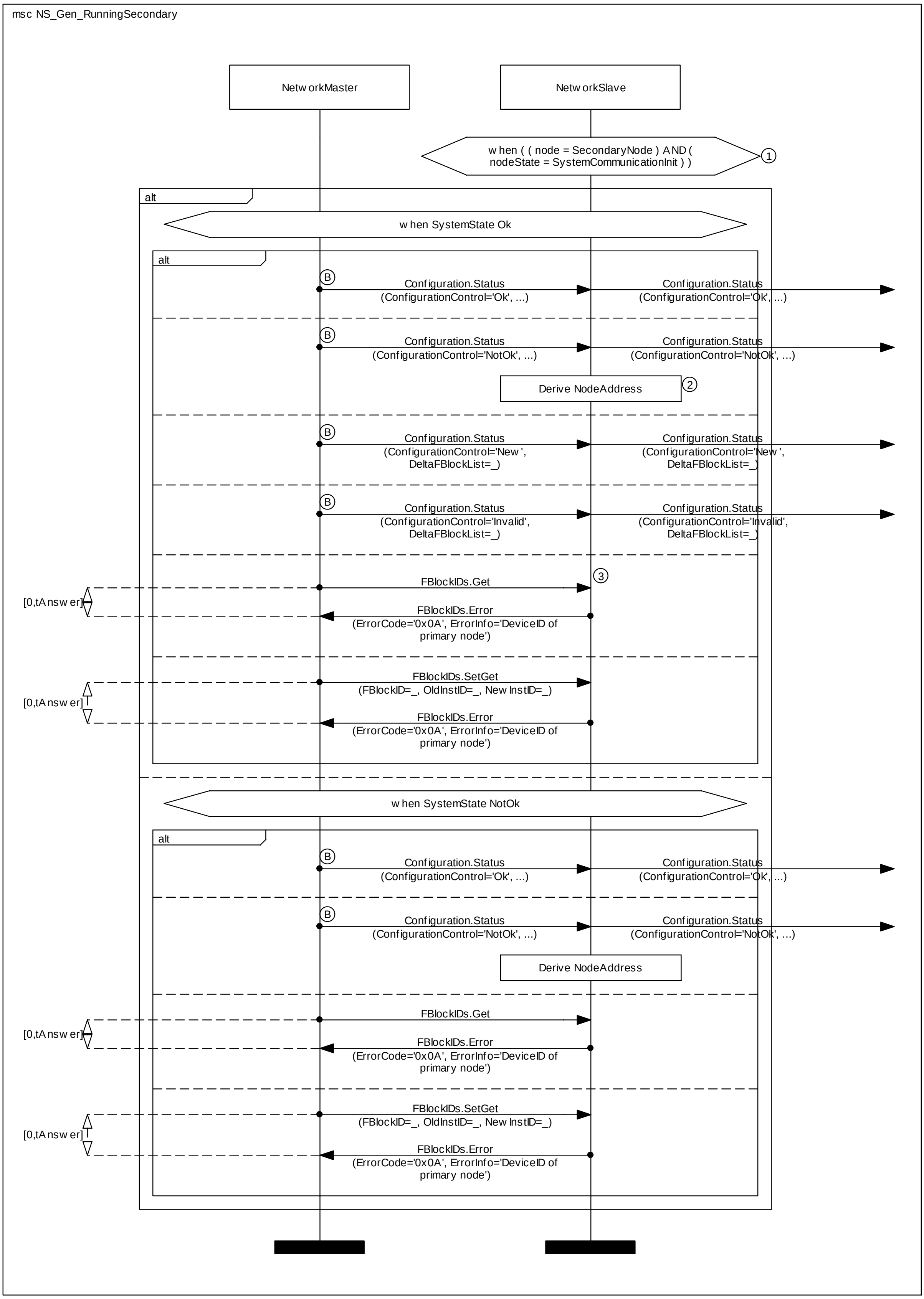

msc NS_Gen_RunningPrimary
Netw orkMaster
Netw orkSlave
w hen ((node = PrimaryNode) AND (nodeState = SystemCommunicationInit))
1
alt
w hen SystemState Ok
alt
B
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
Update decentral registry
B
Configuration.Status (ConfigurationControl='NotOk', ...)
Configuration.Status (ConfigurationControl='NotOk', ...)
NS_Gen_Reset
B
Configuration.Status (ConfigurationControl='New ', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New ', DeltaFBlockList=_)
Update decentral registry
B
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
Update decentral registry
FBlockIDs.Get
[0,tAnsw er]
2
FBlockIDs.Status (FBlockIDList=_)
FBlockIDs.SetGet (FBlockID=_, OldInstID=_, New InstID=_)
[0,tAnsw er]
FBlockIDs.Status (FBlockIDList=_)
w hen SystemState NotOk
alt
B
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
If not already stored the address of the Netw orkMaster should be stored.
(Re-)initialize applications
Update decentral registry
B
Configuration.Status (ConfigurationControl='NotOk', ...)
Configuration.Status (ConfigurationControl='NotOk', ...)
Clear decentral registry
Derive NodeAddress
3
FBlockIDs.Get
[0,tAnsw er]
FBlockIDs.Status (FBlockIDList=_)
FBlockIDs.SetGet (FBlockID=_, OldInstID=_, New InstID=_)
[0,tAnsw er]
FBlockIDs.Status (FBlockIDList=_)

MSC 27: NS_Gen_RunningPrimary

1. The term "PrimaryNode" is used here regardless of an occurrence of a "SecondaryNode" in the system.
2. NetworkSlave must reply within tAnswer after the request has been received.
3. The address should be static, stored, or calculated from position. If no address can be obtained in required time, 0xFFFF should be used.

3.4.4 Running Operation - Secondary Node

General MSC:	NS_Gen_RunningSecondary
Description:	Describes the running (normal) operation of a Secondary Node after the startup sequence has been completed.
Prior Condition:	- SystemCommunicationInit (Startup sequence) has completed.
Initiator:	-
Communication Partners:	- NetworkMaster
Events:	-
Timers/Timing constraints:	- t_{Answer}
Remarks:	- The term “Primary Node” is used regardless of the occurrence of a “Secondary Node” in the system.

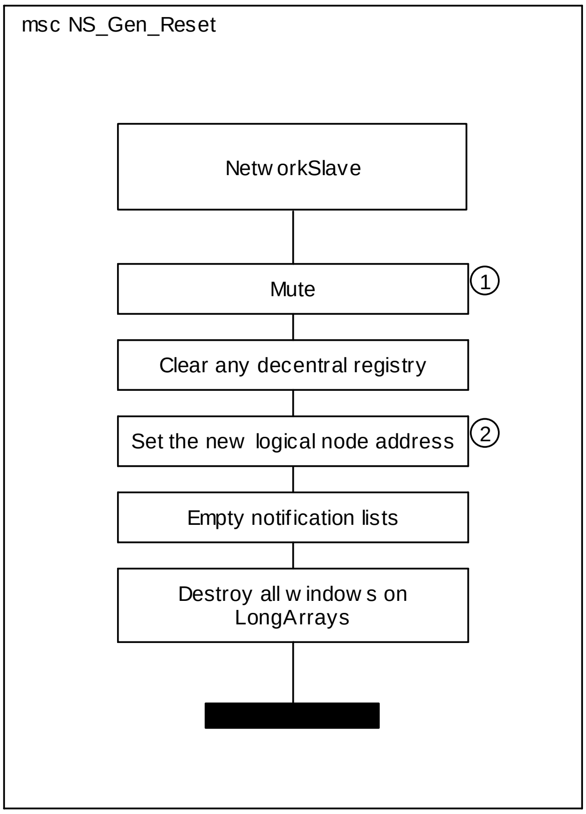

msc NS_Gen_RunningSecondary
NetworkMaster
NetworkSlave
when ((node = SecondaryNode) AND (nodeState = SystemCommunicationInit))
1
alt
when SystemState Ok
alt
B
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
B
Configuration.Status (ConfigurationControl='NotOk', ...)
Configuration.Status (ConfigurationControl='NotOk', ...)
Derive NodeAddress
2
B
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='New', DeltaFBlockList=_)
B
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
FBlockIDs.Get
3
[0,tAnswer]
FBlockIDs.Error (ErrorCode='0x0A', ErrorInfo='DeviceID of primary node')
FBlockIDs.SetGet (FBlockID=_, OldInstID=_, NewInstID=_)
[0,tAnswer]
FBlockIDs.Error (ErrorCode='0x0A', ErrorInfo='DeviceID of primary node')
when SystemState NotOk
alt
B
Configuration.Status (ConfigurationControl='Ok', ...)
Configuration.Status (ConfigurationControl='Ok', ...)
B
Configuration.Status (ConfigurationControl='NotOk', ...)
Configuration.Status (ConfigurationControl='NotOk', ...)
Derive NodeAddress
FBlockIDs.Get
[0,tAnswer]
FBlockIDs.Error (ErrorCode='0x0A', ErrorInfo='DeviceID of primary node')
FBlockIDs.SetGet (FBlockID=_, OldInstID=_, NewInstID=_)
[0,tAnswer]
FBlockIDs.Error (ErrorCode='0x0A', ErrorInfo='DeviceID of primary node')

MSC 28: NS_Gen_RunningSecondary

1. The address should be static, stored, or calculated from position. If no address can be obtained in the required time, 0xFFFF should be used.
2. NetworkSlave must reply within tAnswer after the request has been received.

3.4.5 Reset

General MSC:	NS_Gen_Reset
Description:	Action taken when Configuration.Status(NotOk) is received.
Prior Condition:	-
Initiator:	-
Communication Partners:	-
Events:	-
Timers/Timing constraints:	-
Remarks:	The Network Slave must service requests from the NetworkMaster while waiting for the System State to be set to OK.

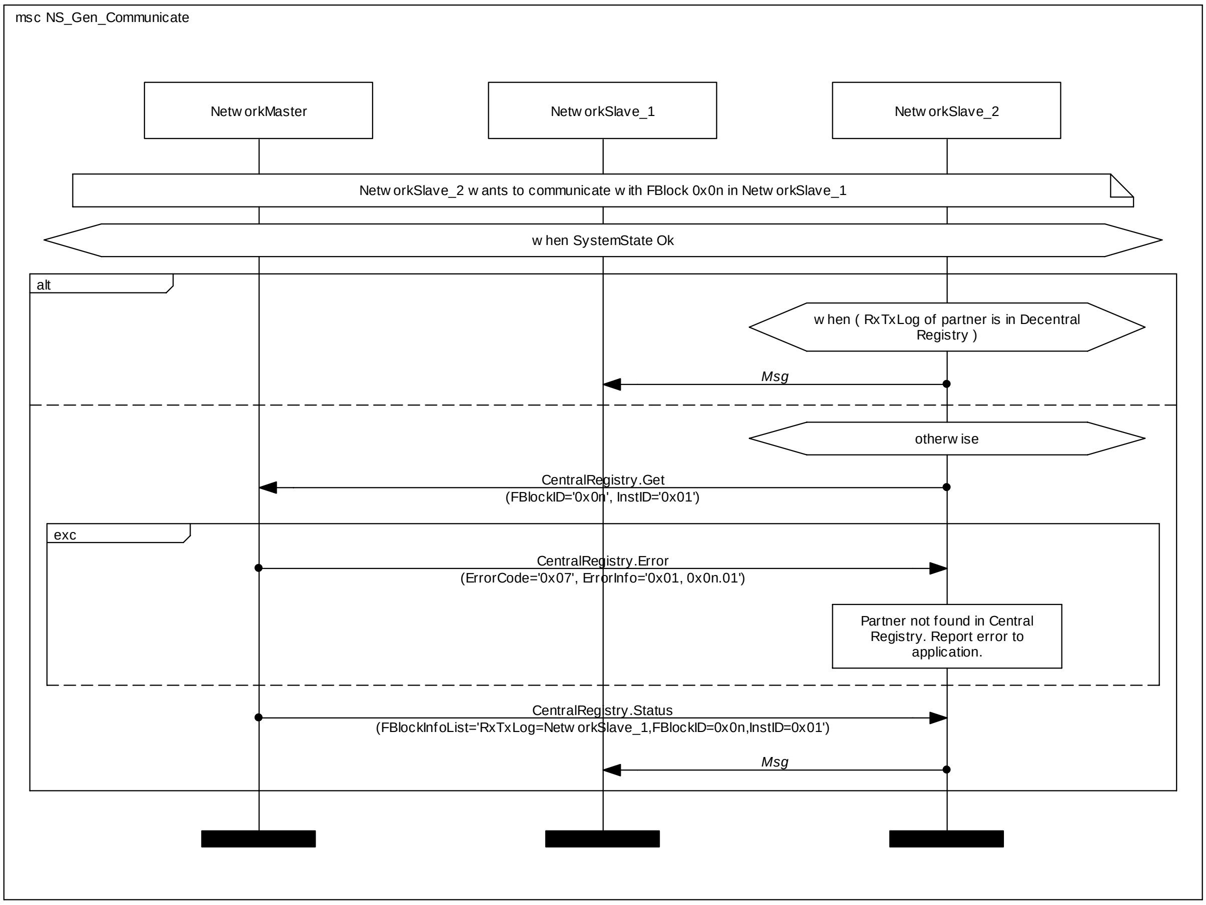

msc NS_Gen_Reset
NetworkSlave
Mute
1
Clear any decentral registry
Set the new logical node address
2
Empty notification lists
Destroy all windows on LongArrays

MSC 29: NS_Gen_Reset

1. Mute and disconnect all synchronous sinks. All synchronous sources must route zeros (signal mute) for a time tCleanChannels, before de-allocate.
2. The address should be static, stored, or dynamic (calculated from position).

3.4.6 Communicate

General MSC:	NS_Gen_Communicate
Description:	Describes how a Network Slave uses its decentral registry or the central registry to find the logical address of its communication partner.
Prior Condition:	- SystemState Ok
Initiator:	-
Communication Partners:	- NetworkMaster - NetworkSlave_1
Events:	-
Timers/Timing constraints:	-
Remarks:	-


msc NS_Gen_Communicate
Netw orkMaster
Netw orkSlave_1
Netw orkSlave_2
Netw orkSlave_2 w ants to communicate w ith FBlock 0x0n in Netw orkSlave_1
w hen SystemState Ok
alt
w hen (RxTxLog of partner is in Decentral Registry)
Msg
otherw ise
CentralRegistry.Get (FBlockID='0x0n', InstID='0x01')
exc
CentralRegistry.Error (ErrorCode='0x07', ErrorInfo='0x01, 0x0n.01')
Partner not found in Central Registry. Report error to application.
CentralRegistry.Status (FBlockInfoList='RxTxLog=Netw orkSlave_1,FBlockID=0x0n,InstID=0x01')
Msg

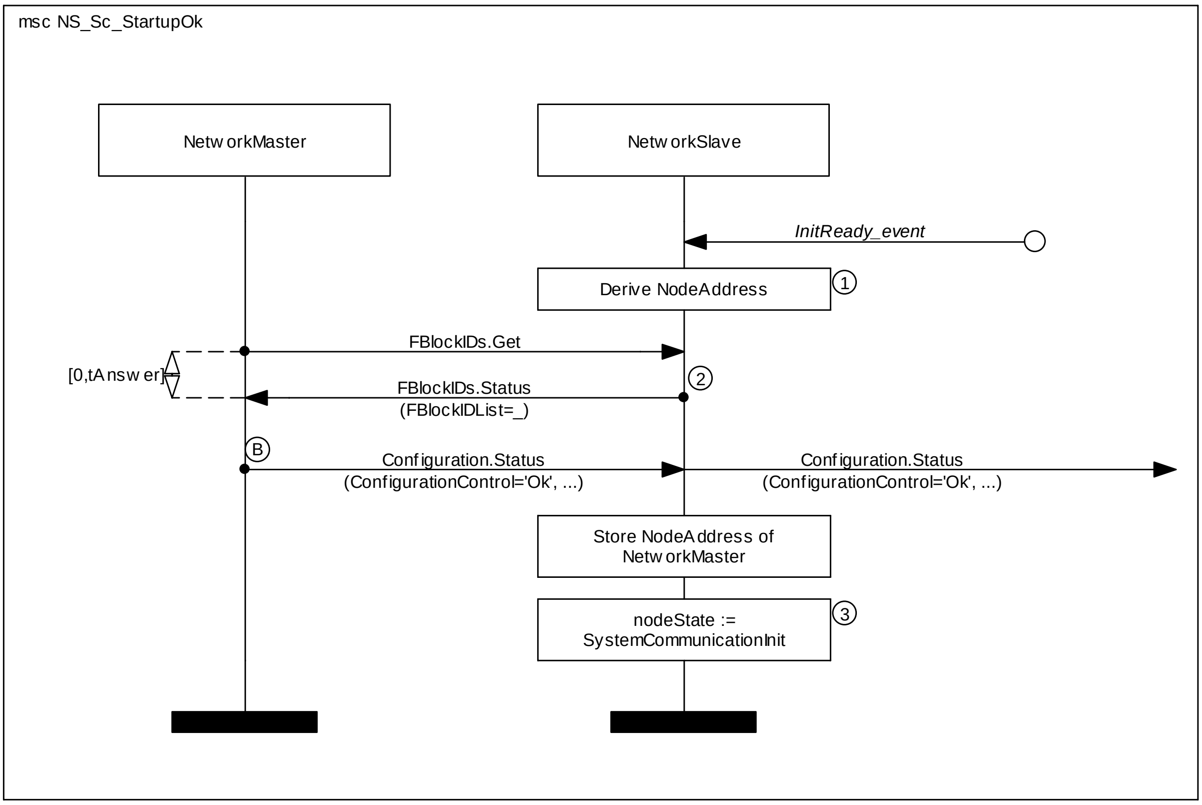
MSC 30: NS_Gen_Communicate

3.5 Network Slave Scenario MSCs

3.5.1 Startup scenarios

3.5.1.1 Startup - Ok

Scenario MSC	NS_Sc_StartupOk
Description:	Describes the startup sequence of a Network Slave when Configuration.Status(Ok) is received during startup.
Prior Condition:	-
Initiator:	-
Communication Partners:	- NetworkMaster
Events:	-Init Ready
Timers/Timing constraints:	- t_{Answer}
Remarks:	-

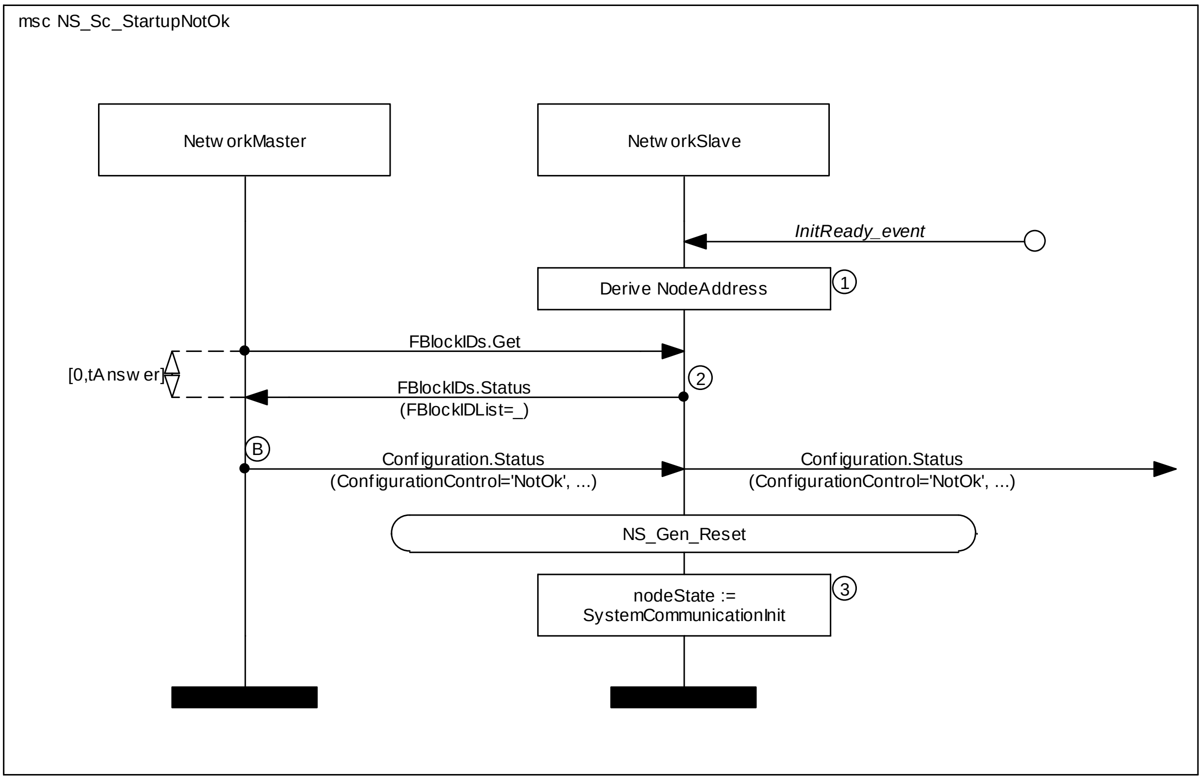


MSC 31: NS_Sc_StartupOk

1. The address should be static or stored. If no address can be obtained in required time, 0xFFFF should be used.
2. NetworkSlave must reply within tAnswer after the request has been received.
3. Initialization of the application with respect to communication has completed.

3.5.1.2 Startup - NotOk

Scenario MSC	NS_Sc_StartupNotOk
Description:	Describes the startup sequence of a Network Slave when Configuration.Status(NotOk) is received during startup.
Prior Condition:	-
Initiator:	-
Communication Partners:	- NetworkMaster
Events:	-Init Ready
Timers/Timing constraints:	- t_{Answer}
Remarks:	-



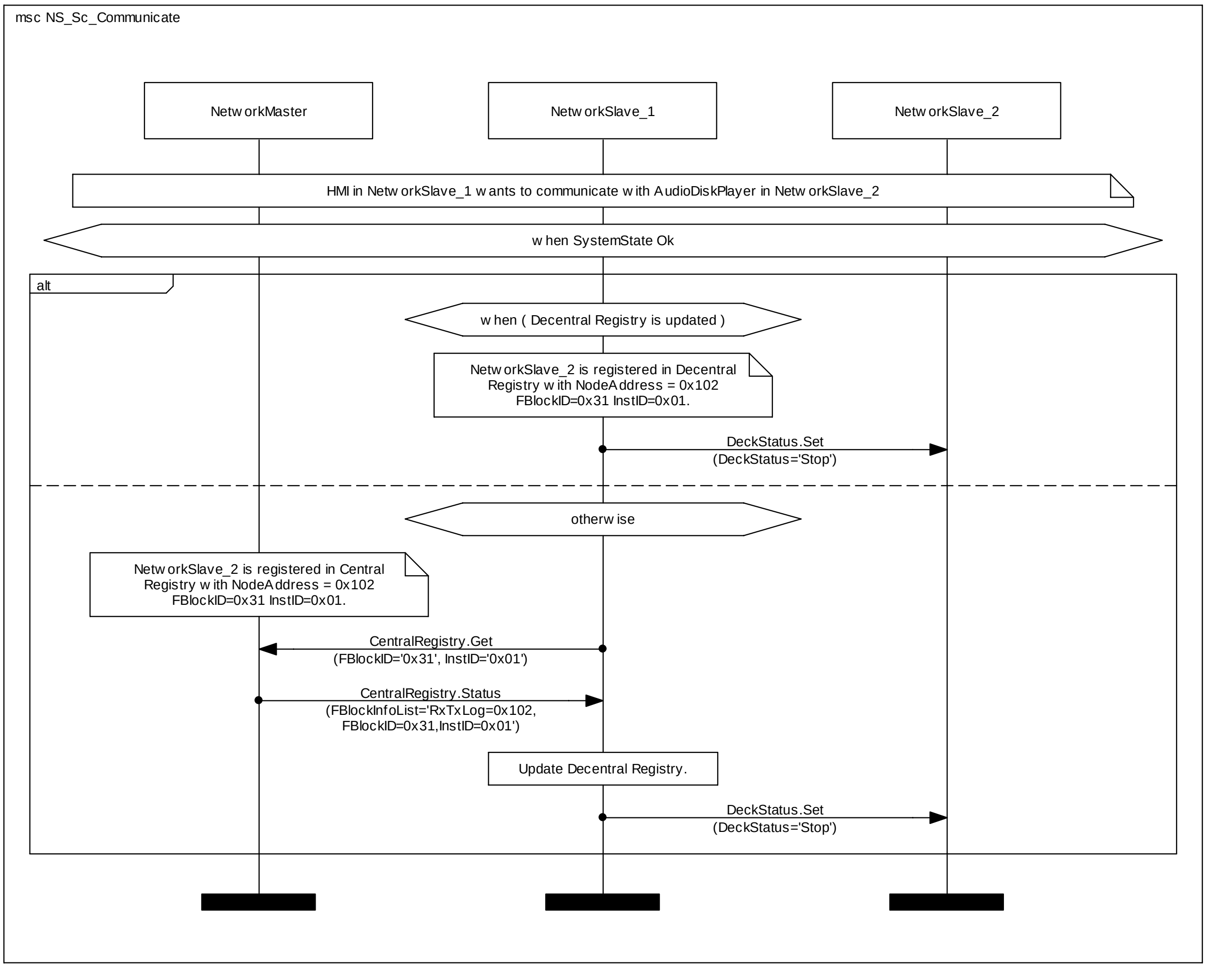
MSC 32: NS_Sc_StartupNotOk

1. The address should be static or stored. If no address can be obtained in the required time, 0xFFFF should be used.
2. NetworkSlave must reply within tAnswer after the request has been received.
3. Initialization of the application with respect to communication has completed.

3.5.2 Communication Scenarios

3.5.2.1 Communicate with partner

Scenario MSC	NS_Sc_Communicate
Description:	Describes how a Network Slave uses its decentral registry or the central registry to find the logical address of its communication partner.
Prior Condition:	- SystemState Ok
Initiator:	-
Communication Partners:	- NetworkMaster - NetworkSlave_2
Events:	-
Timers/Timing constraints:	-
Remarks:	- As an example, HMI and AudioDiskPlayer are used as communication partners.


msc NS_Sc_Communicate
NetworkMaster
NetworkSlave_1
NetworkSlave_2
HMI in NetworkSlave_1 wants to communicate with AudioDiskPlayer in NetworkSlave_2
when SystemState Ok
alt
when (Decentral Registry is updated)
NetworkSlave_2 is registered in Decentral Registry with NodeAddress = 0x102 FBlockID=0x31 InstID=0x01.
DeckStatus.Set (DeckStatus='Stop')
otherwise
NetworkSlave_2 is registered in Central Registry with NodeAddress = 0x102 FBlockID=0x31 InstID=0x01.
CentralRegistry.Get (FBlockID='0x31', InstID='0x01')
CentralRegistry.Status (FBlockInfoList='RxTxLog=0x102, FBlockID=0x31,InstID=0x01')
Update Decentral Registry.
DeckStatus.Set (DeckStatus='Stop')

MSC 33: NS_Sc_Communicate

4 Connection Management

4.1 Introduction

This description of Connection Management takes into account both the older Allocate method as well as the newer Source Connect method. In addition, mixed systems, where some slaves support Allocate and some support Source Connect, are shown.

4.2 Logical Model of Connection Management

In this logical model of Connection Management (please refer to Figure 4-1) the priority and clustered connection handler is added, though not being a part of the specification. The concept is not new; most car manufacturers use the concept of priority handler and clustered connection handler in their proprietary part of the specification. By extending the existing Connection Management with this functionality, the description of the dynamic behavior of the system is simplified.

The extended functionality of the Connection Management is not a description of a specific implementation, but rather a description of its general behavior.

In this specification, the working names used for these two types of Connection Management will be Connection Management (as defined in the MOST Specification of today) and Extended Connection Management.

The dynamics of the Extended Connection Management are not specified here, but the logical name is introduced in order to avoid confusion with different interpretations of the Connection Management. Note that the Extended Connection Management could be implemented in the same node as the Connection Management, or implemented in other nodes (e.g., HMI).

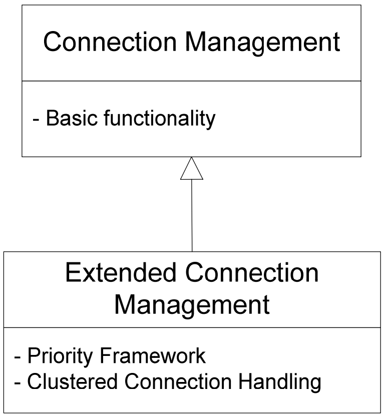


Figure 4-1: Logical Model of Connection Management

4.3 Variables used in Connection Management MSCs

The general MSCs use variables to simplify the MSCs, as well as reducing the total number of MSCs. Table 4-1 shows a list of the variables used in the general ConnectionMaster MSCs.

Variable	Range	Explanation
Error	False, True	Indicates if something fails during connection management.
Progress	None, SourceInfo_Received, Source_Connected, Sink_Connected, SourceActivity_On, Functions_Received	This variable is used to keep track of how far the procedure has progressed.
SourceType	Unknown, SourceConnect, Allocate	Holds information of which method that shall be used.

Table 4-1: Variables used in the general Connection Management MSCs

4.4 Normal Behavior

In this section, MSCs are used to describe Connection Management in normal behavior. The OPTypes of the functions are given in non-Ack format. If desired, the Ack format of the function OPType can also be used.

When building a connection, the following issues are important for the Connection Management:

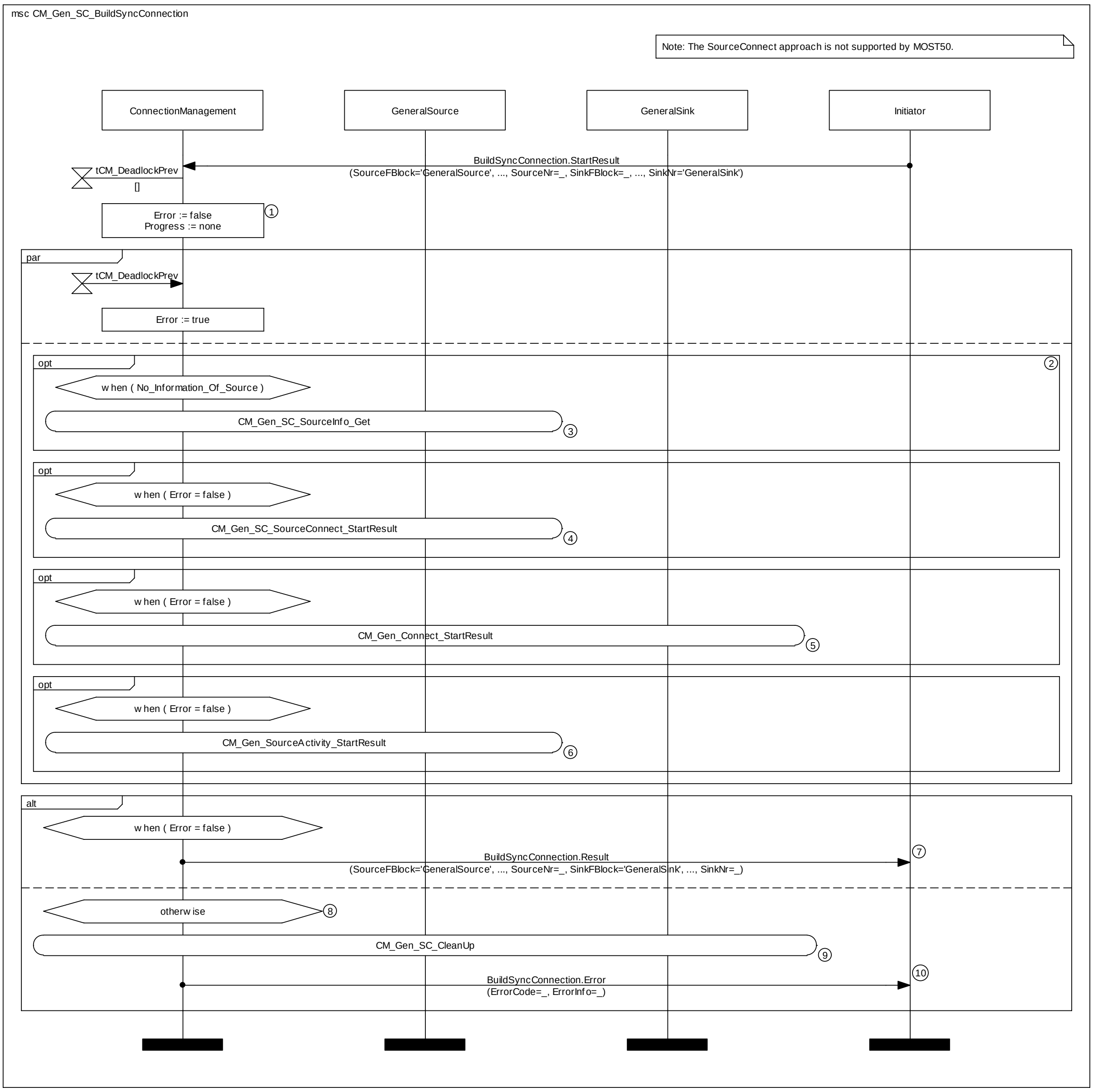
- Is the system mixed or homogenous? That is, are all sources of the same type (Allocate/SourceConnect)?
- Is the Connection Management familiar with the nodes?

In a familiar system, the Connection Management does not have to ask the sources which type of nodes they are. It does not have to ask how many channels different sources need. The Connection Management may be familiar with the nodes either by previously having asked them or by having the information already provided by the system developers.

4.4.1 Connection Management General MSCs

4.4.1.1 BuildSyncConnection in a SourceConnect System

General MSC:	CM_Gen_SC_BuildSyncConnection
Description:	A connection is built between GeneralSource and GeneralSink. The flow is as follows: If information regarding the source is needed, this is collected first. Then the source is connected to the network and the sink is also connected. When the connection is established, SourceActivity may be turned on. A timer is started when the Connection Manager starts this process. If it takes more than the $t_{CM\_DeadlockPrev}$ to do this, the process is aborted. The Connection Management will have to tidy up by removing everything that has been done so far, which is done in the CleanUp MSC.
Prior Condition:	
Initiator:	Any controller
Communication Partners:	Initiator, a source, and a sink.
Events:	-
Timers / Timing Constraints:	$t_{CM\_DeadlockPrev}$
Remarks:	- All sources in the system support the SourceConnect method. - The allocation table of the Timing Master is not used. - Note that the source activity is optional.

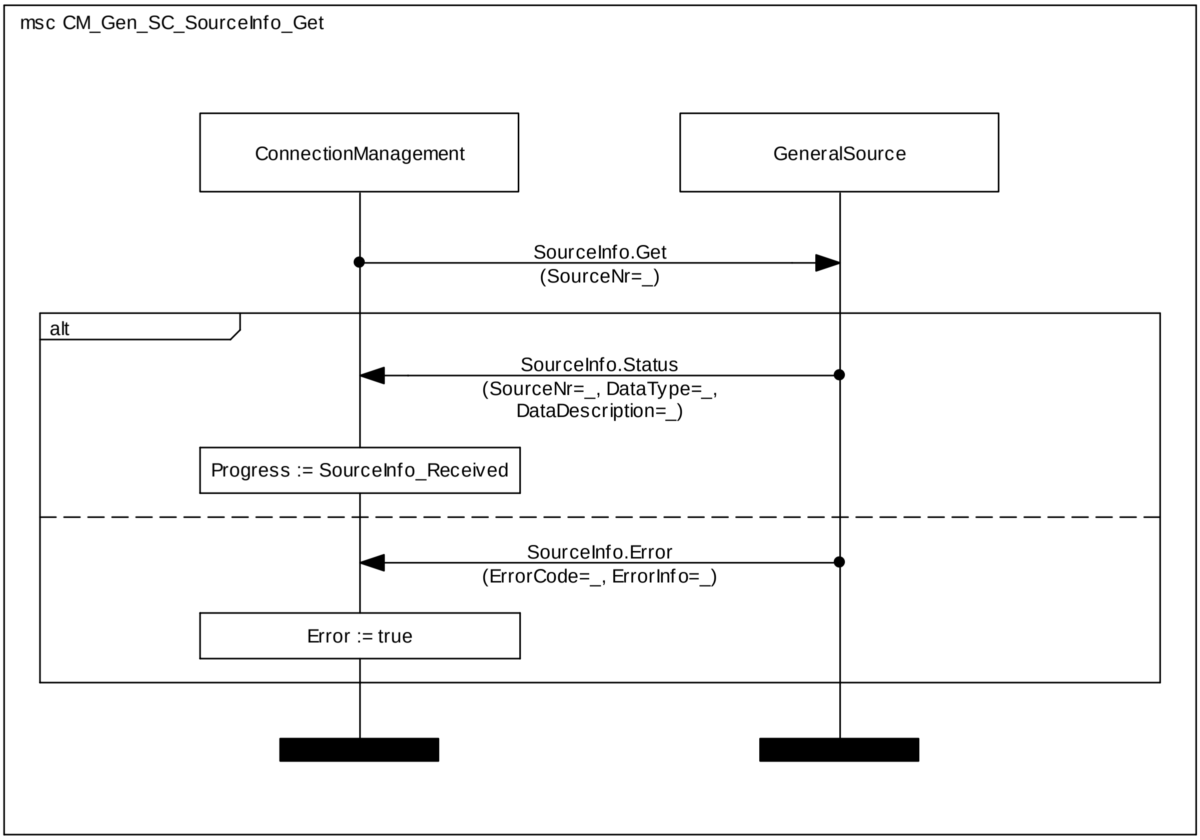

msc CM_Gen_SC_BuildSyncConnection
Note: The SourceConnect approach is not supported by MOST50.
ConnectionManagement
GeneralSource
GeneralSink
Initiator
BuildSyncConnection.StartResult
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock=_, ..., SinkNr='GeneralSink')
tCM_DeadlockPrev
[]
Error := false
Progress := none
1
par
tCM_DeadlockPrev
Error := true
opt
2
when (No_Information_Of_Source)
CM_Gen_SC_SourceInfo_Get
3
opt
when (Error = false)
CM_Gen_SC_SourceConnect_StartResult
4
opt
when (Error = false)
CM_Gen_Connect_StartResult
5
opt
when (Error = false)
CM_Gen_SourceActivity_StartResult
6
alt
when (Error = false)
BuildSyncConnection.Result
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock='GeneralSink', ..., SinkNr=_)
7
otherwise
8
CM_Gen_SC_CleanUp
9
BuildSyncConnection.Error
(ErrorCode=_, ErrorInfo=_)
10

MSC 34: CM_Gen_SC_BuildSyncConnection

1. Initialization of variables. Error is set, if an error occurs, to see where it occurs. Progress is used to keep track of how far the procedure has progressed.
2. If no prior information regarding the channels needed by the source is available.
3. Collect source information.
4. Connect the source to the network.
5. Connect the sink to the network.
6. Turn source activity on.
7. The procedure was carried out successfully.
8. An error was received during the procedure.
9. Clean up all that was made before.
10. Error message contains accumulated error information.

4.4.1.1.1 Retrieving SourceInfo

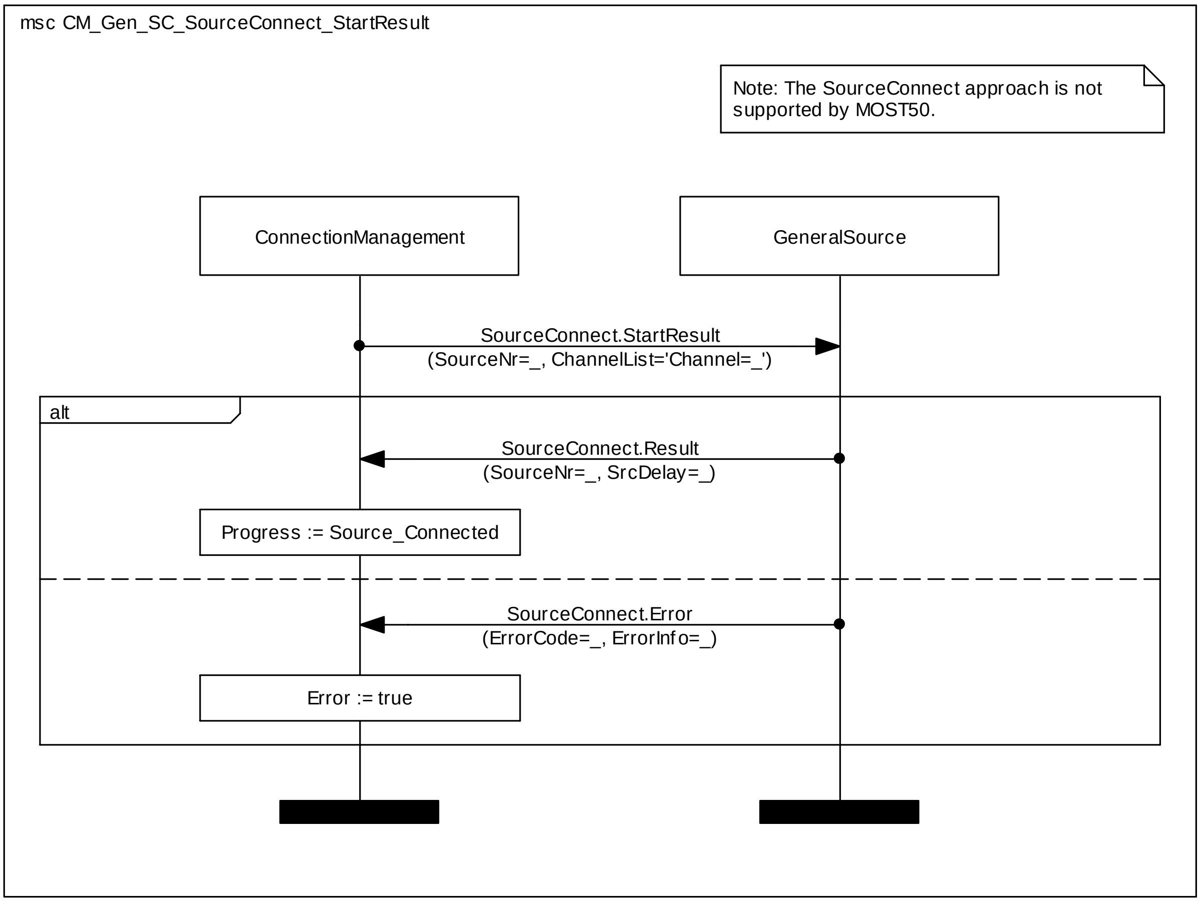
General MSC:	CM_Gen_SC_SourceInfo_Get
Description:	The Connection Management retrieves information regarding the number of channels that is needed by the source. The result of this MSC is saved for the higher level MSC that uses this one.
Prior Condition:	The Connection Management does not know the requirements of this particular source.
Initiator:	Connection Management
Communication Partners:	A source
Events:	-
Timers / Timing Constraints:	-
Remarks:	-



MSC 35: CM_Gen_SC_SourceInfo_Get

4.4.1.1.2 Connecting a Source with SourceConnect

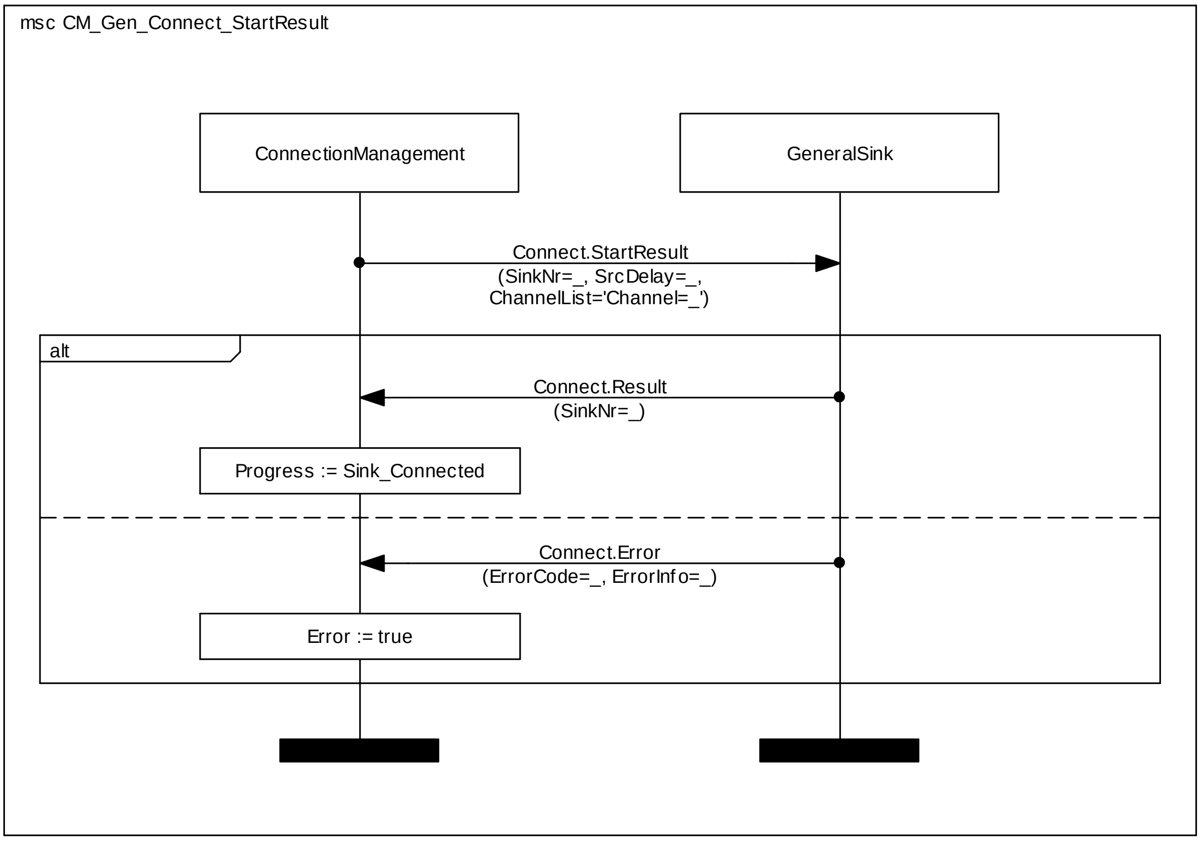
General MSC:	CM_Gen_SC_SourceConnect_StartResult
Description:	The Connection Management commands the source to connect to the specified channels. The result of this MSC is saved for the higher level MSC that uses this one.
Prior Condition:	Connection Management has reserved channels for the source to use.
Initiator:	Connection Management
Communication Partners:	Initiator and a source
Events	-
Timers / Timing Constraints:	-
Remarks:	-


msc CM_Gen_SC_SourceConnect_StartResult
Note: The SourceConnect approach is not supported by MOST50.
ConnectionManagement
GeneralSource
SourceConnect.StartResult
(SourceNr=_, ChannelList='Channel=_')
alt
SourceConnect.Result
(SourceNr=_, SrcDelay=_)
Progress := Source_Connected
SourceConnect.Error
(ErrorCode=_, ErrorInfo=_)
Error := true

MSC 36: CM_Gen_SC_SourceConnect_StartResult

4.4.1.1.3 Connecting a Sink

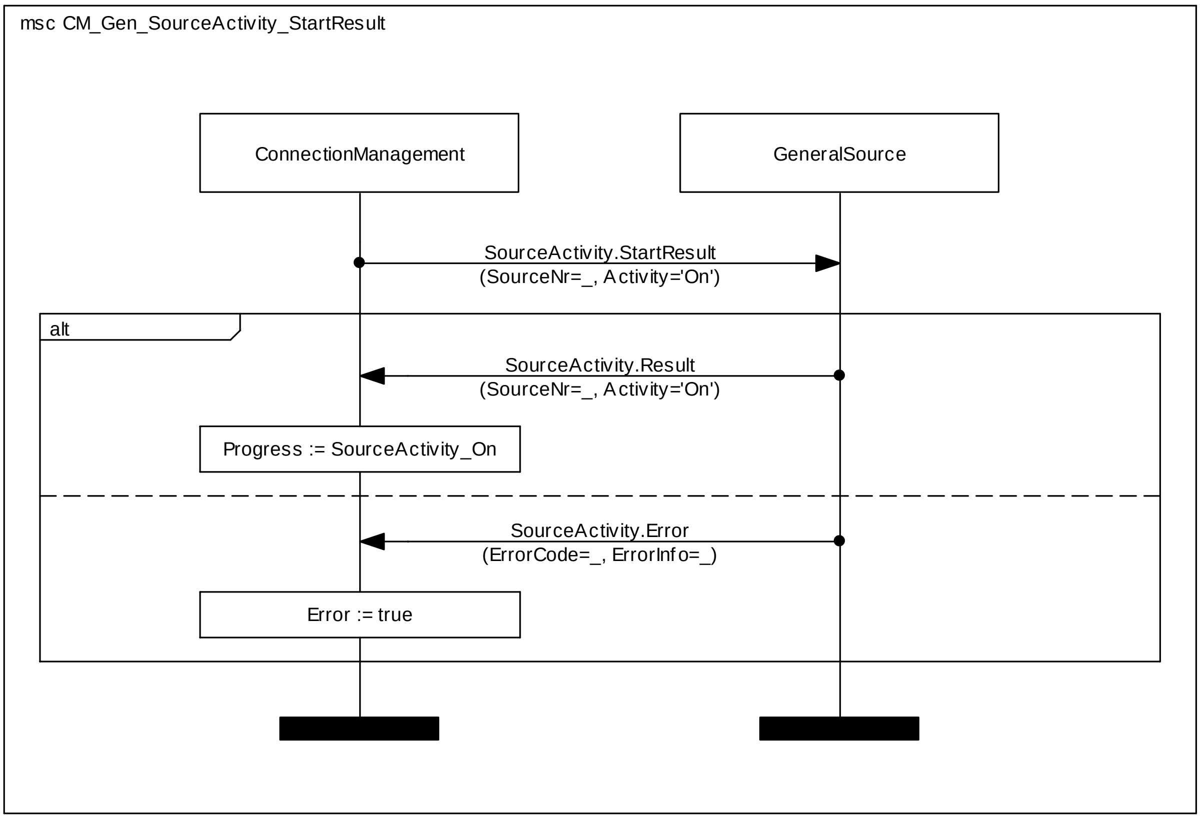
General MSC:	CM_Gen_Connect_StartResult
Description:	The Connection Management commands the sink to connect to the specified channels. The result of this MSC is saved for the higher level MSC that uses this one.
Prior Condition:	The channels that the sink is connecting to are in use by a source.
Initiator:	Connection Management
Communication Partners:	A sink
Events	-
Timers / Timing Constraints:	-
Remarks:	-



MSC 37: CM_Gen_Connect_StartResult

4.4.1.1.4 SourceActivity turned on

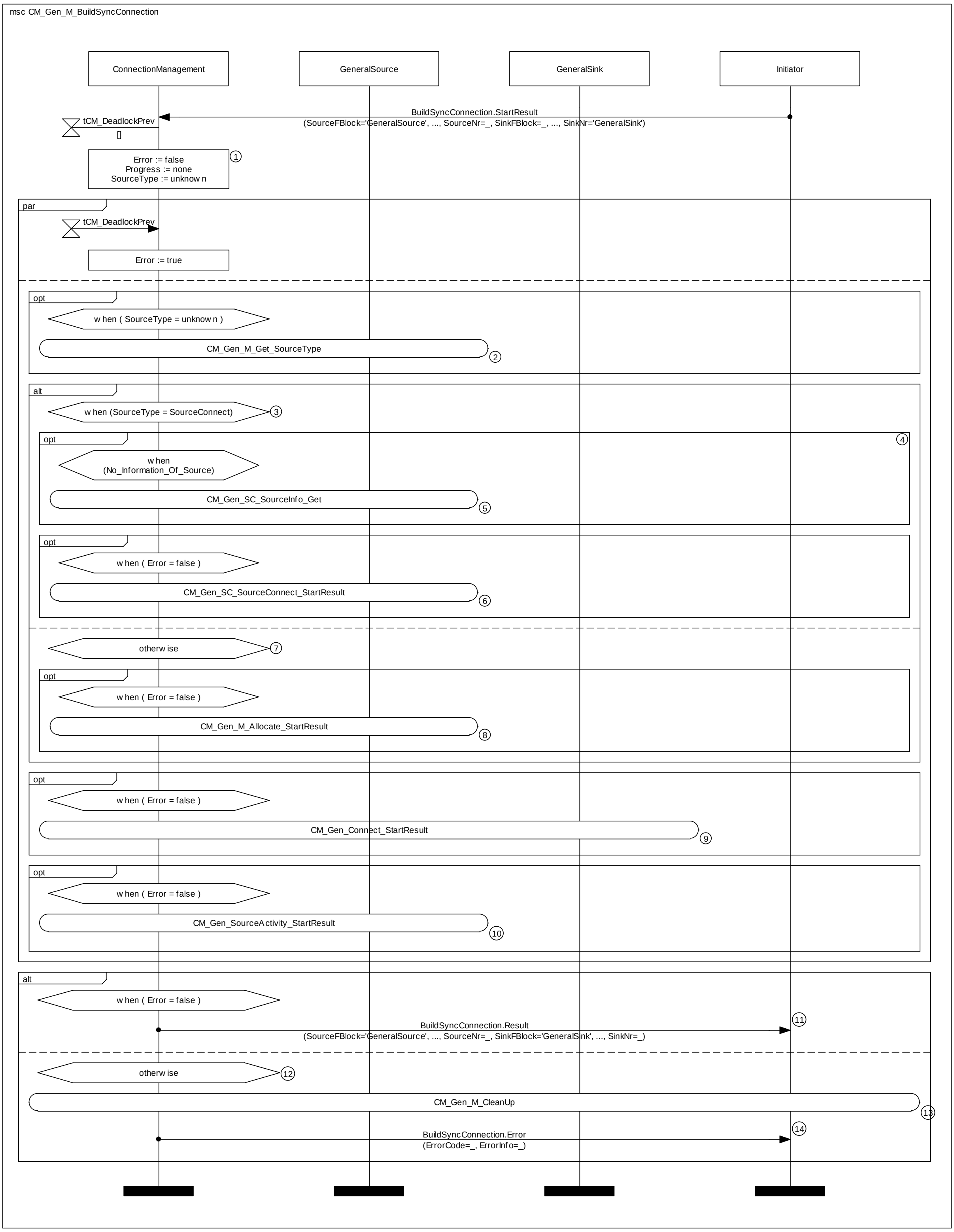
General MSC:	CM_Gen_SourceActivity_StartResult
Description:	The Connection Management commands the source to turn the SourceActivity on. The result of this MSC is saved for the higher level MSC that uses this one.
Prior Condition:	A connection to a sink has been established.
Initiator:	Connection Management
Communication Partners:	A source
Events	-
Timers / Timing Constraints:	-
Remarks:	- Note that the source activity is optional.


msc CM_Gen_SourceActivity_StartResult
ConnectionManagement
GeneralSource
SourceActivity.StartResult
(SourceNr=_, Activity='On')
alt
SourceActivity.Result
(SourceNr=_, Activity='On')
Progress := SourceActivity_On
SourceActivity.Error
(ErrorCode=_, ErrorInfo=_)
Error := true

MSC 38: CM_Gen_SourceActivity_StartResult

4.4.1.2 BuildSyncConnection in a Mixed System

General MSC:	CM_Gen_M_BuildSyncConnection
Description:	A connection is built between GeneralSource and GeneralSink. The flow is as follows: If unknown, the Connection Management first determines if it is dealing with a node using only Allocate or if it supports SourceConnect. If in a SourceConnect connection information regarding the source is needed, this is collected first. Then the source is connected to the network and the sink is also connected. When the connection is established, SourceActivity may be turned on. A timer is started when the Connection Manager starts this process. If it takes more than the $t_{CM\_DeadlockPrev}$ to do this, the process is aborted. The Connection Management will have to tidy up after itself by removing everything that has been done so far, which is done in the CleanUp MSC.
Prior Condition:	Some sources in the system support only Allocate.
Initiator:	Any controller
Communication Partners:	Initiator, a source, and a sink
Events	-
Timers / Timing Constraints:	$t_{CM\_DeadlockPrev}$
Remarks:	- While running CleanUp, there is no reaction to an incoming Abort. The same applies after SourceActivity has been turned on. - Note that the source activity is optional.


msc CM_Gen_M_BuildSyncConnection
ConnectionManagement
GeneralSource
GeneralSink
Initiator
BuildSyncConnection.StartResult
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock=_, ..., SinkNr='GeneralSink')
tCM_DeadlockPrev
[]
Error := false
Progress := none
SourceType := unknow n
1
par
tCM_DeadlockPrev
Error := true
opt
w hen (SourceType = unknow n)
CM_Gen_M_Get_SourceType
2
alt
w hen (SourceType = SourceConnect)
3
opt
4
w hen
(No_Information_Of_Source)
CM_Gen_SC_SourceInfo_Get
5
opt
w hen (Error = false)
CM_Gen_SC_SourceConnect_StartResult
6
otherw ise
7
opt
w hen (Error = false)
CM_Gen_M_Allocate_StartResult
8
opt
w hen (Error = false)
CM_Gen_Connect_StartResult
9
opt
w hen (Error = false)
CM_Gen_SourceActivity_StartResult
10
alt
w hen (Error = false)
BuildSyncConnection.Result
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock='GeneralSink', ..., SinkNr=_)
11
otherw ise
12
CM_Gen_M_CleanUp
13
BuildSyncConnection.Error
(ErrorCode=_, ErrorInfo=_)
14

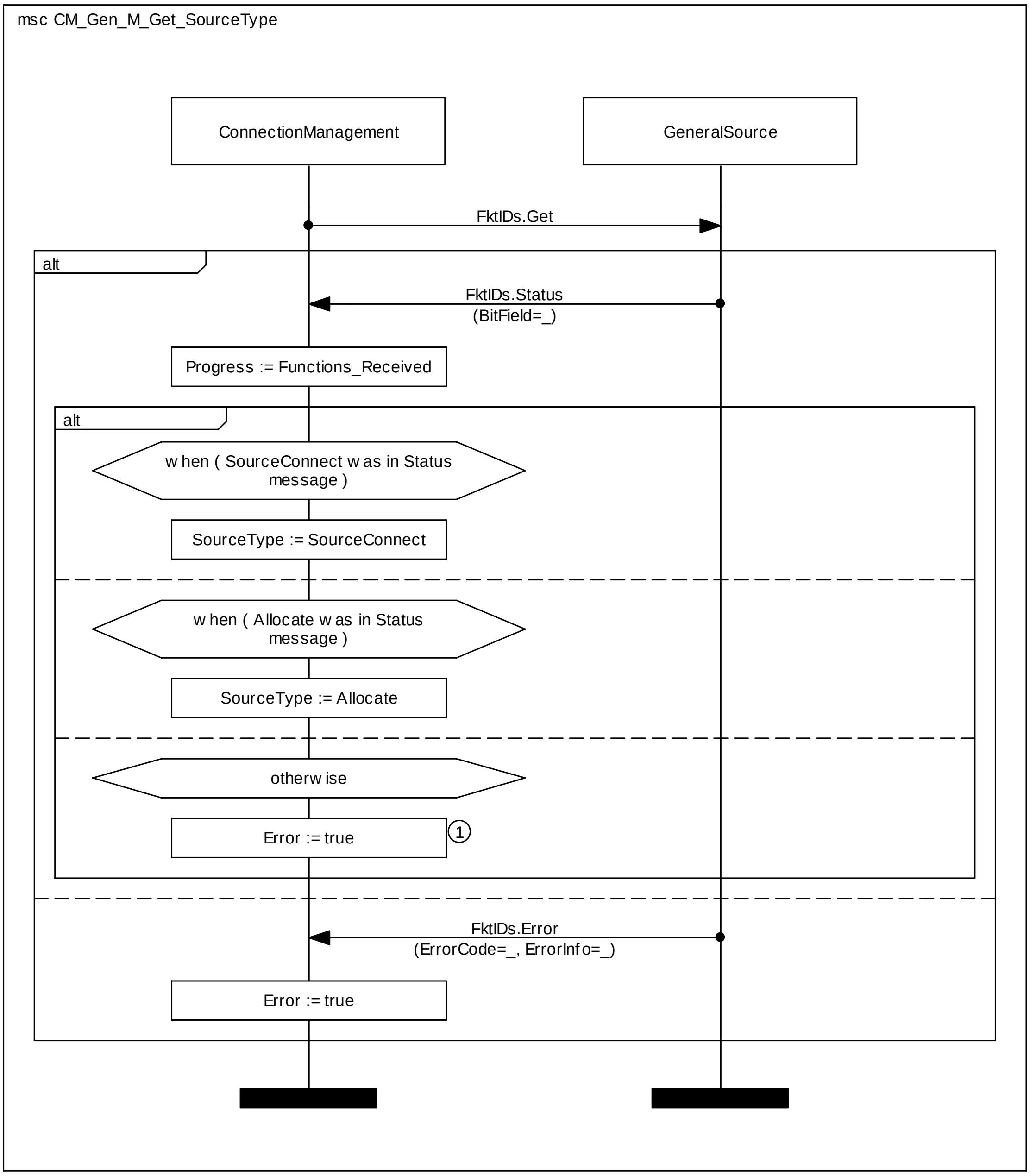
MSC 39: CM_Gen_M_BuildSyncConnection

1. Error is set if an error occurs.
   SourceType tells if it is a SourceConnect or Allocate node. It might be set in advance.
2. Determine the source type, Allocate or SourceConnect.
3. If no prior information regarding the channels needed by the source is available.
4. Collect source information.

5. Connect the source to the network.
6. Tell the source to allocate channels and connect to the network.
7. Connect the sink to the network.
8. Turn source activity on.
9. The procedure was carried out successfully.
10. An error was received during the procedure.
11. Clean up all that was made before.
12. Error message contains accumulated error information.

4.4.1.2.1 Establishing the Source Type

General MSC:	CM_Gen_M_Get_SourceType
Description:	Connection Management asks the source for its function IDs. From this it is deduced whether the source uses SourceConnect or Allocate as its method for connecting to the network. The result of this MSC is saved for the higher level MSC that uses this one.
Prior Condition:	Connection Management does not know if this particular node supports SourceConnect or Allocate.
Initiator:	Connection Management
Communication Partners:	A source
Events	
Timers / Timing Constraints:	
Remarks:	-

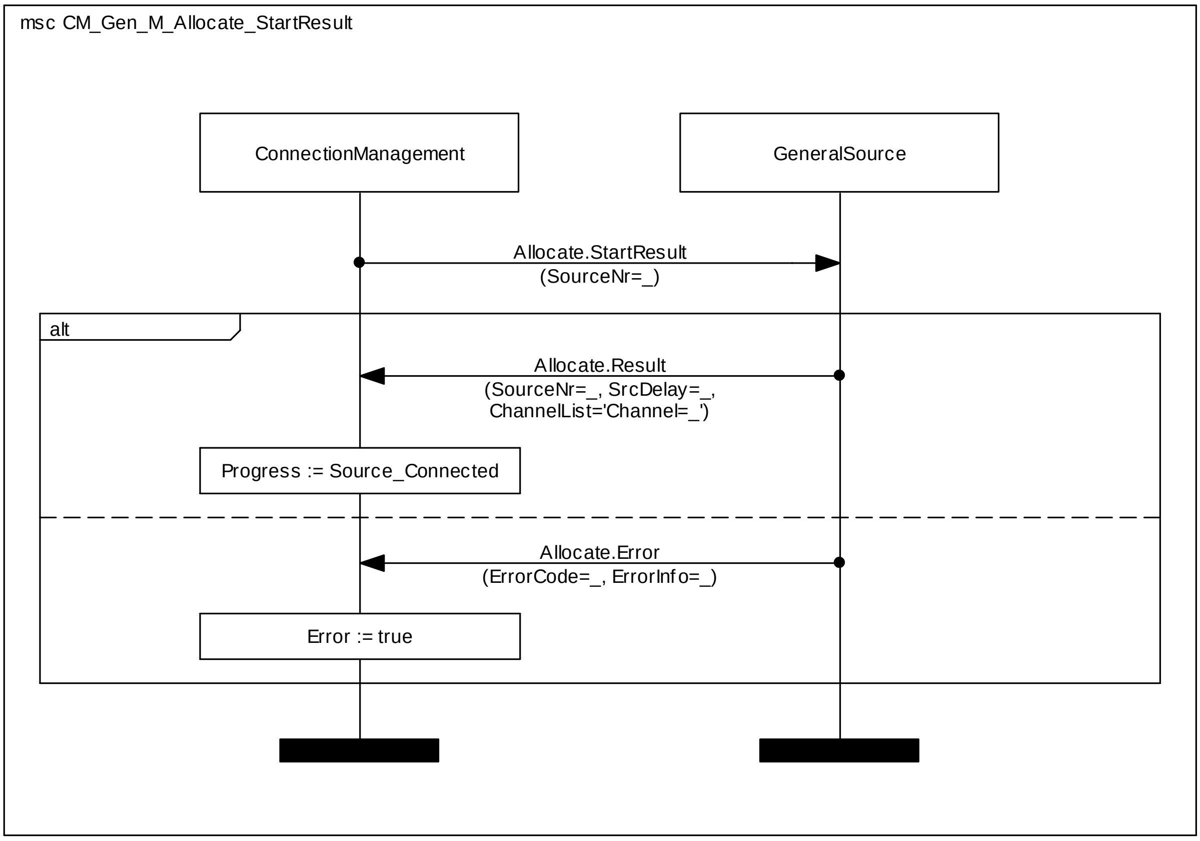

msc CM_Gen_M_Get_SourceType
ConnectionManagement
GeneralSource
FktIDs.Get
alt
FktIDs.Status
(BitField=_)
Progress := Functions_Received
alt
w hen (SourceConnect w as in Status message)
SourceType := SourceConnect
w hen (Allocate w as in Status message)
SourceType := Allocate
otherw ise
Error := true
1
FktIDs.Error
(ErrorCode=_, ErrorInfo=_)
Error := true

MSC 40: CM_Gen_M_Get_SourceType

1. The result is seen as an error if there are no functions for connecting the source.

4.4.1.2.2 Allocate in a Mixed System

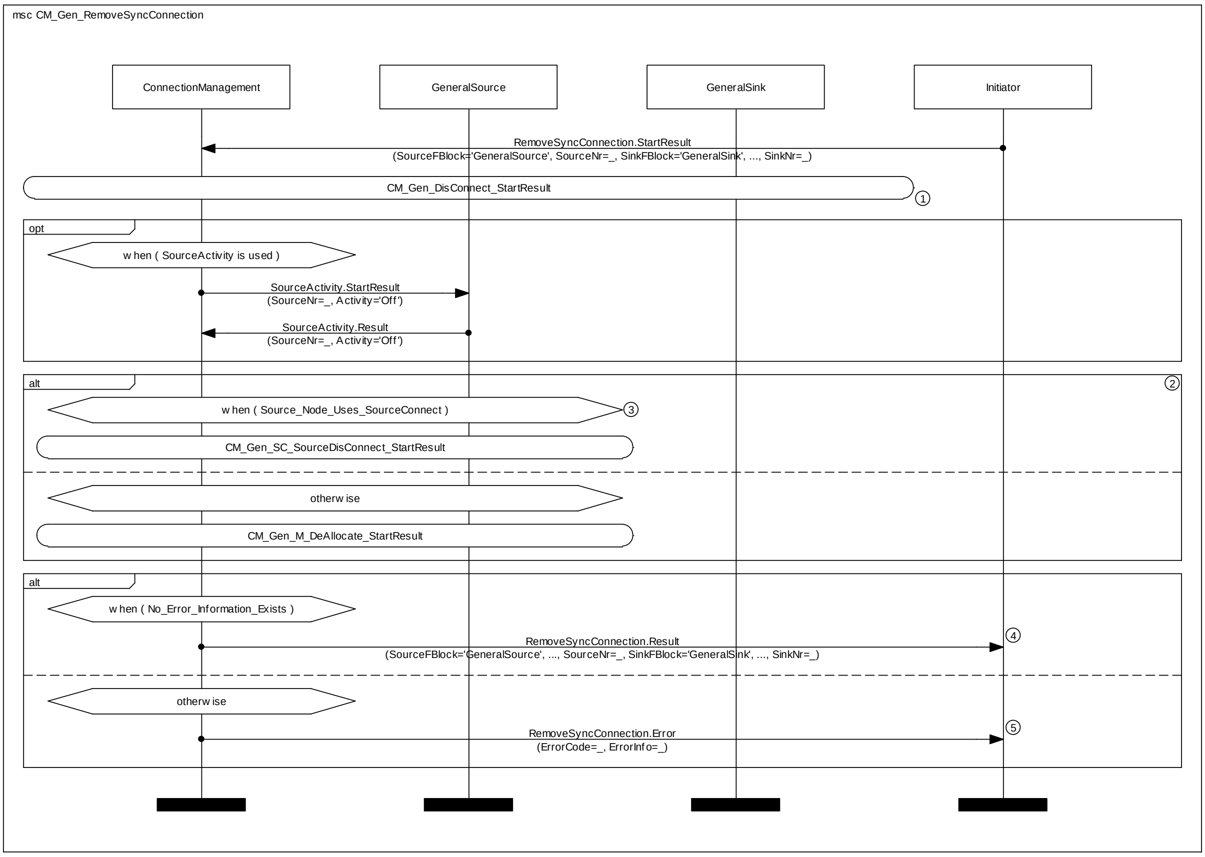
General MSC:	CM_Gen_M_Allocate_StartResult
Description:	The Connection Management tells the source to allocate channels and to connect to them. The result of this MSC is saved for the higher level MSC that uses this one.
Prior Condition:	The source uses Allocate as its method for connection to the network.
Initiator:	Connection Management
Communication Partners:	A source
Events:	
Timer / Timing Constraints:	
Remarks:	-



MSC 41: CM_Gen_M_Allocate_StartResult

4.4.1.3 Removing a Synchronous Connection

General MSC:	CM_Gen_RemoveSyncConnection
Description:	The Connection Management removes a connection. First, the sink is disconnected (and muted). Then depending on whether the source uses SourceConnect or Allocate it is disconnected with SourceDisConnect or DeAllocate. When sending this command, SourceActivity is turned off (if used).
Prior Condition:	A connection between the source and sink exists.
Initiator:	Any controller
Communication Partners:	Initiator, a source, and a sink
Events	-
Timer / Timing Constraints:	-
Remarks:	- There is no way of aborting this procedure. - Note that the source activity is optional.

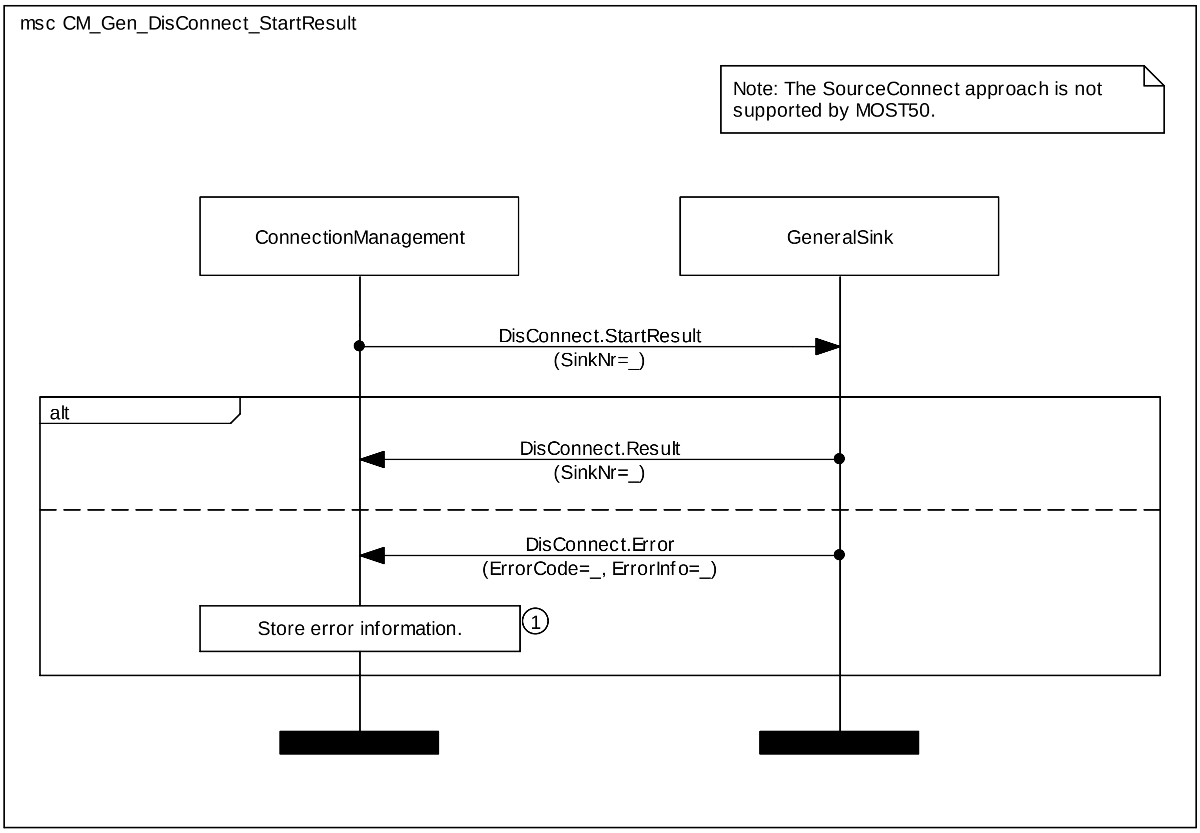

msc CM_Gen_RemoveSyncConnection
ConnectionManagement
GeneralSource
GeneralSink
Initiator
RemoveSyncConnection.StartResult
(SourceFBlock='GeneralSource', SourceNr=_, SinkFBlock='GeneralSink', ..., SinkNr=_)
CM_Gen_DisConnect_StartResult
1
opt
w hen (SourceActivity is used)
SourceActivity.StartResult
(SourceNr=_, Activity='Off')
SourceActivity.Result
(SourceNr=_, Activity='Off')
alt
2
w hen (Source_Node_Uses_SourceConnect)
3
CM_Gen_SC_SourceDisConnect_StartResult
otherw ise
CM_Gen_M_DeAllocate_StartResult
alt
w hen (No_Error_Information_Exists)
RemoveSyncConnection.Result
4
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock='GeneralSink', ..., SinkNr=_)
otherw ise
RemoveSyncConnection.Error
5
(ErrorCode=_, ErrorInfo=_)

MSC 42: CM_Gen_RemoveSyncConnection

1. Note: The SourceConnect approach is not supported by MOST50.
2. The connection manager remembers if it used SourceConnect or Allocate to create the connection and can remove it accordingly.
3. Note: The SourceConnect approach is not supported by MOST50.
4. The procedure was carried out successfully.
5. Parameter contains accumulated error information.

4.4.1.3.1 Disconnecting a Sink

General MSC:	CM_Gen_DisConnect_StartResult
Description:	Connection Management tells the sink to disconnect the specified sink. Error information is saved for the MSC that uses this one.
Prior Condition:	The sink is connected to the network.
Initiator:	Connection Management
Communication Partners:	A sink
Events	-
Timer / Timing Constraints:	-
Remarks:	-

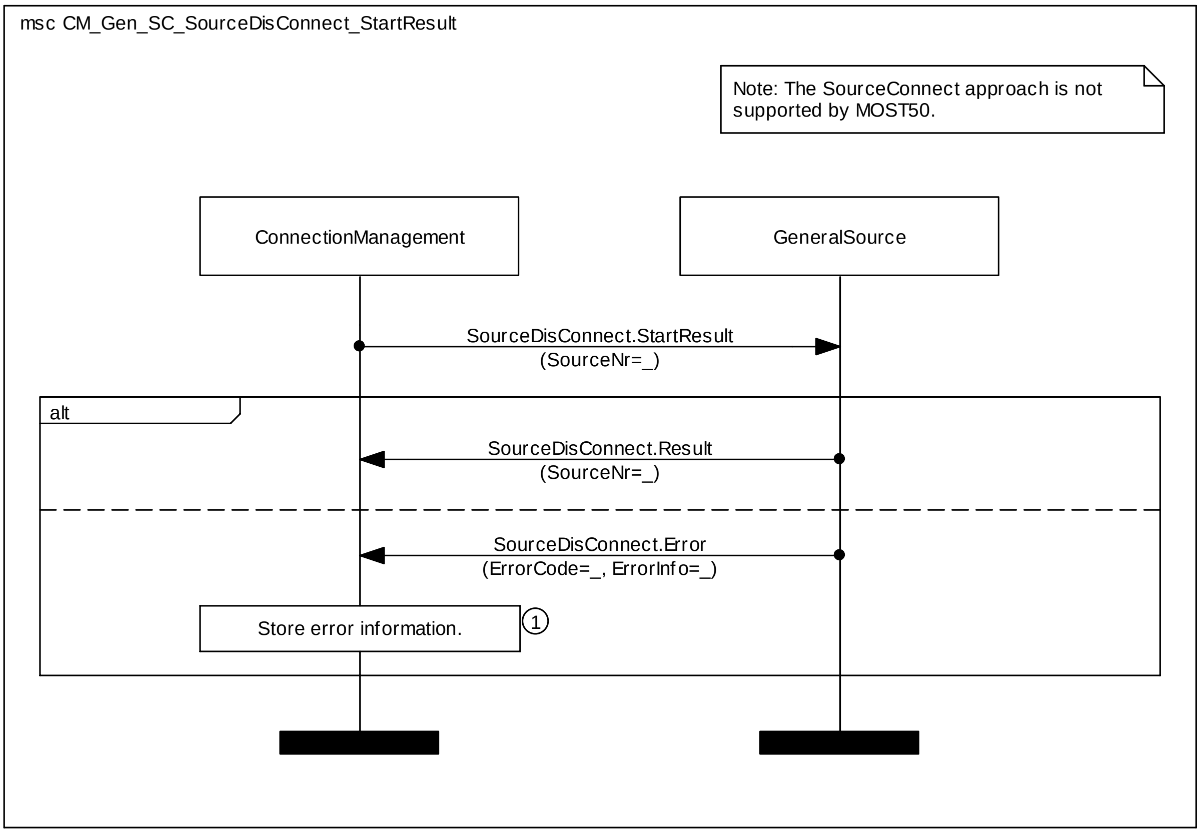


MSC 43: CM_Gen_DisConnect_StartResult

1. Will be reported in BuildSyncConnection.Error or RemoveSyncConnection.Error

4.4.1.3.2 Disconnecting a Source using SourceDisConnect

General MSC:	CM_Gen_SC_SourceDisConnect_StartResult
Description:	Connection Management tells the source to disconnect the specified source. Error information is saved for the MSC that uses this one.
Prior Condition:	The source is connected to the network.
Initiator:	Connection Management
Communication Partners:	A source
Events	-
Timer / Timing Constraints:	-
Remarks:	

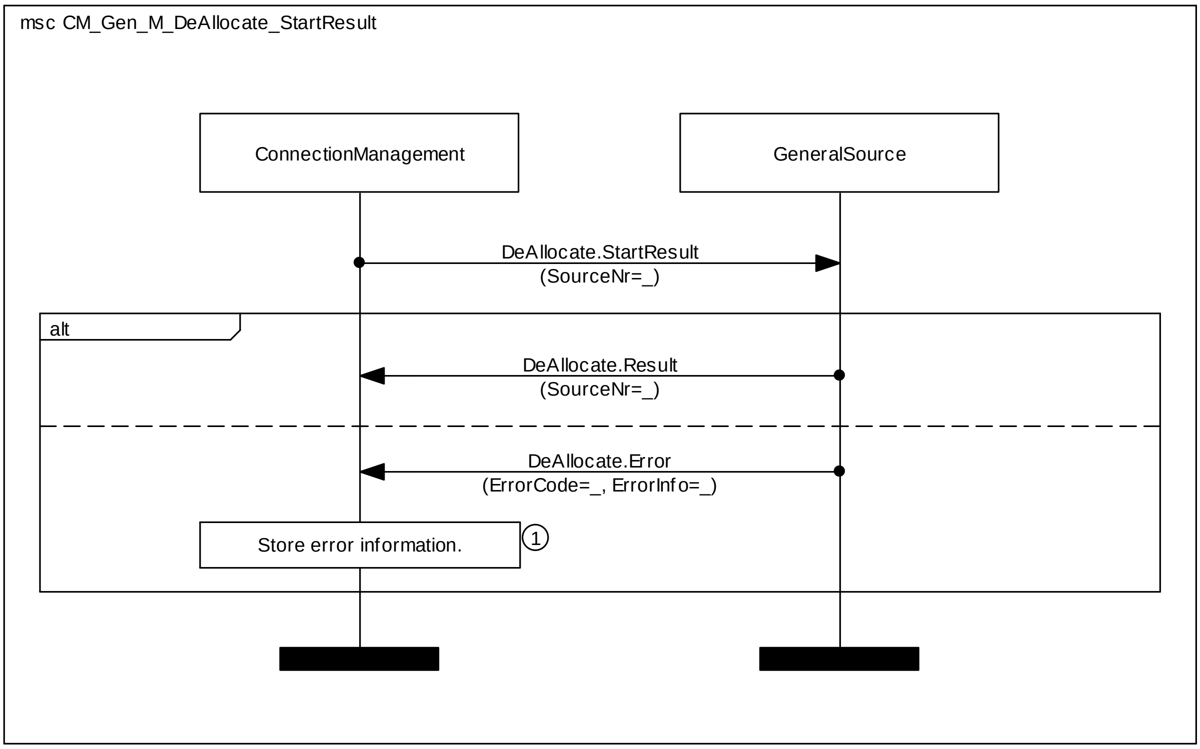

msc CM_Gen_SC_SourceDisConnect_StartResult
Note: The SourceConnect approach is not supported by MOST50.
ConnectionManagement
GeneralSource
SourceDisConnect.StartResult
(SourceNr=_)
alt
SourceDisConnect.Result
(SourceNr=_)
SourceDisConnect.Error
(ErrorCode=_, ErrorInfo=_)
Store error information.
1

MSC 44: CM_Gen_SC_SourceDisConnect_StartResult

1. Will be reported in BuildSyncConnection.Error or RemoveSyncConnection.Error.

4.4.1.3.3 Deallocation Procedure

General MSC:	CM_Gen_M_DeAllocate_StartResult
Description:	Connection Management tells the source to deallocate the specified channels and to disconnect. Error information is saved for the MSC that uses this one.
Prior Condition:	The source is connected to the network and uses the specified channels.
Initiator:	Connection Management
Communication Partners:	A source
Events	-
Timer / Timing Constraints:	-
Remarks:	



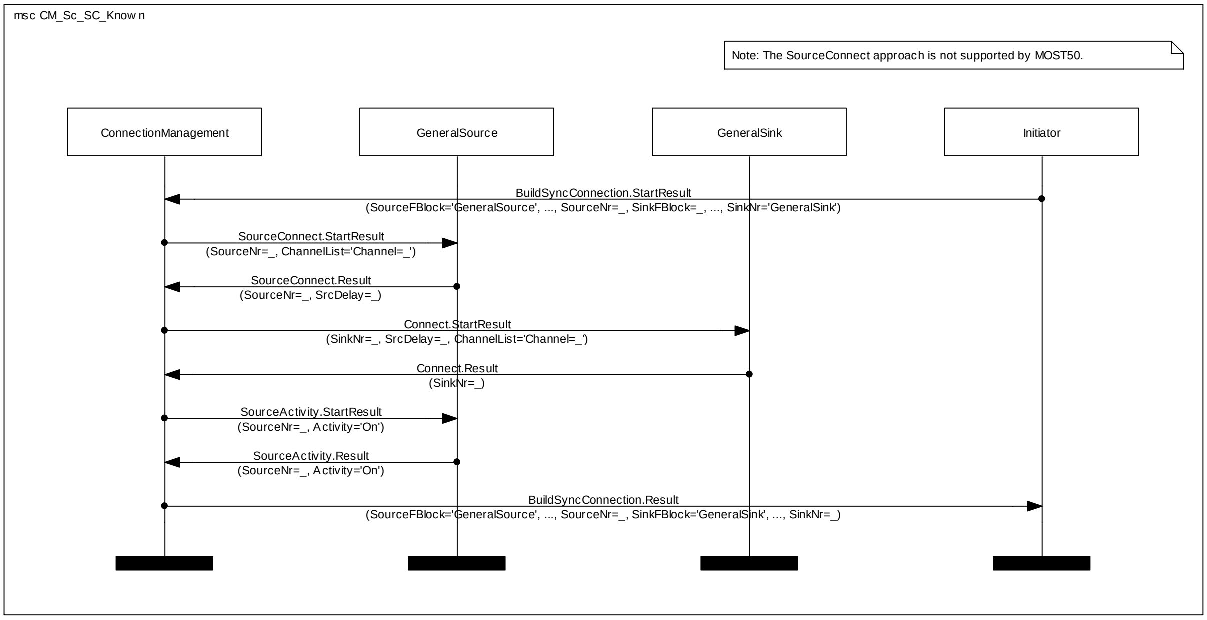
MSC 45: CM_Gen_M_DeAllocate_StartResult

1. Will be reported in BuildSyncConnection.Error or RemoveSyncConnection.Error.

4.4.2 Connection Management Scenario MSCs

4.4.2.1 Building a Connection with a Known Source

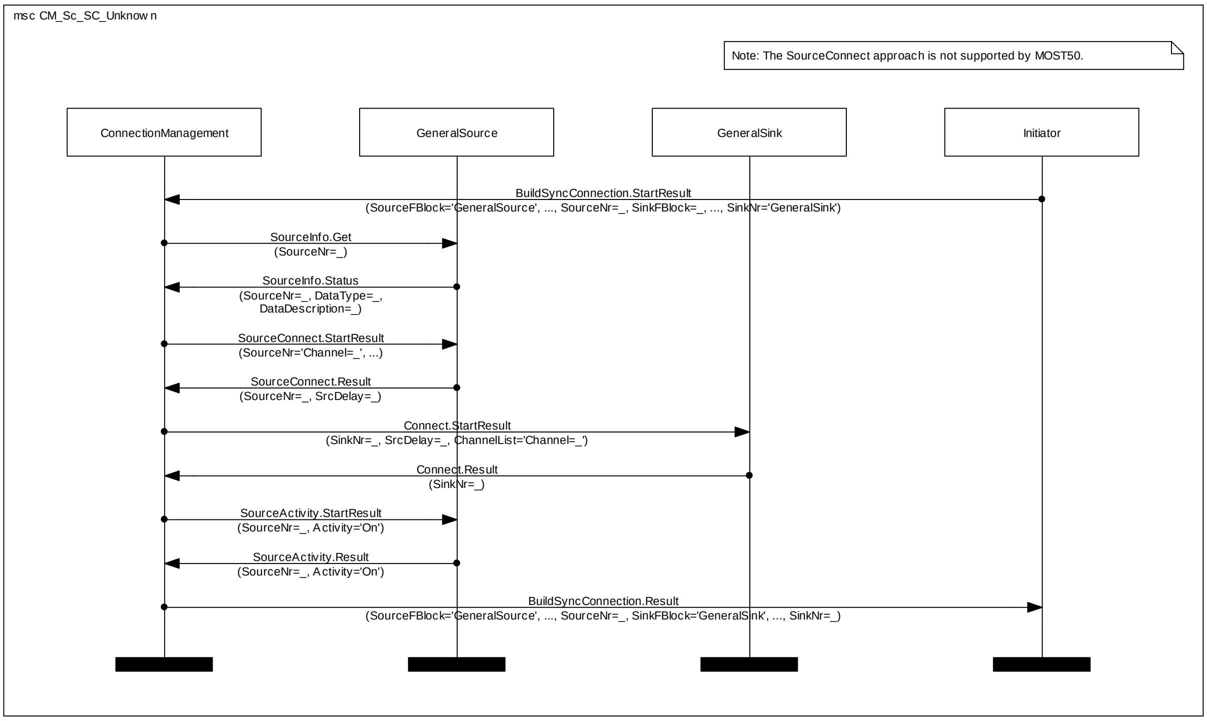
Scenario:	CM_Sc_SC_Known
Description:	Connection Management is familiar with the Source and knows that it uses SourceConnect and it also knows how many channels it needs. Hence, SourceInfo.Get can be skipped.
Prior Condition:	All sources in the system support the SourceConnect method. The allocation table of the Timing Master is not used.
Initiator:	Any controller
Communication Partners:	Initiator, a source, and a sink
Events	-
Remarks:	- This is the fastest way of building a connection. - Note that the source activity is optional.


msc CM_Sc_SC_Know n
Note: The SourceConnect approach is not supported by MOST50.
ConnectionManagement
GeneralSource
GeneralSink
Initiator
BuildSyncConnection.StartResult
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock=_, ..., SinkNr='GeneralSink')
SourceConnect.StartResult
(SourceNr=_, ChannelList='Channel=_')
SourceConnect.Result
(SourceNr=_, SrcDelay=_)
Connect.StartResult
(SinkNr=_, SrcDelay=_, ChannelList='Channel=_')
Connect.Result
(SinkNr=_)
SourceActivity.StartResult
(SourceNr=_, Activity='On')
SourceActivity.Result
(SourceNr=_, Activity='On')
BuildSyncConnection.Result
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock='GeneralSink', ..., SinkNr=_)

MSC 46: CM_Sc_SC_Known

4.4.2.2 SourceConnect with an Unknown Source

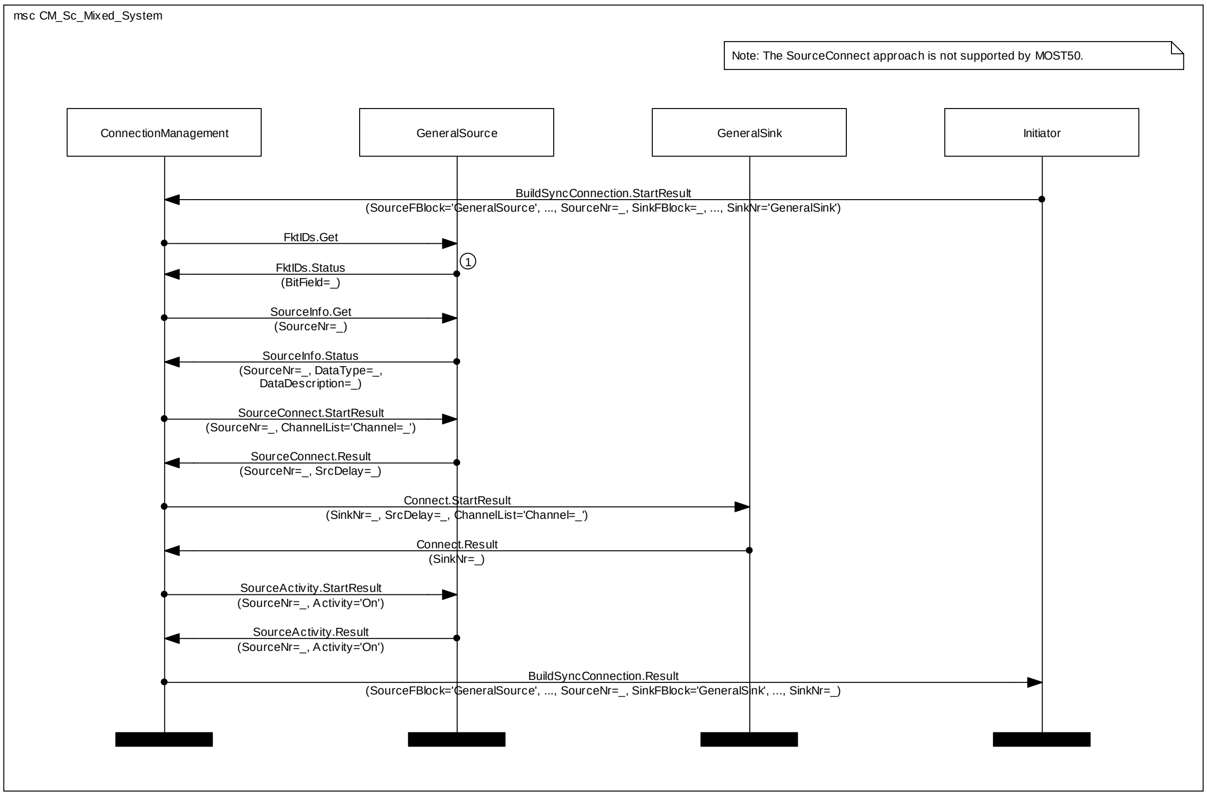
Scenario:	CM_Sc_SC_Unknown
Description:	Connection Management knows that the source uses SourceConnect, but it does not know how many channels it needs.
Prior Condition:	All sources in the system support the SourceConnect method. The allocation table of the Timing Master is not used.
Initiator:	Any controller
Communication Partners:	Controller, a source, and a sink
Events	-
Remarks:	- The SourceInfo information may be kept to speed up connections in the future. - Note that the source activity is optional.


msc CM_Sc_SC_Unknown
Note: The SourceConnect approach is not supported by MOST50.
ConnectionManagement
GeneralSource
GeneralSink
Initiator
BuildSyncConnection.StartResult
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock=_, ..., SinkNr='GeneralSink')
SourceInfo.Get
(SourceNr=_)
SourceInfo.Status
(SourceNr=_, DataType=_,
DataDescription=_)
SourceConnect.StartResult
(SourceNr='Channel=_', ...)
SourceConnect.Result
(SourceNr=_, SrcDelay=_)
Connect.StartResult
(SinkNr=_, SrcDelay=_, ChannelList='Channel=_')
Connect.Result
(SinkNr=_)
SourceActivity.StartResult
(SourceNr=_, Activity='On')
SourceActivity.Result
(SourceNr=_, Activity='On')
BuildSyncConnection.Result
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock='GeneralSink', ..., SinkNr=_)

MSC 47: CM_Sc_SC_Unknown

4.4.2.3 Building a Connection in a Mixed System with Unknown Sources

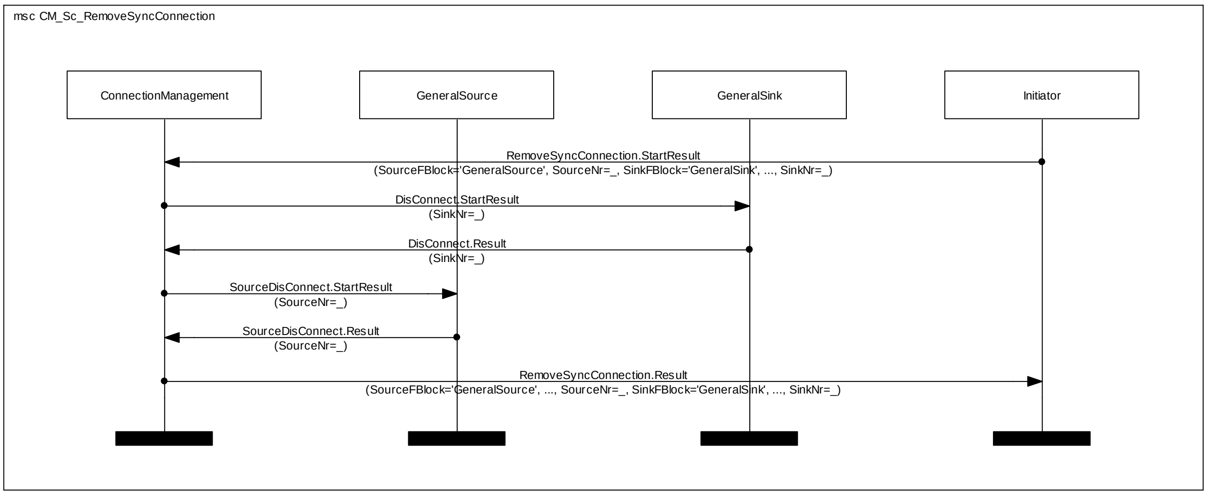
Scenario:	CM_Sc_MixedSystem
Description:	The Connection Management is in a Mixed System with no knowledge of the sources. It finds a SourceConnect node.
Prior Condition:	-
Initiator:	Any controller
Communication Partners:	Controller, a source, and a sink
Events	-
Remarks:	- This is the slowest way of building a connection. - Note that the source activity is optional.


msc CM_Sc_Mixed_System
Note: The SourceConnect approach is not supported by MOST50.
ConnectionManagement
GeneralSource
GeneralSink
Initiator
BuildSyncConnection.StartResult
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock=_, ..., SinkNr='GeneralSink')
FktIDs.Get
1
FktIDs.Status
(BitField=_)
SourceInfo.Get
(SourceNr=_)
SourceInfo.Status
(SourceNr=_, DataType=_,
DataDescription=_)
SourceConnect.StartResult
(SourceNr=_, ChannelList='Channel=_')
SourceConnect.Result
(SourceNr=_, SrcDelay=_)
Connect.StartResult
(SinkNr=_, SrcDelay=_, ChannelList='Channel=_')
Connect.Result
(SinkNr=_)
SourceActivity.StartResult
(SourceNr=_, Activity='On')
SourceActivity.Result
(SourceNr=_, Activity='On')
BuildSyncConnection.Result
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock='GeneralSink', ..., SinkNr=_)

MSC 48: CM_Sc_MixedSystem

4.4.2.4 Removing a Synchronous Connection

Scenario:	CM_Sc_RemoveSyncConnection
Description:	The Connection Management is removing a connection and therefore knows what kind of nodes it uses. It finds a SourceConnect node.
Prior Condition:	-
Initiator:	Any controller
Communication Partners:	Controller, a source, and a sink
Events	-
Remarks:	- Nodes are required to mute when receiving a DisConnect command and to turn SourceActivity off when receiving DeAllocate or SourceDisConnect. - Note that the source activity is optional.


msc CM_Sc_RemoveSyncConnection
ConnectionManagement
GeneralSource
GeneralSink
Initiator
RemoveSyncConnection.StartResult
(SourceFBlock='GeneralSource', SourceNr=_, SinkFBlock='GeneralSink', ..., SinkNr=_)
DisConnect.StartResult
(SinkNr=_)
DisConnect.Result
(SinkNr=_)
SourceDisConnect.StartResult
(SourceNr=_)
SourceDisConnect.Result
(SourceNr=_)
RemoveSyncConnection.Result
(SourceFBlock='GeneralSource', ..., SourceNr=_, SinkFBlock='GeneralSink', ..., SinkNr=_)

MSC 49: CM_Sc_RemoveSyncConnection

4.5 Error Handling

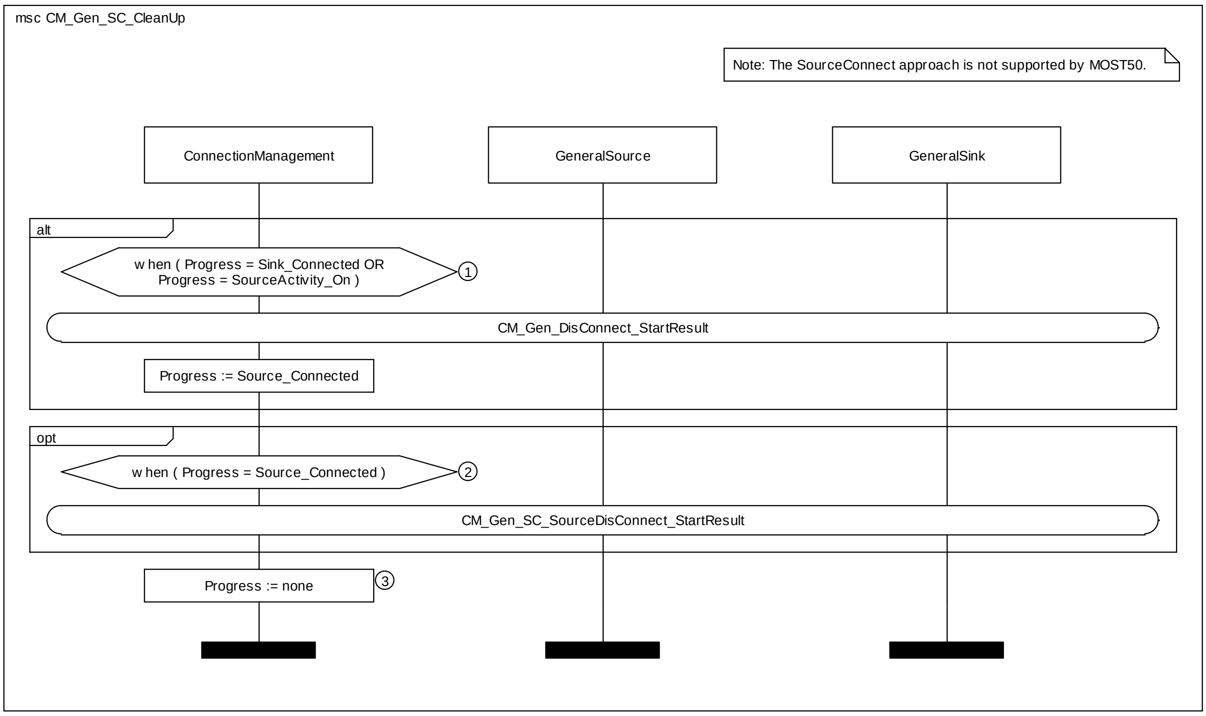
This section describes the behavior of the Connection Management when dealing with error cases.

The function OPTypes within this chapter are given in non-Ack format. If desired, the Ack form of the function OPTypes can be used.

4.5.1 Error Handling General MSCs

4.5.1.1 CleanUp in a SourceConnect System

General MSC:	CM_Gen_SC_CleanUp
Description:	Connection Management unmakes everything that has been made in the connection procedure.
Prior Condition:	An error or a timeout has interrupted the BuildSyncConnection procedure. The system consists only of sources using method SourceConnect to connect to the network.
Initiator:	This MSC is initiated after an abort or error within BuildSyncConnection.
Communication Partners:	A source and a sink
Events	-
Timer / Timing Constraints:	-
Remarks:	There is no way of aborting the CleanUp process.

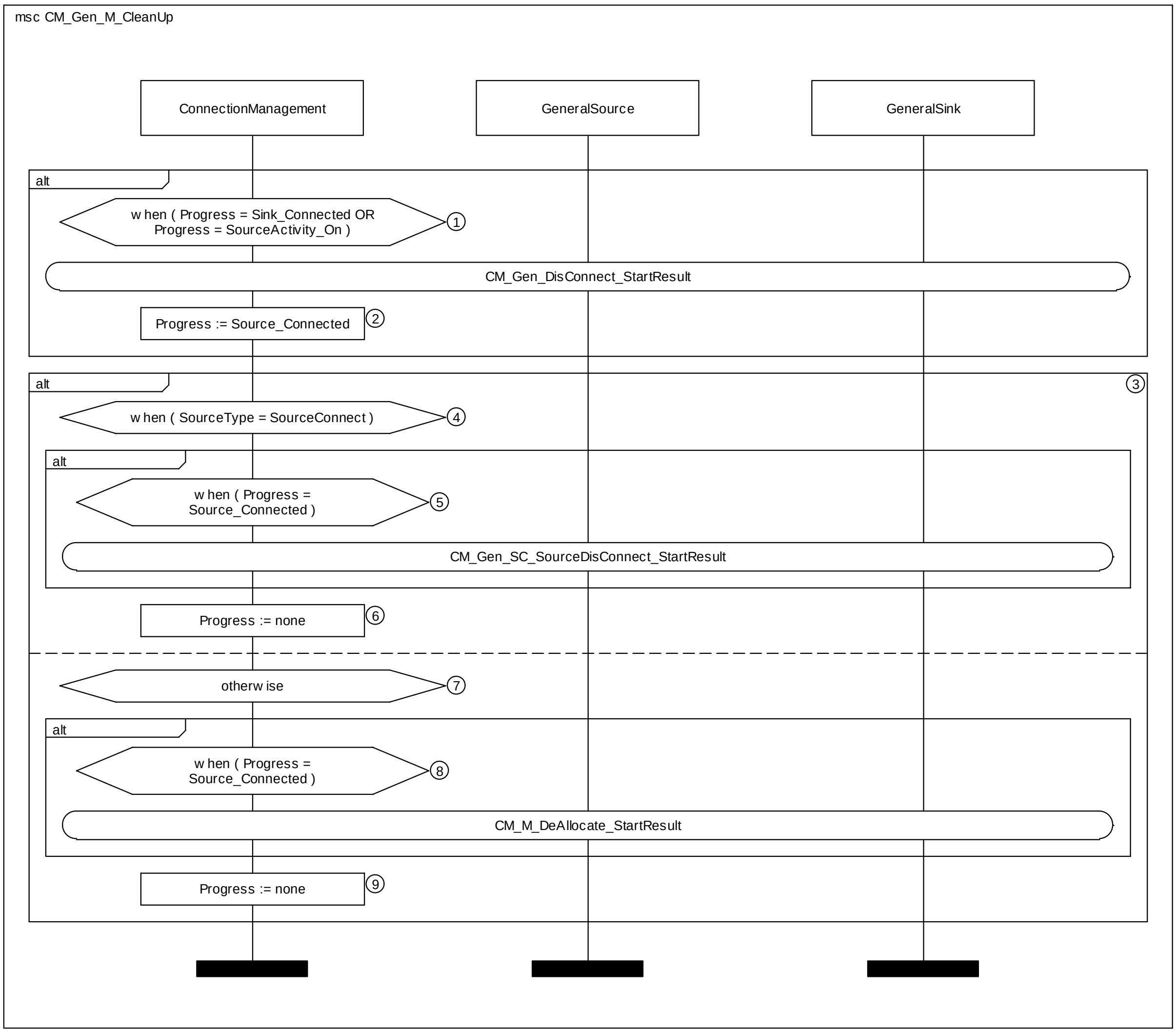

msc CM_Gen_SC_CleanUp
Note: The SourceConnect approach is not supported by MOST50.
ConnectionManagement
GeneralSource
GeneralSink
alt
w hen (Progress = Sink_Connected OR Progress = SourceActivity_On)
1
CM_Gen_DisConnect_StartResult
Progress := Source_Connected
opt
w hen (Progress = Source_Connected)
2
CM_Gen_SC_SourceDisConnect_StartResult
Progress := none
3

MSC 50: CM_Gen_SC_CleanUp

1. Sink has to be disconnected.
2. Source has to be disconnected.
3. CleanUp finished.

4.5.1.2 CleanUp in a Mixed System

General MSC:	CM_Gen_M_CleanUp
Description:	Connection Management unmakes everything that has been made in the connection procedure.
Prior Condition:	An error or abort has interrupted the BuildSyncConnection procedure. The source is disconnected in different ways depending on if it was connected using Allocate or SourceConnect.
Initiator:	This MSC is initiated after an abort or error within BuildSyncConnection.
Communication Partners:	A source and a sink
Events	-
Timer / Timing Constraints:	-
Remarks:	There is no way of aborting the CleanUp process.


msc CM_Gen_M_CleanUp
ConnectionManagement
GeneralSource
GeneralSink
alt
w hen (Progress = Sink_Connected OR Progress = SourceActivity_On)
1
CM_Gen_DisConnect_StartResult
Progress := Source_Connected
2
alt
3
w hen (SourceType = SourceConnect)
4
alt
w hen (Progress = Source_Connected)
5
CM_Gen_SC_SourceDisConnect_StartResult
Progress := none
6
otherw ise
7
alt
w hen (Progress = Source_Connected)
8
CM_M_DeAllocate_StartResult
Progress := none
9

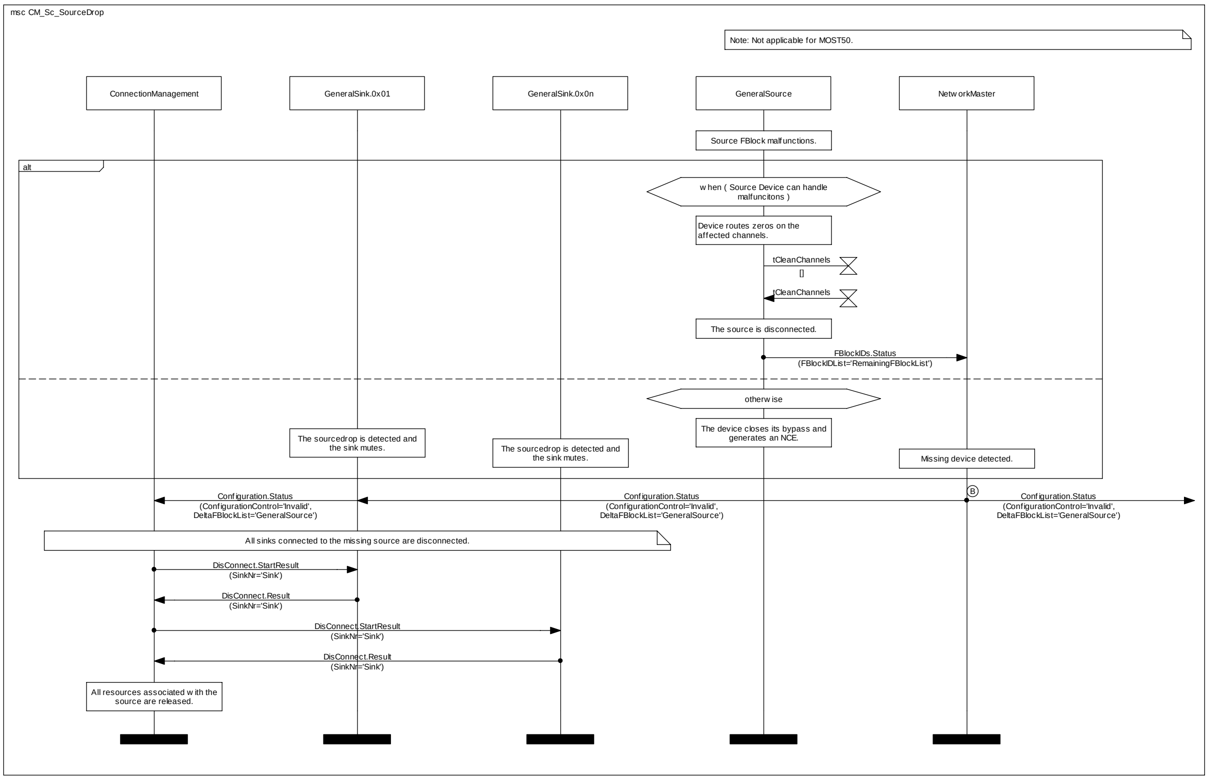
MSC 51: CM_Gen_M_CleanUp

1. Sink has to be disconnected.
2. Continue cleanup.
3. Nodes using SourceConnect and nodes using Allocate are handled differently.
4. Note: The SourceConnect approach is not supported by MOST50.
5. Source has to be disconnected.
6. CleanUp finished.
7. SourceType is Allocate.
8. Source has to be disconnected.
9. CleanUp finished.

4.5.2 Error Handling Scenario MSCs

4.5.2.1 Source drop

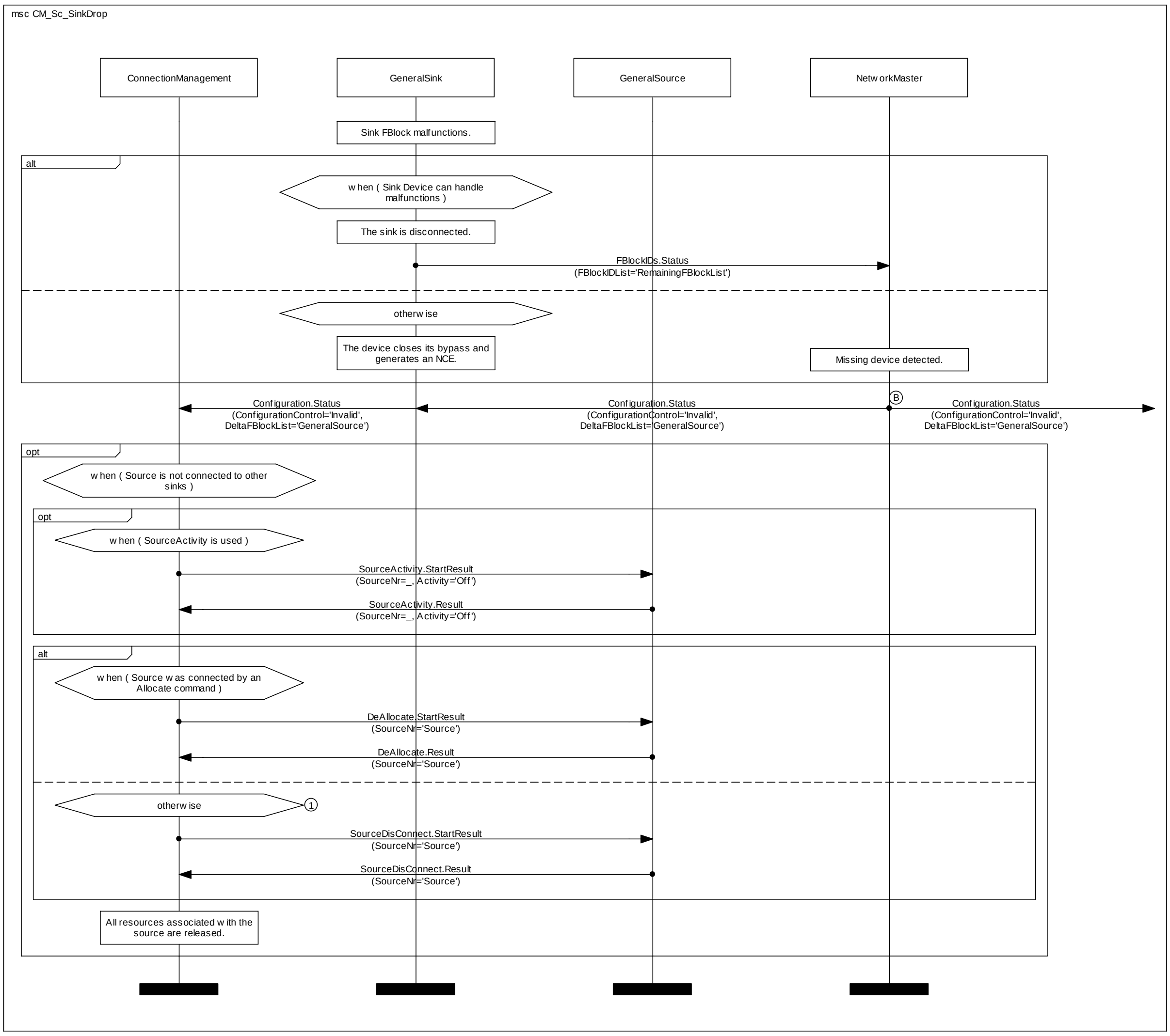
General MSC:	CM_Sc_SourceDrop
Description:	General Source drops out due to an unlikely internal error.
Prior Condition:	-
Initiator:	Any failing General Source
Communication Partners:	-
Events	-
Timer / Timing Constraints:	- $t_{CleanChannels}$
Remarks:	The deallocation of channels for GeneralSource is not possible, since the GeneralSource already is disconnected (dropped out).


msc CM_Sc_SourceDrop
Note: Not applicable for MOST50.
ConnectionManagement
GeneralSink.0x01
GeneralSink.0x0n
GeneralSource
Netw orkMaster
Source FBlock malfunctions.
alt
w hen (Source Device can handle malfuncitons)
Device routes zeros on the affected channels.
tCleanChannels
[]
tCleanChannels
The source is disconnected.
FBlockIDs.Status (FBlockIDList='RemainingFBlockList')
otherw ise
The sourcedrop is detected and the sink mutes.
The sourcedrop is detected and the sink mutes.
The device closes its bypass and generates an NCE.
Missing device detected.
B
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList='GeneralSource')
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList='GeneralSource')
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList='GeneralSource')
All sinks connected to the missing source are disconnected.
DisConnect.StartResult (SinkNr='Sink')
DisConnect.Result (SinkNr='Sink')
DisConnect.StartResult (SinkNr='Sink')
DisConnect.Result (SinkNr='Sink')
All resources associated w ith the source are released.

MSC 52: CM_Sc_SourceDrop

4.5.2.2 Sink drop

General MSC:	CM_Sc_SinkDrop
Description:	General Sink drops out due to an unlikely internal error.
Prior Condition:	-
Initiator:	Any failing General Sink
Communication Partners:	-
Events	-
Timer / Timing Constraints:	-
Remarks:	- Note that the source activity is optional.


msc CM_Sc_SinkDrop
ConnectionManagement
GeneralSink
GeneralSource
Netw orkMaster
Sink FBlock malfunctions.
alt
w hen (Sink Device can handle malfunctions)
The sink is disconnected.
FBlockIDs.Status (FBlockIDList='RemainingFBlockList')
otherw ise
The device closes its bypass and generates an NCE.
Missing device detected.
B
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList='GeneralSource')
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList='GeneralSource')
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList='GeneralSource')
opt
w hen (Source is not connected to other sinks)
opt
w hen (SourceActivity is used)
SourceActivity.StartResult (SourceNr=_, Activity='Off')
SourceActivity.Result (SourceNr=_, Activity='Off')
alt
w hen (Source w as connected by an Allocate command)
DeAllocate.StartResult (SourceNr='Source')
DeAllocate.Result (SourceNr='Source')
otherw ise
1
SourceDisConnect.StartResult (SourceNr='Source')
SourceDisConnect.Result (SourceNr='Source')
All resources associated w ith the source are released.

MSC 53: CM_Sc_SinkDrop

4.6 Extended ConnectionMaster General MSCs

TBD

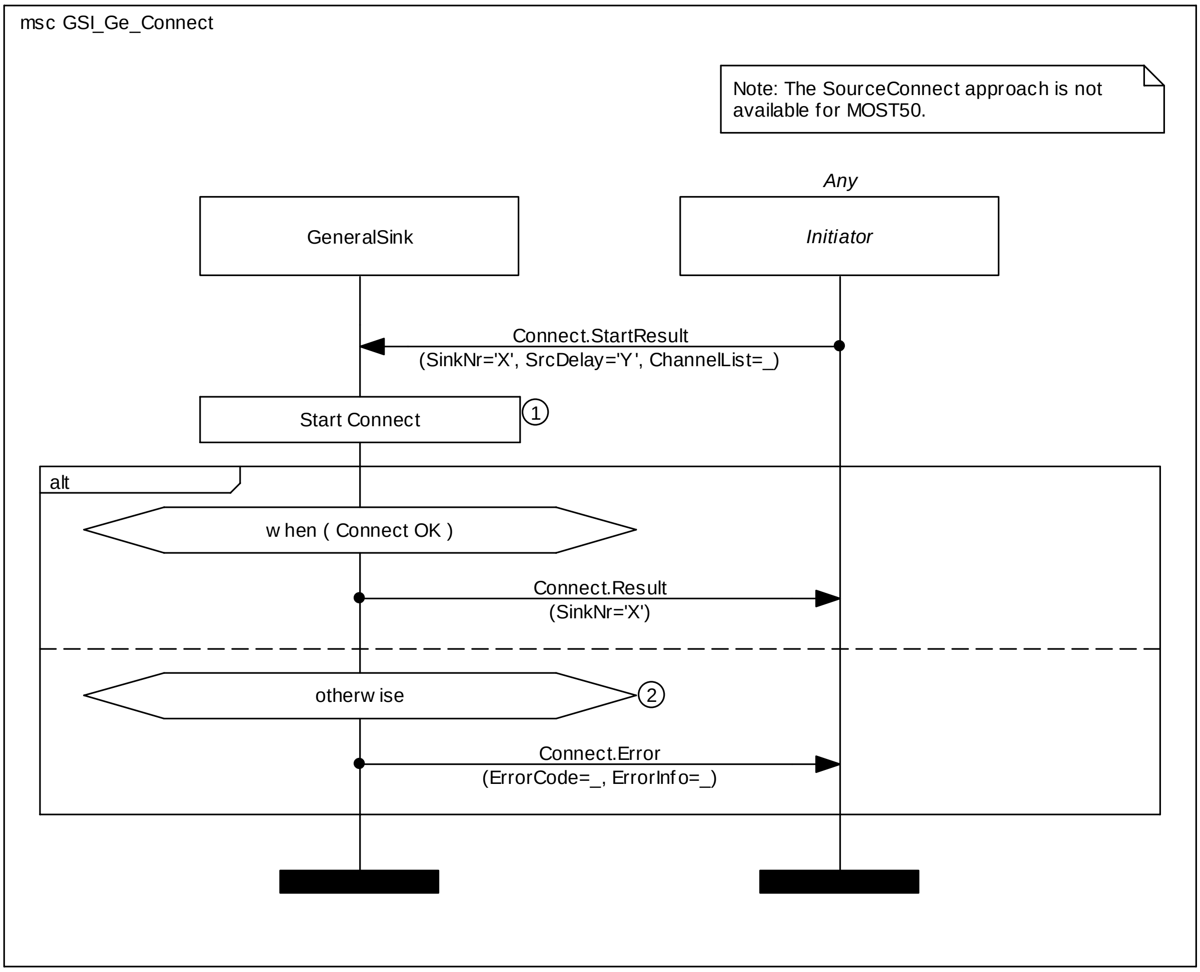
4.7 Extended ConnectionMaster Scenarios

TBD

4.8 General Sink MSCs

4.8.1 Connect

Use Case:	GSI_Ge_Connect
Description:	Request a sink to connect to channels.
Prior Condition:	-
Initiator:	-
Communication Partners:	Initiator
Events:	-
Remarks:	The sink needs to be demuted after it is connected and sourcerouting started.

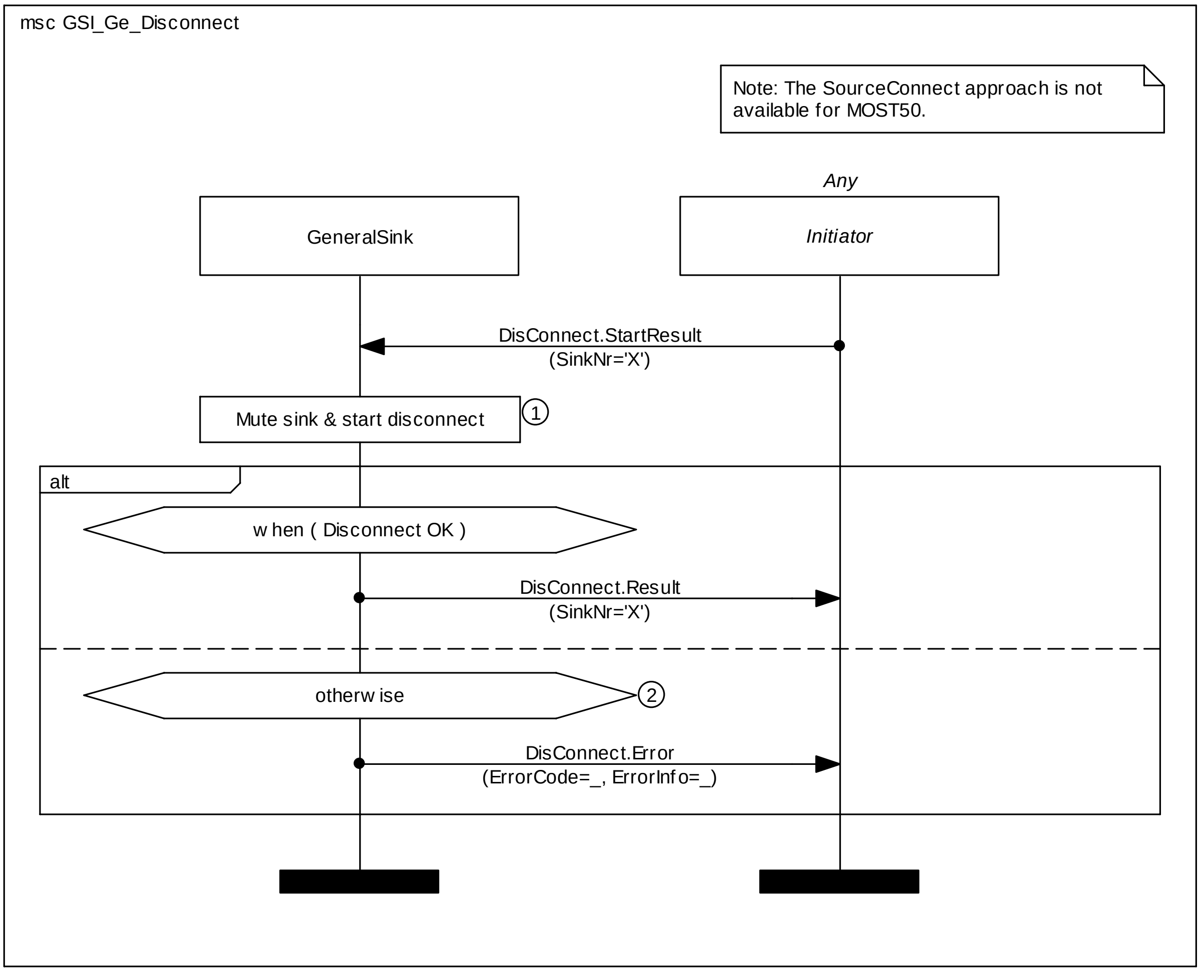


MSC 54: GSI_Ge_Connect

1. Performed regardless of earlier connect status. If already connected to same channels, no action has to be performed; this is a Connect OK.
2. For example, erroneous SinkNr.

4.8.2 Disconnect

Use Case:	GSI_Ge_Disconnect
Description:	Request a sink to disconnect from channels.
Prior Condition:	-
Initiator:	-
Communication Partners:	Initiator
Events:	-
Remarks:	-



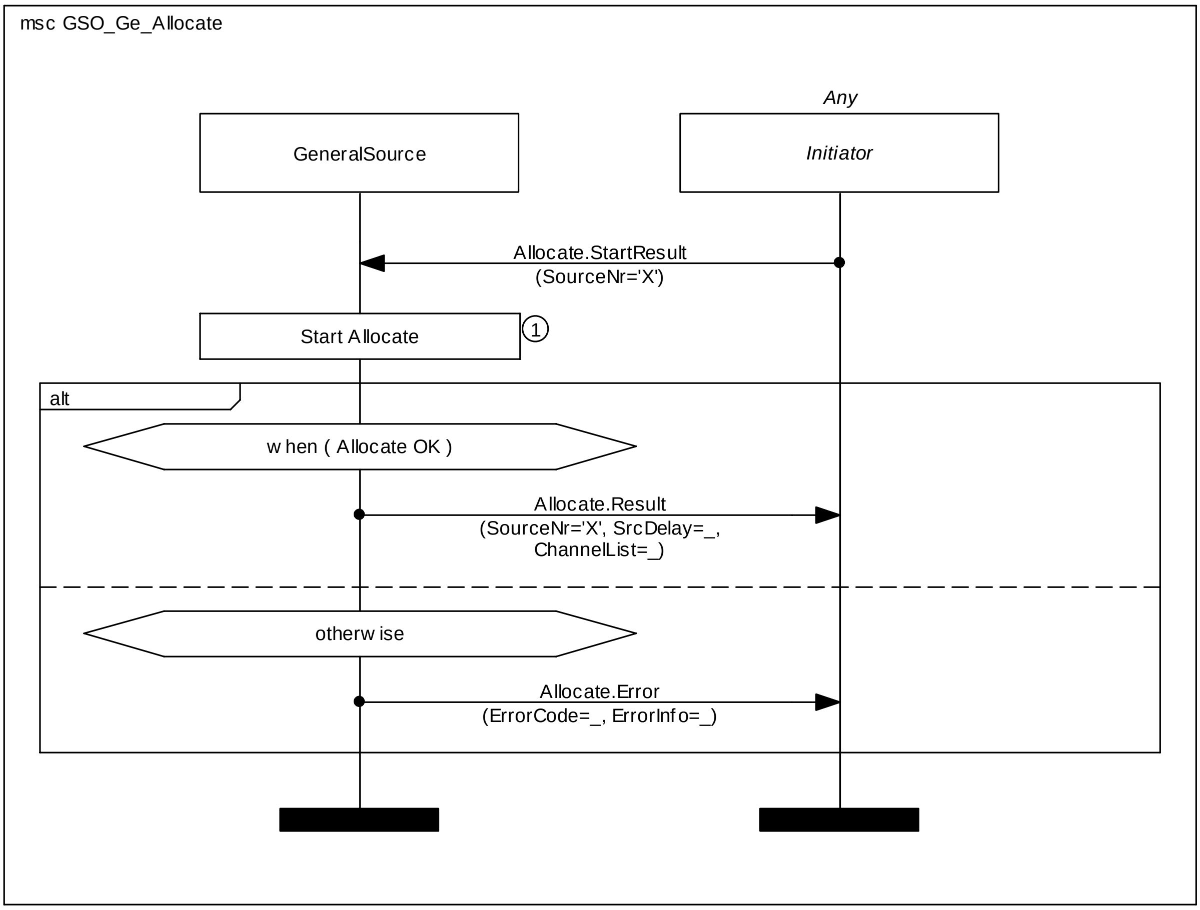
MSC 55: GSI_Ge_Disconnect

1. No action needed if not connected at all; this is treated as a successful disconnect.
2. For example, erroneous SinkNr.

4.9 General Source MSCs

4.9.1 Allocate

Use Case:	GSO_Ge_Allocate
Description:	Request a source to perform allocate.
Prior Condition:	-
Initiator:	-
Communication Partners:	Initiator
Events:	-
Remarks:	- The source should not start routing data until SourceActivity.StartResult(On) is called. - Note that the source activity is optional.

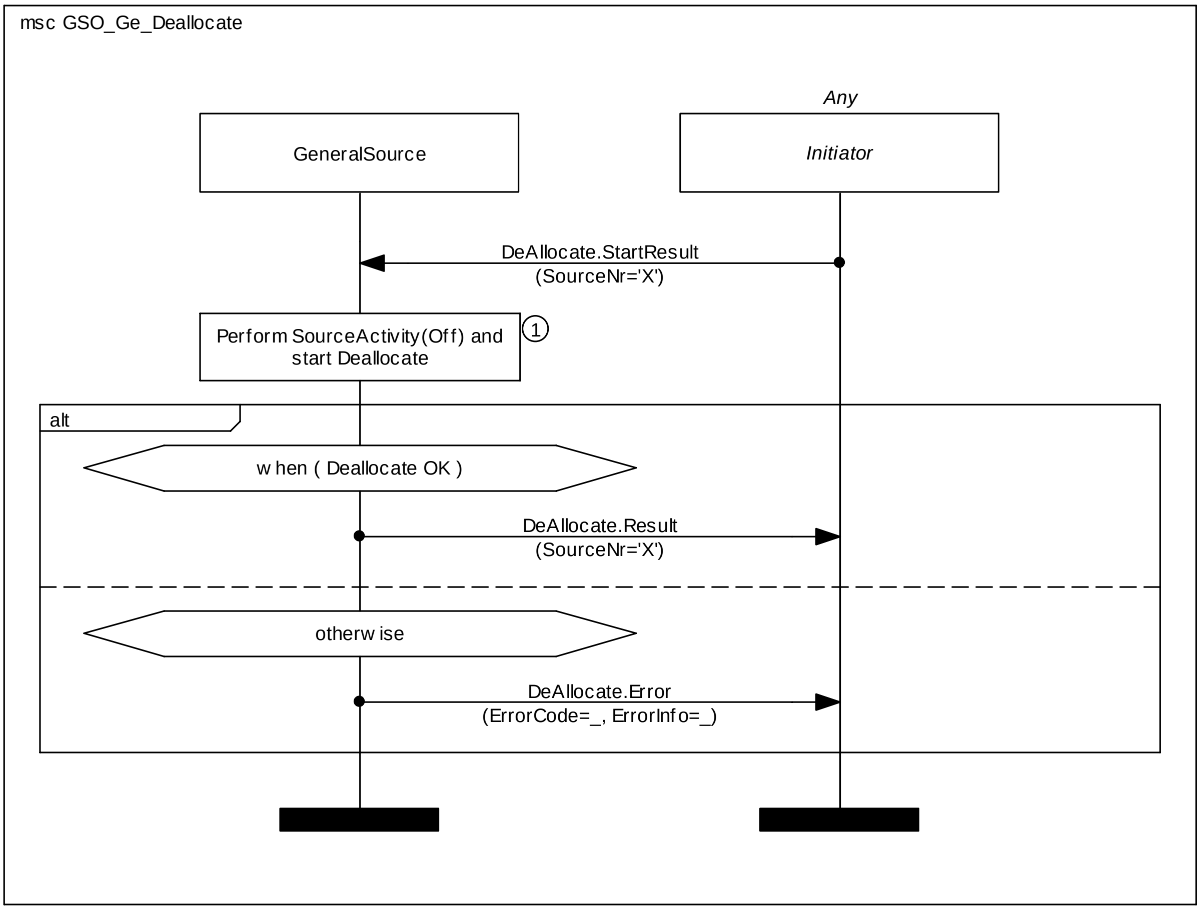


MSC 56: GSO_Ge_Allocate

1. If SourceNr already is allocated, it is treated as a successful Allocate.

4.9.2 Deallocate

Use Case:	GSO_Ge_Deallocate
Description:	Request a source to deallocate.
Prior Condition:	-
Initiator:	-
Communication Partners:	Initiator
Events:	-
Remarks:	The source stops routing and removes its channels.

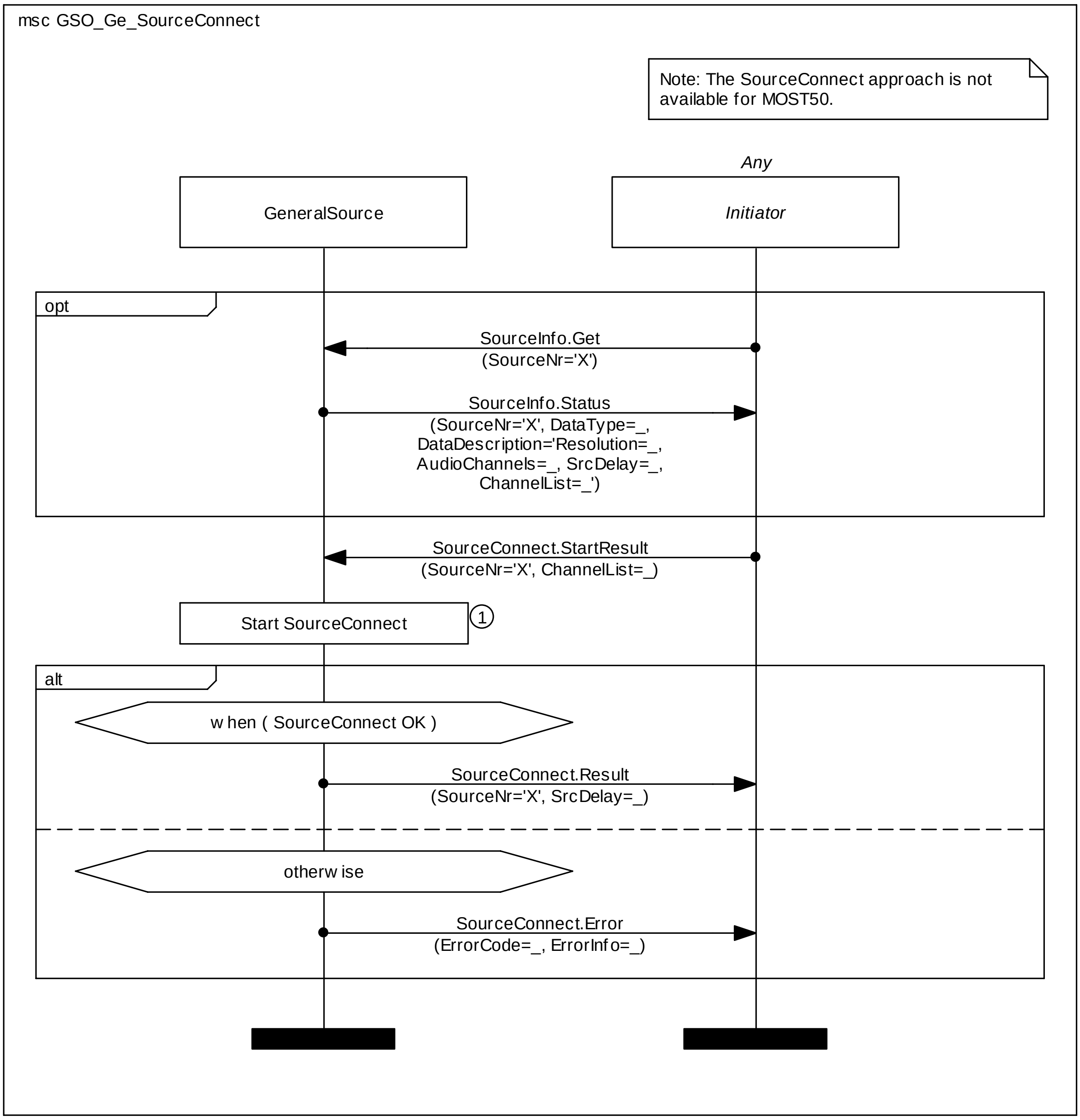


MSC 57: GSO_Ge_Deallocate

1. No action needed if not allocated at all; this is treated as a successful deallocate.

4.9.3 SourceConnect

Use Case:	GSO_Ge_SourceConnect
Description:	Request a source to connect itself to pre-allocated channels.
Prior Condition:	-
Initiator:	-
Communication Partners:	Initiator
Events:	-
Remarks:	- The source should not start routing data until SourceActivity.StartResult(On) is called. - Note that the source activity is optional.

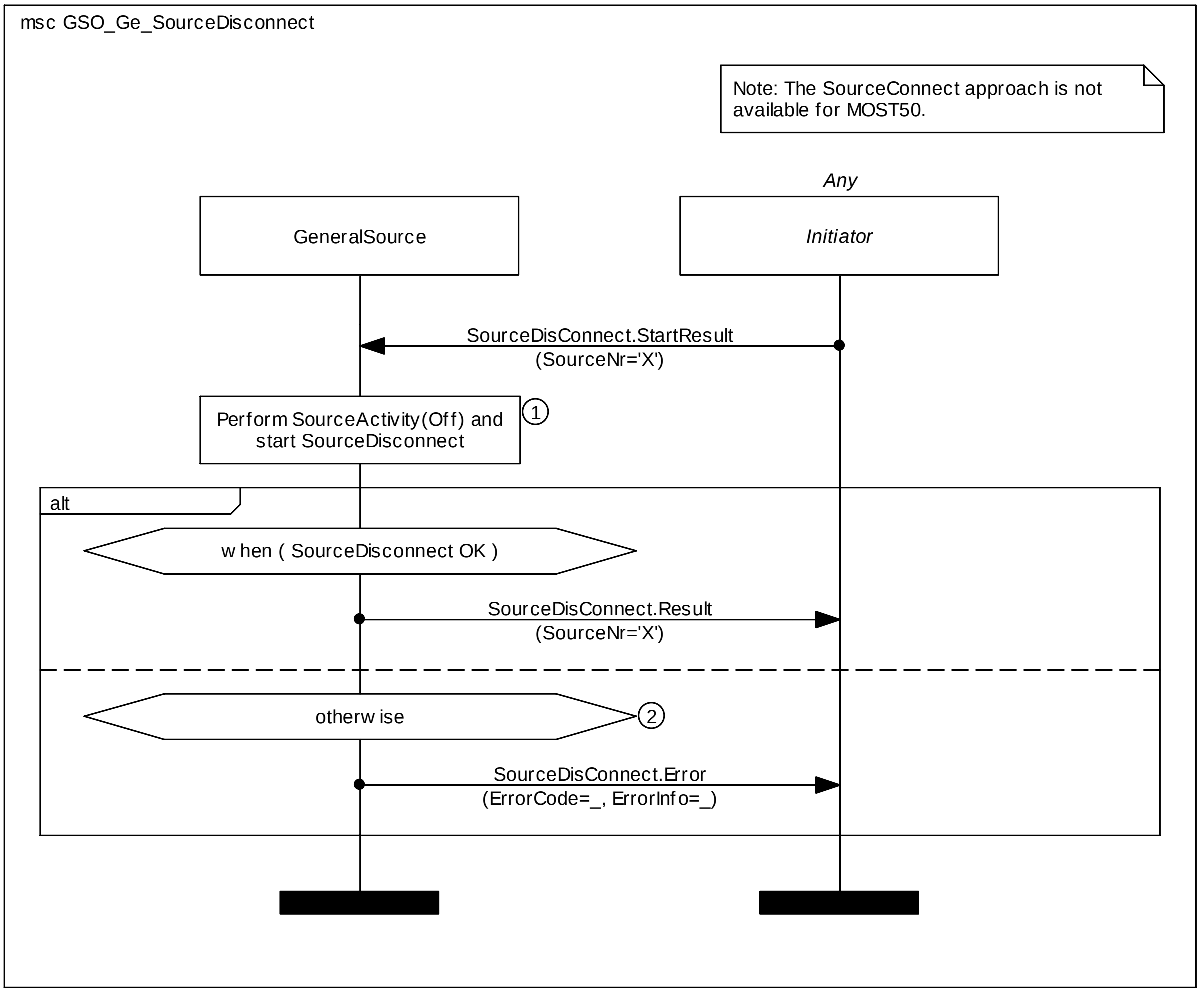

msc GSO_Ge_SourceConnect
Note: The SourceConnect approach is not available for MOST50.
Any
GeneralSource
Initiator
opt
SourceInfo.Get
(SourceNr='X')
SourceInfo.Status
(SourceNr='X', DataType=_,
DataDescription='Resolution=_,
AudioChannels=_, SrcDelay=_,
ChannelList=_')
SourceConnect.StartResult
(SourceNr='X', ChannelList=_)
Start SourceConnect
1
alt
when (SourceConnect OK)
SourceConnect.Result
(SourceNr='X', SrcDelay=_)
otherwise
SourceConnect.Error
(ErrorCode=_, ErrorInfo=_)

MSC 58: GSO_Ge_SourceConnect

1. If SourceNr already is connected, it is treated as a successful SourceConnect.

4.9.4 SourceDisconnect

Use Case:	GSO_Ge_SourceDisconnect
Description:	Request a source to disconnect from the pre-allocated channels.
Prior Condition:	-
Initiator:	-
Communication Partners:	Initiator
Events:	-
Remarks:	The source stops routing.

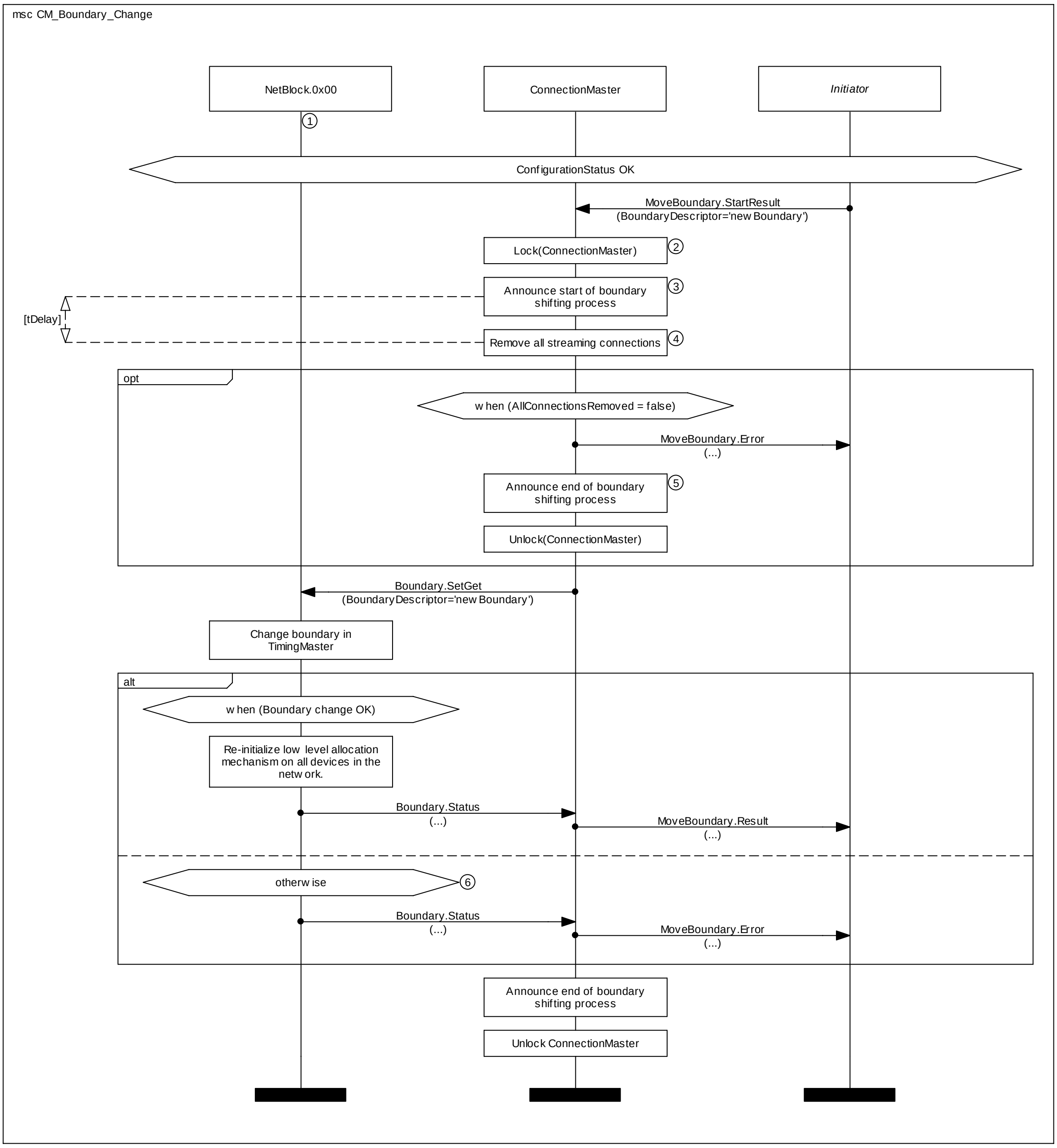

msc GSO_Ge_SourceDisconnect
Note: The SourceConnect approach is not available for MOST50.
Any
GeneralSource
Initiator
SourceDisConnect.StartResult (SourceNr='X')
Perform SourceActivity(Off) and start SourceDisconnect
1
alt
when (SourceDisconnect OK)
SourceDisConnect.Result (SourceNr='X')
otherwise
2
SourceDisConnect.Error (ErrorCode=_, ErrorInfo=_)

MSC 59: GSO_Ge_SourceDisconnect

1. If SourceNr is not connected at all, it is treated as a successful SourceDisconnect.
2. For example, faulty SourceNr.

4.10 Boundary Change

Use Case:	CM_Boundary_Change
Description:	The initiator requests a Boundary change from the ConnectionMaster, which forwards the request to the TimingMaster.
Prior Condition:	Configuration Status OK
Initiator:	-
Communication Partners:	Initiator, ConnectionMaster, TimingMaster
Events:	-
Remarks:	


msc CM_Boundary_Change
NetBlock.0x00
ConnectionMaster
Initiator
1
ConfigurationStatus OK
MoveBoundary.StartResult
(BoundaryDescriptor='new Boundary')
Lock(ConnectionMaster)
2
Announce start of boundary shifting process
3
[tDelay]
Remove all streaming connections
4
opt
w hen (AllConnectionsRemoved = false)
MoveBoundary.Error
(...)
Announce end of boundary shifting process
5
Unlock(ConnectionMaster)
Boundary.SetGet
(BoundaryDescriptor='new Boundary')
Change boundary in TimingMaster
alt
w hen (Boundary change OK)
Re-initialize low level allocation mechanism on all devices in the netw ork.
Boundary.Status
(...)
MoveBoundary.Result
(...)
otherw ise
6
Boundary.Status
(...)
MoveBoundary.Error
(...)
Announce end of boundary shifting process
Unlock ConnectionMaster

MSC 60: CM_Boundary_Change

1. NetBlock with InstID 0x00 = TimingMaster

2. Reject new Build/Remove requests
3. By setting the respective property, all notified devices informed about the forthcoming boundary change and can take appropriate action before streaming connections are removed and the boundary is physically changed. This can be done by implementing the optional property "BoundaryChange" or any other OEM specific solution.
4. tDelay - Optional delay between announcing the start of the boundary change and starting to remove streaming connections should be system configurable.
5. By clearing the respective property, all notified devices are informed that boundary change process is completed. This can be done by implementing the optional property "BoundaryChange" or any other OEM specific solution.
6. The allocation table is empty.

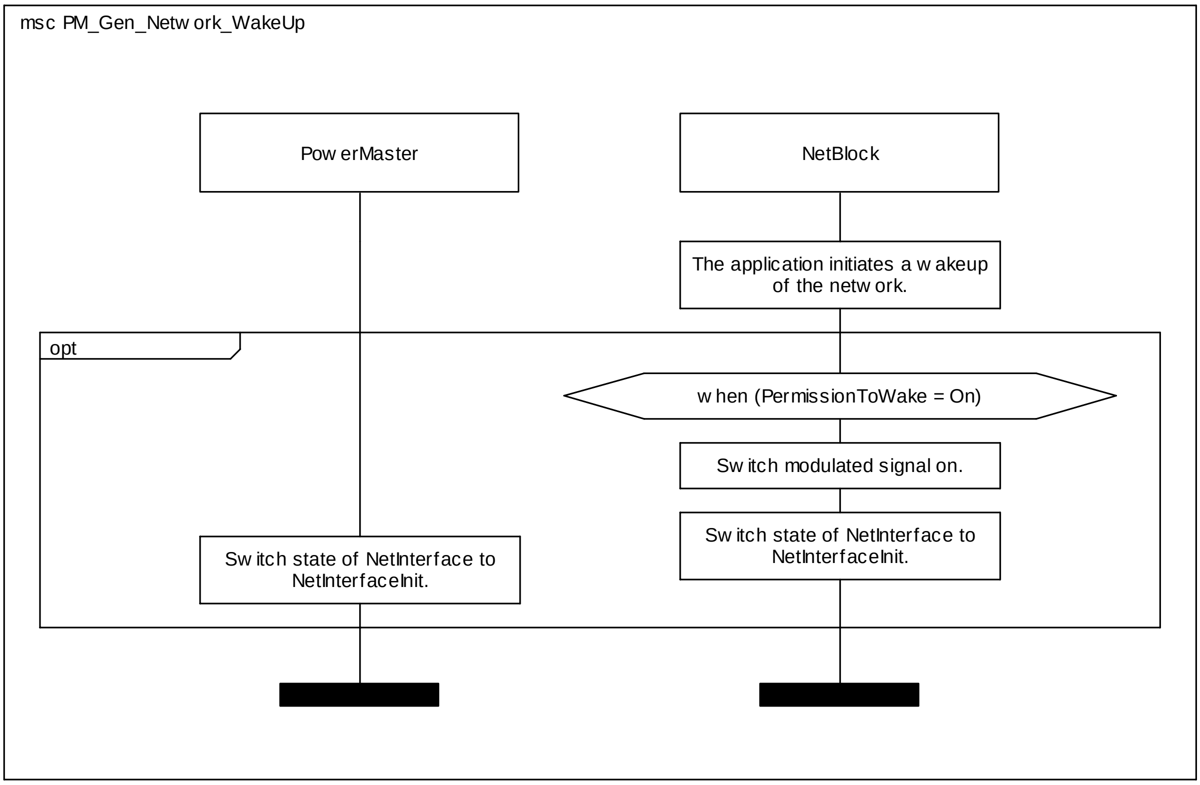
5 Power management

5.1 Introduction

Power management means that the administrative function, which is above the Network Service, wakes and shuts down the MOST network or specific devices. The power management is handled mainly by the PowerMaster that uses NetBlock functions for this purpose.

5.2 Network wake up

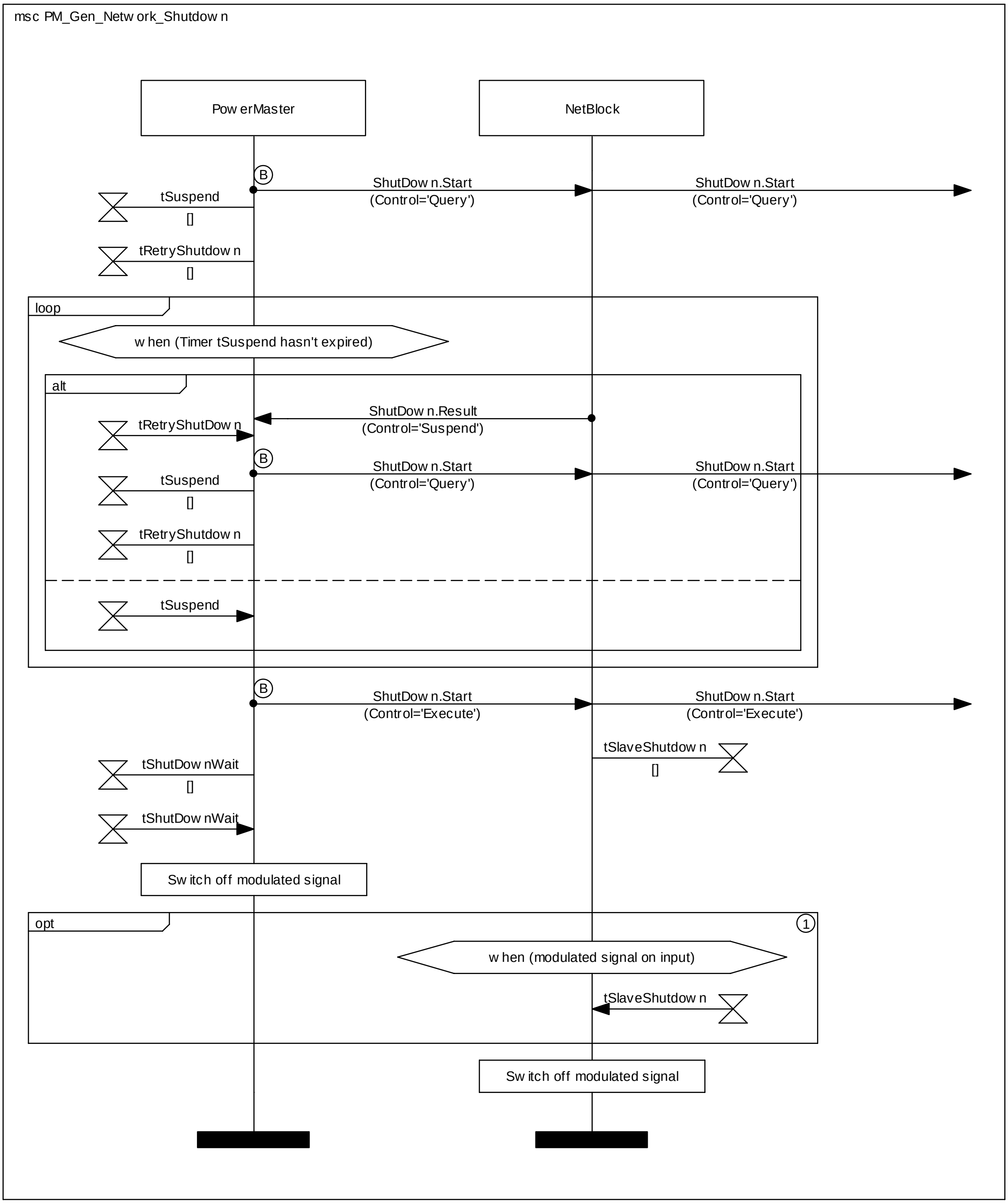
General MSC:	PM_Gen_Network_WakeUp
Description:	This process handles a waking of the network.
Prior Condition:	The NetInterface of the device is in state “Off”.
Initiator:	
Communication Partners:	All NetBlocks
Events:	
Timers / Timing Constraints:	
Remarks:	


msc PM_Gen_Netw ork_WakeUp
Pow erMaster
NetBlock
The application initiates a w akeup of the netw ork.
opt
w hen (PermissionToWake = On)
Sw itch modulated signal on.
Sw itch state of NetInterface to NetInterfaceInit.
Sw itch state of NetInterface to NetInterfaceInit.

MSC 61: PM_Gen_Network_WakeUp

5.3 Network shutdown

General MSC:	PM_Gen_Network_Shutdown
Description:	This process handles a shutdown of the network.
Prior Condition:	
Initiator:	
Communication Partners:	All NetBlocks
Events:	
Timers / Timing Constraints:	- $t_{Suspend}$ - $t_{RetryShutDown}$ - $t_{ShutDownWait}$ - $t_{SlaveShutDown}$
Remarks:	

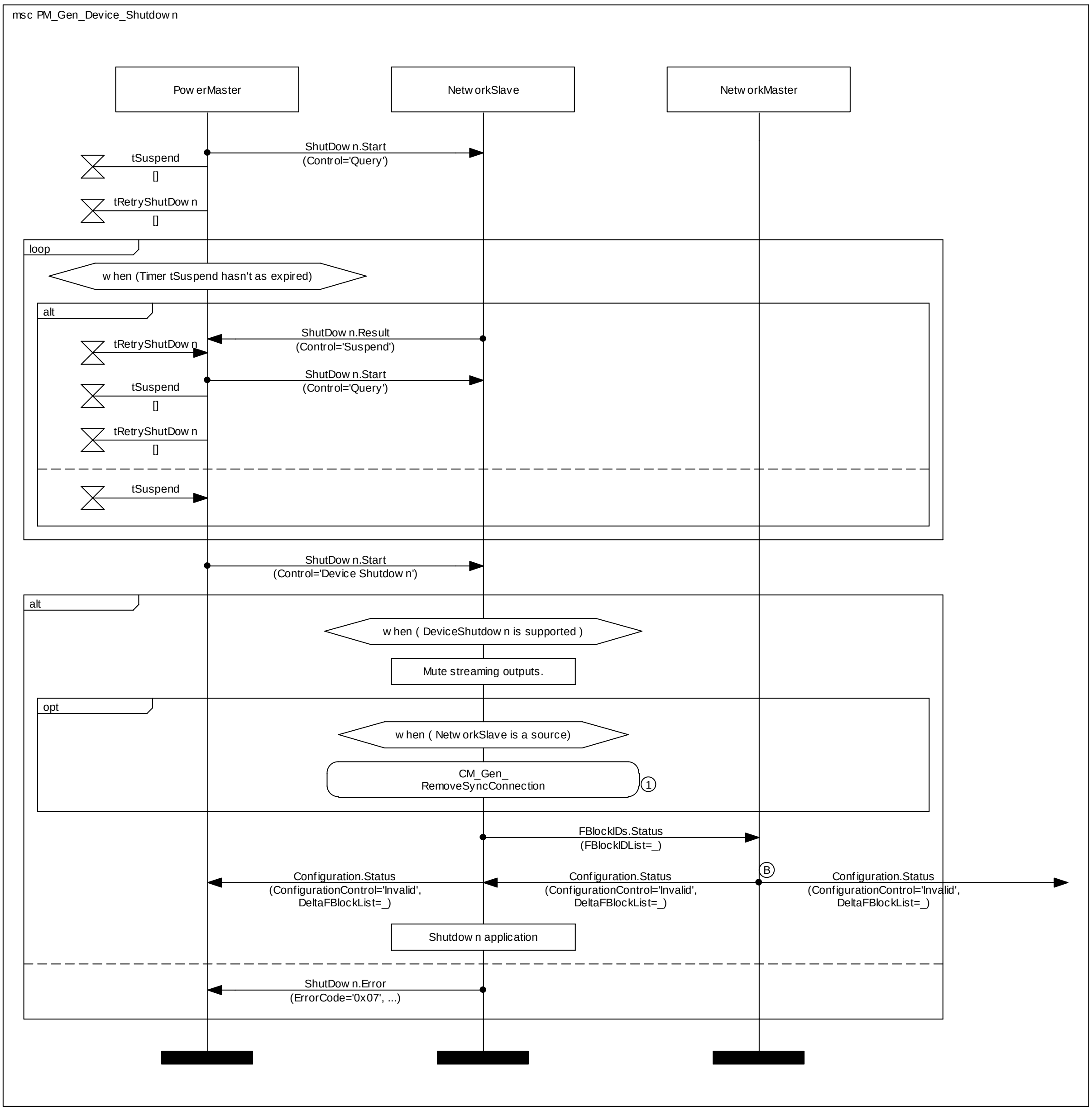


MSC 62: PM_Gen_Network_Shutdown

1. A slave device still receiving a modulated signal on its input may switch off its own modulated signal after tSlaveShutdown.

5.4 Device shutdown

General MSC:	PM_Gen_Device_Shutdown
Description:	This process handles a shutdown of a device that is initiated by the PowerMaster.
Prior Condition:	
Initiator:	PowerMaster
Communication Partners:	All NetBlocks
Events:	
Timers / Timing Constraints:	- $t_{Suspend}$ - $t_{RetryShutDown}$
Remarks:	

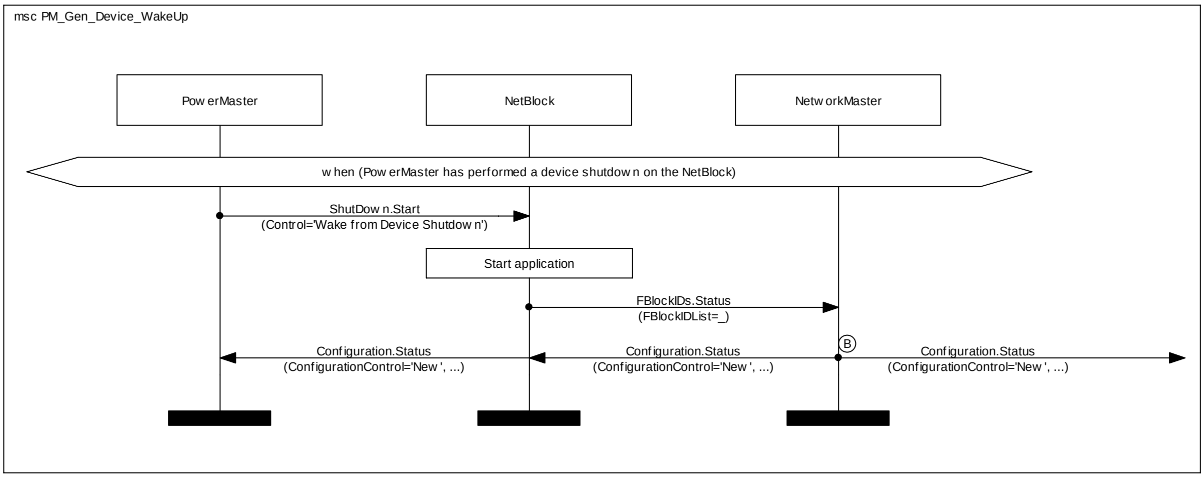

msc PM_Gen_Device_Shutdown
PowerMaster
NetworkSlave
NetworkMaster
ShutDown.Start (Control='Query')
tSuspend
[]
tRetryShutDown
[]
loop
when (Timer tSuspend hasn't as expired)
alt
ShutDown.Result (Control='Suspend')
tRetryShutDown
ShutDown.Start (Control='Query')
tSuspend
[]
tRetryShutDown
[]
tSuspend
ShutDown.Start (Control='Device Shutdown')
alt
when (DeviceShutdown is supported)
Mute streaming outputs.
opt
when (NetworkSlave is a source)
CM_Gen_ RemoveSyncConnection
1
FBlockIDs.Status (FBlockIDList=_)
B
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
Configuration.Status (ConfigurationControl='Invalid', DeltaFBlockList=_)
Shutdown application
ShutDown.Error (ErrorCode='0x07', ...)

MSC 63: PM_Gen_Device_Shutdown

1. This MSC describes how a connection is removed.

5.5 Device wake up

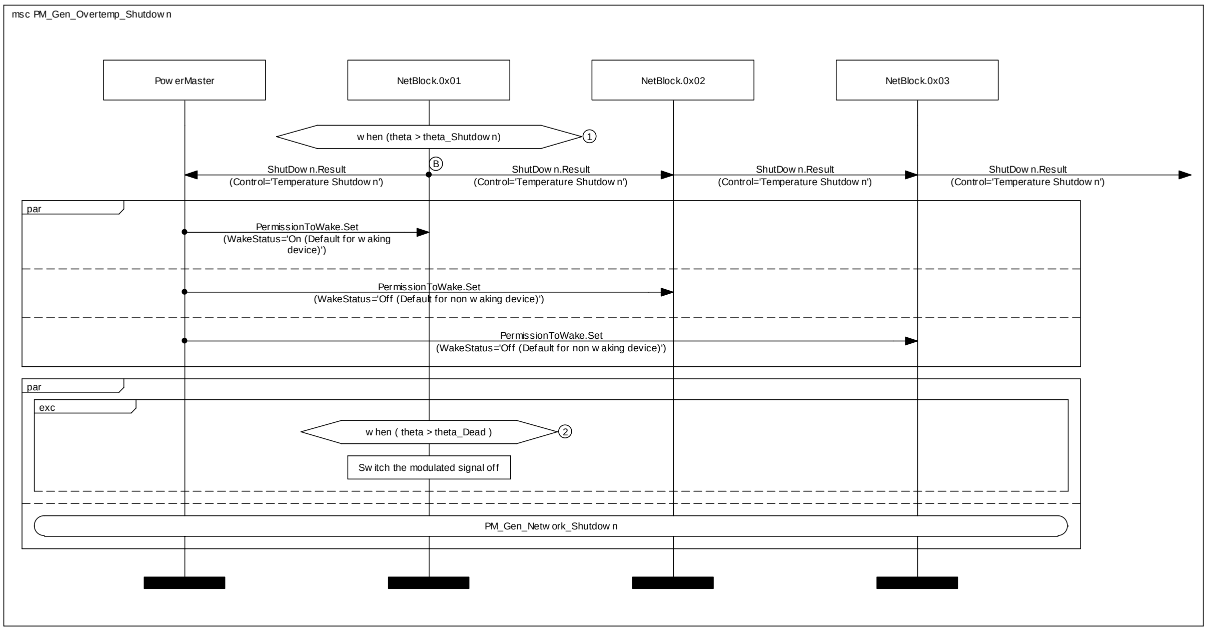
General MSC:	PM_Gen_Device_WakeUp
Description:	This process handles a wake up of a device that is initiated by the PowerMaster.
Prior Condition:	
Initiator:	PowerMaster
Communication Partners:	NetBlock in device that has been shutdown.
Events:	
Timers / Timing Constraints:	
Remarks:	


msc PM_Gen_Device_WakeUp
PowerMaster
NetBlock
NetworkMaster
when (PowerMaster has performed a device shutdown on the NetBlock)
ShutDown.Start
(Control='Wake from Device Shutdown')
Start application
FBlockIDs.Status
(FBlockIDList=_)
B
Configuration.Status
(ConfigurationControl='New', ...)
Configuration.Status
(ConfigurationControl='New', ...)
Configuration.Status
(ConfigurationControl='New', ...)

MSC 64: PM_Gen_Device_WakeUp

5.6 Network shutdown due to over temperature

General MSC:	PM_Gen_Overtemp_Shutdown
Description:	This process handles a shutdown of the network due to over temperature.
Prior Condition:	
Initiator:	NetBlock in device that is overheated.
Communication Partners:	All NetBlocks
Events:	
Timers / Timing Constraints:	
Remarks:	

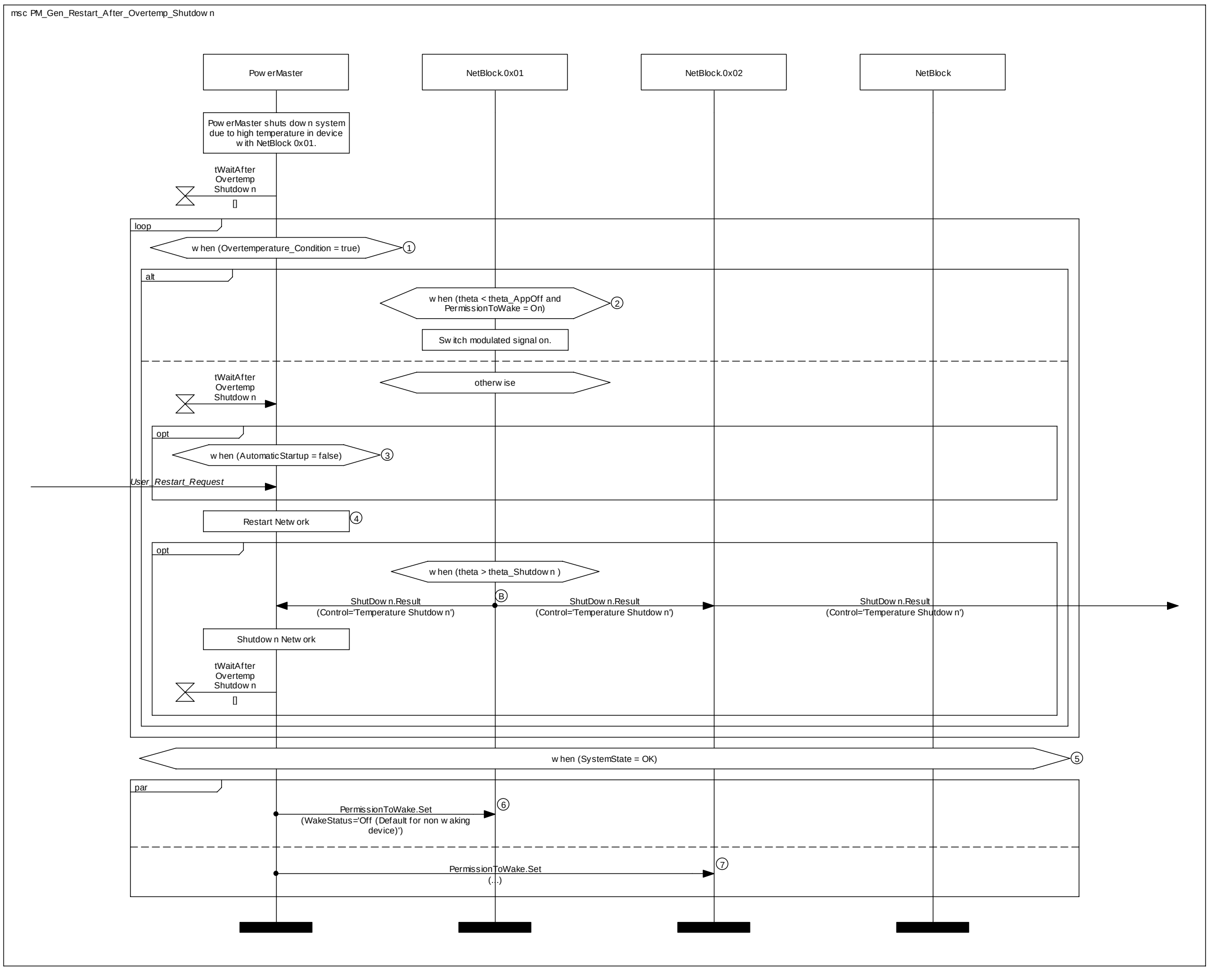

msc PM_Gen_Overtemp_Shutdow n
Pow erMaster
NetBlock.0x01
NetBlock.0x02
NetBlock.0x03
w hen (theta > theta_Shutdow n)
1
B
ShutDow n.Result
(Control='Temperature Shutdow n')
ShutDow n.Result
(Control='Temperature Shutdow n')
ShutDow n.Result
(Control='Temperature Shutdow n')
ShutDow n.Result
(Control='Temperature Shutdow n')
par
PermissionToWake.Set
(WakeStatus='On (Default for w aking device)')
PermissionToWake.Set
(WakeStatus='Off (Default for non w aking device)')
PermissionToWake.Set
(WakeStatus='Off (Default for non w aking device)')
par
exc
w hen (theta > theta_Dead)
2
Sw itch the modulated signal off
PM_Gen_Netw ork_Shutdow n

MSC 65: PM_Gen_Overtemp_Shutdown

1. Temperature near critical limit.
2. Critical temperature reached.

5.7 Network restart after over-temperature shutdown

General MSC:	PM_Gen_Restart_After_Overtemp_Shutdown
Description:	This process handles a network restart after a shutdown of the network due to over-temperature.
Prior Condition:	
Initiator:	PowerMaster
Communication Partners:	All NetBlocks
Events:	
Timers / Timing Constraints:	- $t_{WaitAfterOvertempShutdown}$
Remarks:	


msc PM_Gen_Restart_After_Overtemp_Shutdow n
Pow erMaster
NetBlock.0x01
NetBlock.0x02
NetBlock
Pow erMaster shuts dow n system due to high temperature in device w ith NetBlock 0x01.
tWaitAfter Overtemp Shutdow n
[]
loop
w hen (Overtemperature_Condition = true)
1
alt
w hen (theta < theta_AppOff and PermissionToWake = On)
2
Sw itch modulated signal on.
otherw ise
tWaitAfter Overtemp Shutdow n
opt
w hen (AutomaticStartup = false)
3
User_Restart_Request
Restart Netw ork
4
opt
w hen (theta > theta_Shutdow n)
B
ShutDow n.Result (Control='Temperature Shutdow n')
ShutDow n.Result (Control='Temperature Shutdow n')
ShutDow n.Result (Control='Temperature Shutdow n')
Shutdow n Netw ork
tWaitAfter Overtemp Shutdow n
[]
w hen (SystemState = OK)
5
par
PermissionToWake.Set (WakeStatus='Off (Default for non w aking device)')
6
PermissionToWake.Set (...)
7

MSC 66: PM_Gen_Restart_After_Overtemp_Shutdown

1. Wait for network restart attempts until System State OK is reached and therefore the overtemperature condition is over.
2. The device may wake up the network after cooling down.
3. After tWaitAfterOvertempShutdown, the PowerMaster may restart the network but is not required to do so. This may depend on applicative requests or states.
   If the PowerMaster does not restart the system automatically, a user request may trigger the restart.
4. This action triggers the NetworkMaster to scan the system.

5. A successful network scan was performed by the NetworkMaster.
6. Resets the PermissionToWake property to original state.
7. Resets the PermissionToWake property to original state.

6 Generic Management of Audio (Synchronous data)

6.1 Introduction

Audio or any Synchronous Management could be described being very complex. This has also been experienced by many carmakers. Therefore, a concept of Audio Management is frequently used.
Here we show how introducing this concept as a part of the MOST Specification would simplify the management of Synchronous connections. The concept is equally valid for Video Management or any other Synchronous Management.

6.2 Example Architecture of Synchronous Management

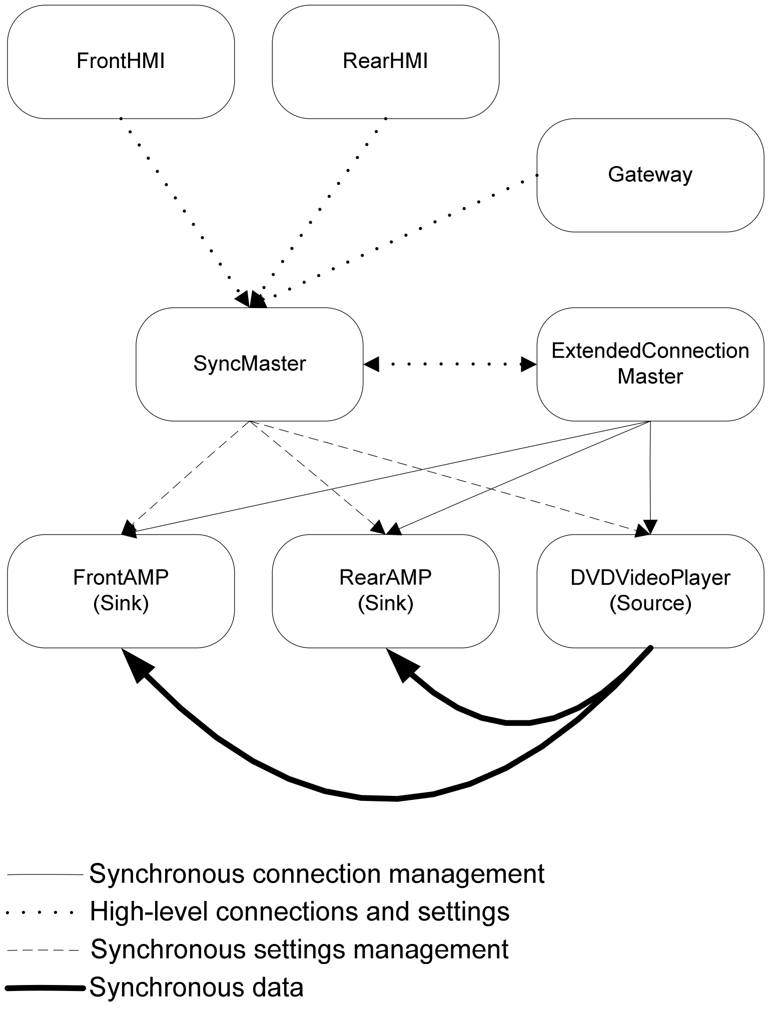

FrontHMI
RearHMI
Gateway
SyncMaster
ExtendedConnection Master
FrontAMP (Sink)
RearAMP (Sink)
DVDVideoPlayer (Source)
Synchronous connection management
High-level connections and settings
Synchronous settings management
Synchronous data

Figure 6-1: Usage of a synchronous master in an advanced system.

The system in figure 6-1 is an example where several human interfaces can use the same synchronous sources and sinks and also other devices that may have a need to interact. The gateway in the example can be a simple node that only sends status notifications to the SyncMaster that, for example, a door is open, which results in some warning signal.

6.3 Logical Model of Streaming Management

Definition 1: Audio Management controls audio properties for one or more amplifiers in the system.

Definition 2: Video Management controls image properties for one or more videos and cameras.

These definitions apply to logical entities that could be implemented in many different ways.

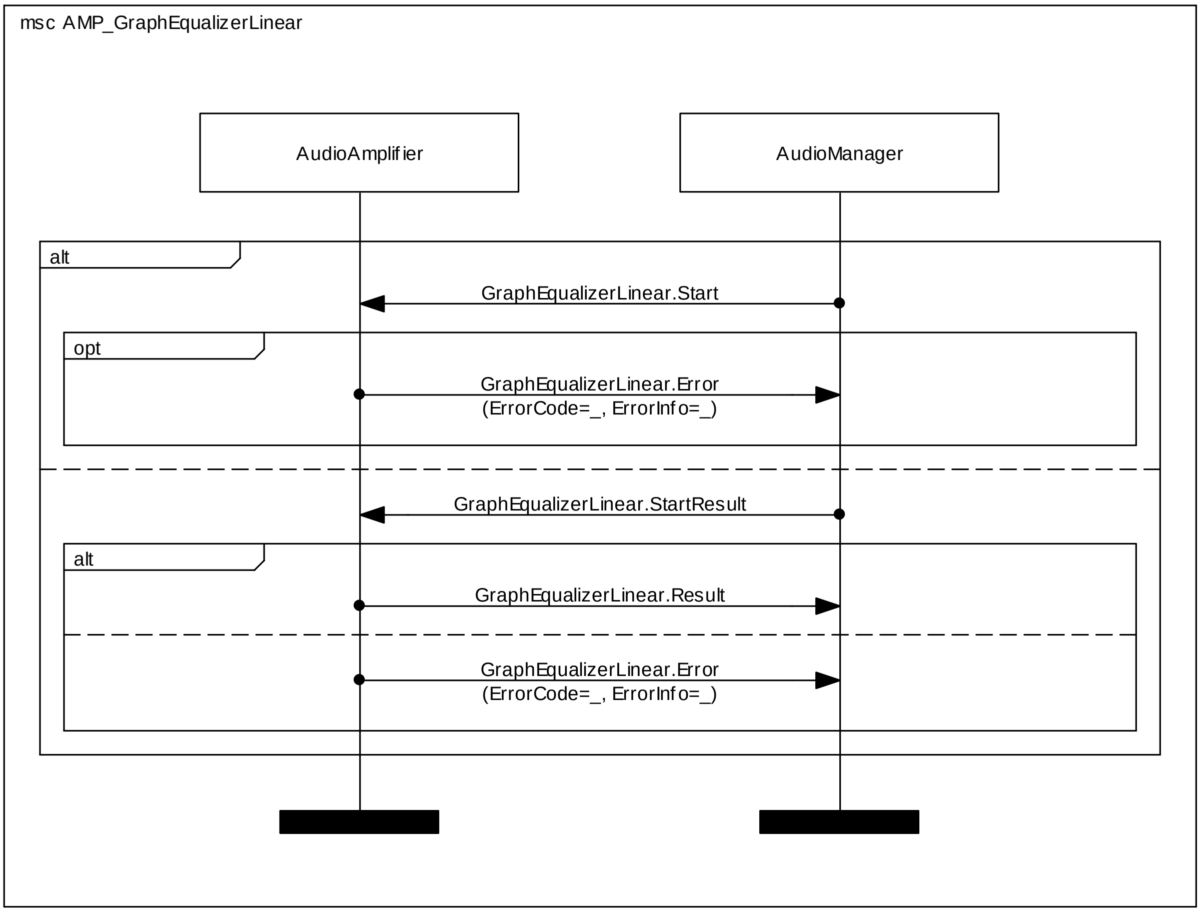
7 Function Blocks

7.1 AudioAmplifier

The functions presented below are exemplary. Hence, the MSCs can be generalized when using similar functions (e.g., replace Balance with Volume).

7.1.1 Methods

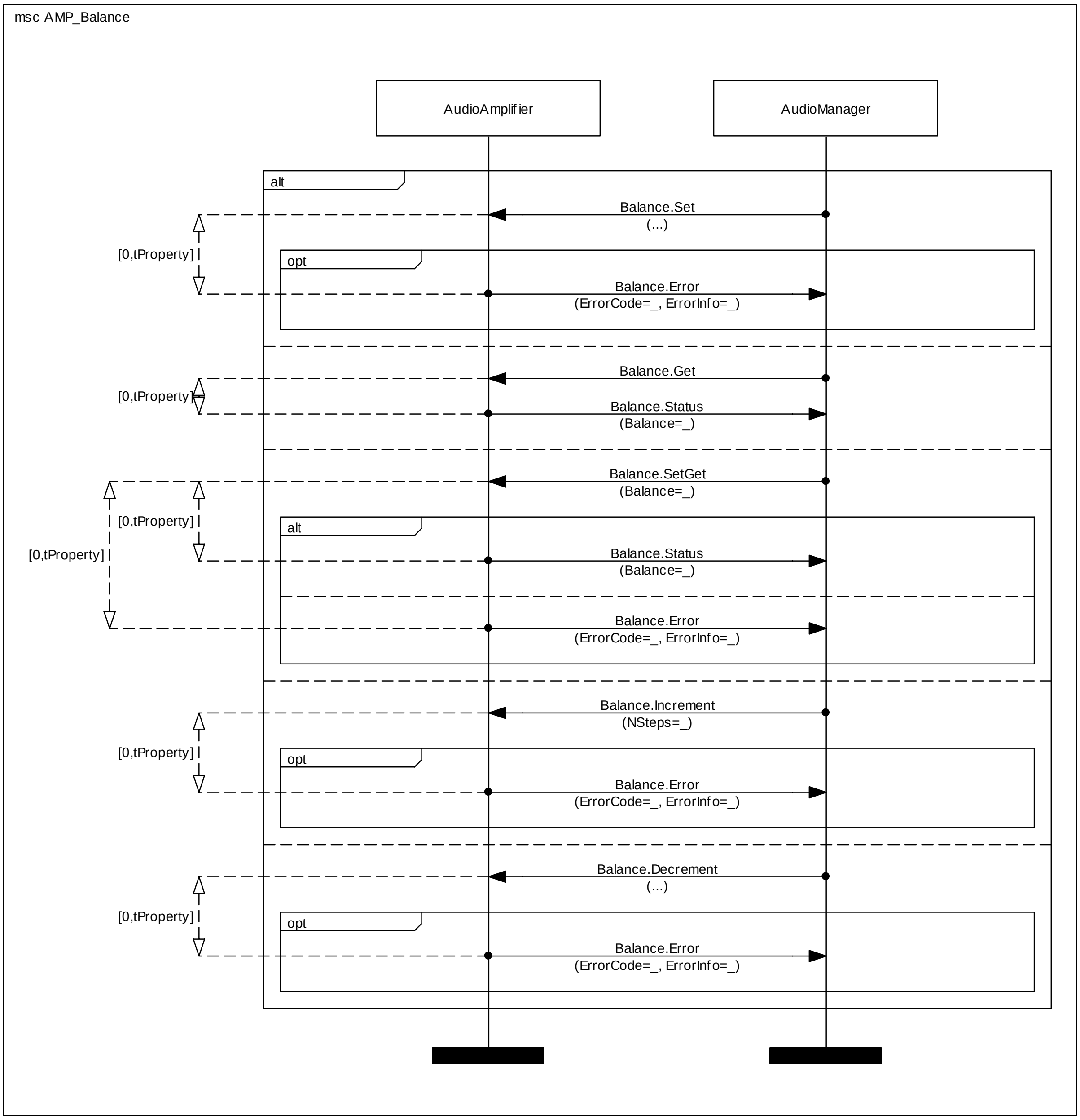
Scenario MSC:	AMP_Sc_GraphEqualizerLinear
Description:	Using the AudioAmplifier method GraphEqualizerLinear.
Prior Condition:	-
Initiator:	AudioManager
Communication Partners:	AudioManager
Events	-
Timers/Timing constraints	-
Remarks:	General for all methods. Processing is run according to what is specified for each function and any result or error should be given within time limits, if specified.


msc AMP_GraphEqualizerLinear
AudioAmplifier
AudioManager
alt
GraphEqualizerLinear.Start
opt
GraphEqualizerLinear.Error
(ErrorCode=_, ErrorInfo=_)
GraphEqualizerLinear.StartResult
alt
GraphEqualizerLinear.Result
GraphEqualizerLinear.Error
(ErrorCode=_, ErrorInfo=_)

MSC 67: AMP_Gen_GraphEqualizerLinear

7.1.2 Balance

Scenario MSC:	AMP_Sc_Balance
Description:	Using the function “Balance” or a general view of a property.
Prior Condition:	-
Initiator:	AudioManager
Communication Partners:	AudioManager
Events	-
Timers/Timing constraints	$t_{Propery}$
Remarks:	General for all properties.

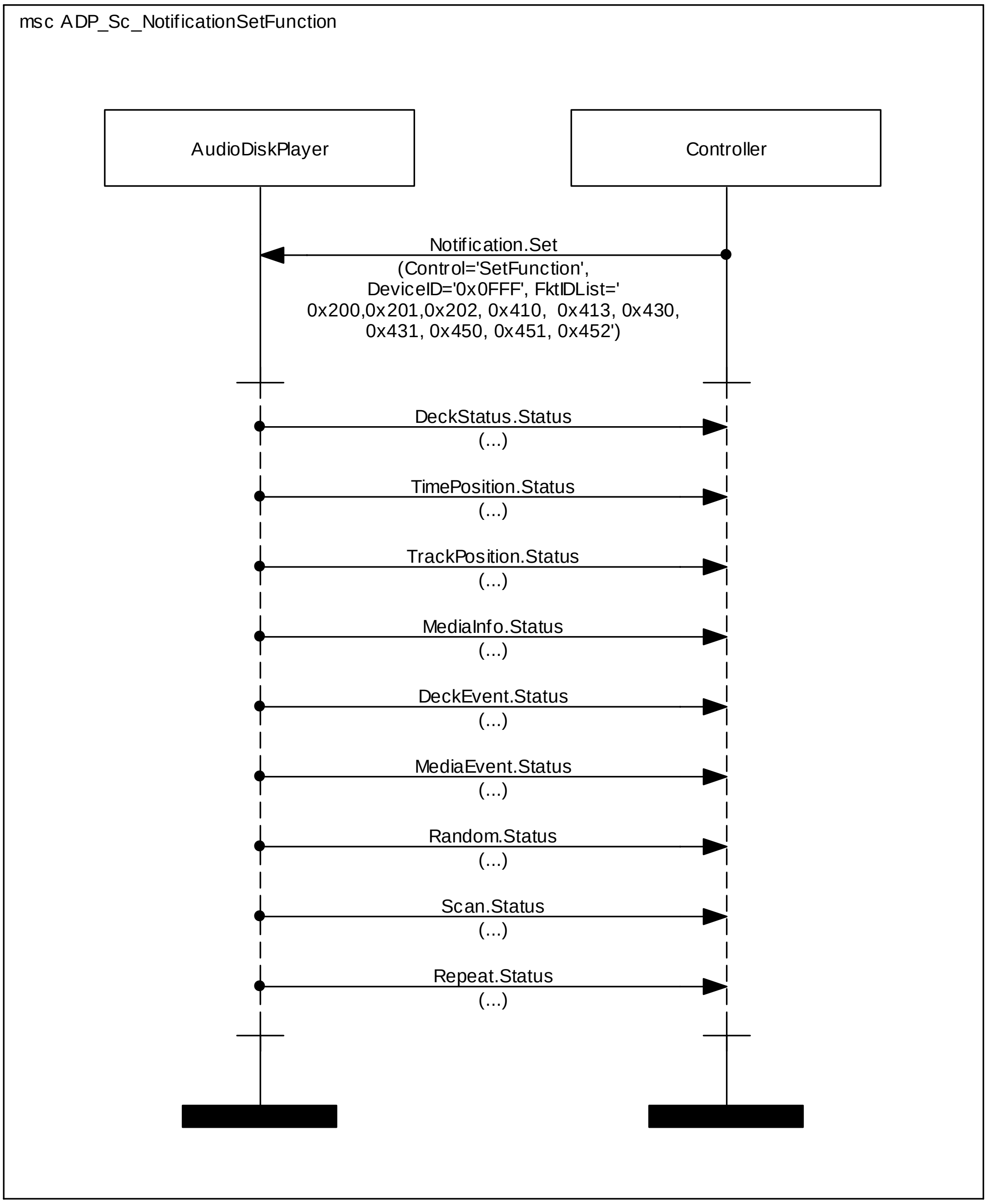


MSC 68: AMP_Gen_Balance

7.2 AudioDiskPlayer

7.2.1 Notification

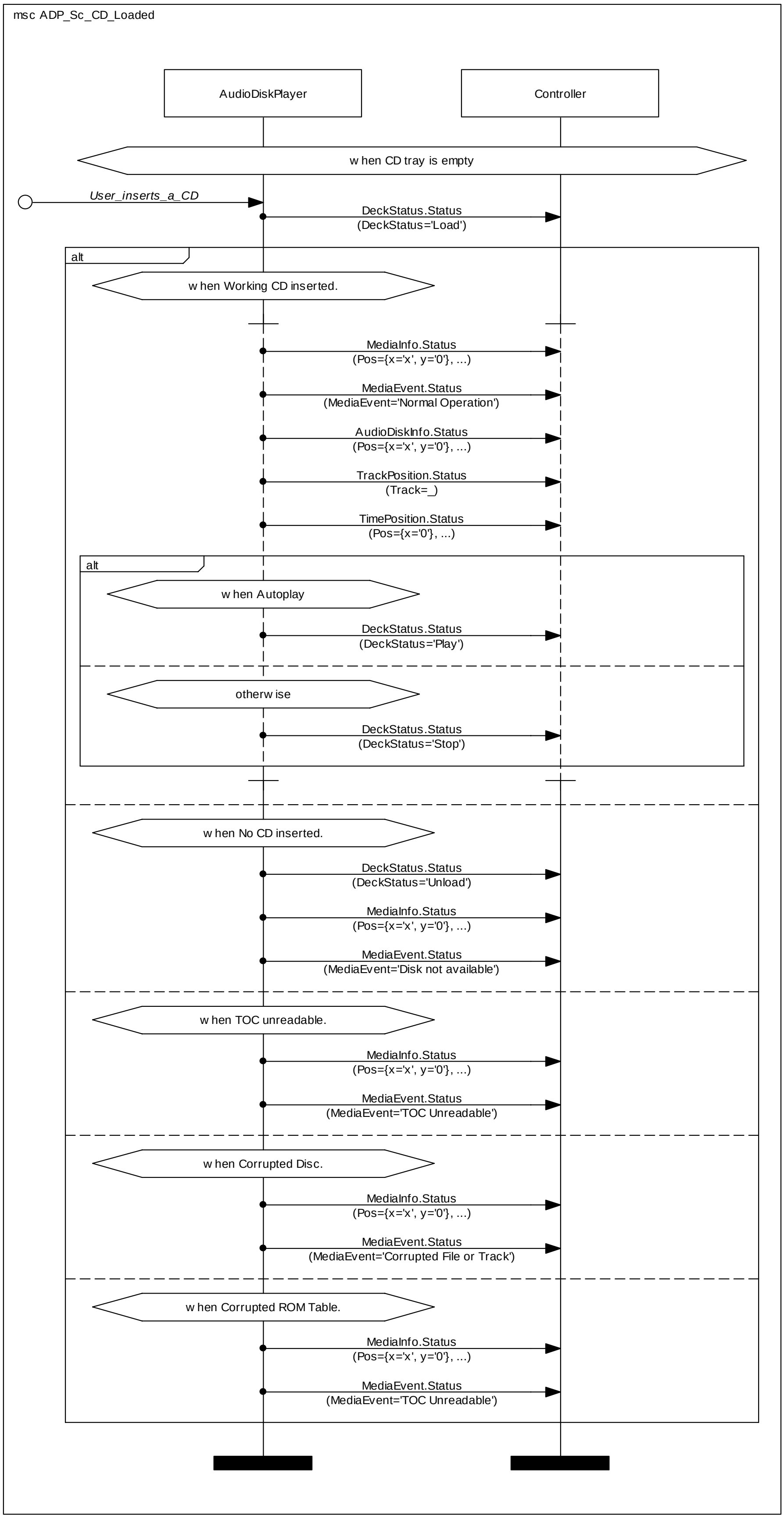
Scenario MSC:	ADP_Sc_NotificationSetFunction
Description:	The controller signs up for notification on different properties. It is up to the system integrator to decide which properties will be signed up for notification.
Prior Condition:	NetworkMaster has issued Configuration.Status(OK).
Initiator:	
Communication Partners:	Controlling device
Events	
Remarks:	


msc ADP_Sc_NotificationSetFunction
AudioDiskPlayer
Controller
Notification.Set (Control='SetFunction', DeviceID='0x0FFF', FktIDList=' 0x200,0x201,0x202, 0x410, 0x413, 0x430, 0x431, 0x450, 0x451, 0x452')
DeckStatus.Status (...)
TimePosition.Status (...)
TrackPosition.Status (...)
MediaInfo.Status (...)
DeckEvent.Status (...)
MediaEvent.Status (...)
Random.Status (...)
Scan.Status (...)
Repeat.Status (...)

MSC 69: ADP_Sc_NotificationSetFunction

7.2.2 Loading a CD

Scenario MSC:	ADP_Sc_CD_Loaded
Description:	The user inserts a CD.
Prior Condition:	The CD tray is empty.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	This use case assumes that autoplay is to be used when a playable CD is found.

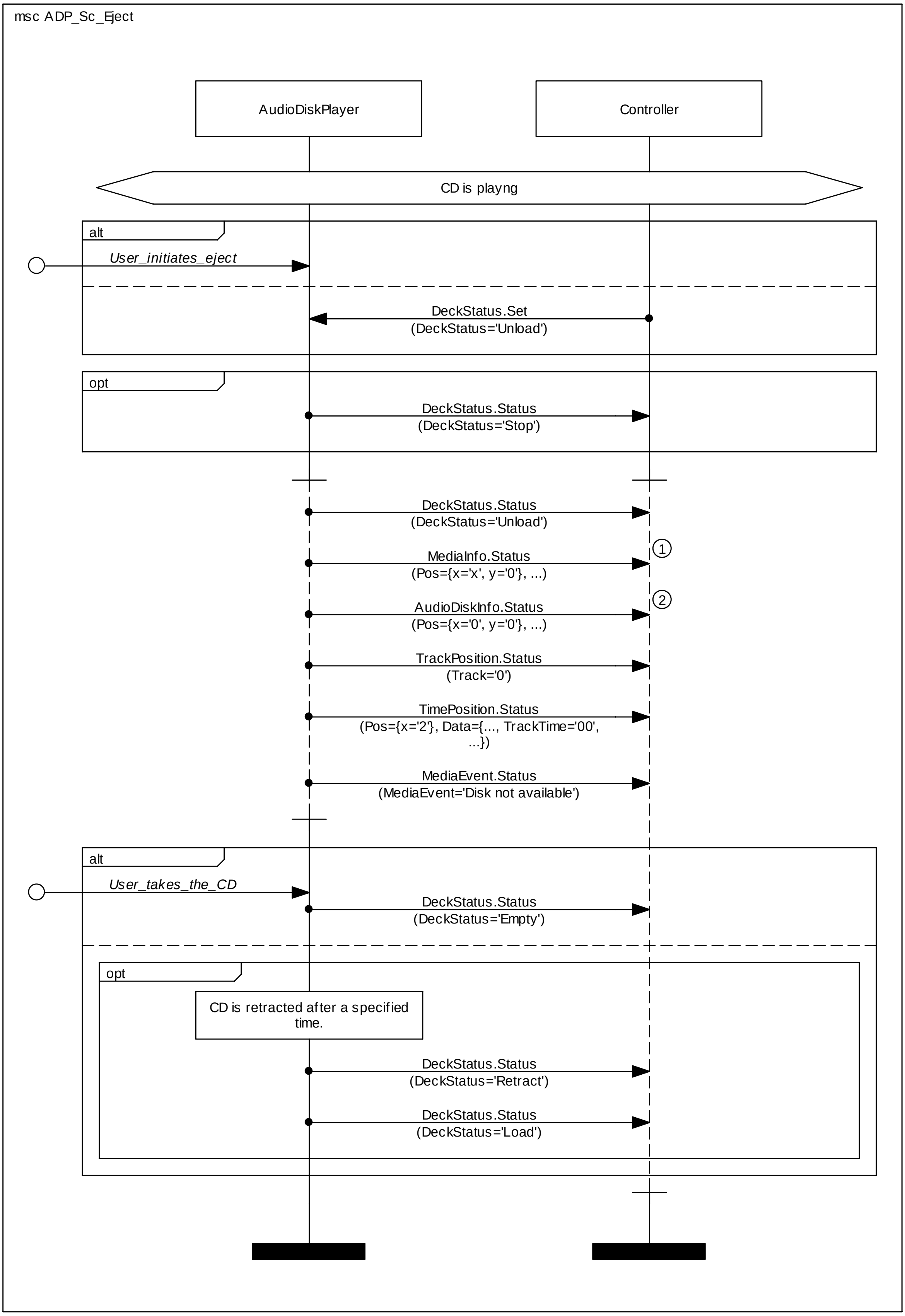


MSC 70: ADP_Sc_CD_Loaded

1. Autoplay is used and MMI is switched to CD on valid discs.

7.2.3 Ejecting a Disc

Scenario MSC:	ADP_Sc_Eject
Description:	The CD is ejected.
Prior Condition:	CD is playing. Notification registered as in 6.2.1.
Initiator:	User or controller
Communication Partners:	Controlling device
Events	
Remarks:	

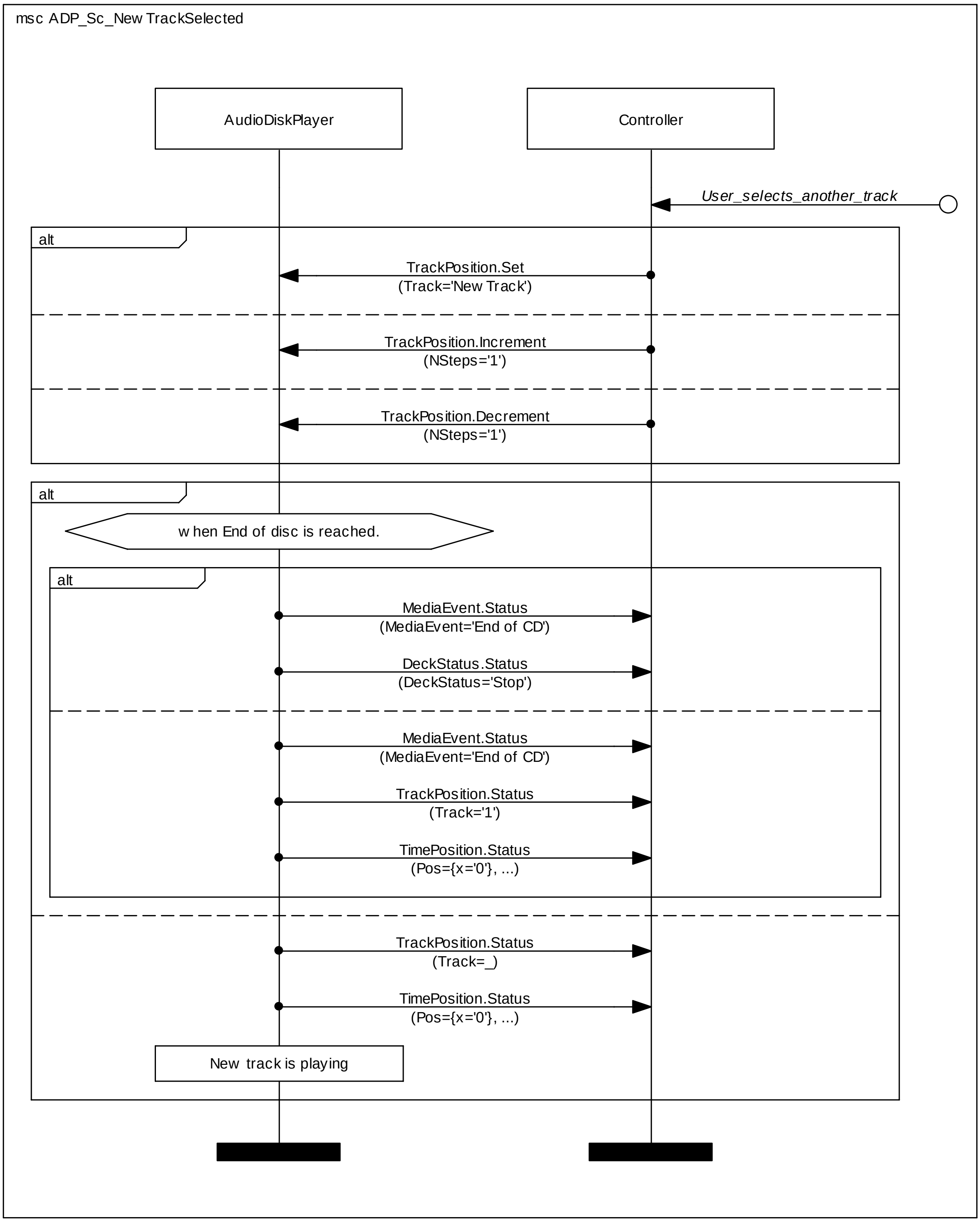


MSC 71: ADP_Sc_Eject

1. MediaInfo for a Single CD Player has a constant default value: Data is empty.
2. Data is empty.

7.2.4 New Track Selected

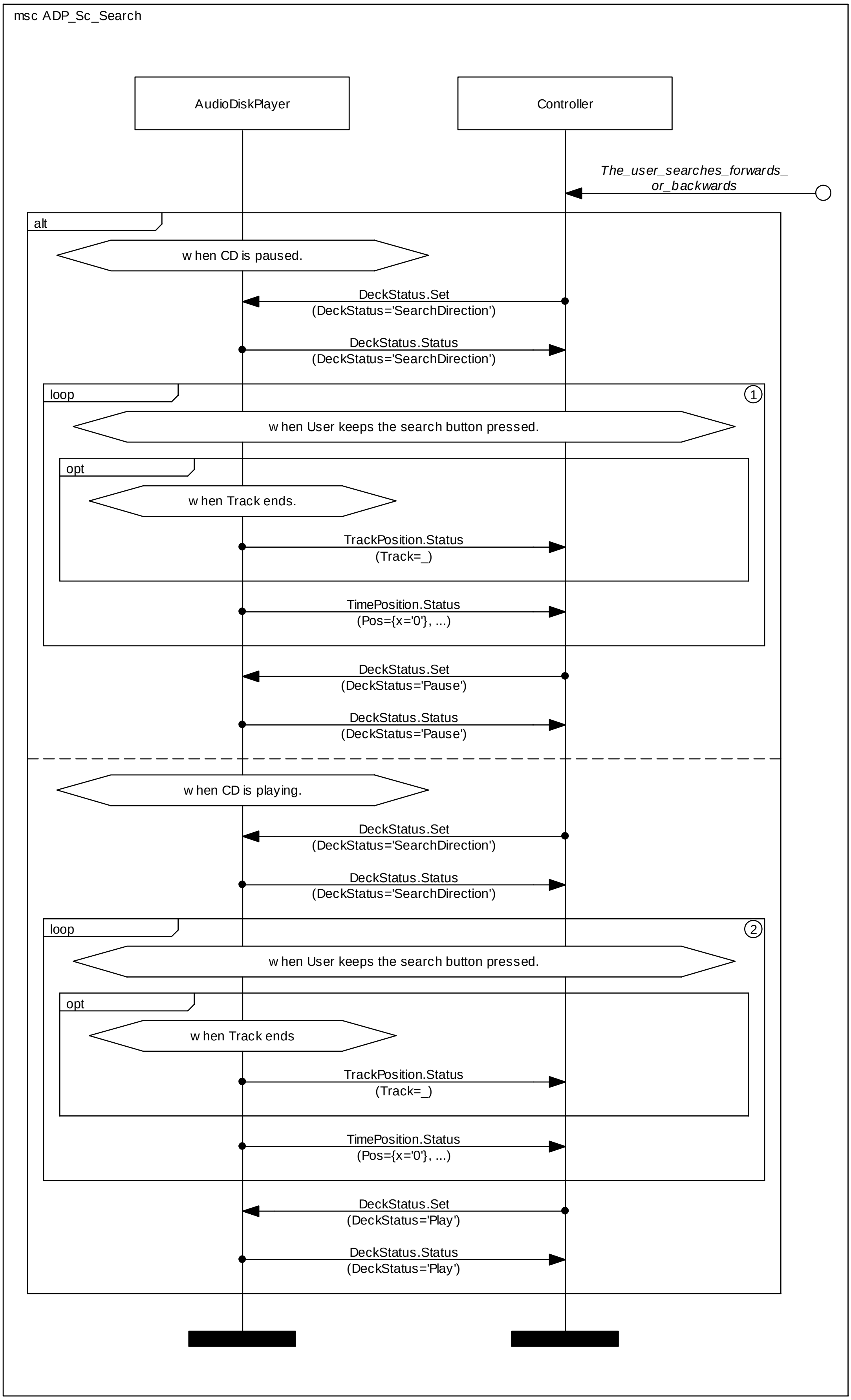
Scenario MSC:	ADP_Sc_NewTrackSelected
Description:	The user selects another track.
Prior Condition:	CD is playing.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	



MSC 72: ADP_Sc_NewTrackSelected

7.2.5 Searching

Scenario MSC:	ADP_Sc_Search
Description:	The user searches forwards or backwards.
Prior Condition:	
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	

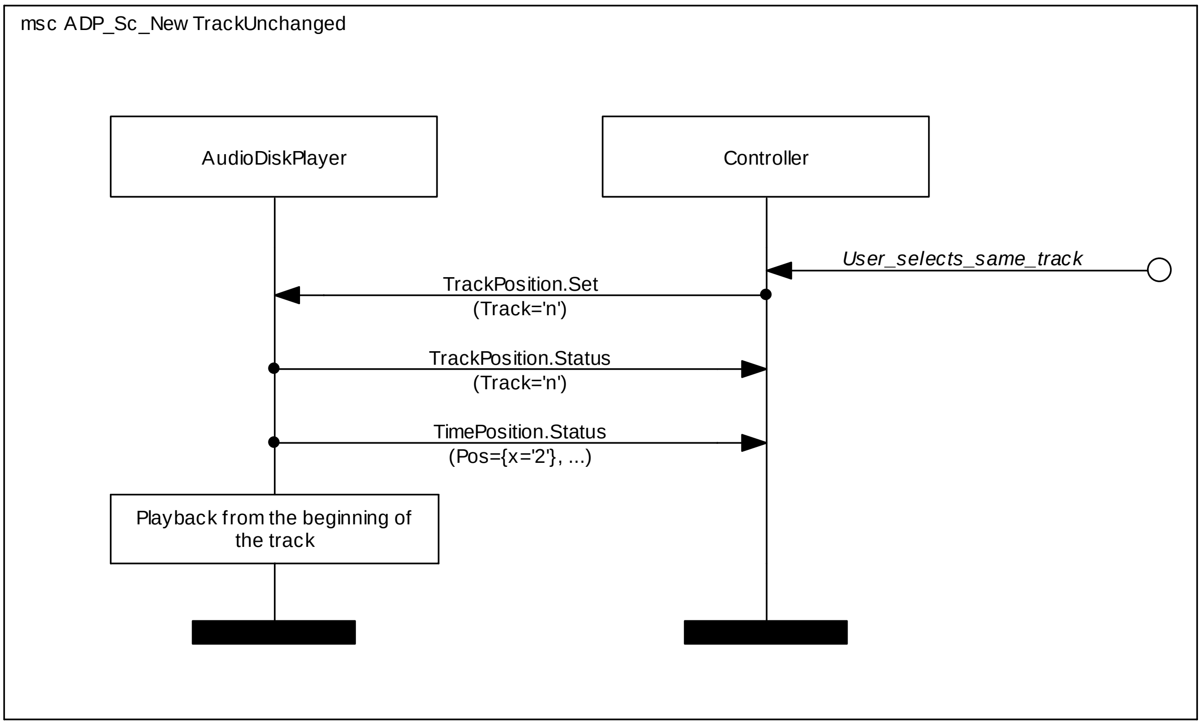


MSC 73: ADP_Sc_Search

1. Triggered for every second that is passed while searching.
2. Triggered for every second that is passed while searching.

7.2.6 New Track Unchanged

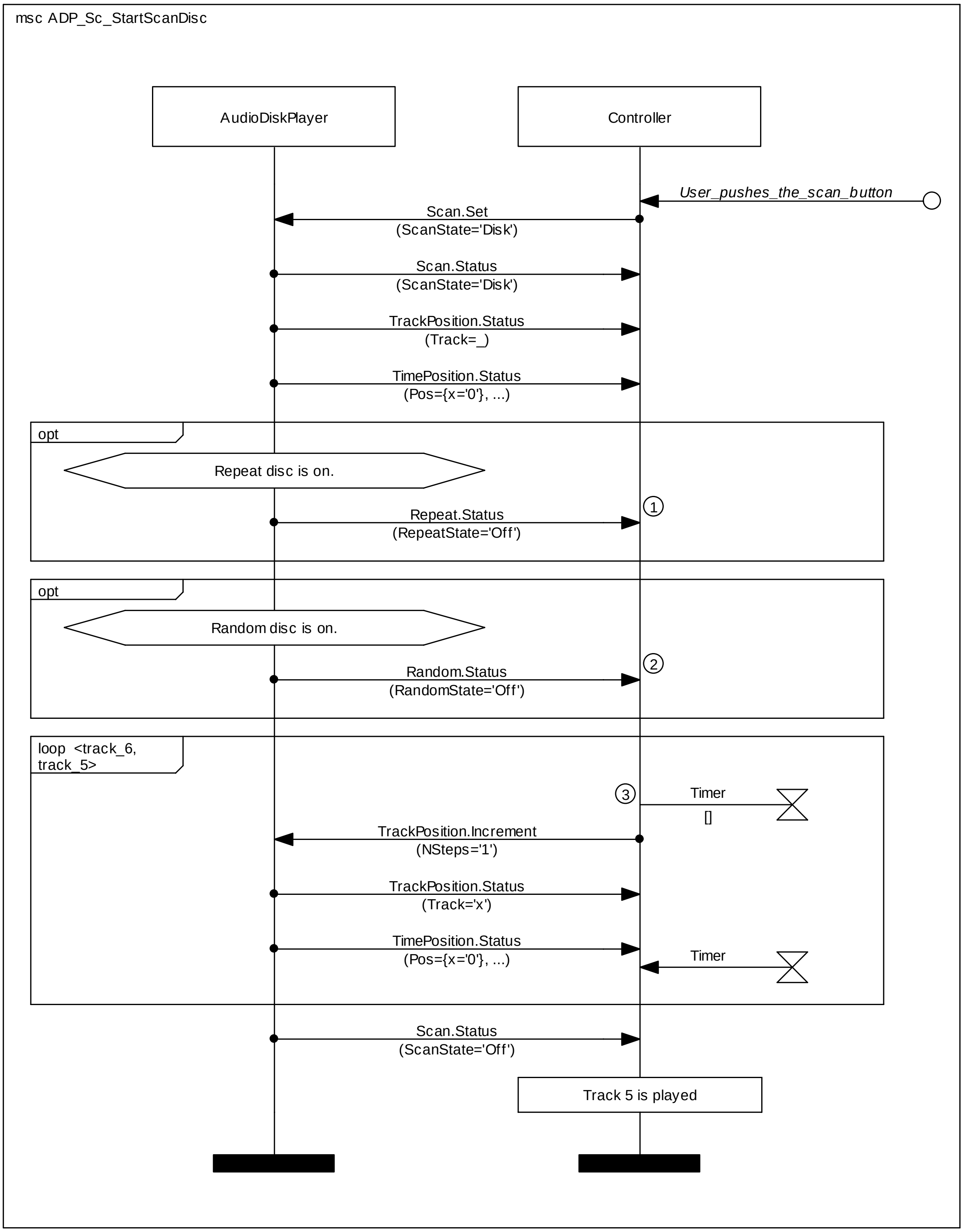
Scenario MSC:	ADP_Sc_NewTrackUnchanged
Description:	The user selects the same track that is playing.
Prior Condition:	Track *n* is playing.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	



MSC 74: ADP_Sc_NewTrackUnchanged

7.2.7 Start Scan Disc

Scenario MSC:	ADP_Sc_StartScanDisc
Description:	The user presses Scan and scans the disc.
Prior Condition:	Scan is off and track number 5 is playing.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	

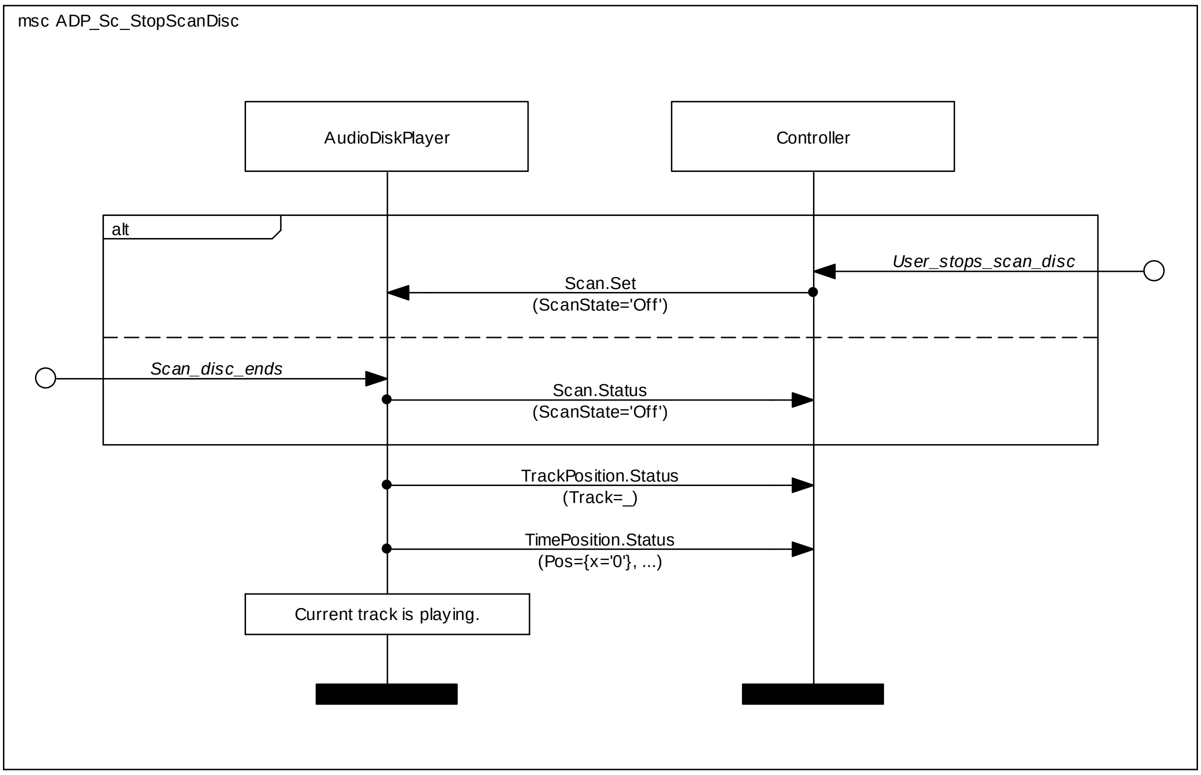


MSC 75: ADP_Sc_StartScanDisc

1. If Repeat is on, it will be turned off.
2. If Random is on, it will be turned off.
3. Timer = each track will be played for a defined time; usually 10 seconds.

7.2.8 Stop Scan Disc

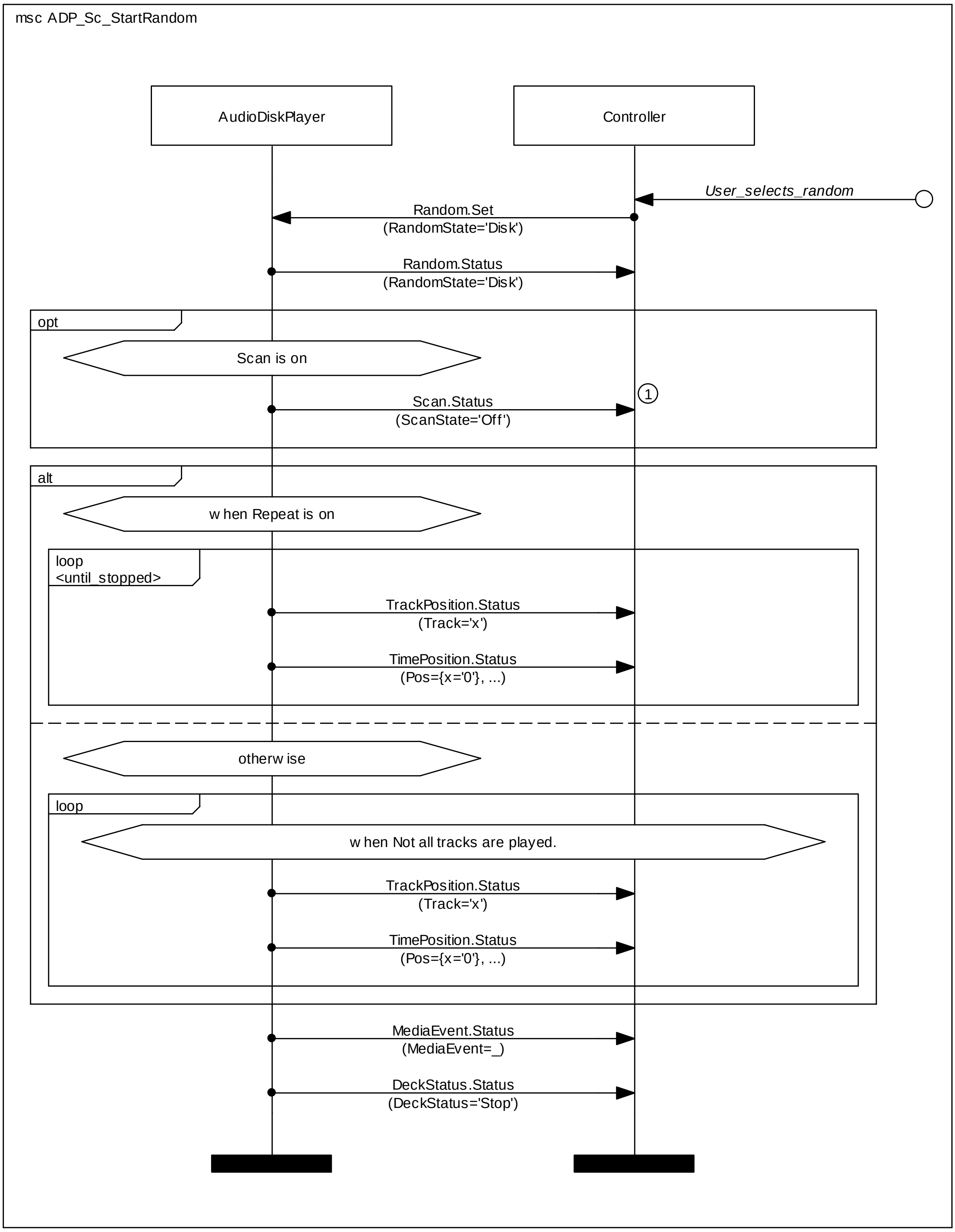
Scenario MSC:	ADP_Sc_StopScanDisc
Description:	The user presses Scan and scans the disc.
Prior Condition:	Scan is off and track number 5 is playing.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	



MSC 76: ADP_Sc_StartScanDisc

7.2.9 Start Random

Scenario MSC:	ADP_Sc_StartRandom
Description:	The user selects random playback.
Prior Condition:	Random is off and a track is being played.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	

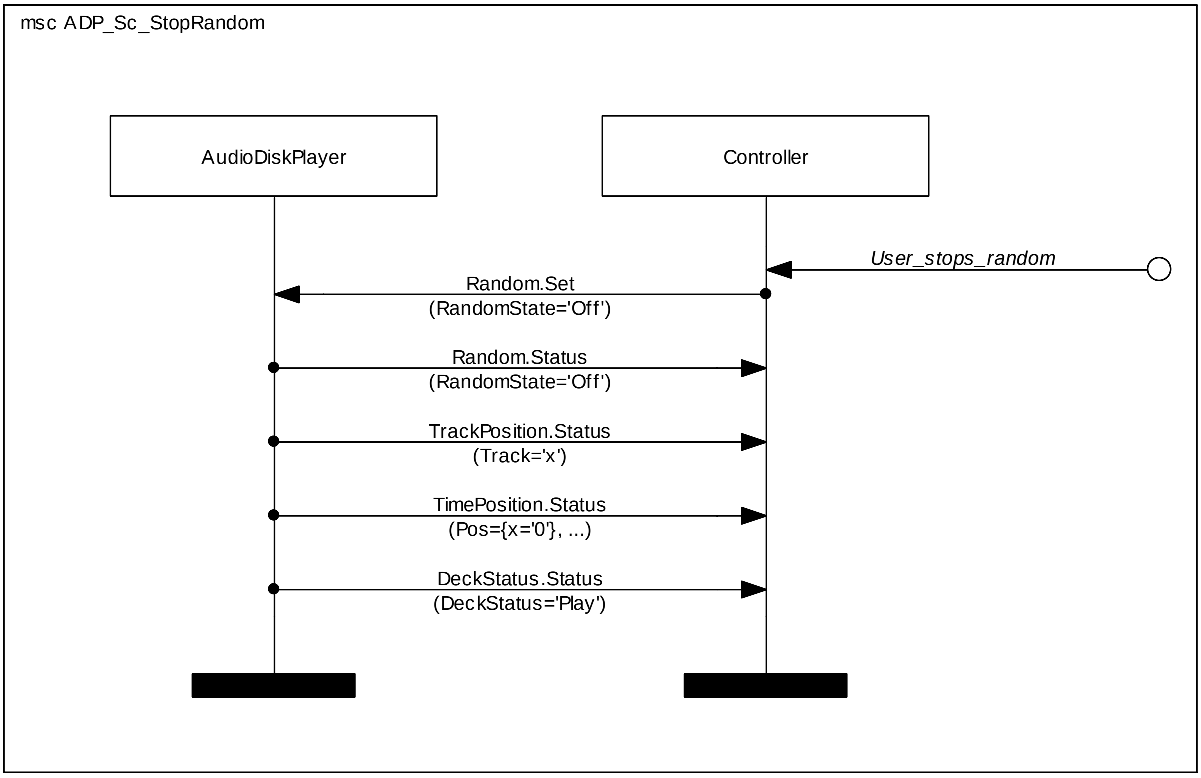


MSC 77: ADP_Sc_StartRandom

1. If Scan is on, it will be turned off.

7.2.10 Stop Random

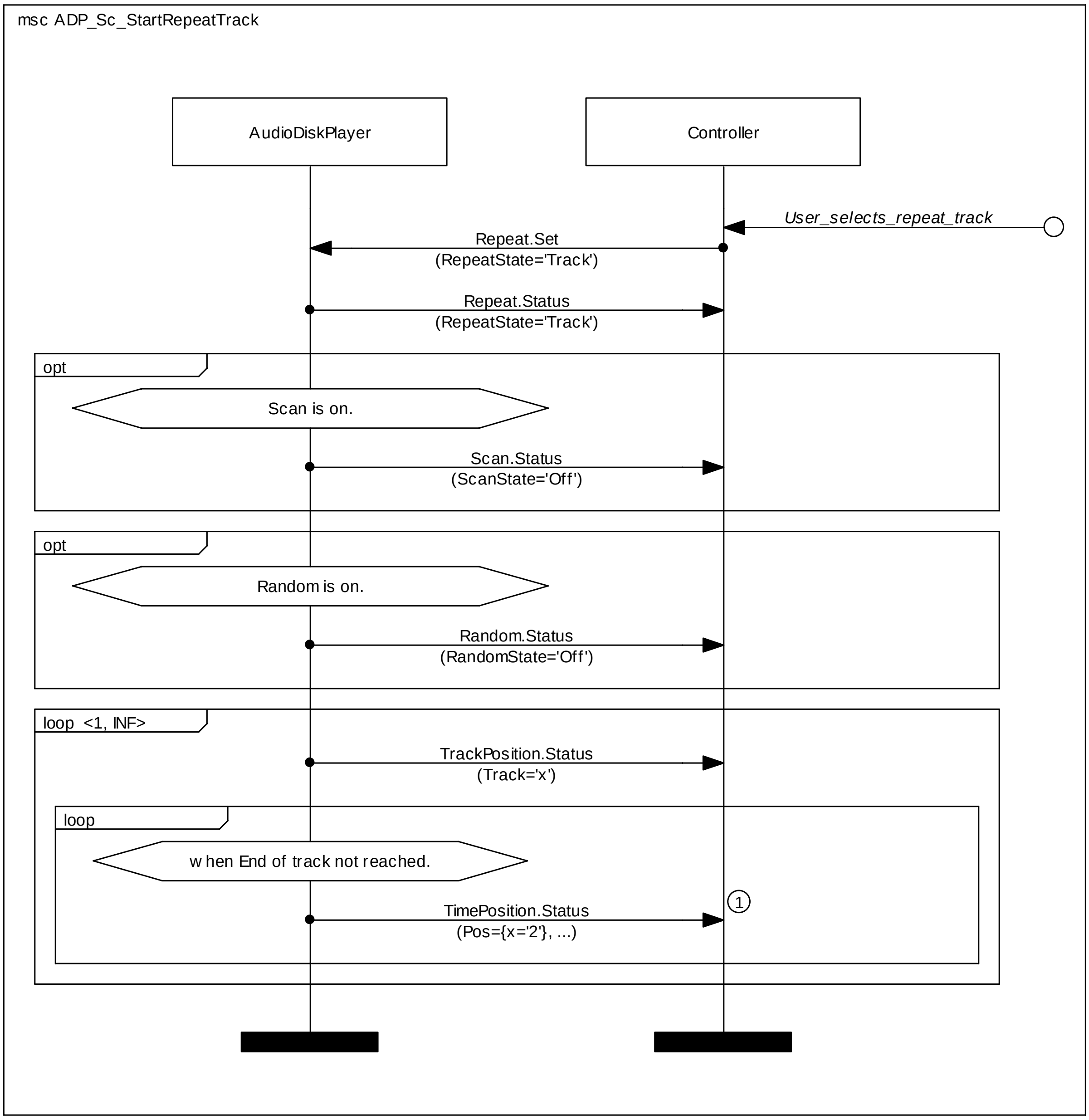
Scenario MSC:	ADP_Sc_StopRandom
Description:	Random playback is stopped. The player assumes normal playback again.
Prior Condition:	Random is on.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	



MSC 78: ADP_Sc_StopRandom

7.2.11 Start Repeat Track

Scenario MSC:	ADP_Sc_StartRepeatTrack
Description:	User selects repeat single track.
Prior Condition:	Repeat track is off and a track is being played.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	

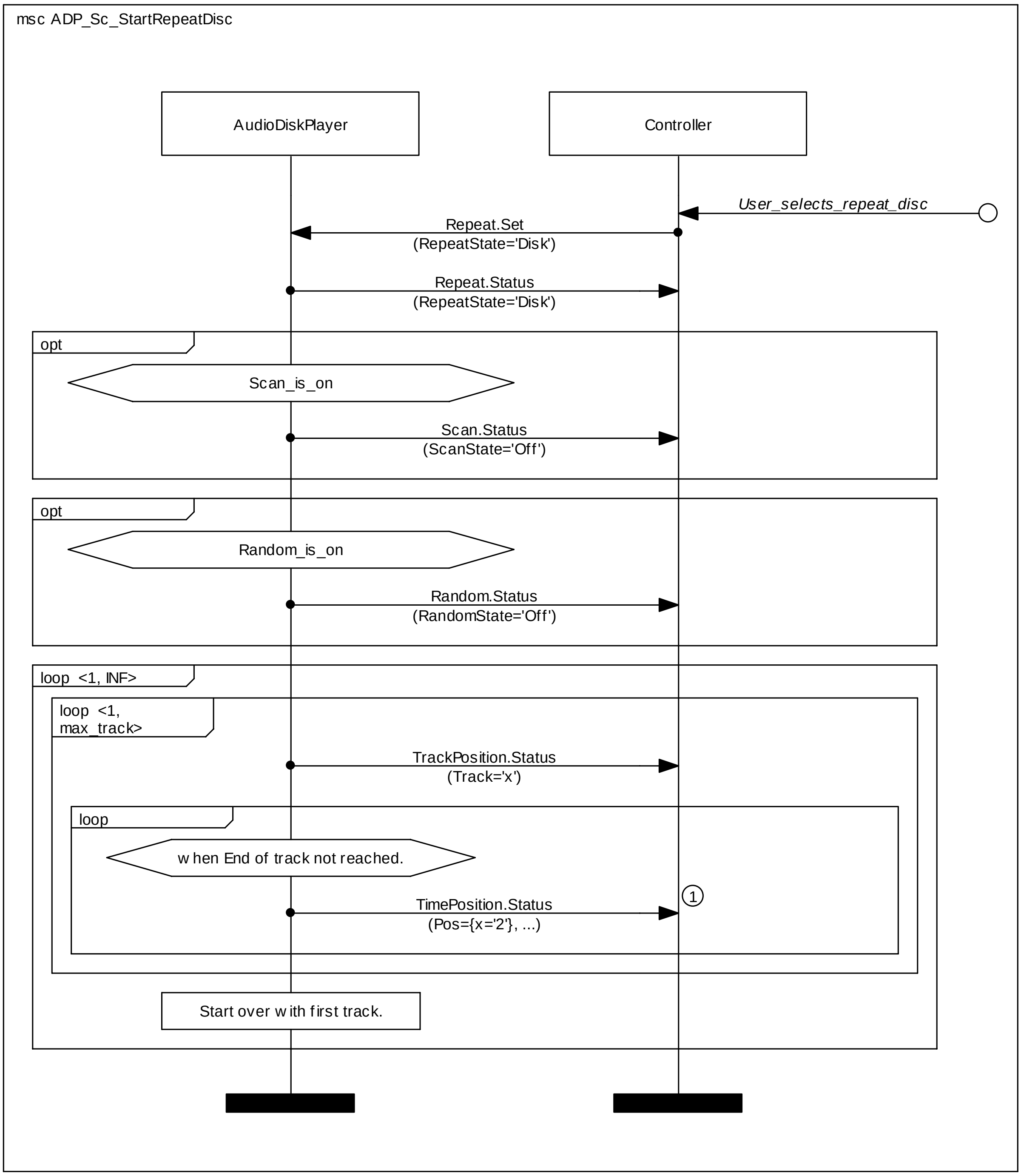

msc ADP_Sc_StartRepeatTrack
AudioDiskPlayer
Controller
User_selects_repeat_track
Repeat.Set (RepeatState='Track')
Repeat.Status (RepeatState='Track')
opt
Scan is on.
Scan.Status (ScanState='Off')
opt
Random is on.
Random.Status (RandomState='Off')
loop <1, INF>
TrackPosition.Status (Track='x')
loop
w hen End of track not reached.
1
TimePosition.Status (Pos={x='2'}, ...)

MSC 79: ADP_Sc_StartRepeatTrack

1. Message is sent every second.

7.2.12 Start Repeat Disc

Scenario MSC:	ADP_Sc_StartRepeatDisc
Description:	User selects repeat disc.
Prior Condition:	Repeat disc is off and a track is being played.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	

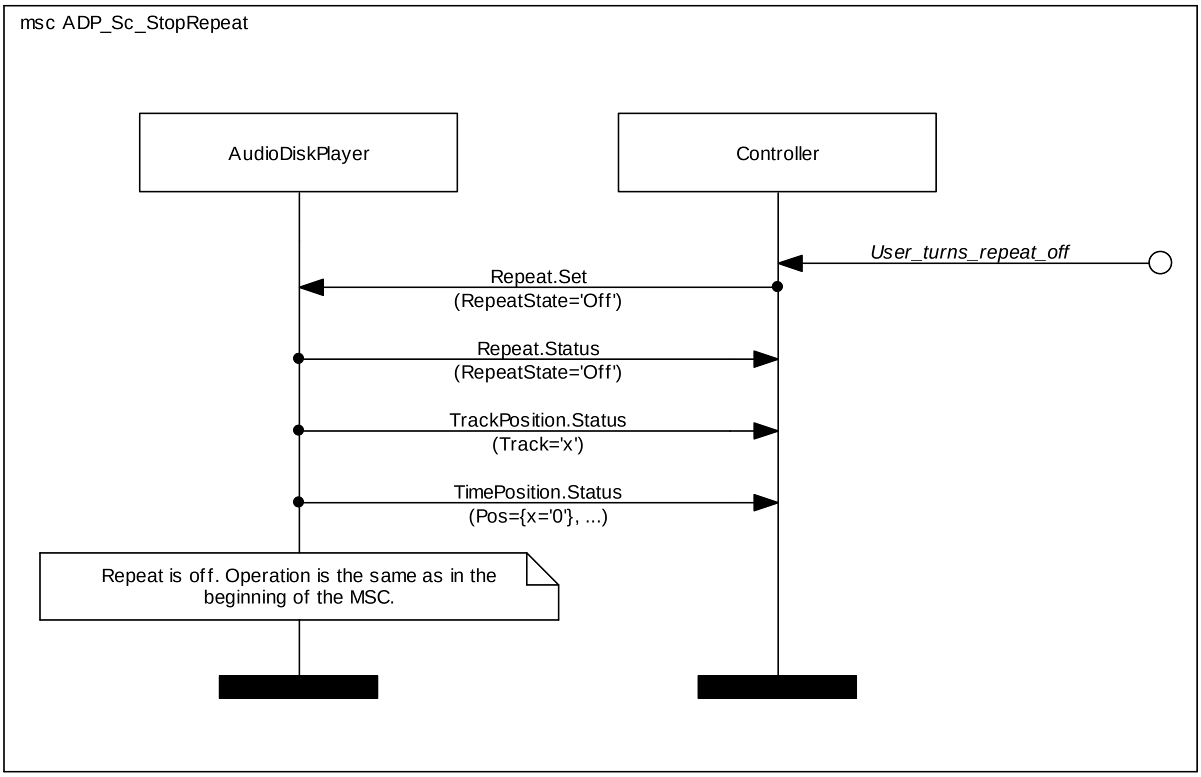


MSC 80: ADP_Sc_StartRepeatDisc

1. Message is sent every second.

7.2.13 Stop Repeat

Scenario MSC:	ADP_Sc_StopRepeat
Description:	User turns repetition off.
Prior Condition:	Repeat disc or track is on.
Initiator:	User
Communication Partners:	Controlling device
Events	
Remarks:	



MSC 81: ADP_Sc_StopRepeat

8 Timers

The definition of timers can be found in the corresponding MOST Specification.

9 Naming Conventions

The names of the MSCs categorize them into different sections. Every name has a prefix that differentiates it from MSCs dealing with other topics. Two parts make up the prefix. The first part consists of an abbreviation or a characteristic name of the topic that the MSC focuses on. The following names exist:

Topic Prefix	Contents of the MSC
ADP	AudioDiskPlayer MSC
AMP	AudioAmplifier MSC
GSI	GeneralSink MSC
GSO	GeneralSource MSC
CM	Connection Management MSC
NM	NetworkMaster MSC
NS	Network Slave MSC

The second part of the prefix is used to differentiate between General MSCs and Scenario MSCs. _Gen_ stands for general and _Sc_ for Scenario.

An example is ADP_Sc_CD_Loaded, which has ADP_Sc as a prefix. This means that the MSC is an AudioDiskPlayer Scenario MSC. (It shows what happens when a CD is loaded).

9.1 Connection Management Naming Conventions

In addition to the above named prefixes of the MSC name, the Connection Management section also uses _SC_ to denote that the MSC is only relevant in an all SourceConnect network. _M_ is used to denote that this MSC is only used in Mixed System networks (networks that use the Timing Master's allocation table). MSC names that have neither _SC_ nor _M_ is used in both types of network.

An example is CM_Gen_M_BuildSyncConnection, which is a Connection Management general MSC. It shows how the Connection Management builds a synchronous connection in a mixed system.

Warning: There are scenarios MSCs that deal only with SourceConnect networks giving them the name CM_Sc_SC_....

10 Appendix A: Index of Figures

11 Appendix B: Index of Tables

12 Appendix C: Index of MSCs

Notes:

Notes: