



Elektrobit

EB tresos[®] AutoCore Generic 8 Memory Stack documentation

product release 8.8.3



Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

Technical support

<https://www.elektrobit.com/support>

Legal disclaimer

Confidential information.

ALL RIGHTS RESERVED. No part of this publication may be copied in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of Elektrobit Automotive GmbH.

All brand names, trademarks, and registered trademarks are property of their rightful owners and are used only for description.

Copyright 2021, Elektrobit Automotive GmbH.

Table of Contents

1. Overview of EB tresos AutoCore Generic 8 Memory Stack documentation	21
2. Supported features	22
2.1. Overview	22
2.2. Supported Crc features	22
2.3. Supported Ea features	22
2.4. Supported Fee features	22
2.5. Supported MemIf features	23
2.6. Supported NvM features	23
3. ACG8 Memory Stack release notes	28
3.1. Overview	28
3.2. Scope of the release	28
3.2.1. Configuration tool	28
3.2.2. AUTOSAR modules	28
3.2.3. EB (Elektrobit) modules	29
3.2.4. MCAL modules and EB tresos AutoCore OS	29
3.3. Module release notes	29
3.3.1. Crc module release notes	29
3.3.1.1. Change log	30
3.3.1.2. New features	34
3.3.1.3. EB-specific enhancements	34
3.3.1.4. Deviations	35
3.3.1.5. Limitations	36
3.3.1.6. Open-source software	36
3.3.2. Ea module release notes	36
3.3.2.1. Change log	36
3.3.2.2. New features	41
3.3.2.3. EB-specific enhancements	41
3.3.2.4. Deviations	42
3.3.2.5. Limitations	43
3.3.2.6. Open-source software	43
3.3.3. Fee module release notes	44
3.3.3.1. Change log	44
3.3.3.2. New features	50
3.3.3.3. EB-specific enhancements	51
3.3.3.4. Deviations	55
3.3.3.5. Limitations	56
3.3.3.6. Open-source software	58
3.3.4. MemAs module release notes	59
3.3.4.1. Change log	59

3.3.4.2. New features	63
3.3.4.3. EB-specific enhancements	63
3.3.4.4. Deviations	63
3.3.4.5. Limitations	63
3.3.4.6. Open-source software	63
3.3.5. MemIf module release notes	63
3.3.5.1. Change log	63
3.3.5.2. New features	67
3.3.5.3. EB-specific enhancements	67
3.3.5.4. Deviations	68
3.3.5.5. Limitations	68
3.3.5.6. Open-source software	69
3.3.6. NvM module release notes	69
3.3.6.1. Change log	69
3.3.6.2. New features	80
3.3.6.3. EB-specific enhancements	81
3.3.6.4. Deviations	87
3.3.6.5. Limitations	93
3.3.6.6. Open-source software	94
4. ACG8 Memory Stack user guide	95
4.1. Overview	95
4.2. Background information	96
4.2.1. Functional description	97
4.2.2. NvM block model	97
4.2.3. Block identification	100
4.2.4. Identifying logical blocks in NV memory	101
4.2.4.1. Example	101
4.2.5. Block management	102
4.2.6. Working with the different management types	102
4.2.6.1. Native block	102
4.2.6.1.1. Example: Instrument cluster	102
4.2.6.2. Redundant block	106
4.2.6.2.1. Example: Instrument cluster extended	106
4.2.6.3. Dataset block	109
4.2.6.3.1. Example	109
4.2.7. Memory configuration identification	114
4.2.8. Implicit and explicit synchronization	115
4.2.8.1. Implicit synchronization	116
4.2.8.2. Explicit synchronization	117
4.2.9. Prioritization of RAM buffers	119
4.2.10. Data recovery for blocks with reconfigured length	119
4.2.10.1. Configuration options	120

4.2.11. Working with the abstraction modules Ea and Fee	120
4.2.11.1. Functional description	120
4.2.11.1.1. Underlying memory access	120
4.2.11.1.2. Memory lifetime management	120
4.2.11.1.3. Data consistency checking	121
4.2.11.1.4. Invalid block handling	121
4.2.11.1.5. Inconsistent block handling	122
4.2.11.1.6. Garbage collection	122
4.2.11.1.7. Initialization and internal operations	122
4.2.11.1.8. Production errors	123
4.2.11.2. Memory layout	123
4.2.11.2.1. Ea memory layout	123
4.2.11.2.2. Fee memory layout	125
4.2.11.3. Patterns for data consistency	128
4.2.11.3.1. Functional description	128
4.2.11.3.2. Memory layout with consistency patterns	129
4.2.11.3.3. Configuration options	131
4.2.11.4. Configurable number of sections	132
4.2.11.4.1. Functional description	132
4.2.11.4.2. Configuration options	132
4.2.11.5. Secured section switch	132
4.2.11.5.1. Functional description	132
4.2.11.5.2. Configuration options	133
4.2.11.6. Section erase counter	133
4.2.11.6.1. Configuration options	133
4.2.11.7. Abort erase	133
4.2.11.7.1. Functional description	133
4.2.11.7.2. Configuration options	133
4.2.11.8. Freeze activities	134
4.2.11.8.1. Functional description	134
4.2.11.8.2. Configuration options	134
4.2.11.9. Switch non-configured blocks	134
4.2.11.9.1. Functional description	134
4.2.11.9.2. Configuration options	135
4.2.11.10. Write custom	135
4.2.11.10.1. Functional description	135
4.2.11.10.2. Configuration options	136
4.2.11.11. Read custom	136
4.2.11.11.1. Functional description	136
4.2.11.11.2. Configuration options	136
4.2.11.12. Cancel section erase	137
4.2.11.12.1. Functional description	137

4.2.11.12.2. Configuration options	137
4.2.11.13. Emergency block	137
4.2.11.13.1. Functional description	137
4.2.11.13.2. Configuration options	138
4.2.11.14. FIs alignment compatibility	138
4.2.11.14.1. Functional description	138
4.2.11.14.2. Configuration options	138
4.2.11.15. Small section support	139
4.2.11.15.1. Functional description	139
4.2.11.15.2. Configuration options	139
4.2.11.16. Drivers with vendorId and vendorApilInfix support	139
4.2.11.16.1. Functional description	139
4.2.11.16.2. Configuration options	140
4.2.11.17. Initialize in loop	140
4.2.11.17.1. Functional description	140
4.2.11.17.2. Configuration options	140
4.2.12. Multi-core support for NvM	140
4.2.12.1. Functional description	140
4.2.12.2. Configuration options	142
4.2.13. RAM block state after soft reset	142
4.2.13.1. Functional description	142
4.2.13.2. Configuration options	143
4.3. Configuring the memory stack	143
4.3.1. Background information	143
4.3.2. Configuration steps	143
4.4. The Memory Stack Editor	144
4.4.1. Background information	144
4.4.2. Using the Memory Stack Editor	145
4.4.2.1. Memory Stack Editor window layout	146
4.4.2.2. Configuring the memory stack with the Memory Stack Editor	147
4.5. The Memory Stack Automation Unattended Wizard	148
4.5.1. Background information	148
4.5.2. Using the Memory Stack Automation Unattended Wizard	148
4.5.3. Configuring the memory stack with the Memory Stack Automation	149
5. ACG8 Memory Stack module references	150
5.1. Overview	150
5.1.1. Notation in EB module references	150
5.1.1.1. Default value of configuration parameters	150
5.1.1.2. Range information of configuration parameters	150
5.2. Crc	151
5.2.1. Configuration parameters	151
5.2.1.1. CommonPublishedInformation	151

5.2.1.2. CrcGeneral	154
5.2.1.3. PublishedInformation	161
5.2.2. Recommended configurations	161
5.2.2.1. CrcRecConfigurationCrc16Crc32	162
5.2.2.1.1. CrcGeneral	162
5.2.3. Application programming interface (API)	162
5.2.3.1. Macro constants	162
5.2.3.1.1. CRC_AR_RELEASE_MAJOR_VERSION	162
5.2.3.1.2. CRC_AR_RELEASE_MINOR_VERSION	162
5.2.3.1.3. CRC_AR_RELEASE_REVISION_VERSION	163
5.2.3.1.4. CRC_E_PARAM_DATA	163
5.2.3.1.5. CRC_GET_VERSION_INFO_API_ID	163
5.2.3.1.6. CRC_INSTANCE_ID	163
5.2.3.1.7. CRC_MODULE_ID	163
5.2.3.1.8. CRC_SW_MAJOR_VERSION	163
5.2.3.1.9. CRC_SW_MINOR_VERSION	164
5.2.3.1.10. CRC_SW_PATCH_VERSION	164
5.2.3.1.11. CRC_VENDOR_ID	164
5.2.3.1.12. DBG_CRC_CALCULATECRC16_ENTRY	164
5.2.3.1.13. DBG_CRC_CALCULATECRC16_EXIT	164
5.2.3.1.14. DBG_CRC_CALCULATECRC32_ENTRY	164
5.2.3.1.15. DBG_CRC_CALCULATECRC32_EXIT	165
5.2.3.1.16. DBG_CRC_CALCULATECRC64_ENTRY	165
5.2.3.1.17. DBG_CRC_CALCULATECRC64_EXIT	165
5.2.3.1.18. DBG_CRC_CALCULATECRC8H2F_ENTRY	165
5.2.3.1.19. DBG_CRC_CALCULATECRC8H2F_EXIT	165
5.2.3.1.20. DBG_CRC_CALCULATECRC8_ENTRY	165
5.2.3.1.21. DBG_CRC_CALCULATECRC8_EXIT	166
5.2.3.1.22. DBG_CRC_GETVERSIONINFO_ENTRY	166
5.2.3.1.23. DBG_CRC_GETVERSIONINFO_EXIT	166
5.2.3.2. Functions	166
5.2.3.2.1. Crc_CalculateCRC16	166
5.2.3.2.2. Crc_CalculateCRC32	167
5.2.3.2.3. Crc_CalculateCRC64	167
5.2.3.2.4. Crc_CalculateCRC8	168
5.2.3.2.5. Crc_CalculateCRC8H2F	169
5.2.3.2.6. Crc_GetVersionInfo	169
5.2.4. Integration notes	170
5.2.4.1. Exclusive areas	170
5.2.4.2. Production errors	170
5.2.4.3. Memory mapping	170
5.2.4.4. Integration requirements	171

5.3. Ea	171
5.3.1. Configuration parameters	171
5.3.1.1. CommonPublishedInformation	172
5.3.1.2. EaBlockConfiguration	175
5.3.1.3. EaGeneral	177
5.3.1.4. EaPublishedInformation	181
5.3.1.5. PublishedInformation	183
5.3.2. Recommended configurations	183
5.3.2.1. EaRecConfiguration	183
5.3.2.1.1. EaBlockConfiguration_0	184
5.3.2.1.2. EaBlockConfiguration_1	184
5.3.3. Application programming interface (API)	184
5.3.3.1. Macro constants	184
5.3.3.1.1. DBG_EA_CANCEL_ENTRY	184
5.3.3.1.2. DBG_EA_CANCEL_EXIT	185
5.3.3.1.3. DBG_EA_ERASEIMMEDIATEBLOCK_ENTRY	185
5.3.3.1.4. DBG_EA_ERASEIMMEDIATEBLOCK_EXIT	185
5.3.3.1.5. DBG_EA_GETJOBRESULT_ENTRY	185
5.3.3.1.6. DBG_EA_GETJOBRESULT_EXIT	185
5.3.3.1.7. DBG_EA_GETSTATUS_ENTRY	185
5.3.3.1.8. DBG_EA_GETSTATUS_EXIT	185
5.3.3.1.9. DBG_EA_GETVERSIONINFO_ENTRY	186
5.3.3.1.10. DBG_EA_GETVERSIONINFO_EXIT	186
5.3.3.1.11. DBG_EA_INIT_ENTRY	186
5.3.3.1.12. DBG_EA_INIT_EXIT	186
5.3.3.1.13. DBG_EA_INVALIDATEBLOCK_ENTRY	186
5.3.3.1.14. DBG_EA_INVALIDATEBLOCK_EXIT	186
5.3.3.1.15. DBG_EA_JOBENDNOTIFICATION_ENTRY	187
5.3.3.1.16. DBG_EA_JOBENDNOTIFICATION_EXIT	187
5.3.3.1.17. DBG_EA_JOBERRORNOTIFICATION_ENTRY	187
5.3.3.1.18. DBG_EA_JOBERRORNOTIFICATION_EXIT	187
5.3.3.1.19. DBG_EA_JOBRESULT	187
5.3.3.1.20. DBG_EA_MAINFUNCTION_ENTRY	187
5.3.3.1.21. DBG_EA_MAINFUNCTION_EXIT	187
5.3.3.1.22. DBG_EA_MAINSTATE	188
5.3.3.1.23. DBG_EA_READ_ENTRY	188
5.3.3.1.24. DBG_EA_READ_EXIT	188
5.3.3.1.25. DBG_EA_SETMODE_ENTRY	188
5.3.3.1.26. DBG_EA_SETMODE_EXIT	188
5.3.3.1.27. DBG_EA_STATUS	188
5.3.3.1.28. DBG_EA_WRITE_ENTRY	189
5.3.3.1.29. DBG_EA_WRITE_EXIT	189

5.3.3.1.30. EA_API_CANCEL	189
5.3.3.1.31. EA_API_ERASEIMMEDIATEBLOCK	189
5.3.3.1.32. EA_API_GETJOBRESULT	189
5.3.3.1.33. EA_API_GETSTATUS	189
5.3.3.1.34. EA_API_GETVERSIONINFO	189
5.3.3.1.35. EA_API_INIT	190
5.3.3.1.36. EA_API_INVALIDATEBLOCK	190
5.3.3.1.37. EA_API_JOBENDNOTIFICATION	190
5.3.3.1.38. EA_API_JOBERRORNOTIFICATION	190
5.3.3.1.39. EA_API_MAINFUNCTION	190
5.3.3.1.40. EA_API_READ	190
5.3.3.1.41. EA_API_SETMODE	190
5.3.3.1.42. EA_API_WRITE	191
5.3.3.1.43. EA_AR_RELEASE_MAJOR_VERSION	191
5.3.3.1.44. EA_AR_RELEASE_MINOR_VERSION	191
5.3.3.1.45. EA_AR_RELEASE_REVISION_VERSION	191
5.3.3.1.46. EA_E_BUSY	191
5.3.3.1.47. EA_E_BUSY_INTERNAL	191
5.3.3.1.48. EA_E_INVALID_BLOCK_LEN	191
5.3.3.1.49. EA_E_INVALID_BLOCK_NO	192
5.3.3.1.50. EA_E_INVALID_BLOCK_OFS	192
5.3.3.1.51. EA_E_INVALID_CANCEL	192
5.3.3.1.52. EA_E_INVALID_DATA_PTR	192
5.3.3.1.53. EA_E_INVALID_MODE	192
5.3.3.1.54. EA_E_UNINIT	192
5.3.3.1.55. EA_INSTANCE_ID	192
5.3.3.1.56. EA_MODULE_ID	193
5.3.3.1.57. EA_SW_MAJOR_VERSION	193
5.3.3.1.58. EA_SW_MINOR_VERSION	193
5.3.3.1.59. EA_SW_PATCH_VERSION	193
5.3.3.1.60. EA_VENDOR_ID	193
5.3.3.2. Functions	193
5.3.3.2.1. Ea_Cancel	193
5.3.3.2.2. Ea_EraseImmediateBlock	194
5.3.3.2.3. Ea_GetJobResult	194
5.3.3.2.4. Ea_GetStatus	195
5.3.3.2.5. Ea_GetVersionInfo	195
5.3.3.2.6. Ea_Init	196
5.3.3.2.7. Ea_InvalidateBlock	196
5.3.3.2.8. Ea_JobEndNotification	197
5.3.3.2.9. Ea_JobErrorNotification	197
5.3.3.2.10. Ea_MainFunction	197

5.3.3.2.11. Ea_Read	197
5.3.3.2.12. Ea_SetMode	198
5.3.3.2.13. Ea_Write	198
5.3.4. Integration notes	199
5.3.4.1. Exclusive areas	199
5.3.4.2. Production errors	199
5.3.4.3. Memory mapping	199
5.3.4.4. Integration requirements	200
5.3.4.4.1. dev.Ea.IntegrationRestrictions	200
5.4. Fee	200
5.4.1. Configuration parameters	200
5.4.1.1. CommonPublishedInformation	202
5.4.1.2. FeeBlockConfiguration	205
5.4.1.3. FeeDefensiveProgramming	207
5.4.1.4. FeeGeneral	210
5.4.1.5. FeeSectionConfiguration	222
5.4.1.6. FeeDemEventParameterRefs	223
5.4.1.7. FeePublishedInformation	224
5.4.1.8. PublishedInformation	226
5.4.2. Recommended configurations	227
5.4.2.1. FeeRecConfiguration	227
5.4.2.1.1. FeeBlockConfiguration_0	227
5.4.2.1.2. FeeBlockConfiguration_1	227
5.4.3. Application programming interface (API)	227
5.4.3.1. Macro constants	228
5.4.3.1.1. DBG_FEE_CANCELSECTIONERASE_ENTRY	228
5.4.3.1.2. DBG_FEE_CANCELSECTIONERASE_EXIT	228
5.4.3.1.3. DBG_FEE_CANCEL_ENTRY	228
5.4.3.1.4. DBG_FEE_CANCEL_EXIT	228
5.4.3.1.5. DBG_FEE_ERASEIMMEDIATEBLOCK_ENTRY	228
5.4.3.1.6. DBG_FEE_ERASEIMMEDIATEBLOCK_EXIT	228
5.4.3.1.7. DBG_FEE_FREEZEACTIVITIES_ENTRY	229
5.4.3.1.8. DBG_FEE_FREEZEACTIVITIES_EXIT	229
5.4.3.1.9. DBG_FEE_GETERASECOUNTERVALUE_ENTRY	229
5.4.3.1.10. DBG_FEE_GETERASECOUNTERVALUE_EXIT	229
5.4.3.1.11. DBG_FEE_GETJOBRESULT_ENTRY	229
5.4.3.1.12. DBG_FEE_GETJOBRESULT_EXIT	229
5.4.3.1.13. DBG_FEE_GETSTATUS_ENTRY	229
5.4.3.1.14. DBG_FEE_GETSTATUS_EXIT	230
5.4.3.1.15. DBG_FEE_GETVERSIONINFO_ENTRY	230
5.4.3.1.16. DBG_FEE_GETVERSIONINFO_EXIT	230
5.4.3.1.17. DBG_FEE_INIT_ENTRY	230

5.4.3.1.18. DBG_FEE_INIT_EXIT	230
5.4.3.1.19. DBG_FEE_INVALIDATEBLOCK_ENTRY	230
5.4.3.1.20. DBG_FEE_INVALIDATEBLOCK_EXIT	231
5.4.3.1.21. DBG_FEE_JOBENDNOTIFICATION_ENTRY	231
5.4.3.1.22. DBG_FEE_JOBENDNOTIFICATION_EXIT	231
5.4.3.1.23. DBG_FEE_JOBERRORNOTIFICATION_ENTRY	231
5.4.3.1.24. DBG_FEE_JOBERRORNOTIFICATION_EXIT	231
5.4.3.1.25. DBG_FEE_MAINFUNCTION_ENTRY	231
5.4.3.1.26. DBG_FEE_MAINFUNCTION_EXIT	232
5.4.3.1.27. DBG_FEE_NEXTSTATE	232
5.4.3.1.28. DBG_FEE_READ_CUSTOM_ENTRY	232
5.4.3.1.29. DBG_FEE_READ_CUSTOM_EXIT	232
5.4.3.1.30. DBG_FEE_READ_ENTRY	232
5.4.3.1.31. DBG_FEE_READ_EXIT	232
5.4.3.1.32. DBG_FEE_SETMODE_ENTRY	232
5.4.3.1.33. DBG_FEE_SETMODE_EXIT	233
5.4.3.1.34. DBG_FEE_STATE	233
5.4.3.1.35. DBG_FEE_WRITE_CUSTOM_ENTRY	233
5.4.3.1.36. DBG_FEE_WRITE_CUSTOM_EXIT	233
5.4.3.1.37. DBG_FEE_WRITE_ENTRY	233
5.4.3.1.38. DBG_FEE_WRITE_EXIT	233
5.4.3.1.39. FEE_AR_RELEASE_MAJOR_VERSION	234
5.4.3.1.40. FEE_AR_RELEASE_MINOR_VERSION	234
5.4.3.1.41. FEE_AR_RELEASE_REVISION_VERSION	234
5.4.3.1.42. FEE_CANCELSECTIONERASE_API_ID	234
5.4.3.1.43. FEE_CANCEL_API_ID	234
5.4.3.1.44. FEE_ERASEIMMEDIATEBLOCK_API_ID	234
5.4.3.1.45. FEE_FREEZEACTIVITIES_API_ID	234
5.4.3.1.46. FEE_GETJOBRESULT_API_ID	235
5.4.3.1.47. FEE_GETSTATUS_API_ID	235
5.4.3.1.48. FEE_GET_ERASE_COUNTER_API_ID	235
5.4.3.1.49. FEE_GET_VERSION_INFO_API_ID	235
5.4.3.1.50. FEE_INIT_API_ID	235
5.4.3.1.51. FEE_INVALIDATEBLOCK_API_ID	235
5.4.3.1.52. FEE_JOBENDNOTIFICATION_API_ID	236
5.4.3.1.53. FEE_JOBERRORNOTIFICATION_API_ID	236
5.4.3.1.54. FEE_MAINFUNCTION_API_ID	236
5.4.3.1.55. FEE_MODULE_ID	236
5.4.3.1.56. FEE_READ_API_ID	236
5.4.3.1.57. FEE_READ_CUSTOM_API_ID	236
5.4.3.1.58. FEE_SETMODE_API_ID	236
5.4.3.1.59. FEE_SW_MAJOR_VERSION	237

5.4.3.1.60. FEE_SW_MINOR_VERSION	237
5.4.3.1.61. FEE_SW_PATCH_VERSION	237
5.4.3.1.62. FEE_VENDOR_ID	237
5.4.3.1.63. FEE_WRITE_API_ID	237
5.4.3.1.64. FEE_WRITE_CUSTOM_API_ID	237
5.4.3.2. Functions	238
5.4.3.2.1. Fee_Cancel	238
5.4.3.2.2. Fee_CancelSectionErase	238
5.4.3.2.3. Fee_EraseImmediateBlock	238
5.4.3.2.4. Fee_FreezeActivities	239
5.4.3.2.5. Fee_GetEraseCounterValue	239
5.4.3.2.6. Fee_GetJobResult	240
5.4.3.2.7. Fee_GetStatus	240
5.4.3.2.8. Fee_GetVersionInfo	241
5.4.3.2.9. Fee_Init	241
5.4.3.2.10. Fee_InvalidateBlock	242
5.4.3.2.11. Fee_JobEndNotification	242
5.4.3.2.12. Fee_JobErrorNotification	242
5.4.3.2.13. Fee_MainFunction	243
5.4.3.2.14. Fee_Read	243
5.4.3.2.15. Fee_ReadCustom	244
5.4.3.2.16. Fee_SetMode	244
5.4.3.2.17. Fee_Write	245
5.4.3.2.18. Fee_WriteCustom	245
5.4.4. Integration notes	246
5.4.4.1. Exclusive areas	246
5.4.4.2. Production errors	246
5.4.4.3. Memory mapping	246
5.4.4.4. Integration requirements	247
5.4.4.4.1. lim.Fee.EB_INTREQ_Fee_0001	247
5.4.4.4.2. lim.Fee.EB_INTREQ_Fee_0002	247
5.4.4.4.3. lim.Fee.EB_INTREQ_Fee_0003	247
5.4.4.4.4. lim.Fee.EB_INTREQ_Fee_0004	247
5.4.4.4.5. lim.Fee.EB_INTREQ_Fee_0005	248
5.4.4.4.6. lim.Fee.EB_INTREQ_Fee_0006	248
5.4.4.4.7. lim.Fee.EB_INTREQ_Fee_0007	248
5.4.4.4.8. lim.Fee.EB_INTREQ_Fee_0008	248
5.4.4.4.9. lim.Fee.EB_INTREQ_Fee_0009	249
5.4.4.4.10. lim.Fee.EB_INTREQ_Fee_0010	249
5.4.4.4.11. lim.Fee.EB_INTREQ_Fee_0011	249
5.4.4.4.12. lim.Fee.EB_INTREQ_Fee_0012	249
5.5. MemIf	249

5.5.1. Configuration parameters	250
5.5.1.1. CommonPublishedInformation	250
5.5.1.2. MemIfGeneral	253
5.5.1.3. PublishedInformation	254
5.5.2. Application programming interface (API)	255
5.5.2.1. Type definitions	255
5.5.2.1.1. MemIf_CancelFctPtrType	255
5.5.2.1.2. MemIf_EraseImmedBlockFctPtrType	255
5.5.2.1.3. MemIf_GetJobResultFctPtrType	255
5.5.2.1.4. MemIf_GetStatusFctPtrType	256
5.5.2.1.5. MemIf_InvalidateBlockFctPtrType	256
5.5.2.1.6. MemIf_JobResultType	256
5.5.2.1.7. MemIf_ModeType	256
5.5.2.1.8. MemIf_ReadFctPtrType	256
5.5.2.1.9. MemIf_SetModeFctPtrType	256
5.5.2.1.10. MemIf_StatusType	257
5.5.2.1.11. MemIf_WriteFctPtrType	257
5.5.2.2. Macro constants	257
5.5.2.2.1. DBG_MEMIF_CANCEL_ENTRY	257
5.5.2.2.2. DBG_MEMIF_CANCEL_EXIT	257
5.5.2.2.3. DBG_MEMIF_ERASEIMMEDIATEBLOCK_ENTRY	257
5.5.2.2.4. DBG_MEMIF_ERASEIMMEDIATEBLOCK_EXIT	258
5.5.2.2.5. DBG_MEMIF_GETJOBRESULT_ENTRY	258
5.5.2.2.6. DBG_MEMIF_GETJOBRESULT_EXIT	258
5.5.2.2.7. DBG_MEMIF_GETSTATUS_ENTRY	258
5.5.2.2.8. DBG_MEMIF_GETSTATUS_EXIT	258
5.5.2.2.9. DBG_MEMIF_GETVERSIONINFO_ENTRY	258
5.5.2.2.10. DBG_MEMIF_GETVERSIONINFO_EXIT	259
5.5.2.2.11. DBG_MEMIF_INVALIDATEBLOCK_ENTRY	259
5.5.2.2.12. DBG_MEMIF_INVALIDATEBLOCK_EXIT	259
5.5.2.2.13. DBG_MEMIF_READ_ENTRY	259
5.5.2.2.14. DBG_MEMIF_READ_EXIT	259
5.5.2.2.15. DBG_MEMIF_SETMODE_ENTRY	259
5.5.2.2.16. DBG_MEMIF_SETMODE_EXIT	259
5.5.2.2.17. DBG_MEMIF_WRITE_ENTRY	260
5.5.2.2.18. DBG_MEMIF_WRITE_EXIT	260
5.5.2.2.19. MEMIF_AR_RELEASE_MAJOR_VERSION	260
5.5.2.2.20. MEMIF_AR_RELEASE_MINOR_VERSION	260
5.5.2.2.21. MEMIF_AR_RELEASE_REVISION_VERSION	260
5.5.2.2.22. MEMIF_BLOCK_INCONSISTENT	260
5.5.2.2.23. MEMIF_BLOCK_INVALID	261
5.5.2.2.24. MEMIF_BROADCAST_ID	261

5.5.2.2.25. MEMIF_BUSY	261
5.5.2.2.26. MEMIF_BUSY_INTERNAL	261
5.5.2.2.27. MEMIF_CANCEL_ID	261
5.5.2.2.28. MEMIF_ERASEIMMEDIATEBLOCK_ID	261
5.5.2.2.29. MEMIF_E_PARAM_DEVICE	262
5.5.2.2.30. MEMIF_E_PARAM_NULL_PTR	262
5.5.2.2.31. MEMIF_GETJOBRESULT_ID	262
5.5.2.2.32. MEMIF_GETSTATUS_ID	262
5.5.2.2.33. MEMIF_GETVERSIONINFO_ID	262
5.5.2.2.34. MEMIF_IDLE	262
5.5.2.2.35. MEMIF_INSTANCE_ID	263
5.5.2.2.36. MEMIF_INVALIDATEBLOCK_ID	263
5.5.2.2.37. MEMIF_JOB_CANCELED	263
5.5.2.2.38. MEMIF_JOB_FAILED	263
5.5.2.2.39. MEMIF_JOB_OK	263
5.5.2.2.40. MEMIF_JOB_OK_SIZE_DECREASED	263
5.5.2.2.41. MEMIF_JOB_OK_SIZE_INCREASED	264
5.5.2.2.42. MEMIF_JOB_PENDING	264
5.5.2.2.43. MEMIF_MODE_FAST	264
5.5.2.2.44. MEMIF_MODE_SLOW	264
5.5.2.2.45. MEMIF_MODULE_ID	264
5.5.2.2.46. MEMIF_READ_ID	264
5.5.2.2.47. MEMIF_SETMODE_ID	265
5.5.2.2.48. MEMIF_SW_MAJOR_VERSION	265
5.5.2.2.49. MEMIF_SW_MINOR_VERSION	265
5.5.2.2.50. MEMIF_SW_PATCH_VERSION	265
5.5.2.2.51. MEMIF_UNINIT	265
5.5.2.2.52. MEMIF_VENDOR_ID	265
5.5.2.2.53. MEMIF_WRITE_ID	266
5.5.2.3. Objects	266
5.5.2.3.1. MemIf_CancelFctPtr	266
5.5.2.3.2. MemIf_EraseImmediateBlockFctPtr	266
5.5.2.3.3. MemIf_GetJobResultFctPtr	266
5.5.2.3.4. MemIf_InvalidateBlockFctPtr	266
5.5.2.3.5. MemIf_ReadFctPtr	266
5.5.2.3.6. MemIf_WriteFctPtr	267
5.5.2.4. Functions	267
5.5.2.4.1. MemIf_Cancel	267
5.5.2.4.2. MemIf_EraseImmediateBlock	267
5.5.2.4.3. MemIf_GetJobResult	268
5.5.2.4.4. MemIf_GetStatus	268
5.5.2.4.5. MemIf_GetVersionInfo	269

5.5.2.4.6. MemIf_InvalidateBlock	270
5.5.2.4.7. MemIf_Read	270
5.5.2.4.8. MemIf_SetMode	271
5.5.2.4.9. MemIf_Write	271
5.5.3. Integration notes	272
5.5.3.1. Exclusive areas	272
5.5.3.2. Production errors	272
5.5.3.3. Memory mapping	272
5.5.3.4. Integration requirements	273
5.6. NvM	273
5.6.1. Configuration parameters	273
5.6.1.1. CommonPublishedInformation	274
5.6.1.2. NvMBlockDescriptor	277
5.6.1.3. NvMTargetBlockReference	308
5.6.1.4. NvMEaRef	308
5.6.1.5. NvMFeeRef	308
5.6.1.6. NvMDefensiveProgramming	309
5.6.1.7. NvMCommon	312
5.6.1.8. NvMCommonCryptoSecurityParameters	325
5.6.1.9. NvMServiceAPI	328
5.6.1.10. NvmDemEventParameterRefs	330
5.6.1.11. ReportToDem	336
5.6.1.12. MultiCoreCallout	346
5.6.1.13. PublishedInformation	350
5.6.2. Pre-configurations	350
5.6.2.1. NvMPreConfiguration	350
5.6.2.1.1. NvMBlock_ConfigID	351
5.6.3. Application programming interface (API)	351
5.6.3.1. Type definitions	351
5.6.3.1.1. NvM_ASR32_BlockIdType	351
5.6.3.1.2. NvM_ASR32_RequestResultType	351
5.6.3.1.3. NvM_ASR40_BlockIdType	352
5.6.3.1.4. NvM_ASR40_RequestResultType	352
5.6.3.1.5. NvM_ASR42_BlockIdType	352
5.6.3.1.6. NvM_ASR42_RequestResultType	352
5.6.3.1.7. NvM_BlockIdType	352
5.6.3.1.8. NvM_RequestResultType	352
5.6.3.2. Macro constants	353
5.6.3.2.1. DBG_NVM_ASR40_CANCELJOBS_ENTRY	353
5.6.3.2.2. DBG_NVM_ASR40_CANCELJOBS_EXIT	353
5.6.3.2.3. DBG_NVM_ASR40_ERASENBLOCK_ENTRY	353
5.6.3.2.4. DBG_NVM_ASR40_ERASENBLOCK_EXIT	353

5.6.3.2.5. DBG_NVM_ASR40_GETDATAINDEX_ENTRY	353
5.6.3.2.6. DBG_NVM_ASR40_GETDATAINDEX_EXIT	353
5.6.3.2.7. DBG_NVM_ASR40_GETERRORSTATUS_ENTRY	354
5.6.3.2.8. DBG_NVM_ASR40_GETERRORSTATUS_EXIT	354
5.6.3.2.9. DBG_NVM_ASR40_INVALIDATENVBLOCK_ENTRY	354
5.6.3.2.10. DBG_NVM_ASR40_INVALIDATENVBLOCK_EXIT	354
5.6.3.2.11. DBG_NVM_ASR40_READBLOCK_ENTRY	354
5.6.3.2.12. DBG_NVM_ASR40_READBLOCK_EXIT	354
5.6.3.2.13. DBG_NVM_ASR40_RESTOREBLOCKDEFAULTS_ENTRY	355
5.6.3.2.14. DBG_NVM_ASR40_RESTOREBLOCKDEFAULTS_EXIT	355
5.6.3.2.15. DBG_NVM_ASR40_SETBLOCKPROTECTION_ENTRY	355
5.6.3.2.16. DBG_NVM_ASR40_SETBLOCKPROTECTION_EXIT	355
5.6.3.2.17. DBG_NVM_ASR40_SETDATAINDEX_ENTRY	355
5.6.3.2.18. DBG_NVM_ASR40_SETDATAINDEX_EXIT	355
5.6.3.2.19. DBG_NVM_ASR40_SETRAMBLOCKSTATUS_ENTRY	356
5.6.3.2.20. DBG_NVM_ASR40_SETRAMBLOCKSTATUS_EXIT	356
5.6.3.2.21. DBG_NVM_ASR40_WRITEBLOCK_ENTRY	356
5.6.3.2.22. DBG_NVM_ASR40_WRITEBLOCK_EXIT	356
5.6.3.2.23. DBG_NVM_ASR42_READPRAMBLOCK_ENTRY	356
5.6.3.2.24. DBG_NVM_ASR42_READPRAMBLOCK_EXIT	356
5.6.3.2.25. DBG_NVM_ASR42_RESTOREPRAMBLOCKDEFAULTS_ENTRY	357
5.6.3.2.26. DBG_NVM_ASR42_RESTOREPRAMBLOCKDEFAULTS_EXIT	357
5.6.3.2.27. DBG_NVM_ASR42_WRITEPRAMBLOCK_ENTRY	357
5.6.3.2.28. DBG_NVM_ASR42_WRITEPRAMBLOCK_EXIT	357
5.6.3.2.29. DBG_NVM_CANCELWRITEALL_ENTRY	357
5.6.3.2.30. DBG_NVM_CANCELWRITEALL_EXIT	357
5.6.3.2.31. DBG_NVM_FIRSTINITALL_ENTRY	358
5.6.3.2.32. DBG_NVM_FIRSTINITALL_EXIT	358
5.6.3.2.33. DBG_NVM_GETVERSIONINFO_ENTRY	358
5.6.3.2.34. DBG_NVM_GETVERSIONINFO_EXIT	358
5.6.3.2.35. DBG_NVM_GLOBALCALLLEVEL	358
5.6.3.2.36. DBG_NVM_GLOBALERRORSTATUS	358
5.6.3.2.37. DBG_NVM_GLOBALGENERICSTATUS	359
5.6.3.2.38. DBG_NVM_GLOBALWORKINGSTATUS	359
5.6.3.2.39. DBG_NVM_INIT_ENTRY	359
5.6.3.2.40. DBG_NVM_INIT_EXIT	359
5.6.3.2.41. DBG_NVM_JOBENDNOTIFICATION_ENTRY	359
5.6.3.2.42. DBG_NVM_JOBENDNOTIFICATION_EXIT	359
5.6.3.2.43. DBG_NVM_JOBERRORNOTIFICATION_ENTRY	360
5.6.3.2.44. DBG_NVM_JOBERRORNOTIFICATION_EXIT	360
5.6.3.2.45. DBG_NVM_MAINFUNCTION_ENTRY	360
5.6.3.2.46. DBG_NVM_MAINFUNCTION_EXIT	360

5.6.3.2.47. DBG_NVM_READALL_ENTRY	360
5.6.3.2.48. DBG_NVM_READALL_EXIT	360
5.6.3.2.49. DBG_NVM_SETBLOCKLOCKSTATUS_ENTRY	361
5.6.3.2.50. DBG_NVM_SETBLOCKLOCKSTATUS_EXIT	361
5.6.3.2.51. DBG_NVM_VALIDATEALL_ENTRY	361
5.6.3.2.52. DBG_NVM_VALIDATEALL_EXIT	361
5.6.3.2.53. DBG_NVM_WRITEALL_ENTRY	361
5.6.3.2.54. DBG_NVM_WRITEALL_EXIT	361
5.6.3.2.55. NVM_AR_RELEASE_MAJOR_VERSION	362
5.6.3.2.56. NVM_AR_RELEASE_MINOR_VERSION	362
5.6.3.2.57. NVM_AR_RELEASE_REVISION_VERSION	362
5.6.3.2.58. NVM_BLOCK_INVALID	362
5.6.3.2.59. NVM_BLOCK_VALID	362
5.6.3.2.60. NVM_CANCEL_JOBS_API_ID	362
5.6.3.2.61. NVM_CANCEL_WRITE_ALL_API_ID	363
5.6.3.2.62. NVM_CRYPTO_HOOKS_API_ID	363
5.6.3.2.63. NVM_ENABLE_BC_API_ID	363
5.6.3.2.64. NVM_ERASE_NV_BLOCK_API_ID	363
5.6.3.2.65. NVM_E_BLOCK_CONFIG	363
5.6.3.2.66. NVM_E_BLOCK_LOCKED	363
5.6.3.2.67. NVM_E_BLOCK_PENDING	364
5.6.3.2.68. NVM_E_BLOCK_WITHOUT_DEFAULTS	364
5.6.3.2.69. NVM_E_CRYPTO_FAILED	364
5.6.3.2.70. NVM_E_CRYPTO_WRITE_FAILED	364
5.6.3.2.71. NVM_E_NOT_INITIALIZED	364
5.6.3.2.72. NVM_E_PARAM_ADDRESS	364
5.6.3.2.73. NVM_E_PARAM_BLOCK_DATA_IDX	365
5.6.3.2.74. NVM_E_PARAM_BLOCK_ID	365
5.6.3.2.75. NVM_E_PARAM_BLOCK_TYPE	365
5.6.3.2.76. NVM_E_PARAM_DATA	365
5.6.3.2.77. NVM_E_PARAM_POINTER	365
5.6.3.2.78. NVM_E_WRITE_ONCE_STATUS_UNKNOWN	365
5.6.3.2.79. NVM_FIRST_INIT_ALL_API_ID	366
5.6.3.2.80. NVM_GET_DATA_INDEX_API_ID	366
5.6.3.2.81. NVM_GET_ERROR_STATUS_API_ID	366
5.6.3.2.82. NVM_GET_VERSION_INFO_API_ID	366
5.6.3.2.83. NVM_INVALIDATE_NV_BLOCK_API_ID	366
5.6.3.2.84. NVM_MODULE_ID	366
5.6.3.2.85. NVM_READ_ALL_API_ID	367
5.6.3.2.86. NVM_READ_BLOCK_API_ID	367
5.6.3.2.87. NVM_READ_PRAM_BLOCK_API_ID	367
5.6.3.2.88. NVM_REQ_BLOCK_SKIPPED	367

5.6.3.2.89. NVM_REQ_CANCELED	367
5.6.3.2.90. NVM_REQ_CANCELLED	367
5.6.3.2.91. NVM_REQ_INTEGRITY_FAILED	368
5.6.3.2.92. NVM_REQ_NOT_OK	368
5.6.3.2.93. NVM_REQ_NV_INVALIDATED	368
5.6.3.2.94. NVM_REQ_OK	368
5.6.3.2.95. NVM_REQ_PENDING	368
5.6.3.2.96. NVM_REQ_REDUNDANCY_FAILED	369
5.6.3.2.97. NVM_REQ_RESTORED_DEFAULTS	369
5.6.3.2.98. NVM_REQ_RESTORED_FROM_ROM	369
5.6.3.2.99. NVM_RESTORE_BLOCK_DEFAULTS_API_ID	369
5.6.3.2.100. NVM_RESTORE_PRAM_BLOCK_DEFAULTS_API_ID	369
5.6.3.2.101. NVM_SET_BLOCK_LOCK_STATUS_API_ID	369
5.6.3.2.102. NVM_SET_BLOCK_PROTECTION_API_ID	370
5.6.3.2.103. NVM_SET_DATA_INDEX_API_ID	370
5.6.3.2.104. NVM_SET_RAM_BLOCK_STATUS_API_ID	370
5.6.3.2.105. NVM_SW_MAJOR_VERSION	370
5.6.3.2.106. NVM_SW_MINOR_VERSION	370
5.6.3.2.107. NVM_SW_PATCH_VERSION	370
5.6.3.2.108. NVM_VALIDATE_ALL_API_ID	371
5.6.3.2.109. NVM_VENDOR_ID	371
5.6.3.2.110. NVM_WRITE_ALL_API_ID	371
5.6.3.2.111. NVM_WRITE_BLOCK_API_ID	371
5.6.3.2.112. NVM_WRITE_PRAM_BLOCK_API_ID	371
5.6.3.2.113. NvM_CancelJobs	371
5.6.3.2.114. NvM_EraseNvBlock	372
5.6.3.2.115. NvM_GetDataIndex	372
5.6.3.2.116. NvM_GetErrorStatus	372
5.6.3.2.117. NvM_InvalidateNvBlock	372
5.6.3.2.118. NvM_ReadBlock	372
5.6.3.2.119. NvM_ReadPRAMBlock	372
5.6.3.2.120. NvM_RestoreBlockDefaults	373
5.6.3.2.121. NvM_RestorePRAMBlockDefaults	373
5.6.3.2.122. NvM_SetBlockProtection	373
5.6.3.2.123. NvM_SetDataIndex	373
5.6.3.2.124. NvM_SetRamBlockStatus	373
5.6.3.2.125. NvM_WriteBlock	373
5.6.3.2.126. NvM_WritePRAMBlock	374
5.6.3.3. Functions	374
5.6.3.3.1. NvM_ASR32_EraseNvBlock	374
5.6.3.3.2. NvM_ASR32_GetDataIndex	375
5.6.3.3.3. NvM_ASR32_GetErrorStatus	375

5.6.3.3.4. NvM_ASR32_InvalidateNvBlock	376
5.6.3.3.5. NvM_ASR32_ReadBlock	377
5.6.3.3.6. NvM_ASR32_RestoreBlockDefaults	378
5.6.3.3.7. NvM_ASR32_SetBlockProtection	380
5.6.3.3.8. NvM_ASR32_SetDataIndex	380
5.6.3.3.9. NvM_ASR32_SetRamBlockStatus	381
5.6.3.3.10. NvM_ASR32_WriteBlock	382
5.6.3.3.11. NvM_ASR40_CancelJobs	383
5.6.3.3.12. NvM_ASR40_EraseNvBlock	383
5.6.3.3.13. NvM_ASR40_GetDataIndex	385
5.6.3.3.14. NvM_ASR40_GetErrorStatus	385
5.6.3.3.15. NvM_ASR40_InvalidateNvBlock	386
5.6.3.3.16. NvM_ASR40_ReadBlock	387
5.6.3.3.17. NvM_ASR40_RestoreBlockDefaults	388
5.6.3.3.18. NvM_ASR40_SetBlockProtection	390
5.6.3.3.19. NvM_ASR40_SetDataIndex	390
5.6.3.3.20. NvM_ASR40_SetRamBlockStatus	391
5.6.3.3.21. NvM_ASR40_WriteBlock	392
5.6.3.3.22. NvM_ASR42_EraseNvBlock	393
5.6.3.3.23. NvM_ASR42_GetDataIndex	394
5.6.3.3.24. NvM_ASR42_GetErrorStatus	395
5.6.3.3.25. NvM_ASR42_InvalidateNvBlock	396
5.6.3.3.26. NvM_ASR42_ReadBlock	397
5.6.3.3.27. NvM_ASR42_ReadPRAMBlock	398
5.6.3.3.28. NvM_ASR42_RestoreBlockDefaults	400
5.6.3.3.29. NvM_ASR42_RestorePRAMBlockDefaults	401
5.6.3.3.30. NvM_ASR42_SetBlockProtection	402
5.6.3.3.31. NvM_ASR42_SetDataIndex	403
5.6.3.3.32. NvM_ASR42_SetRamBlockStatus	404
5.6.3.3.33. NvM_ASR42_WriteBlock	405
5.6.3.3.34. NvM_ASR42_WritePRAMBlock	406
5.6.3.3.35. NvM_CancelWriteAll	407
5.6.3.3.36. NvM_DisableBlockCheckMechanism	408
5.6.3.3.37. NvM_EnableBlockCheck	408
5.6.3.3.38. NvM_FirstInitAll	408
5.6.3.3.39. NvM_GetVersionInfo	409
5.6.3.3.40. NvM_Init	409
5.6.3.3.41. NvM_JobEndNotification	410
5.6.3.3.42. NvM_JobErrorNotification	410
5.6.3.3.43. NvM_MainFunction	411
5.6.3.3.44. NvM_ReadAll	411
5.6.3.3.45. NvM_ReadBlockHook	414

5.6.3.3.46. NvM_SetBlockLockStatus	415
5.6.3.3.47. NvM_ValidateAll	415
5.6.3.3.48. NvM_WriteAll	416
5.6.3.3.49. NvM_WriteBlockHook	417
5.6.4. Integration notes	418
5.6.4.1. Exclusive areas	418
5.6.4.1.1. SCHM_NVM_EXCLUSIVE_AREA_0	418
5.6.4.2. Production errors	418
5.6.4.3. Memory mapping	420
5.6.4.4. Integration requirements	421
5.6.4.4.1. lim.NvM.EB_INTREQ_NvM_0001	421
5.6.4.4.2. lim.NvM.EB_INTREQ_NvM_0002	421
5.6.4.4.3. lim.NvM.EB_INTREQ_NvM_0003	422



1. Overview of EB tresos AutoCore Generic 8 Memory Stack documentation

Welcome to the EB tresos AutoCore Generic 8 Memory Stack (ACG8 Memory Stack) product documentation.

This document provides:

- ▶ [Chapter 2, “Supported features”](#): list of features supported by the ACG8 Memory Stack
- ▶ [Chapter 3, “ACG8 Memory Stack release notes”](#): release notes for the ACG8 Memory Stack modules
- ▶ [Chapter 4, “ACG8 Memory Stack user guide”](#): containing background information and instructions
- ▶ [Chapter 5, “ACG8 Memory Stack module references”](#): information about configuration parameters and the application programming interface

2. Supported features

2.1. Overview

This chapter provides an overview of the ACG8 Memory Stack and the features that are currently supported.

[Section 2.2, “Supported Crc features”](#) contains an overview of `Crc` features.

[Section 2.3, “Supported Ea features”](#) contains an overview of `Ea` features.

[Section 2.4, “Supported Fee features”](#) contains an overview of `Fee` features.

[Section 2.5, “Supported MemIf features”](#) contains an overview of `MemIf` features.

[Section 2.6, “Supported NvM features”](#) contains an overview of `NvM` features.

2.2. Supported Crc features

- ▶ Feature complete according to AUTOSAR 4.0.3
- ▶ CRC64 (ECMA-182 standard) implemented according to AUTOSAR 4.4.0

2.3. Supported Ea features

- ▶ Feature complete according to AUTOSAR 4.0.3

2.4. Supported Fee features

- ▶ Feature complete according to AUTOSAR 4.0.3
- ▶ **Patterns for data consistency:** this feature is part of ACG8 Memory Stack (Feature Package 1)
 - ▶ **End patterns for consistency:** each successful write request triggered by `Fee` is marked by writing an end pattern after the actual data has been successfully written.
 - ▶ **Start and end patterns for consistency:** each write request triggered by `Fee` is marked by writing a start pattern before writing the actual data, and an end pattern after the actual data has been successfully written.

ACG8 Memory Stack (Feature Package 1) is included in ACG8 Memory Stack.

- ▶ **Configurable number of sections:** The number of sections used by Fee to store data can be configured. The goal is to have higher flexibility and increase the robustness of the implementation.
- ▶ **Freeze activities:** This vendor-specific interface can be used in order to stop the processing of any request in the memory stack.
- ▶ **Switch non-configured blocks:** During a section switch, Fee recognizes blocks that are valid but not configured and switches these blocks as well. The configuration of the memory stack in application and bootloader does not have to be identical. An update on application side does not require an update in the bootloader.
- ▶ **Secured section switch:** No blocks are lost during a section switch. To ensure this, the section switch operation takes higher priority under some conditions and user requests are no longer accepted. If errors occur during the section switch, Fee can restart the section switch and ensure that all blocks are successfully copied to avoid block losses.
- ▶ **Section erase counter:** Fee provides a counter that provides information about the lifetime of the flash and the number of erase cycles for each section.
- ▶ **Abort erase:** To ensure that immediate data is written as quickly as possible, immediate write requests can interrupt an ongoing erase section operation.

2.5. Supported MemIf features

- ▶ Feature complete according to AUTOSAR 4.0.3

2.6. Supported NvM features

- ▶ Features implemented according to AUTOSAR 4.0.3 with selected features implemented according to AUTOSAR 4.2.1
- ▶ **Block management types:**
 - ▶ **Native NVRAM blocks:** block management type that consists of one NV block, a RAM block, an administrative block and an optional ROM block.
 - ▶ **Redundant NVRAM blocks:** block management type that consists of two NV blocks, a RAM block, an administrative block and an optional ROM block.
 - ▶ **Dataset NVRAM blocks:** block management type that consists of a configurable number of NV blocks, a RAM block, an administrative block and an configurable number of ROM blocks.
- ▶ **General features:**
 - ▶ **API configuration classes:** support for different configuration classes in order to provide the possibility of adapting the NvM module to limited hardware resources.

- ▶ **Block prioritization:** support for priority-based job processing.
- ▶ **NVRAM block consistency check:** support for implicit techniques to check the data consistency of NVRAM blocks.
- ▶ **NVRAM block write protection:** support for a configurable mechanism of write protection for the NV part of NVRAM block.
- ▶ **Static block ID check:** support for a configurable mechanism of storing and verifying a static identifier during write and read requests.
- ▶ **Write verification:** support for a configurable mechanism of reading back and verifying the data after each write request.
- ▶ **Retry mechanism for read and write requests:** support for the configuration of the retry mechanism if failure is detected during read or write requests.
- ▶ **Validation of RAM block data:** support for the validation of the RAM block depending on different API user requests.
- ▶ **(Quasi) parallel access to NvM:**
 - ▶ **Standard job queuing mechanism:** support for the use of an independent queue for processing only standard priority jobs.
 - ▶ **Immediate job queuing mechanism:** support for the use of an independent queue for processing only immediate jobs.
- ▶ **Recovery of NVRAM blocks:**
 - ▶ **Implicit recovery of RAM block with (ROM) default data:** support for the automatic recovery of default data in the RAM block when failure is detected during read requests.
 - ▶ **Explicit recovery of RAM with (ROM) default data:** support for the recovery of default data in the RAM block based on user request.
 - ▶ **Recovery of redundant blocks:** support of automatically recovering redundant blocks when inconsistencies are detected.
- ▶ **Communication between application and NvM:**
 - ▶ **Implicit synchronization during single-block requests:** support for implicit synchronization according to AUTOSAR specifications.
 - ▶ **Implicit synchronization during multi-block requests:** support for implicit synchronization according to AUTOSAR specifications.
 - ▶ **Explicit synchronization during single-block requests:** support for explicit synchronization according to AUTOSAR specifications.
 - ▶ **Explicit synchronization during multi-block requests:** support for explicit synchronization according to AUTOSAR specifications.
- ▶ **Error reporting:**

- ▶ **Development errors reporting:** support for the configuration of reporting development errors.
- ▶ **Extended production errors:** support for the configuration of reporting production errors.
- ▶ **Synchronous single-block requests:**
 - ▶ **Handling index of dataset NVRAM blocks with `NvM_SetDataIndex()`, `NvM_GetDataIndex()`:** support for user interfaces to select and read the active instance of DATASET blocks.
 - ▶ **NVRAM block protection with `NvM_SetBlockProtection()`:** support for the user interface to enable/disable block protection.
 - ▶ **NVRAM block status with `NvM_GetErrorStatus()`:** Support for the user interface to request the current block result for a specific NVRAM block.
 - ▶ **Validation of NVRAM blocks with `NvM_SetRamBlockStatus()`:** support for the user interface to update the status of the RAM block.
 - ▶ **Locking of NVRAM blocks with `NvM_SetBlockLockStatus()`:** support for the user interface to lock a NVRAM block for further requests.
 - ▶ **Cancel ongoing requests with `NvM_CancelJobs()`:** support for the user interface to cancel pending requests for a specific block.
- ▶ **Asynchronous single-block requests:**
 - ▶ **Copy NV block data to RAM with `NvM_ReadBlock()` or `NvM_ReadPRAMBlock()`:** support for the user interface to read data from non-volatile memory and copy it to the RAM block.
 - ▶ **Copy RAM data to corresponding NV block with `NvM_WriteBlock()` or `NvM_WritePRAMBlock()`:** support for the user interfaces to write the data from the RAM block to the non-volatile memory.
 - ▶ **Restore default data to RAM block with `NvM_RestoreBlockDefaults()` or `NvM_RestorePRAM-BlockDefaults()`:** support for the user interface to copy default data to the RAM block.
 - ▶ **Erase a NV block with `NvM_EraseNvBlock()`:** support for the user interface to erase the content of a NVRAM block in the non-volatile memory.
 - ▶ **Invalidate a NV block with `NvM_InvalidateNvBlock()`:** support for the user interface to invalidate the content of a NVRAM block in the non-volatile memory.
- ▶ **Asynchronous multi-block requests:**
 - ▶ **Read all NVRAM blocks at startup with `NvM_ReadAll()`:** support for the interface to read multiple configured blocks during startup.
 - ▶ **Write all NVRAM blocks at shutdown with `NvM_WriteAll()`:** support for the interface to write multiple configured blocks during shutdown.
 - ▶ **Auto-validation of all NVRAM blocks with `NvM_ValidateAll()`:** support for the interface to auto-validate multiple configured blocks.
 - ▶ **Cancel running multi-block write request with `NvM_CancelWriteAll()`:** support for the interface to stop an ongoing multiple block write request in the shutdown phase.

► **Callbacks and optional interfaces:**

- **Job end notification:** support for the configuration of a callback to be called by the lower layer to notify that a job finished without error.
- **Job error notification:** support for the configuration of a callback to be called by the lower layer to notify that a job finished with error.
- **Single-block job end notification:** support for the configuration of a block-specific callback function to notify the upper layer that a job finished.
- **Multi-block job end notification:** support for the configuration of a common callback to notify that a multi-block request finished.
- **Block initialization callback:** support for block-specific callback function to request default data.
- **RAM to NvM and NvM to RAM copy callbacks:** support for the configuration of block-specific callback functions.

► **Service interfaces:**

- **Client-server NvM_Service interface (ASR3.2, ASR4.0.3, ASR4.2.1):** support for configurable client-server interfaces to be published for a software component in order to invoke NvM services.
- **Client-server NvM_NotifyJobFinished interface (ASR3.2, ASR4.0.3, ASR4.2.1):** support for configurable client-server interfaces to be published for a software component in order to notify that a job finished.
- **Client-server NvM_NotifyInitBlock interface (ASR3.2, ASR4.0.3, ASR4.2.1):** support for configurable client-server interfaces to be published for a software component in order to provide default data.
- **Client-server NvM_Admin interface (ASR3.2, ASR4.0.3, ASR4.2.1):** support for configurable client-server interfaces to be published for a software component in order to update administrative information.
- **Client-server NvM_Mirror interface (ASR3.2, ASR4.0.3, ASR4.2.1):** support for configurable client-server interfaces to be published for a software component in order to exchange data between the internal mirror buffer and the user RAM.

► **EB-specific extensions:**

- **Hook functions for read/write operations:** support for the configuration of additional hook functions to be called during read/write requests when data is copied to/from permanent RAM.
- **Optimized RAM consumption (NvmUserProvidesSpaceForBlockAndCrc):** support for a block-specific configuration parameter to let the user decide if CRC shall be stored together with the data.
- **Unified RAM mirror and internal buffer:** support for sharing the same RAM area for both the RAM mirror and the NvM internal buffer.
- **Improved handling of redundant blocks (NvMRedundantRecovery):** support for mechanism of reporting the loss of redundancy as a result of the block.



- ▶ **Extra block size checks:** support for the configuration of a block size check.
- ▶ **Support for symbolic port names to easily change configurations:** support for the deletion of a port without renumbering and reconnecting all existing ports.

3. ACG8 Memory Stack release notes

3.1. Overview

This chapter provides the ACG8 Memory Stack product specific release notes. General release notes that are applicable to all products are provided in the EB tresos AutoCore Generic documentation. Refer to the general release notes in addition to the product release notes documented here.

3.2. Scope of the release

3.2.1. Configuration tool

Your release of EB tresos AutoCore is compatible with the release of the EB tresos Studio configuration tool:

- ▶ EB tresos Studio: 28.1.0 b210701-0227

3.2.2. AUTOSAR modules

The following table lists the AUTOSAR modules that are part of this ACG8 Memory Stack release.

Module name	AUTOSAR version and revision	SWS version and revision	Module version	Supplier
Crc	4.0.3 []	4.2.0 [3]	6.11.13	Elektrobit Automotive GmbH
Ea	4.0.3 []	2.0.0 [3]	5.12.14	Elektrobit Automotive GmbH
Fee	4.0.3 []	2.0.0 [3]	6.14.13	Elektrobit Automotive GmbH
Memlf	4.0.3 []	1.4.0 [3]	5.11.11	Elektrobit Automotive GmbH

Module name	AUTOSAR version and revision	SWS version and revision	Module version	Supplier
NvM	4.0.3 []	3.2.0 [3]	6.17.22	Elektrobit Automotive GmbH

Table 3.1. Hardware-Independent Modules specified by the AUTOSAR standard

3.2.3. EB (Elektrobit) modules

The following table lists all modules which are part of this release but are not specified by the AUTOSAR standard. These modules include tooling developed by EB or they may hold files shared by all other modules.

Module name	Module version	Supplier
MemAs	1.2.0	Elektrobit Automotive GmbH

Table 3.2. Modules not specified by the AUTOSAR standard

3.2.4. MCAL modules and EB tresos AutoCore OS

For information about MCAL modules and OS, refer to the respective documentation, which is available as PDF at `$TRESOS_BASE/doc/3.0_EB_tresos_AutoCore_OS` and `$TRESOS_BASE/doc/5.0_MCAL_modules`¹. It is also available in the online help in EB tresos Studio. Browse to the folders `EB tresos AutoCore OS` and `MCAL modules`.

3.3. Module release notes

3.3.1. Crc module release notes

- ▶ AUTOSAR R4.0 Rev 3
- ▶ AUTOSAR SWS document version: 4.2.0
- ▶ Module version: 6.11.13.B439717

¹`$TRESOS_BASE` is the location at which you installed EB tresos Studio.

- ▶ Supplier: Elektrobit Automotive GmbH

3.3.1.1. Change log

This chapter lists the changes between different versions.

Module version 6.11.13

2021-06-25

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.12

2021-03-05

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.11

2020-10-23

- ▶ New feature implemented: ASR compliant CRC64
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.10

2020-06-19

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.8

2020-02-21

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.7

2019-10-11

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.6

2019-06-14

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.5

2019-02-15

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.4

2018-10-25

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.3

2018-05-25

Module version 6.11.2

2018-02-16

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.1

2017-09-22

- ▶ Switch to MISRA-C:2012. It does not affect module functionality.

Module version 6.11.0

2017-03-31

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.10.0

2016-11-04

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.9.0

2016-05-25

- ▶ Added support for Debug & Trace with custom header file configurable via parameter `BaseDbgHeader-File`

Module version 6.8.0

2016-02-10

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.7.0

2015-11-06

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.6.0

2014-10-02

- ▶ Updated module version for ACG-7.2.0 release

Module version 6.5.0

2014-04-25

- ▶ Added support for function tracing via AUTOSAR Debugging

Module version 6.4.0

2013-09-13

- ▶ Removed the `NULL_PTR` check from all Crc API's except `Crc_GetVersionInfo()`

Module version 6.3.1

2013-06-14

- ▶ Changed the parameter type from `INTEGER` to `ENUMERATION` for `CrcXXTableSize` configuration parameters

Module version 6.3.0

2013-02-08

- ▶ `NULL` pointer check added for each Crc API
- ▶ Provided BSWMD for Crc

Module version 6.2.0

2012-10-12

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.1.0

2012-06-15

- ▶ Removed `CrcVersionInfoApi` from CRC configuration schema with respect to AUTOSAR version 4.0.3

Module version 6.0.1

2012-03-16

- ▶ Added labels for the CRC8H2F entries in the xdm
- ▶ Provided recommended configuration for Crc where CRC16 and CRC32 are enabled

Module version 6.0.0

2011-03-11

- ▶ Update to AUTOSAR Release 2.1

- ▶ Removed empty functions for hardware support
- ▶ Update to AUTOSAR Release 3.0
- ▶ Re-factored module implementation, performed code size and run time optimization
- ▶ Further module implementation clean up and maintainability improvements
- ▶ The availability of `Crc_GetVersionInfo()` depends now on the configuration
- ▶ Restructured the source code to increase compile time. All module configuration switches are now available via the module API
- ▶ ASCCRC-147 Fixed known issue: Changed storage class of `Crc_DataPtr` from `CRC_APPL_CONST` to `CRC_APPL_DATA`
- ▶ Module conforms to AUTOSAR R3.1
- ▶ Added support for CRC8
- ▶ Added support for two table sizes with either 16 or 256 entries
- ▶ Added labels and extended the parameter descriptions to improve user documentation
- ▶ Updated module to be compliant to AUTOSAR R4.0 Rev 2

3.3.1.2. New features

- ▶ No new features have been added since the last release.

3.3.1.3. EB-specific enhancements

This chapter lists the enhancements provided by the module.

- ▶ Configurable number of table entries

Description:

For the table-based CRC calculation methods two variants with either 16 or 256 table entries are supported and can be selected at configuration time.

Rationale:

This enhancement offers an intermediate algorithm that uses only a small precalculated table. This small precalculated table is faster than the full calculation at runtime and also offers a significantly smaller ROM footprint compared to the large table method.

The small table has a rather small ROM footprint but is still roughly four times faster than the runtime calculation method. The algorithm using the large table is approximately double as fast compared to the one using small tables at the cost of using 16 times more ROM to store the precalculated table entries.

- ▶ Support for detection of development error

Description:

Vendor-specific configuration parameter `CrcDevErrorDetect` is implemented to determine whether a development error shall be reported or not.

Rationale:

This enhancement enables the detection and reporting of development errors if a `NULL` pointer is passed as parameter in Crc APIs.

- ▶ Function tracing support via AUTOSAR Debugging

Description:

The module Crc supports tracing of function entry and exit via the EB Dbg module.

Function tracing records following parameters for each function:

- ▶ function name
- ▶ values of the function arguments
- ▶ point in time of function invocation
- ▶ point in time of function termination
- ▶ return value of the function

3.3.1.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

- ▶ Hardware-based CRC (reference to product description: ASCPD-44)

Description:

The hardware-based CRC calculation is not supported in this generic Crc module. This feature may be added as part of a customization project by EB.

Requirements:

`SWS_Crc_00033`, `SWS_Crc_00045`, `SWS_Crc_00009`, `SWS_Crc_00010`, `ECUC_Crc_00025`, `ECUC_Crc_00026`, `ECUC_Crc_00031`, `ECUC_Crc_00030`

- ▶ The Crc module does not provide the `GetVersionInfo` as macro.

Rationale:

An implementation of `GetVersionInfo` as function is more robust.

Requirements:

SWS_Crc_00017

3.3.1.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- ▶ For this module no limitations are known.

3.3.1.6. Open-source software

Crc does not use open-source software.

3.3.2. Ea module release notes

- ▶ AUTOSAR R4.0 Rev 3
- ▶ AUTOSAR SWS document version: 2.0.0
- ▶ Module version: 5.12.14.B439717
- ▶ Supplier: Elektrobit Automotive GmbH

3.3.2.1. Change log

This chapter lists the changes between different versions.

Module version 5.12.14

2021-06-25

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.13

2021-03-05

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.12

2020-10-23

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.11

2020-06-19

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.9

2020-02-21

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.8

2019-10-11

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ ASCEA-459 Fixed known issue: Incorrect mapping of pointer variable leads to alignment issues at runtime

Module version 5.12.7

2019-06-14

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.6

2019-04-18

- ▶ ASCFEE-449 Fixed known issue: EA and EEP are no longer compatible due to AUTOSAR 4.0.3 / 4.2.-2 differences.



Module version 5.12.5

2019-02-15

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.4

2018-10-25

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.3

2018-06-20

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.2

2018-02-16

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.1

2017-09-21

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.12.0

2017-03-31

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.11.0

2016-11-04



- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.10.0

2016-05-25

- ▶ Added support for Debug & Trace with custom header file configurable via parameter `BaseDbgHeader-File`

Module version 5.9.0

2016-02-10

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.8.0

2015-11-06

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Implemented AUTOSAR Bugzilla RfC #58294: MEMIF_BLOCK_INCONSISTENT is returned for blocks not found in memory

Module version 5.7.0

2014-10-02

- ▶ Updated module version for ACG-7.2.0 release

Module version 5.6.0

2014-04-25

- ▶ Added support for function tracing and variable debugging via AUTOSAR Debugging

Module version 5.5.1

2013-09-13

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.5.0

2013-06-14

- ▶ Updated the code generation by reporting used EEPROM memory space to the user

Module version 5.4.0

2013-02-08

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.3.0

2012-10-12

- ▶ ASCEA-304 Fixed known issue: The `EaBlockOverhead` published parameter value is empty in EB tresos Studio 13.0
- ▶ Changed the top-level structure of the software-component description in the ARXML files from `/AUTOSAR/Ea` to `/AUTOSAR_Ea`

Module version 5.2.0

2012-06-15

- ▶ ASCEA-288 Fixed known issue: `Ea_MainFunction` is not invoked by `Rte`
- ▶ Added `EA_E_BUSY DET` reporting if module status is `MEMIF_BUSY`
- ▶ Changed development error name from `EA_E_NOT_INITIALIZED` to `EA_E_UNINIT` with respect to AUTOSAR 4.0.3

Module version 5.1.0

2012-03-16

- ▶ Updated naming scheme for `#defines` for symbolic name values to AUTOSAR 4.0 rev3 naming scheme
- ▶ Provided BSWMD for `Ea`

Module version 5.0.0

2011-09-30

- ▶ Initial AUTOSAR 4.0 version

3.3.2.2. New features

- ▶ No new features have been added since the last release.

3.3.2.3. EB-specific enhancements

This chapter lists the enhancements provided by the module.

- ▶ Configuration parameter `EaMainFunctionPeriod`

Description:

The vendor-specific configuration parameter `EaMainFunctionPeriod` is implemented to specify the period with which `Ea_MainFunction` shall be invoked by the BSW scheduler.

Rationale:

The BSW scheduler of the Rte requires the `PERIOD` attribute of a timing event which triggers a schedulable entity in the Basic Software Module description. The Rte calculates the activation time of the OS schedule table and the execution time of a BSW main function within an OS task.

- ▶ Reporting of used EEPROM memory space

Description:

The used EEPROM memory space with respect to the total Ea blocks configured is reported as an information message to users while code is generated if EEPROM size is sufficient. It is reported as an error message if the size is not sufficient.

Rationale:

This enhancement helps users to be aware of possible future problems if the EEPROM size is not sufficient.

- ▶ Function tracing support via AUTOSAR Debugging

Description:

The Ea module supports tracing of function entry and exit via the EB Dbg module.

Function tracing records following parameters for each function:

- ▶ function name
- ▶ values of the function arguments

- ▶ point in time of function invocation
- ▶ point in time of function termination
- ▶ return value of the function
- ▶ Variable Debugging

Description:

The Ea module supports debugging of variables that contain the module's state, job result, and status.

Variables available for debugging:

- ▶ `Ea_MainState` (of type `Ea_MainStateType`)
- ▶ `Ea_Status` (of type `MemIf_StatusType`)
- ▶ `Ea_JobResult` (of type `MemIf_JobResultType`)

3.3.2.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

- ▶ Initialization check in main function

Description:

If the `Ea_MainFunction` is called while the module is not yet initialized, the main function returns immediately without performing any functionality and without raising any DET error. This initialization check is always performed independent of the development error detection setting.

Rationale:

The SchM module may schedule the module's main function before the module is initialized. This would result in lots of DET errors during startup. Therefore `Ea_MainFunction` does not throw a DET error if the module is not yet initialized and simply returns in this case.

- ▶ No support for internal management operations

Description:

In AUTOSAR 4.0, a new job requested when Ea is `MEMIF_BUSY_INTERNAL` shall be rejected only if the internal management operation cannot be suspended or aborted. Otherwise, Ea shall accept the request and start the request after suspending/aborting the internal management operation. However, in the current implementation, the module status `MEMIF_BUSY_INTERNAL` is not considered.

(This deviation applies to the AUTOSAR R4.0.3 specification only.)

Requirements:

EA022, EA025, EA182, EA186, EA187, EA188, EA189, EA183, EA184, EA185, EA017, EA128, EA179, EA181, EA174, EA166, EA180, EA073

- ▶ `DataBufferPtr` type is `const uint8*` in `Ea_Write` API

Description:

In contrast to EA087, `DataBufferPtr` type in AUTOSAR EA SWS V2.0.0, is `const uint8*` in the `Ea_Write` API.

Rationale:

To make `Ea_Write` compatible with `NvM_WriteBlock`, `DataBufferPtr` type must be `const uint8*`. http://www.autosar.org/bugzilla/show_bug.cgi?id=55397 has been raised in AUTOSAR Bugzilla to fix the issue in Ea SWS.

Requirements:

EA087

- ▶ `Ea_GetVersionInfo` is not realized as a macro

Description:

In contrast to EA082 the function `Ea_GetVersionInfo` is not realized as a macro if source code for caller and callee of the function `Ea_GetVersionInfo` is available.

Rationale:

The support for this function is in line with EB-implementation. A macro is not used in any case.

Requirements:

EA082

3.3.2.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- ▶ No limitations are reported

3.3.2.6. Open-source software

Ea does not use open-source software.

3.3.3. Fee module release notes

- ▶ AUTOSAR R4.0 Rev 3
- ▶ AUTOSAR SWS document version: 2.0.0
- ▶ Module version: 6.14.13.B439717
- ▶ Supplier: Elektrobit Automotive GmbH

3.3.3.1. Change log

This chapter lists the changes between different versions.

Module version 6.14.13

2021-06-25

- ▶ Changed custom APIs: reject jobs for configured blocks
- ▶ Internal module improvement. This module version update does not affect module functionality.

Module version 6.14.12

2021-03-05

- ▶ ASCFEE-632 New feature: Added Support for drivers that contain vendorId and vendorApiInfix
- ▶ ASCFEE-641 New feature: Added Small Section Support
- ▶ ASCFEE-611 New feature: Added Initialize in loop
- ▶ Behavior improvement: Fee_WriteCustom() accepts a different size for non configured block than the already existing size.
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.14.11

2020-10-23

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ ASCFEE-606 New feature: Added Dynamic block length
- ▶ ASCFEE-612 Fee Blocks can be lost when an immediate block is erased in section 0



Module version 6.14.10

2020-06-19

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.14.8

2020-04-24

- ▶ ASCFEE-606 New feature: Added Fee_ReadCustom API
- ▶ ASCFEE-607 New feature: Caching reconfigured not-configured blocks

Module version 6.14.7

2020-02-21

- ▶ ASCFEE-585 Compatibility with FIs that requires only 32b aligned addresses. Internal module improvement. This module version update does not affect module functionality ASCFEE-603 Fixed known issue: Fee_WriteCustom() API does not work when the maximum number of blocks is used

Module version 6.14.6

2020-01-31

- ▶ ASCFEE-596 Fixed known issue: Fee Config ID is not generated correctly

Module version 6.14.5

2019-12-12

- ▶ ASCFEE-458 Write emergency block feature.
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ ASCFEE-592 Fixed known issue: Fee generates incorrect configurations that trigger continuous switches at run-time.

Module version 6.14.4

2019-10-11

- ▶ ASCFEE-547 API extension to cancel ongoing page erase.

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.14.3

2019-06-14

- ▶ ASCFEE-542 Fee API to write non-configured blocks
- ▶ Behavior improvement: If section erase fails, allow requested jobs to be executed, then retry.
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.14.2

2019-04-18

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ ASCFEE-486 Fixed known issue: FEE and FLS are no longer compatible due to AUTOSAR 4.0.3 / 4.-2.2 differences.

Module version 6.14.1

2019-02-15

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.14.0

2018-10-24

- ▶ Flexibility of memory layout (New features: Configurable number of sections, Switch not configured blocks, Freeze activities).

Module version 6.13.3

2018-10-23

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ ASCFEE-479 Fixed known issue: FEE shall support a virtual page size of 256 bytes configured.
- ▶ ASCFEE-469 Fixed known issue: User write failure occurs during section switch, which leads to an over-writing of already existing flash
- ▶ ASCFEE-463 ASCFEE-463 Removed license check for Fee.ConsistencyPatterns



- ▶ ASCFEE-497 Fixed known issue: Failure of an immediate user write request during a sections switch causes the Fee to get stuck
- ▶ ASCFEE-485 Support for FEE configuration to use only a part of the FLS address space configured in the FLS driver.
- ▶ ASCFEE-456 Switch not configured blocks.
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ ASCFEE-510 Internal module improvement. This module version update does not affect module functionality

Module version 6.13.2

2018-05-25

Module version 6.13.1

2018-02-16

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.13.0

2017-09-22

- ▶ Update Job handling when internal operations are ongoing.
- ▶ Switch to MISRA-C:2012. It does not affect module functionality.

Module version 6.12.0

2017-03-31

- ▶ ASCFEE-412 Fixed known issue: Fee could perform an out of bounds read access
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.11.1

2016-12-14

- ▶ ASCFEE-407 Fixed known issue: Fee could not compile with certain compilers

Module version 6.11.0

2016-11-04

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.10.1

2016-07-13

- ▶ Implemented data consistency pattern mechanism and additional robustness improvements

Module version 6.10.0

2016-05-25

- ▶ Added support for Debug & Trace with custom header file configurable via parameter `BaseDbgHeader-File`

Module version 6.9.0

2016-02-08

- ▶ Improved the configuration parameters descriptions

Module version 6.8.0

2015-11-06

- ▶ Improved Fee robustness during start-up. Removed callout function in case the status of the sections can not be read from flash memory and implemented erase reaction in Fee
- ▶ ASCFEE-323 Fixed known issue: An already erased section may be erased again
- ▶ Added support for code flash sectors according to Renesas vendor specific parameter
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Implemented AUTOSAR Bugzilla RfC #58294: MEMIF_BLOCK_INCONSISTENT is returned for blocks that are not found in memory

Module version 6.7.0

2015-06-19

- ▶ Improved the Fee robustness if flash failure occurs during a write / invalidate request. Improved the Fee robustness if the section headers are FEE_SECTION_INCONSISTENT and FEE_SECTION_FULL
- ▶ ASCFEE-297 Fixed known issue: Fee remains in IDLE state without notifying upper layers if flash driver is corrupted
- ▶ ASCFEE-305 Fixed known issue: The Fee module doesn't check the module status before a mode switch is performed
- ▶ Improved the Fee robustness regarding the detection of erased sections. Implemented the flash memory erase counter functionality
- ▶ Improved Fee robustness in case reading the internal management data of a block fails during start-up

Module version 6.6.0

2014-10-02

- ▶ Updated module version for ACG-7.2.0 release

Module version 6.5.0

2014-04-25

- ▶ Added support for function tracing and variable debugging via AUTOSAR Debugging

Module version 6.4.0

2013-09-13

- ▶ Moved `DBG_FEE_STATE()` in `Fee_Init()` to just before initialization is completed

Module version 6.3.0

2013-06-25

- ▶ ASCFEE-249 Fixed known issue: The module behavior may be undefined if `Fee_Init()` is preempted by `Fee_MainFunction()`
- ▶ ASCFEE-262 Fixed known issue: The function `Fee_JobErrorNotification()` incorrectly sets the Fee flash job result to `MEMIF_JOB_FAILED` if the job result is not equal to `MEMIF_JOB_CANCELED` or `MEMIF_JOB_PENDING`

Module version 6.2.1

2013-02-08

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.2.0

2012-10-12

- ▶ Changed `Fee_Init()` to an asynchronous interface
- ▶ ASCFEE-196 Fixed known issue: Compiler warnings may occur due to incorrect size of `FEE_SECTION_STATUS_ERASED` when the Fls published value `FlsErasedValue` is larger than one byte
- ▶ Changed the top-level structure of the software-component description in the ARXML files from `/AUTOSAR/Fee` to `/AUTOSAR_Fee`

Module version 6.1.0

2012-06-15

- ▶ Added new Development Error `FEE_E_INVALID_CANCEL` in case module status is `MEMIF_BUSY` from `Fee_Cancel`
- ▶ ASCFEE-183 Fixed known issue: `Fee_MainFunction` is not invoked by `Rte`

Module version 6.0.0

2012-03-16

- ▶ Updated naming scheme for `#defines` for symbolic name values to AUTOSAR 4.0 Rev 3 naming scheme
- ▶ Updated Fee with respect to AUTOSAR R4.0 Rev 2
- ▶ Provided BSWMD for Fee

Module version 5.0.0

2011-09-30

- ▶ Initial AUTOSAR 4.0 version

3.3.3.2. New features

- ▶ No new features have been added since the last release.

3.3.3.3. EB-specific enhancements

This chapter lists the enhancements provided by the module.

- ▶ Jump table and bootloader support

Description:

The global and static variables of the Fee are implemented as elements of a single structure. The Integrator is able to map these variables to a dedicated memory or Flash section of his choice.

Rationale:

This enhancement implements the HIS requirements HisFee0003 and HisFee0004 which enable the EB Fee to be used from multiple applications by using jump tables, e.g. for the use in a bootloader.

- ▶ Configuration parameter `FeeMainFunctionPeriod`

Description:

The vendor-specific configuration parameter `FeeMainFunctionPeriod` is implemented to specify the period with which `Fee_MainFunction` shall be invoked by the BSW scheduler.

Rationale:

The BSW scheduler of the Rte requires the `PERIOD` attribute of a timing event which triggers a schedulable entity in the Basic Software Module description. The Rte calculates the activation time of the OS schedule table and the execution time of a BSW main function within an OS task.

- ▶ Function tracing support via AUTOSAR Debugging

Description:

The module Fee supports tracing of function entry and exit via the EB Dbg module.

Function tracing records following parameters for each function:

- ▶ function name
 - ▶ values of the function arguments
 - ▶ point in time of function invocation
 - ▶ point in time of function termination
 - ▶ return value of the function
- ▶ Variable Debugging

Description:

The Fee module supports debugging of variables containing the module's state and the next action to be performed.

Variables available for debugging:

- ▶ Fee_State (of type Fee_State_t)
- ▶ Fee_NextState (of type Fee_State_t)

Fee_State_t is defined as an enum with the following possible states:

- ▶ FEE_UNINIT
- ▶ FEE_IDLE
- ▶ FEE_INIT_READ_HEADER0
- ▶ FEE_INIT_READ_HEADER1
- ▶ FEE_INIT_ERASE
- ▶ FEE_INIT_FILL_CACHE
- ▶ FEE_INIT_CHK_LASTBLOCK
- ▶ FEE_READ_BLOCKDATA
- ▶ FEE_WRITE_BLOCK_INFO
- ▶ FEE_WRITE_BLOCK_DATA
- ▶ FEE_WRITE_SECTION_ACTIVE
- ▶ FEE_WRITE_SECTION_FULL
- ▶ FEE_WRITE_SECTION_EMPTY
- ▶ FEE_WRITE_SECTION_NOT_EMPTY
- ▶ FEE_WRITE_SECTION_ERASE_COUNTER
- ▶ FEE_INVALIDATE
- ▶ FEE_SECTION_SWITCHING
- ▶ FEE_SS_READ_DATA
- ▶ FEE_SS_COPY_INFO
- ▶ FEE_SS_COPY_DATA
- ▶ FEE_SS_WRITE_COPIED
- ▶ FEE_SS_ERASE_SECTION
- ▶ FEE_HANDLE_SECTION_HEADER_WRITE_FAILURE
- ▶ FEE_WRITE_BLOCK_INFO_START_PATTERN
- ▶ FEE_WRITE_BLOCK_DATA_START_PATTERN
- ▶ FEE_WRITE_BLOCK_INFO_END_PATTERN
- ▶ FEE_WRITE_BLOCK_DATA_END_PATTERN

- ▶ `FEE_WRITE_SECTION_ACTIVE_END_PATTERN`
- ▶ `FEE_WRITE_SECTION_FULL_END_PATTERN`
- ▶ `FEE_WRITE_SECTION_COPIED_END_PATTERN`
- ▶ `FEE_WRITE_SECTION_EMPTY_END_PATTERN`
- ▶ `FEE_WRITE_SECTION_ERASE_COUNTER_END_PATTERN`
- ▶ `FEE_NUM_STATES`
- ▶ `FEE_STATE_INVALID`

▶ Production error reporting

Description:

The vendor-specific parameter `FeeProdErrorDetect` allows enabling or disabling the detection of production errors. In case production error detection is enabled and Fee internal operations fail production errors can be reported to DEM. For details regarding the production errors please refer to the EB user documentation.

Rationale:

If errors occur during the the internal operations, Fee does not have any mechanism to report these error to the upper layer. Production errors have been added to notify the user.

▶ Flash memory erase counter

Description:

The vendor-specific configuration parameter `FeeEraseCounter` enables or disables the `Fee_GetEraseCounterValue()` API. The interface returns the approximate number of times Fee sections have been erased. The counter represents the number of erase of both sections, it is not section specific. Under unexpected working conditions, if the flash data cannot be recovered, the value of the erase counter will also be lost.

Rationale:

This feature gives a possibility to estimate how many times the flash has been erased and the user can get information regarding the lifetime of the flash device.

▶ Cancel Section Erase

Description:

The vendor-specific configuration parameter `FeeCancelSectionErase` enables or disables the `Fee_CancelSectionErase()` API. The interface allows the user to cancel an ongoing section erase if the highest internal priority is not active.

Rationale:

This feature gives a possibility to stop an ongoing section erase and free Fee/FIs, if this is allowed by the internal state of Fee.

► Write emergency block

Description:

The vendor-specific configuration parameter `FeeCriticalBlock` give highest priority to a write job of the block it refers to. This block, when requested, is written immediately. A successful write of this block freezes Fee internally. An erase-immediate request for this block unfreezes Fee.

► FIs alignment compatibility

Description:

FIs needs aligned addresses: The vendor-specific configuration parameter `FeeUseBufferForJobs` enables or disables the compatibility with FIs drivers that require 32bit alignment for addresses that are passed as API parameters.

► `Fee_WriteCustom` API

Description:

The vendor-specific configuration parameter `FeeWriteCustomApi` enables the API `Fee_WriteCustom` that allows the user to write not-configured blocks.

► `Fee_ReadCustom` API

Description:

The vendor-specific configuration parameter `FeeReadCustomApi` enables the API `Fee_ReadCustom` that allows the user to read not-configured blocks.

► Caching reconfigured not-configured blocks

Description:

Newer instances of not-configured blocks that are found with reconfigured size will be cached at initialization.

► Allow Small Section

Description:

The vendor-specific parameter `FeeEnableSmallSectionSize` allows enabling or disabling the Fee Section size to be smaller than the total size of the blocks, in case the number of sections configured is more than 3 sections and the total size of the blocks fits in the $(\text{number of sections})/2$ sections.

Rationale:

When the number of sections configured is more than 3 it is not mandatory to have the size of the section larger than the total size of the blocks configured as the large number of sections will compensate for the small size of each section.

► Dynamic block length

Description:

With this feature, Fee can detect and switch valid blocks that have a different size in configuration. This means that for these blocks the configuration changed and the user wants to recover the existing data anyway. Otherwise their data would be lost. There are two ways the block length can be changed: by increasing the size or by decreasing the size. - If the block size is decreased in the current configuration Fee will restore as many data as it's requested, and sets the result to MEMIF_JOB_OK_SIZE_DECREASED. There is some extra data in the flash memory for the block, but the user doesn't have the space for it anymore, and it doesn't need it. - If the block size is increased in the current configuration Fee will restore the entire data from flash memory for this block and will add some padding bytes, in order to fill the user's bigger buffer. The padding byte value is the bitwise complement value of the last byte of data found in flash for this block. The user is responsible to manage this data recovery situation. The first successful write job for a reconfigured block will return everything to standard behavior concerning this feature for that particular block. Fee will switch these blocks always with the size that is written in flash and not by the current configuration. To use the feature, set the configuration parameters FeeDynamicBlockLength.

3.3.3.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

► Initialization check in main function

Description:

If the main function is called while the module is not yet initialized the main function returns immediately without performing any functionality and without raising any Det error. This initialization check is always performed independent of the development error detection setting.

Rationale:

The SchM module may schedule the module's main function before the module is initialized. This would result in lots of Det errors during startup. Therefore the module's main function does not throw a Det error if the module is not yet initialized and simply returns in this case.

► Variable tracing by AUTOSAR Debugging

Description:

Only the module's state is available for debugging. The state gives information about the module's status and the current action. As long as the module's state is only used by the Fee module, it is not declared in the module's header file `Fee.h`. No information related to the job result and the block meta information are available for debugging.

Requirements:

FEE130, FEE131, FEE132

- `DataBufferPtr` type is `const uint8*` in `Fee_Write` API

Description:

In contrast to FEE088 in AUTOSAR Fee SWS V2.0.0, `DataBufferPtr` type is `const uint8*` in `Fee_Write` API.

Rationale:

In NvM R4.0.3 the parameter `NvM_SrcPtr` of `NvM_WriteBlock()` is changed to `const uint8*`, to make `Fee_Write()` compatible with `NvM_WriteBlock()`, `DataBufferPtr` type must be `const uint8*`. http://www.autosar.org/bugzilla/show_bug.cgi?id=55397 has been raised in AUTOSAR Bugzilla to fix the issue in the Fee SWS.

Requirements:

FEE088

- Internal Management Operation Handling

Description:

Fee internal management operation management is according to AUTOSAR Fee SWS V4.3.0. Requirements related to internal management operations handling that are exclusive to previous AUTOSAR specs are not supported any more.

Requirements:

FEE179, FEE180, FEE181, FEE182

3.3.3.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- **Integration requirement: EB_INTREQ_Fee_0001**

Integration restriction and recommendation

Description:

The EB memory stack modules NvM, Ea, and Fee make only limited use of the callback calls from their underlying modules. This also means that callbacks from the Fls to the Fee are not synchronously forwarded to the NvM. During the integration one has to make sure that the NvM, Ea, and Fee main functions are only called from the same task context so that they cannot preempt each other.

Rationale:

This approach enables a simple and lock-free implementation resulting in smaller code.

► **Integration requirement: EB_INTREQ_Fee_0002**

Contiguous and ascending Flash sectors

Description:

The Flash sectors configured in `Fee_FlsSectorList` of the flash driver configuration should be contiguous if more than one sector is configured and they shall be in the ascending order of their addresses.

Rationale:

The logic used for calculating the Fee section size is dependent upon this configuration.

► Integrity of block management data

Description:

Integrity of the data block is ensured by writing management information before and after writing the actual user data. If an unexpected shutdown occurs during the writing of management data (block info) before the actual block write, the module retrieves the old block. If an unexpected shutdown occurs during the writing of management data (block number) after the actual block write, the module cannot retrieve that block's data. This block shall be reported as inconsistent.

Rationale:

The module ensures the integrity of blocks stored in flash by storing extra block management data.

► **Integration requirement: EB_INTREQ_Fee_0003**

Flash virtual page size re-configuration

Description:

If the flash virtual page size is reconfigured, the Fee module cannot retrieve the data blocks which are present in the flash.

Rationale:

Internal management information size and aligned block size are dependent on the virtual page size.

► **Integration requirement: EB_INTREQ_Fee_0004**

Flash erase value limitation

Description:

Only the first byte of the FIs published parameter `FlsErasedValue` is used by the Fee module.

Rationale:

The FIs erased value is not expected to be different from one byte to another.

► **Integration requirement: EB_INTREQ_Fee_0005**

Integration restriction and recommendation

Description:

During the integration, the responsible for the MemStack modules has to make sure that the MemStack modules main functions are only called from the same task context and only from one task context so that they cannot preempt each other.

Rationale:

This approach enables a simple and lock-free implementation resulting in smaller code.

► **Integration requirement: EB_INTREQ_Fee_0006**

Integration restriction and recommendation

Description:

During integration testing, the responsible for the MemStack modules must make sure that the `FEE_E_CANCEL_WHILE_MF` Det runtime error is never reported.

Rationale:

The error indicates that a call to `Fee_Cancel` was done while the `Fee_MainFunction` was ongoing and interrupted by the context which triggered the call to the `Fee_Cancel`.

3.3.3.6. Open-source software

Fee does not use open-source software.

3.3.4. MemAs module release notes

- ▶ Module version: 1.2.0.B439717
- ▶ Supplier: Elektrobit Automotive GmbH

3.3.4.1. Change log

This chapter lists the changes between different versions.

Module version 1.2.0

2020-04-09

- ▶ Fixed issue with Memory Stack Configurator changing NVM Priority block value when it's set to 128.
- ▶ Updated the default value for the new created Block Configuration via Memory Stack Configurator.
- ▶ Add Open source statement in the release notes

Module version 1.1.7

2020-03-05

Module version 1.1.6

2020-12-18

- ▶ Fixed problem with Memory Module Configurator overwriting block identifiers

Module version 1.1.5

2020-10-23

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 1.1.4

2020-06-19

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 1.1.3

2020-05-22

- ▶ Memory Stack Configurator doesn't copy value from "Number of NVRAM Copies for Block" to "Number of ROM Blocks" anymore.
- ▶ Fixed issue with Memory Stack Configurator changing NVM Priority block value when it's set to 128.
- ▶ Updated the default value for the new created Block Configuration via Memory Stack Configurator.
- ▶ Add Open source statement in the release notes

Module version 1.1.2

2020-02-21

- ▶ Implement support for NvMCryptoExtraInfoSize
- ▶ Fixed unexpected error occurring during unattended wizard run

Module version 1.1.1

2019-10-11

- ▶ Now Memory Stack Wizard calculates the NvMNvramDeviceId based on the FlsDriverIndex/EepDriverIndex

Module version 1.1.0

2018-10-26

- ▶ Added functionality for Memory Stack Automation, unattended wizard

Module version 1.0.14

2018-09-13

- ▶ ASCMEMAS-224 Fixed issue: MemAs will handle correctly the newly added NvM blocks

Module version 1.0.13

2018-06-22

- ▶ ASCMEMAS-199 Fixed issue: MemAs will ignore all NvBlockNeeds which are part of a NvmBlocDescriptor

- ▶ Added functionality to support multiple instances of Fee and Ea.

Module version 1.0.12

2017-12-15

Module version 1.0.11

2017-09-19

- ▶ Added functionality for the `IMPORTER_INFO` category of Fee's module parameters to set the `CALCULATED` value if the parameters are calculated inside the Memory Stack Editor

Module version 1.0.10

2017-03-03

Module version 1.0.9

2016-11-04

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 1.0.8

2016-10-19

Module version 1.0.7

2016-05-25

- ▶ Updated setting of `NvmBlockLength` which is now based on the `PER-INSTANCE-MEMORY-SIZE` of the corresponding `SWC-IMPLEMENTATION`
- ▶ Updated setting of `TargetBlockType` which now will be set to `FEE_BLOCK` only if both the `Fee` Module and the `Fls` Module are available otherwise it will be set to `EA_BLOCK` but only if both the `Ea` Module and the `Eep` Module are available.

Module version 1.0.6

2016-01-15

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 1.0.5

2014-04-25

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 1.0.4

2013-05-13

- ▶ ASCMEMAS-122 Fixed known issue: When duplicating the NvM block with block ID 1 the newly created block is write-protected
- ▶ ASCMEMAS-123 Fixed known issue: MemAs does not detect duplicate block names
- ▶ Stabilized the module for mass production usage

Module version 1.0.3

2013-02-08

- ▶ Added support for CRC8

Module version 1.0.2

2012-10-12

- ▶ Improved the module so that decreasing the number of `DataSelection` bits is reflected consistently

Module version 1.0.1

2012-05-29

- ▶ Changed the parameters `NvMBlockCrcType`, `NvMRamBlockDataAddress`, `NvMRomBlockDataAddress`, `NvMSelectBlockForReadAll`, and `NvMSelectBlockForWriteAll` of module NvM to be optional

Module version 1.0.0

2012-03-16

- ▶ Initial AUTOSAR 4.0 version

3.3.4.2. New features

- ▶ No new features have been added since the last release.

3.3.4.3. EB-specific enhancements

This module is not part of the AUTOSAR specification.

3.3.4.4. Deviations

This module is not part of the AUTOSAR specification.

3.3.4.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- ▶ For this module no limitations are known.

3.3.4.6. Open-source software

The MemAs module does not use open-source software.

3.3.5. MemIf module release notes

- ▶ AUTOSAR R4.0 Rev 3
- ▶ AUTOSAR SWS document version: 1.4.0
- ▶ Module version: 5.11.11.B439717
- ▶ Supplier: Elektrobit Automotive GmbH

3.3.5.1. Change log

This chapter lists the changes between different versions.



Module version 5.11.11

2021-06-25

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.11.10

2021-03-05

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.11.9

2020-10-23

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Add 2 new return types for new NvM Block Size Changed Feature.

Module version 5.11.8

2020-06-19

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.11.6

2020-02-21

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.11.5

2019-10-11

- ▶ Increment patch version for ACG-8.7.1

Module version 5.11.4

2019-06-14

- ▶ Increment version number



Module version 5.11.3

2019-02-15

- ▶ MemIf function pointer arrays ordered by underlying device index parameter (FlsDriverIndex/EepDriverIndex)
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.11.2

2018-10-25

- ▶ Fix bug where MemIf generates incorrect API when multiple underlying devices are used.
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.11.1

2017-05-25

Module version 5.11.0

2017-09-22

- ▶ Fix bug where MemIf generates incorrect API when multiple underlying devices are used.

Module version 5.10.0

2016-11-04

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.9.0

2016-05-25

- ▶ Added support for Debug & Trace with custom header file configurable via parameter `BaseDbgHeaderFile`

Module version 5.8.0

2016-02-05

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.7.0

2015-11-06

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.6.0

2015-02-20

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.5.0

2014-10-02

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.4.0

2014-04-25

- ▶ Added support for function tracing via AUTOSAR Debugging

Module version 5.3.2

2013-06-14

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.3.1

2013-02-08

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 5.3.0

2012-10-12

- ▶ Changed the top-level structure of the software-component description in the Arxml files from /AUTOSAR/MemIf to /AUTOSAR_MemIf

Module version 5.2.0

2012-06-15

- ▶ Removed MEMIF_E_INCOMPATIBLE_VERSION development error as per AUTOSAR requirements 4.0.3

Module version 5.1.0

2012-03-16

- ▶ Provided BSWMD for MemIf

Module version 5.0.0

2011-09-30

- ▶ Initial AUTOSAR 4.0 version

3.3.5.2. New features

- ▶ No new features have been added since the last release.

3.3.5.3. EB-specific enhancements

This chapter lists the enhancements provided by the module.

- ▶ Function tracing support via AUTOSAR Debugging

Description:

The module MemIf supports tracing of function entry and exit via the EB Dbg module.

Function tracing records the following parameters for each function:

- ▶ function name
- ▶ values of the function arguments
- ▶ point in time of function invocation
- ▶ point in time of function termination

- ▶ return value of the function

3.3.5.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

- ▶ Header inclusion hierarchy

Description:

Contrary to the requirement MemIf002, the MemIf uses a modified header inclusion structure:

- ▶ The header `MemIf_Cfg.h` does not include `MemIf_Types.h`.
- ▶ The Fee and Ea headers are included from `MemIf_Cfg.h` instead of `MemIf.h`.

Rationale:

The inclusion of `MemIf_Types.h` does not depend on any configuration parameter. Therefore including this header in `MemIf_Cfg.h` is counter-intuitive. On the other hand, the inclusion of the Ea and Fee headers depends on the configuration and these inclusion directives need to be generated in the `MemIf_Cfg.h` file.

Requirements:

MemIf002

- ▶ `DataBufferPtr` type is `const uint8*` in `MemIf_Write` API

Description:

In contrast to the requirement MemIf040 in AUTOSAR MemIf SWS V1.4.0, `DataBufferPtr` type is `const uint8*` in the `MemIf_Write` API.

Rationale:

To make `MemIf_Write()` compatible with `NvM_WriteBlock()`, the `DataBufferPtr` type must be `const uint8*`. See Bugzilla entry http://www.autosar.org/bugzilla/show_bug.cgi?id=55397.

Requirements:

MemIf040

3.3.5.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- ▶ For this module no limitations are known.

3.3.5.6. Open-source software

MemIf does not use open-source software.

3.3.6. NvM module release notes

- ▶ AUTOSAR R4.0 Rev 3
- ▶ AUTOSAR SWS document version: 3.2.0
- ▶ Module version: 6.17.22.B439717
- ▶ Supplier: Elektrobit Automotive GmbH

3.3.6.1. Change log

This chapter lists the changes between different versions.

Module version 6.17.22

2021-06-25

- ▶ ASCNVM-1438 Fixed known issue : NvM native block reference to not existing lower layer block
- ▶ ASCNVM-1427 Fixed known issue : Erroneous handling of redundant blocks configured with a crypto hook
- ▶ Internal module improvement. This module version update does not affect module functionality.

Module version 6.17.21

2021-04-26

- ▶ ASCNVM-1418 Fixed known issue : The vendor-specific PreWrite verification causes that a redundant NVRAM block is not written
- ▶ ASCNVM-1419 Feature Improvement : Return Pending for multicore blocks is not written
- ▶ Updated default values restoring according to ASR R20-11

Module version 6.17.20

2021-03-05

- ▶ New feature : Change behavior for PowerOnReset handling of NvM admin header
- ▶ Internal module improvement. This module version update does not affect module functionality.
- ▶ ASCNVM-1410 Fixed known issue : The NvM generator does not generate correct NVM_ASSERT_STC statements if array is used
- ▶ Internal module improvement. This module version update does not affect module functionality.
- ▶ ASCNVM-1414 Fixed known issue : NvM does not handle redundant blocks correctly during WriteAll

Module version 6.17.18

2020-10-23

- ▶ Internal module improvement. This module version update does not affect module functionality.
- ▶ New feature : Data recovery for Blocks with reconfigured length

Module version 6.17.17

2020-08-07

- ▶ Internal module improvement. This module version update does not affect module functionality.
- ▶ ASCNVM-1395 Fixed known issue : A deadlock occurs if a job for a block with an ID greater than 255 is canceled
- ▶ Internal module improvement. This module version update does not affect module functionality.
- ▶ Internal module improvement. This module version update does not affect module functionality.

Module version 6.17.15

2020-06-19

- ▶ Internal module improvement. This module version update does not affect module functionality.
- ▶ Added New Crypto Hooks functionality.
- ▶ ASCNVM-1388 Fixed known issue : Queuing mechanism does not work for queues which are greater than 256
- ▶ Added New API 'NvM_DisableBlockCheckMechanism' .
- ▶ Internal module improvement. This module version update does not affect module functionality.

Module version 6.17.12

2020-04-24

- ▶ Internal module improvement. This module version update does not affect module functionality.

Module version 6.17.11

2020-02-21

- ▶ ASCNVM-1367 Fixed known issue : NvM doesn't compile on 64 Bits platforms ASCNVM-1370 Fixed known issue : NvM_FirstInitAll does not set a final job result for Invalidated blocks. Internal module improvement. This module version update does not affect module functionality.

Module version 6.17.10

2019-10-11

- ▶ Asynchronous CRC calculation queue in FIFO order implemented
- ▶ CancelWriteAll operation changed. Pending block stopped if Fls operation has not started.
- ▶ Internal module improvement. This module version update does not affect module functionality.

Module version 6.17.9

2019-06-14

- ▶ ASCNVM-1343 Fixed known issue : References to callback functions NvMReadRamBlockFromNvCallback and NvMWriteRamBlockToNvCallback are generated when NvMBlockUseSyncMechanism is set to false
- ▶ Update 'WriteBlockOnce' feature according to ASR 4.3.1 .
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.17.8

2019-04-04

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ ASCNVM-1332 Fixed known issue : NvM API NvM_ASR40_SetDataIndex fails to compile on a particular configuration.

Module version 6.17.7

2019-02-15

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.17.6

2019-01-23

- ▶ Added feature: Update of NvM queue sizes
- ▶ Internal module improvement. This module version update does not affect module functionality.
- ▶ Usage of non continuous NvBlock identifier (redir/search)
- ▶ First initialization of NV blocks.
- ▶ Added user callout function for passed production errors.
- ▶ ASCNVM-1310 Fixed known issue : NvM can get stuck in an infinite loop during BlockCheck mechanism.

Module version 6.17.5

2018-10-25

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ ASCNVM-1279 Fixed known issue: Variables NvM_CalcCrc_CalcBuffer and NvM_ConfigurationId are mapped to a wrong section.
- ▶ ASCNVM-1285 Fixed known issue: AutoRepair block mechanism doesn't repair NV Blocks correctly.
- ▶ ASCNVM-1299 Fixed known issue: Block Specific Features 'AutoValidation' and 'CRCComparison' cannot work together

Module version 6.17.4

2018-05-25

Module version 6.17.3

2018-03-09

- ▶ ASCNVM-1263 Fixed known issue: AutorepairRAM for feature BlockCheck doesn't work as expected.

Module version 6.17.2

2018-02-16



- ▶ Added alternative for detecting software change (NvMSoftwareChangeCallout). Block1 that stores ConfigurationId has configurable size now. ConfigurationId range changed.
- ▶ Added user callout function for production errors.
- ▶ Added Background Block Check functionality.

Module version 6.17.1

2017-12-15

- ▶ NvM.xdm updated. Other VSMD violations are covered by deviations.
- ▶ ASCNVM-1202 Fixed known issue: Block size check is not working when multiple underlying modules of the same type are used
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ Add feature for 'CRC Comparison mechanism before write'. Add feature for 'Data Comparison mechanism before write'.
- ▶ Multicore extension for NvM. Add multicore extension for NvM_CancelJobs API.

Module version 6.17.0

2017-09-22

- ▶ Added PRAM APIs for Nvm block Read/Write/RestoreBlockDefaults operations
- ▶ Updated job handling when internal operations are ongoing.
- ▶ NvM reporting of DET errors changed to be in sync with ASR 4.2.2.
- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.16.0

2017-03-31

- ▶ Internal module improvement. This module version update does not affect module functionality

Module version 6.15.0

2016-11-04

- ▶ ASCNVM-1136 Fixed known issue: Multi-block operations do not process VALID blocks if NvM_SetRam-BlockStatus() API is disabled
- ▶ ASCNVM-1141 Fixed known issue: NvM could discard queued immediate requests when other immediate requests are cancelled
- ▶ ASCNVM-1143 Fixed known issue: NvM generates wrong service interface for ReadBlock and Restore-BlockDefaults when NvMEnableASR32ServiceAPI is enabled and generation mode is different than generate_asr32_swcd
- ▶ ASCNVM-1155 Fixed known issue: Single block callback is not called by NvM for cancelled requests done from another single block callback
- ▶ ASCNVM-1156 Fixed known issue: NvM_CancelWriteAll request could be lost if it is triggered while NvM_WriteAll is currently being processed
- ▶ ASCNVM-1157 Fixed known issue: Blocks configured with Ram Block Crc could unnecessarily be read from Nv memory after a SW reset
- ▶ ASCNVM-1158 Fixed known issue: NvM could ignore asynchronous single block jobs requested during startup or shutdown
- ▶ ASCNVM-1155 Fixed known issue: Single block callback is not called by NvM for cancelled requests done from another single block callback

Module version 6.14.0

2016-05-25

- ▶ ASCNVM-1088 Fixed known issue: NvM_RestoreBlockDefaults fails for blocks configured with explicit synchronization and initialization callback
- ▶ ASCNVM-1079 Fixed known issue: Single block requests do not use the provided temporary RAM for blocks configured with explicit synchronization
- ▶ ASCNVM-1093 Fixed known issue: Multi-block notification is not called by NvM if the multi-block request is interrupted by an immediate request
- ▶ ASCNVM-1095 Fixed known issue: The result of NvM_RestoreBlockDefaults is wrongly set to NVM_REQ_OK if the CRC cannot be calculated in one step
- ▶ ASCNVM-1099 Fixed known issue: NvM does not calculate CRC again if static block Id failed and retry is performed
- ▶ Changed processing of mirror retry mechanism according to AUTOSAR4.2
- ▶ Updated mechanism of reporting production errors for read requests
- ▶ ASCNVM-1106 Fixed known issue: NvM could perform less read/write retries than configured for a redundant block
- ▶ ASCNVM-1107 Fixed known issue: The result of NvM_ReadAll is incorrectly set to NVM_REQ_NOT_OK if blocks are read as INVALID

- ▶ Added vendor specific configuration parameter for configuring returned value by the lower layer for blocks not found in the memory
- ▶ ASCNVM-1110 Fixed known issue: The short name of the operation EraseNvBlock is wrongly exported by NvM
- ▶ ASCNVM-1116 Fixed known issue: NvM can write wrong data while restoring redundancy of a redundant block when lower layer becomes BUSY or BUSY_INTERNAL
- ▶ ASCNVM-1117 Fixed known issue: The result of a user single block request could be wrongly changed from NVM_REQ_PENDING during interruptions of a multi-block requests by immediate block user request
- ▶ Added support for Debug & Trace with custom header file configurable via parameter `BaseDbgHeader-File`
- ▶ ASCNVM-1113 Fixed known issue: Unnecessary warning could be reported by NvM if block size check is enabled
- ▶ Implemented `NvM_ValidateAll()` functionality according to AUTOSAR 4.2 software specifications
- ▶ ASCNVM-1121 Fixed known issue: Multiple calls of `NvM_CancelJobs` for the same block could lead to a wrong `NVM_REQ_CANCELED` result for the first processed job
- ▶ ASCNVM-1122 Fixed known issue: During read and restore requests, CRC is copied to a temporary RAM for blocks with `NvMUserProvidesSpaceForBlockAndCrc`
- ▶ ASCNVM-1119 Fixed known issue: The returned value of `NvM_CancelJobs` is `E_OK` even if the single block request cannot be cancelled
- ▶ Changed old memory sections used in NvM

Module version 6.13.0

2016-02-05

- ▶ Internal module improvement. This module version update does not affect module functionality
- ▶ ASCNVM-1061 Fixed known issue: Corrected the error messages issued by NvM when the configuration of block 1 is invalid
- ▶ Added a block-specific configuration parameter which allows the user to select the AUTOSAR version of the R-Port interfaces called by NvM
- ▶ ASCNVM-1066 Fixed known issue: NvM processes cancelled single-block requests
- ▶ ASCNVM-1071 Fixed known issue: Unexpected behavior could be observed when cancelling immediate requests
- ▶ ASCNVM-1081 Fixed known issue: Restore defaults requests do not copy CRC to the RAM block
- ▶ ASCNVM-1080 Fixed known issue: Write protection is wrongly enabled during read requests of blocks configured with `NvMWriteBlockOnce`
- ▶ ASCNVM-1077 Fixed known issue: Invalid `DataIndex` is accepted by `NvM_SetDataIndex`

- ▶ ASCNVM-1078 Fixed known issue: If configuration ID mismatch occurs, NvM does not process the extended runtime preparation correctly

Module version 6.12.0

2015-11-20

- ▶ Exported NvM service IDs to NvM users
- ▶ Implemented a new vendor specific configuration parameter NvMExportBlockLengths, which enables or disables the exporting of NVRAM block lengths to NvM users
- ▶ Added consistency checks regarding the pre-configuration of Configuration ID block
- ▶ ASCNVM-1049 Fixed known issue: NvM restores default data in case of invalidated blocks
- ▶ Improved the NvMExtraBlockChecks functionality regarding blocks with NvMUserProvidesSpaceForBlockAndCrc configuration parameter enabled. If an old NvM configuration is imported, after updating to this module version please manually change the NvMRamBlockDataAddress configuration parameter of the Configuration ID block from &NvM_ConfigurationId[0] to &NvM_ConfigurationId using a text editor.
- ▶ ASCNVM-1054 Fixed known issue: The size of the permanent RAM block configured for block 1 is not correctly calculated
- ▶ ASCNVM-1057 Fixed known issue: NvM_EraseNvBlock() shall not depend on job prioritization parameter

Module version 6.11.0

2015-06-19

- ▶ ASCNVM-1014 Fixed known issue: Data is copied to RAM during NvM_ReadBlock even if the NV block is invalid or inconsistent for blocks of type NVM_BLOCK_DATASET
- ▶ Improved the warning messages issued by the NvMExtraBlockChecks feature
- ▶ The queue length generated by RTE for NvM user is not correct
- ▶ Blocks configured with explicit synchronization have to be processed during multi-block operations
- ▶ Improved the handling of redundant NVRAM blocks
- ▶ ASCNVM-1033 Fixed known issue: CRC is not copied to user RAM during read requests

Module version 6.10.0

2015-02-25

- ▶ Implemented AUTOSAR Bugzilla RfC #62202
- ▶ ASCNVM-998 Fixed known issue: NvMSingleBlockCallback is not called when a single block request is cancelled

- ▶ ASCNVM-1009 Fixed known issue: Multi-block requests do not get preempted by an immediate block write request
- ▶ Added AUTOSAR 4.2.1 interface wrappers
- ▶ Implemented AUTOSAR Bugzilla RfC #52034
- ▶ ASCNVM-1017 Fixed known issue: Cancelled single-block requests are removed from the queues only after all other requests have been completely processed

Module version 6.9.0

2015-01-07

- ▶ ASCNVM-982 Fixed known issue: `NvM_CancelWriteAll()` cancels running `NvM_WriteAll()` as well as `NvM_ReadAll()` requests
- ▶ ASCNVM-988 Fixed known issue: User-defined multi-block callback function is not always called if `NvM_WriteAll` is queued and `NvM_CancelWriteAll` requested
- ▶ ASCNVM-987 Fixed known issue: Write verification size is not correctly calculated
- ▶ ASCNVM-994 Fixed known issue: After a `NvM_CancelWriteAll` request, the `BswM_NvM_CurrentJobMode` notification to BswM is not called when expected
- ▶ ASCNVM-996 Fixed known issue: `NvMRepeatMirrorOperations` configured to "0" may lead to an endless loop

Module version 6.8.0

2014-10-02

- ▶ ASCNVM-947 Fixed known issue: `NvM_ReadAll` for blocks of management type `NVM_BLOCK_REDUNDANT` does not read the second block and uses default data
- ▶ ASCNVM-948 Fixed known issue: `NvM_NvToRamCopyCallbackType` and `NvM_RamToNvCopyCallbackType` are referenced, but never defined
- ▶ ASCNVM-955 Fixed known issue: `NvM_ReadAll()` causes RAM data corruption for blocks with explicit synchronization enabled and permanent RAM configured
- ▶ ASCNVM-977 Fixed known issue: Data in permanent RAM might be incorrect for handling of improved redundancy
- ▶ Updated module version for ACG-7.2.0 release

Module version 6.7.0

2014-04-25

- ▶ Added support for function tracing and variable debugging via AUTOSAR Debugging

- ▶ Removed unnecessarily generated empty lines from the generated file
- ▶ ASCNVM-897 Fixed known issue: The result of blocks is updated during a read multi-block operation, even if they are not selected to be processed by `NvM_ReadAll`
- ▶ ASCNVM-903 Fixed known issue: `NvM_ReadAll()` returns `NVM_REQ_NOT_OK` if reading the configuration ID block (block 1) returns `NVM_REQ_REDUNDANCY_FAILED`
- ▶ ASCNVM-914 Fixed known issue: Conflicting attributes of `NvM_MainFunction` definition determines a Rte verification error
- ▶ ASCNVM-918 Fixed known issue: After executing `NvM_ReadAll()`, the result of block 0 (multi-block request) is set to `NVM_REQ_OK`, but the result of one block selected to be processed during `NvM_ReadAll()` may remain pending
- ▶ ASCNVM-921 Fixed known issue: Fixed values for `NvMRomBlockDataAddress` and `NvMRamBlockDataAddress` attributes of block 1 are not enforced by XDM check

Module version 6.6.0

2013-09-20

- ▶ ASCNVM-851 Fixed known issue: After a redundant block invalidation, the job result is `NVM_REQ_NOT_OK`
- ▶ ASCNVM-860 Fixed known issue: After a request to erase a redundant block, the job result is `NVM_REQ_NOT_OK`
- ▶ ASCNVM-867 Fixed known issue: Preprocessor error generated by missing memory sections in the NvM BSW module description
- ▶ ASCNVM-874 Fixed known issue: A compiler error may be reported due to missing memory sections in `NvM_ASR40_Bswmd.arxml`
- ▶ ASCNVM-825 Fixed known issue: Uninitialized variable `NvM_GlobalGenericStatus` can cause unexpected behavior in NvM module
- ▶ Added description of production errors to the module reference documentation
- ▶ ASCNVM-877 Fixed known issue: The job result `NVM_REQ_REDUNDANCY_FAILED` is still returned after a corrupted redundant block has been successfully written to NV memory with new data

Module version 6.5.0

2013-06-25

- ▶ ASCNVM-789 Fixed known issue: `NvM_SetRamBlockStatus()` is not synchronous
- ▶ ASCNVM-823 Fixed known issue: Writing a block fails when the Ea or Fee is busy with internal operations
- ▶ ASCNVM-829 Fixed known issue: An overflow of the array `NvM_Queue_Immediate[]` is not handled while `NvMJobPrioritization` and `NvMBlockUseSyncMechanism` are enabled

Module version 6.4.0

2013-02-08

- ▶ ASCNVM-717 Fixed known issue: The CRC calculation type CRC8 is not supported by the NvM module
- ▶ Implemented possible errors for the client server operations `NvMNotifyJobFinished` and `NvMNotifyInitBlock`
- ▶ Changed SWCD internal so that the Rte interface of `NvM_MainFunction` is generated without ASR32/ASR40 infixes
- ▶ ASCNVM-776 Fixed known issue: If no `NvMSingleBlockCallback` is configured, the job finished callbacks configured via Rte are not called

Module version 6.3.0

2012-10-19

- ▶ ASCNVM-701 Fixed known issue: If no `NvMInitBlockCallback` is configured, the initialization callbacks configured via Rte are not called
- ▶ Changed to restrict the use of configuration parameter `NvMUserProvidesSpaceForBlockAndCrc` for permanent RAM blocks only
- ▶ ASCNVM-716 Fixed known issue: Error message may occur with correct `NvMBlockJobPriority` and `Ea/FeeImmediateData` parameter configuration
- ▶ Changed the top-level structure of the software-component description in the arxml files from `/AUTOSAR/NvM` to `/AUTOSAR_NvM`
- ▶ Added AUTOSAR 3.2 support of Rte Interface and SWCD

Module version 6.2.0

2012-06-15

- ▶ Modified behavior of NvM when a block is locked and added new development error `NVM_E_BLOCK_LOCKED`
- ▶ Updated input parameters in `NvM_WriteBlock()`, C-interface and port interface
- ▶ ASCNVM-660 Fixed known issue: The multiplicity of some NvM parameters do not conform to the AUTOSAR 4.0 specification
- ▶ Changed the NvM layout and grouped configuration parameters in tabs
- ▶ ASCNVM-674 Fixed known issue: A warning or an Rte generation error may occur because the package `SwcBswMappings` is located at the wrong position in the Basic Software Module Description
- ▶ ASCNVM-687 Fixed known issue: Rte generation fails if API configuration class is 1 and Rte service port is true

- ▶ Added NvM BswM interaction feature

Module version 6.1.0

2012-03-16

- ▶ Changed implementation of `NvM_ReadBlock()` and `NvM_RestoreBlockDefaults()` to set the job result to `NVM_REQ_RESTORED_FROM_ROM` also
- ▶ Changed implementation of `NvM_WriteAll()` to process blocks with configuration parameter `NvMS-electBlockForWriteAll` set to true
- ▶ Changed the Det error in `NvM_GetVersionInfo` from `NVM_E_PARAM_DATA` to `NVM_E_PARAM_POINTER`
- ▶ Updated `NvM_SetDataIndex` to return `E_OK` in production mode for native and redundant blocks
- ▶ Added support for production error `NVM_E_QUEUE_OVERFLOW`
- ▶ Changed implementation of `NvM_ReadBlock()` to handle redundant blocks to recover failed NV block data from the uncorrupt block
- ▶ Updated `NvM_WriteAll` to write block 1 as the last block
- ▶ Added support for production error `NVM_E_WRITE_PROTECTED`
- ▶ Updated immediate request handling, ongoing `NvM_WriteAll` shall not be interrupted
- ▶ Added `Ecum_CB_NfyNvMJobEnd` as default configuration value of `NvmMultiBlockCallback`
- ▶ Added BSWMD for NvM
- ▶ ASCNVM-625 Fixed known issue: The production error symbolic names for the Dem events are defined twice and lead to compilation errors
- ▶ Improved redundant block handling
- ▶ ASCNVM-616 Fixed known issue: Write-once protection without effect
- ▶ ASCNVM-644 Fixed known issue: The explicit synchronization between application and NVRAM manager via ports is not supported

Module version 6.0.0

2011-09-30

- ▶ Initial AUTOSAR 4.0 version

3.3.6.2. New features

- ▶ No new features have been added since the last release.

3.3.6.3. EB-specific enhancements

This chapter lists the enhancements provided by the module.

► New mechanism : Cryptography Hooks for Read/Write NvM Blocks

Description: A new mechanism in which blocks can be written as encrypted into NV and can be decrypted immediately after Read.

CryptoHooks General Configuration:

New Parameter General : 'NvMEnableCryptoSecurityHooks

Description : Enables the usage of crypto security hooks for handling of NvM Blocks's data.

New Parameter General : 'NvMCryptoReadHook

Description : Configures the callback API to apply the corresponding crypto algorithm for a NvM Block after Read.

New Parameter General : 'NvMCryptoWriteHook

Description : Configures the callback API to apply the corresponding crypto algorithm for a NvM Block before Write.

Each Block can be selected to be processed by the Read/Write Hooks :

New Parameter per NvM Block : 'NvMEnableBlockCryptoSecurityHandling

Description : Enables crypto handling of NvM user data corresponding to this NvM Block.

New Parameter per NvM Block : 'NvMCryptoExtraInfoSize

Description : The parameter shall be considered when calculating the actual block size when calling lower layers. This information can be either a MAC, a Hash or a Crypto Initialization Vector plus a MAC.

► New API : NvM_DisableBlockCheckMechanism

Description: Enables/Disables the 'BlockCheck' mechanism.

Description:

When called with 'TRUE' the mechanism will be disabled.

When called with 'FALSE' the mechanism will be enabled.

As for default the NvM starts with the mechanism enabled.

► Data Comparison Mechanism

Description:



The vendor-specific parameter `NvMPreWriteDataComp` enables the use of the data comparison mechanism before a block is written.

With the `NvMPreWriteDataComp` enabled, when processing a Write Job, triggered from any NvM APIs(`WriteBlock`, `WritePRAMBlock`, `WriteAll`) the Block Reads from NV into an internal buffer and a comparison will be done with the data stored currently in RAM that is to be written. If the data is equal then the Write Job will be finished without actual writing but with `Job_OK`, but if the data is different, then the Write Job will proceed as normal.

Rationale:

This feature is useful to save EEPROM/FEE Flash from wear out. Also it improves runtime in configuration that might trigger NV Write jobs more often than the data is actually changed.

► Cancel Internal Operations of underlying modules

Description:

When processing a job with immediate priority the vendor-specific parameter `NvMCancelInternalOperations` enables the checking of status of all configured memory devices and a cancel request, `MemIf_Cancel`, will be issued for all devices with status different then `MEMIF_IDLE`.

Rationale:

This feature is needed to avoid concurrent access of two underlying modules to the same HW driver. For example an immediate job assigned to module `FEE_1` is pending to be executed but we have an additional underlying module, `FEE_2` which is already executing a job on the same HW driver. To be sure the immediate job is executed immediate we need to cancel the internal handling of `FEE_2` also.

► Hook feature

Description:

The vendor-specific parameter `NvMReadBlockHook` enables the use of a hook function for `NvM_ReadBlock()` / `NvM_ReadAll()`. The vendor-specific parameter `NvMWriteBlockHook` enables the use of a hook function for `NvM_WriteBlock()` / `NvM_WriteAll()`.

The hook feature is supported only for blocks configured with a permanent RAM.

If hook features are enabled, only NV block data is updated by users. Read hook function is called after reading NV data and write hook function is called before writing NV data.

Rationale:

Hook features are useful for application that needs encrypt/decrypt data.

► Extra block size checks

Description:

The vendor-specific parameter `NvMExtraBlockChecks` enables the compiler to check if the configured block lengths (including CRC if enabled by configuration) match the size of the RAM or ROM variables associated to the blocks if a size can be determined by the `sizeof()` C-function. The check is done at compile time and no additional resources are required.

Rationale:

Wrongly configured block sizes lead to hard-to-find problems during development. These checks help to find these misconfigurations.

- Enhanced production error reporting

Description:

An enhanced production error reporting mechanism has been introduced. This allows to configure the following options independently for each Dem event:

- Report production errors to the *Diagnostics Event Manager* (Dem).
- Report production errors to the *Development Error Tracer* (Det) as development errors.
- Do not report production errors at all.

If a production error is redirected towards the Det, you may configure the reported Det error-ID.

Rationale:

This enhancement implements the HIS requirements concerning fault operation and error detection: HisGen0007, HisGen0008 and HisGen0009.

- Optimized RAM consumption

Description:

A new vendor-specific parameter `NvmUserProvidesSpaceForBlockAndCrc` has been introduced to configure where the CRC is stored for each block in case of a permanent RAM block.

If set to true, NvM stores the CRC in the RAM block itself. In this case, the user of the block reserves memory directly behind the RAM block to be used by the NvM for storing CRC.

If set to false, NvM uses an internal buffer to store the block data and CRC before writing or reading from the Nv memory.

Rationale:

The data and CRC are stored together as single block in the underlying memory abstraction modules (Fee or Ea). Therefore, NvM needs to copy data and CRC to an internal buffer before writing to NV memory. The size of this internal buffer depends on the length of the largest block configured in NvM. Therefore,

enabling `NvmUserProvidesSpaceForBlockAndCrc` for large blocks saves RAM because the size of the internal buffer can be reduced to fit the smaller blocks. It also reduces runtime by not having to copy to and from the internal buffer.

► Unified RAM mirror and internal buffer

Description:

In order to reduce the RAM usage of the NvM module, the RAM Mirror (defined by AUTOSAR NvM SWS) and NvM's internal buffer share the same RAM space.

Rationale:

As stated in the NvM SWS, the RAM Mirror size shall be equal to the size of the largest NVRAM block configured with explicit synchronization enabled. By using the same memory location for the two buffers, the resource consumption of the NvM module is reduced.

Note:

If *Write Verification* or *Data Comparison Mechanism* is enabled for at least one NVRAM block, an additional RAM buffer (*Write Verification Buffer*) is allocated. The size of this buffer is equal to the maximum value of the `NVM_WRITE_VERIFICATION_DATA_SIZE` parameter.

► Enhanced the *Advanced Recovery* feature

Description:

The vendor-specific general parameter `NvmAdvancedRecovery` has been made block-specific so that it can be configured differently for each block.

If `NvmAdvancedRecovery` is set to true for a block, NvM ensures that irrespective of write-protection, data recovered from the default data of the configured ROM block or user-provided data (`NvmInitBlockCallback`) during a read operation is written to NV memory during the execution of `NvM_WriteAll`.

If `NvmAdvancedRecovery` is set to false for a block, NvM does not automatically write the recovered data to NV memory if the block is write-protected or data has been recovered from the redundant NV block.

Rationale:

Implicit and explicit error recovery, as explained in the SWS, does not apply to blocks which are write-protected. Hence for such blocks, recovered data is not written to NV memory resulting in data loss.

Therefore, enabling `NvmAdvancedRecovery` does recover the block data when implicit error recovery is not possible.

► Improved handling of redundant blocks

Description:

The handling of redundant NVRAM blocks during a read operation has been improved to read both blocks associated with a redundant block. If one copy has been read successfully and the other copy is detected as corrupted, the job result is set to `NVM_REQ_OK` or `NVM_REQ_REDUNDANCY_FAILED`, depending on the value of the configuration parameter `NvMRedundantRecovery`.

If both copies of a redundant block have been read successfully, additionally the CRCs of both copies are compared. If this comparison fails, the block result is set to `NVM_REQ_OK` or `NVM_REQ_REDUNDANCY_FAILED` depending on the value of the configuration parameter `NvMRedundantRecovery`.

During the processing of `NvM_WriteBlock`, both NV blocks of a redundant block are always written. During the processing of `NvM_WriteAll`, both NV blocks of a redundant block are written only if the RAM block is marked as *VALID* and *CHANGED*.

If only one NV block of a NVRAM Block of the type `NVM_BLOCK_REDUNDANT` has been previously read successfully, no error is reported to DEM and the redundancy will be restored during the next `NvM_WriteBlock` or `NvM_WriteAll` request. If the block is marked as changed or `NvM_WriteBlock` is currently being processed, both copies are written with data from RAM. If the block is not marked as changed the block is automatically repaired with data from the correct NV block.

► Configuration parameter `NvMMainFunctionCycleTime`

Description:

The vendor-specific configuration parameter `NvMMainFunctionCycleTime` is implemented to specify the period with which `NvM_MainFunction` shall be invoked by the BSW scheduler.

Rationale:

The BSW scheduler of the Rte requires the *PERIOD* attribute of a timing event which triggers a schedulable entity in the Basic Software Module description. The Rte calculates the activation time of the OS schedule table and the execution time of a BSW main function within an OS task.

► Configuration parameter `NvMUserHeader`

Description:

The vendor-specific configuration parameter `NvMUserHeader` is implemented to specify user-specific header files for providing user dependent definitions and declaration (either directly or indirectly).

Rationale:

The user has to define and declare symbols (RAM and ROM blocks, Callbacks) based on the requirement. The list `NvMUserHeader` allows users to include the header files containing user defined symbols.

► Function tracing support via AUTOSAR Debugging

Description:

The module NvM supports tracing of function entry and exit via the EB Dbg module.

Function tracing records following parameters for each function:

- ▶ function name
- ▶ values of the function arguments
- ▶ point in time of function invocation
- ▶ point in time of function termination
- ▶ return value of the function
- ▶ Variable Debugging

Description:

The NvM module supports debugging of variables that contain module's state, job results and status.

Variables available for debugging:

- ▶ `NvM_GlobalWorkingStatus` (16 bits bitfield):
 - ▶ bit 0: RAM data has been changed
 - ▶ bit 1: block write protected
 - ▶ bit 2: block is invalidated
 - ▶ bit 3: block is locked
 - ▶ bit 4: Rom data was loaded
 - ▶ bit 5: permanent write protection
 - ▶ bit 6: block address changed
 - ▶ bit 7: no loss of redundancy
 - ▶ bit 8: CRC mismatch
 - ▶ bit 9: temporary RAM block used
 - ▶ bit 10: block inconsistent
 - ▶ bit 11: static block ID check failed
 - ▶ bit 12: NvM mirror used
 - ▶ bit 13: `SetRamBlockStatusChangedFlag`
- ▶ `NvM_GlobalErrorStatus` (8 bits bitfield equivalent with `NvM_RequestResultType`)
- ▶ `NvM_GlobalGenericStatus` (8 bits bitfield):
 - ▶ bit 0: Initialization flag
 - ▶ bit 1: Cancel Write All flag

- ▶ bit 2: Main Function Processing flag
- ▶ bit 3: Polling Mode Ignore flag
- ▶ bit 4: Dynamic Cofiguration flag
- ▶ bit 5: Multi Request Keep Job Pending flag
- ▶ bit 6: Block request Canceled Flag
- ▶ `NvM_GlobalCallLevel` (of type `uint8`: store the call level of the currently processed state machine)
- ▶ Export block lengths

Description:

The vendor-specific parameter `NvMExportBlockLengths` enables or disables the exporting of lengths for all configured NVRAM blocks. The exported lengths correspond to the value of the `NvMNvBlockLength` configuration parameter of each block.

Rationale:

The configured block lengths represent internal information of the NvM module. By enabling this configuration parameter, NvM users can avoid duplicating information regarding the lengths of the NVRAM blocks.

- ▶ New mechanism : Data recovery for Blocks with reconfigured length

Description: A new mechanism in which blocks can be read from the NV memory even if their configured length has changed,

Description : When after a Read the lower layer returns the job result as `MEMIF_JOB_OK_SIZE_DECREASED` or `MEMIF_JOB_OK_SIZE_INCREASED` the NvM shall be able to forward the truncated data as if the Block was read successful. If Crc is configured for a block the NvM shall consider it passed and will not do it in this instance. The NvM shall report the the upper layer the job results `NVM_REQ_OK_BLK_DECREASED` or `NVM_REQ_OK_BLK_INCREASED`.

3.3.6.4. Deviations

This chapter lists the deviations of the module from the AUTOSAR standard.

- ▶ Parameter `NvMBlockUseCRCCompMechanism` shall have a multiplicity of "0..1".

Description:

Parameter `NvMBlockUseCRCCompMechanism` shall have a multiplicity of "0..1" as opposed to a multiplicity of "1" as specified by NVRAM Manager Release 4.3.0 says.

Rationale:

Because the parameter is optional, it can also have a multiplicity of "0".

Requirements:

ECUC_NvM_00556

► Description:

In contrast to the AUTOSAR NVRAM Manager specification 4.3.0, the Crc compare functionality shall be provided also to NVRAM blocks with parameter NvMCalcRamBlockCrc set to False.

Rationale:

The parameter NvMCalcRamBlockCrc has no influence over the CRC compare functionality.

Requirements:

NvM.Crc.Comparison_CalcRamBlock

► DEM error NVM_E_HARDWARE reporting is not supported.

Description:

In contrast to the AUTOSAR NVRAM Manager specification, the NVM_E_HARDWARE production error is not reported by NvM.

Rationale:

For MEMIF_JOB_FAILED, MEMIF_BLOCK_INCONSISTENT or a CRC mismatch reported by the lower layer, NvM is reporting extended production errors according to AUTOSAR specification. From architecture point of view NvM cannot detect HW errors as it is HW independent.

Requirements:

SWS_NvM_00835

► Configuration variant VARIANT-LINK-TIME is not supported

Description:

In contrast to the requirement NVM727, the NvM module does not support the configuration variant VARIANT-LINK-TIME. The VARIANT-LINK-TIME is designed for the use cases where parameters are fixed at link time.

Requirements:

NVM727

► NvMNvramDeviceId maximum configurable value is limited to 7

Description:

In contrast to requirement NVM035_Conf, the NvM module supports only `NvMNvramDeviceId` values up to 7 instead of 254.

Requirements:

NVM035_Conf

- ▶ The `NvMWriteVerificationDataSize` maximum configurable value is 65535

Description:

In contrast to the requirement NVM538_Conf, the NvM module allows a maximum value of 65535 instead of 65536 for the `NvMWriteVerificationDataSize` parameter.

Rationale:

http://www.autosar.org/bugzilla/show_bug.cgi?id=47033 has been raised in AUTOSAR Bugzilla to fix the issue in NvM SWS. Since the maximum value of block length(`NvMNvBlockLength`) is 65535, write verification performs on block length. Thus the maximum configurable value of `NvMWriteVerificationDataSize` shall be 65535.

Requirements:

NVM538_Conf

- ▶ The `NvMNvramBlockIdentifier` minimum configurable value is 1 instead of 2

Description:

In contrast to the requirement NVM481_Conf, the NvM module allows a minimum value of 1 for the `NvMNvramBlockIdentifier` parameter.

Requirements:

NVM481_Conf

- ▶ Operations in port interfaces for notifications

Description:

In contrast to the AUTOSAR specification, the operations of the client-server interfaces `NvMNotifyJobFinished` and `NvMNotifyInitBlock` specify the possible error `E_NOT_OK`. Therefore, the compatibility check in the Rte would fail if a port of the NvM refers to one of these interfaces that is connected to a port of a SwC that refers to a correct interface of the same name.

This problem does not occur in case the respective interfaces of the NvM are referred directly by the port declarations of the SwC.

Rationale:

The C-interfaces `NvM_SingleBlockCallbackFunction` and `InitBlockCallbackFunction` as specified in the AUTOSAR specification have `Std_ReturnType` as return type. To be able to link to the respective C-interfaces, the client-server interfaces `NvMNotifyJobFinished` and `NvMNotifyInitBlock` have been adapted by adding the possible error `E_NOT_OK`.

Requirements:

NVM735, NVM736

- Symbolic port name support

Description:

The port names provided by the NvM are not named by their numeric index of the blocks as suggested by the AUTOSAR NvM SWS. Instead the ports are postfixed by the symbolic name of the related configuration container.

Rationale:

With symbolic names, port names do not change when ports are deleted or inserted and renumbered. Therefore ports must not be re-connected.

- Production errors may be switched off

Description:

In contrast to the requirement NVM189, the reporting of production errors may be switched off by configuration.

Rationale:

See the rationale in the list of module enhancements related to *Enhanced production error reporting*.

Requirements:

NVM189

- `NvM_CancelWriteAll` (reference to product description: ASCPD-46)

Description:

In contrast to the AUTOSAR NVRAM Manager specification, the function `NvM_CancelWriteAll()` is implemented as a synchronous function instead of an asynchronous one.

Rationale:

Since `NvM_CancelWriteAll()` must not be queued as per requirement NVM420, `NvM_CancelWriteAll()` does not need to be an asynchronous function. This should be changed in the AUTOSAR NVRAM specification.

Requirements:

NVM458

- ▶ Polling mode (reference to product description: ASCPD-45)

Description:

The NvM and the memory stack modules Ea and Fee do not work reliably with interrupts calls made from the memory drivers Fls and Eep. The modules Ea, Fee, Fls, Eep must be configured so that Ea and Fee poll the underlying Fls and Eep drivers. Additionally the NvM, Ea, and Fee main functions must be called from one task so that they cannot preempt each other.

The configuration parameter `NvmPollingMode` is ignored. The callback functions `NvM_JobEndNotification()` and `NvM_JobErrorNotification()` can be used, although the configuration parameter `NvmPollingMode` is enabled. These functions are only linked to the user application if they are invoked.

Requirements:

NVM441,NVM440,NVM501_Conf

- ▶ Parameter `NvMNvramBlockIdentifier` shall accept minimum value 1.

Description:

Parameter `NvMNvramBlockIdentifier` shall have the range: `min = 1 max = 2^(16- NVM_DATASET_SELECTION_BITS) - 1` as Specification of NVRAM Manager Release 4.3.0 says.

Rationale:

The reserved block that stores the configuration identifier must have `NvMNvramBlockIdentifier = 1`, and it is preconfigured, therefore the lower boundary should be 1. The user can still only configure this parameter starting with value 2.

Requirements:

NVM481_Conf

- ▶ Parameter `NvMWriteVerificationDataSize` shall have multiplicity 0..1.

Description:

Parameter `NvMWriteVerificationDataSize` shall have multiplicity 0..1 because its availability is dependent on the value of parameter `NvMWriteVerification`.

Rationale:

If parameter `NvMWriteVerification` is configured `TRUE`, `NvMWriteVerificationDataSize` will be enabled. If parameter `NvMWriteVerification` is configured `FALSE`, `NvMWriteVerificationDataSize` will be disabled.

Requirements:

`NvM.NvMWriteVerificationDataSize_Conf`

- ▶ Vendor specific parameters shall not be added in alphabetical order.

Description:

Additional vendor specific parameter definitions (using `ParameterTypes`), container definitions and references shall NOT be added to the VSMD according to the alphabetical order.

Rationale:

The warning is known and accepted.

Requirements:

`EcucSws_1014`

- ▶ The parameter `NvMCompiledConfigId` size is extended to `uint64` size.

Description:

The parameter `NvMCompiledConfigId` shall have the same size with the `ConfigId` read from NV memory. The `ConfigId` read from NV memory is stored in the reserved Block 1, which has configurable size. Therefore `NvMCompiledConfigId` must have a maximum range of 0..9223372036854775807.

Rationale:

Size of `configId` block shall be configurable (not limited to 2 bytes). Please see ticket ASCNVM-1212.

Requirements:

`NVM492_Conf`

- ▶ Configuration id's comparison shall be done very early at `ReadAll`.

Description:

The comparison between the stored configuration ID and the compiled configuration ID shall be done as the first step of the job of the function `NvM_ReadAll` during startup, after reading Block1.

Rationale:

Should be done very early at ReadAll process, but not before reading Block1 from NvRAM, otherwise the comparison can not take place without knowing the stored configuration id (Block1's content).

Requirements:

NvM073

3.3.6.5. Limitations

This chapter lists the limitations of the module. Refer to the module references chapter *Integration notes*, subsection *Integration requirements* for requirements on integrating this module.

- ▶ Restriction on starting new jobs

Description:

A new asynchronous job request can be started only after the single-block callback notification for a previous request for the same block is received.

Rationale:

A new job request would be accepted even before the callback for a previous job is issued. But in such cases, the receiver of the notification cannot know for which request the callback was invoked.

- ▶ Limitation on R-Ports called by NvM

Description:

NvM can only invoke one version of the generated R-Port interfaces. This limitation refers to the following interfaces:

- ▶ NvMNotifyInitBlock
- ▶ NvMNotifyJobFinished
- ▶ ReadRamBlockFromNvm
- ▶ WriteRamBlockToNvm

Rationale:

It is possible to enable the generation of multiple AUTOSAR version service APIs and call-backs by configuration. In this case, NvM can not determine during run-time which version of the service APIs was called for triggering requests, and thus it can not invoke the same version of the call-back functions. The NvM user has the possibility to specify the expected call-back functions version using the NvMRPortInterfacesASRVersion configuration parameter.

- ▶ Limitation on single block callbacks for cancelled requests



Description:

If a pending single-block request for an NVRAM block is cancelled and another request for the same block is queued before the next invocation of `NvM_MainFunction()`, the single-block callback for the former (cancelled) request will not be called.

Rationale:

This limitation allows for a more efficient implementation, as NvM does not have to store the history of cancelled requests for each NVMRAM block.

3.3.6.6. Open-source software

NvM does not use open-source software.

4. ACG8 Memory Stack user guide

4.1. Overview

This user guide describes the memory stack of EB tresos AutoCore. From this user guide you will learn about the basic concepts of the memory stack. You will also learn how to configure the stack.

The memory stack consists of the following modules:

- ▶ the `NvM` module
- ▶ the `Crc` module
- ▶ the `MemIf` module
- ▶ the `Ea` module
- ▶ the `Fee` module
- ▶ the `Eep` module
- ▶ the `Fls` module

[Figure 4.1, “Memory stack overview”](#) shows the memory stack modules in the context of the AUTOSAR layered architecture.

[Section 4.2, “Background information”](#) explains the concepts of the memory stack in EB tresos AutoCore.

[Section 4.3, “Configuring the memory stack”](#) provides step-by-step instructions on how to configure the memory stack.

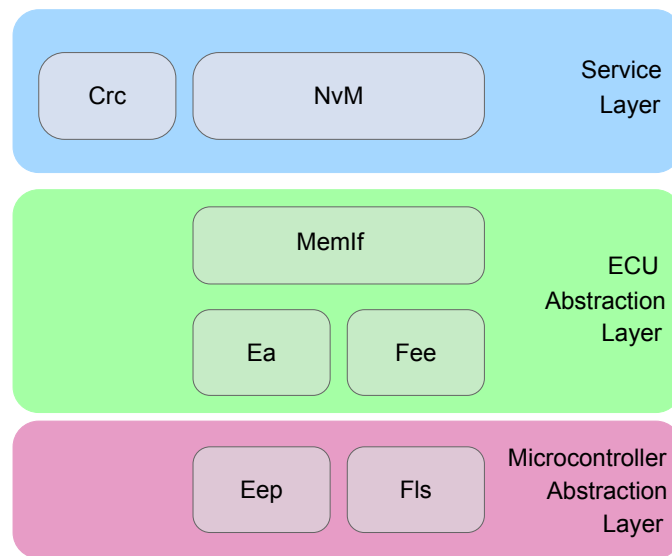


Figure 4.1. Memory stack overview

4.2. Background information

This chapter explains the concepts of the memory stack in EB tresos AutoCore. The following topics are described:

- ▶ [Section 4.2.1, “Functional description”](#) gives an overview of basic functions in the memory stack
- ▶ [Section 4.2.2, “NvM block model”](#) describes the various block types used
- ▶ [Section 4.2.3, “Block identification”](#) describes how blocks are identified
- ▶ [Section 4.2.5, “Block management”](#) describes the block management concept
- ▶ [Section 4.2.6, “Working with the different management types”](#) explains working with the block management types, including examples
- ▶ [Section 4.2.7, “Memory configuration identification”](#) describes the concept of the configuration ID and how this is processed by the **NvM**
- ▶ [Section 4.2.8, “Implicit and explicit synchronization”](#) describes the concepts of implicit and explicit synchronization between **NvM** and applications
- ▶ [Section 4.2.9, “Prioritization of RAM buffers”](#) describes the prioritization mechanism used by **NvM** for different types of RAM blocks.
- ▶ [Section 4.2.11, “Working with the abstraction modules Ea and Fee”](#) describes additional functional aspects particular to the **Ea** and **Fee**

4.2.1. Functional description

The memory stack modules provide services to store and maintain non-volatile (NV) data in an automotive environment. The `NVRAM Manager` module manages all data requests from different application tasks to access the data in the NV memory.

The `NVRAM Manager` works with the `Memory Abstraction Interface (MemIf)` module to access the `EEPROM Abstraction (Ea)` and `Flash EEPROM Emulation (Fee)`. This means that the `NVRAM Manager` is independent of the hardware memory and will support different types of NV memory devices. In addition, the `Crc` module provides routines to calculate a checksum that can be stored together with the data in NV memory.

The `NVRAM Manager` handles asynchronous and synchronous requests to manage data such as

- ▶ Read
- ▶ Write
- ▶ Erase
- ▶ Validation
- ▶ Status report

The data management is handled in terms of blocks. The following sections describe the block concept [Section 4.2.2, “NvM block model”](#), block identification [Section 4.2.3, “Block identification”](#) and block management [Section 4.2.5, “Block management”](#). [Section 4.2.6, “Working with the different management types”](#) gives some examples of how to work with the different block management types.

4.2.2. NvM block model

A block is a contiguous sequence of bytes. This can be subdivided into a *data block* and a *CRC block*. The data block contains the user data and the CRC block contains the CRC checksum calculated from the data in the corresponding data block.

Depending on the type of storage, there are different types of blocks:

- ▶ ROM block
- ▶ permanent RAM block
- ▶ temporary RAM block

If a data block is located in ROM, it is called *ROM data block*. A *ROM block* comprises one or more ROM data blocks. In this case the ROM data blocks must be located consecutively in the ROM. The ROM block is a grouping of data blocks that represents one or several instances of a given user/logical block in ROM. The configuration parameter `NvMRomBlockDataAddress` defines the address of the ROM block which is equal to the address of the first ROM data block. No CRC block can be used with a ROM block.

If a data block is located in RAM, it is called *RAM data block*. There are two types of RAM data block: the *permanent RAM data block* and the *temporary RAM data block*. The address of the permanent RAM data block is defined by the configuration parameter `NvMRamBlockDataAddress`. The address of a temporary RAM data block is not defined by a configuration parameter. This is provided to the `NVRAM Manager` by a parameter in the API services, for example, to read, write and restore block data.

Permanent RAM blocks and the RAM blocks used in explicit synchronization should be placed in the "POWER_ON_INIT" section. If there is not enough space available, there are 2 alternative solutions:

- ▶ 1) Before calling the `ReadAll` function call the `NvM_SetRamBlockStatus(BlockID, FALSE)` API for all the blocks that do not have the ram block placed in the "POWER_ON_INIT" section. This will make them invalid and will force the `NvM` to process them again during `NvM_ReadAll`.
- ▶ 2) Set the "NvMBlockUseSetRamBlockStatus" parameter to `FALSE` for all the blocks that do not have the ram block placed in the "POWER_ON_INIT" section. This will force `NvM_ReadAll` to always read them at warm start-up and will also always write them during `WriteAll`.

A temporary RAM cannot be used with PRAM asynchronous block requests (`NvM_ReadPRAMBlock()`, `NvM_WritePRAMBlock()`, and `NvM_RestorePRAMBlockDefaults()`). These interfaces shall be used for blocks that are configured with permanent RAM (`NvMRamBlockDataAddress`) or explicit synchronization.

If the use of CRC is configured for a block, it is called *RAM CRC* and is stored in permanent RAM for the duration of the application's run-time. By default, the RAM space for the CRC blocks is provided internally by the `NVRAM Manager`. However, the space for this can be reserved in the application's RAM area if the parameter `NvMUserProvidesSpaceForBlockAndCrc` is enabled in the configuration.

WARNING



Space for CRC must be provided if you use a temporary RAM block for a block with `NvMUserProvidesSpaceForBlockAndCrc` enabled

If you configure an `NVRAM` block with permanent RAM and you enable `NvMUserProvidesSpaceForBlockAndCrc` for that block, if you then use temporary RAM in an asynchronous single block request you must ensure that the size of the temporary RAM provided is large enough to include both the size of the data and the size of the CRC (`data length + CRC length`).

A *RAM block* is composed of one or more RAM data blocks and optionally of a RAM CRC block. The RAM block is a grouping of data blocks that represents one or several instances of a given user/logical block in RAM.

The configuration parameter `NvMRamBlockDataAddress` defines the address of the first permanent RAM data block in the RAM block.

1. If `NvMRamBlockDataAddress` is not defined (empty string), no permanent RAM data block is assigned to the RAM block. The user must assign a temporary RAM data block for each requested operation for this block.
2. If a temporary RAM data block is assigned by an API request and a permanent RAM data block is also configured, the temporary RAM data block has higher priority and is used for this specific operation.

In addition, the NvM maintains an *Administrative block*. This is used internally to store information about the block, for example, the status of the block, the write protection status for the block and the index for the block data to be used.

NOTE



NvM does not maintain a complete block mirror

The NvM maintains administrative information about a block. It does not maintain a complete block *mirror*. Operations on block data are carried out directly either using the permanent RAM block address configured or the temporary RAM address provided by the application.

The configuration parameter `NvMNvBlockLength` defines the size of the user data (in bytes) which is contained in a RAM data block and which must be stored in non-volatile memory. This size does not include the space for the CRC when the CRC is also to be stored. In the underlying abstraction layer, however, the block lengths configured define the size to include the space for both the block data and the CRC. The block length is configured in the `Ea` using configuration parameter `EaBlockSize`, and in the `Fee` using the parameter `FeeBlockSize`.

An *NVRAM block* describes the whole structure that is needed in order to manage and store an NV block. This is, in effect, the complete configuration that is needed for the NV block, including all copies in ROM and RAM and also the administrative block.

An overview of the `NVRAM Manager` block model is illustrated in [Figure 4.2, “NVRAM Manager block model overview”](#).

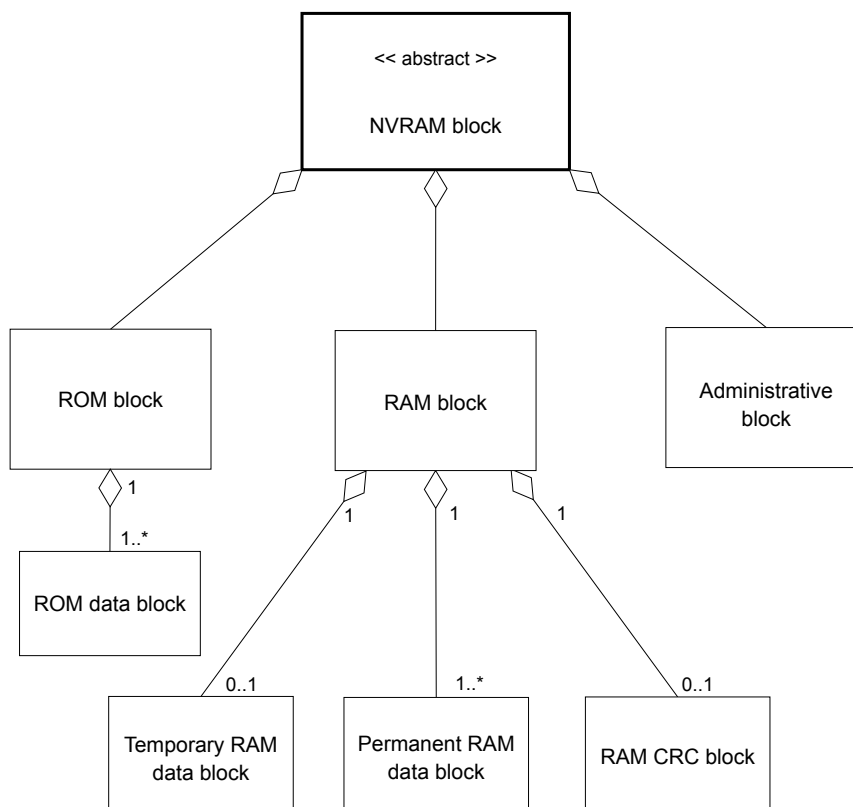


Figure 4.2. NVRAM Manager block model overview

4.2.3. Block identification

From the user application, the `NvMNvBlockIdentifier` is used to specify the NVRAM block to be accessed. When a block is configured with the parameter `NvMBlockManagementType` set to `NVM_BLOCK_REDUNDANT` or `NVM_BLOCK_DATASET` then the *data index* is used to specify the particular NV data block within the NVRAM block. The data index to be used can be managed by the user application using the API functions `NvM_GetDataIndex()` and `NvM_SetDataIndex()` that are provided for this purpose. In order to access a particular data block, the `NvMNvBlockIdentifier` is used by the NVRAM Manager to locate the block in the internal NVRAM Manager block descriptor table. The `NvMNvBlockBaseNumber` that is configured for the block is then combined with the data index in order to specify the logical block address. For details on how to access blocks in NV memory, refer to section [Section 4.2.4, “Identifying logical blocks in NV memory”](#).

The NVRAM Manager then passes this logical block address to the underlying modules `Ea` or `Fee`. Then, the physical address of the NV block is calculated from the logical block address by those modules. The NVRAM Manager has no information about the physical address of the data blocks in non-volatile memory. It only uses logical addresses defined by `NvMNvBlockBaseNumber` and the data index.

4.2.4. Identifying logical blocks in NV memory

The following parameters are used by the `NVRAM Manager` to calculate the logical block address of an NV block in the underlying modules:

- ▶ `NvMNVBlockBaseNumber`
- ▶ `NvMDatasetSelectionBits`

In the underlying abstraction modules `Ea` or `Fee`, the parameters `EaBlockNumber` or `FeeBlockNumber` are used.

The logical address of an NV block has 16 bits. This is calculated from the `NvMNVBlockBaseNumber` of the corresponding NVRAM block and the data index of this NV block. The parameter `NvMDatasetSelectionBits` defines the number of bits that can be used for block indexing. These bits are the least significant bits in the 16-bit logical address. The number of bits selected i.e. configured using `NvMDatasetSelectionBits`, must be enough to index the number of NV data blocks associated with the NVRAM block. If n bits are used for data selection, then the $16-n$ most significant bits of the 16 bit logical block address contain the `NvMNVBlockBaseNumber` shifted left by n bits. This is illustrated in [Figure 4.3, “Calculation of the logical block address”](#).

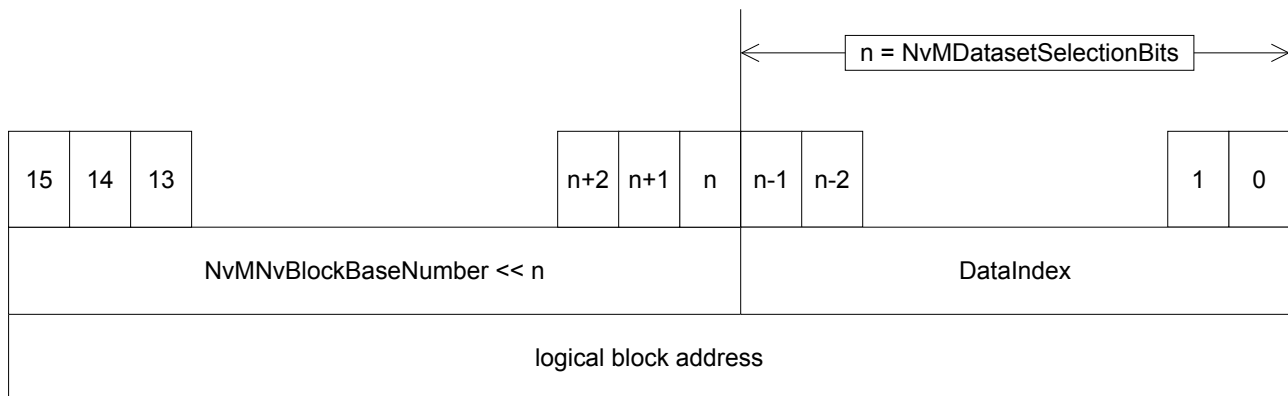


Figure 4.3. Calculation of the logical block address

4.2.4.1. Example

Assume that the parameter `NvMDatasetSelectionBits` is configured to 5 and NV block 3 of the NVRAM block 2 shall be addressed in EEPROM. In block 2, the `NvMNVBlockBaseNumber` is configured to be 6. Therefore, you have an `NvMNVBlockBaseNumber` of 6 and a data index of 3. Using the formula:

```
EaBlockNumber = (NvMNVBlockBaseNumber <<
                  NvMDatasetSelectionBits) + DataIndex
```

this results in a logical block address of 0x00C3.

4.2.5. Block management

As mentioned earlier, a RAM block or ROM block can have one or more associated data blocks. The *block management type* determines the number of these data blocks that can possibly be configured. There are three block management types:

- ▶ native block : this type consists of one data block only
- ▶ redundant block : this type consists of two data blocks
- ▶ dataset block : this type consists of one or several data blocks. The number of possible data blocks is determined by the parameter `NvmDatasetSelectionBits`

4.2.6. Working with the different management types

The following sections describe the different block management types and provide simple examples to explain these.

4.2.6.1. Native block

To define a native block, set the configuration parameter `NvMBlockManagementType` to `NVM_BLOCK_NATIVE`. Optionally, a ROM block can be configured, but it must contain exactly one ROM data block. The NVRAM block must contain exactly one NV block. This means: the parameter `NvMNvBlockNum` must be set to one.

4.2.6.1.1. Example: Instrument cluster

An instrument cluster in a vehicle calculates the average value of the speed, engine rotation and engine temperature from the beginning of the vehicle's life onwards. This means that when the vehicle is started, the current average values must be taken as initial values for the calculation of subsequent values. If the vehicle is stopped the last calculated average values must be stored permanently in non-volatile memory to be available for the next start of the vehicle. For this purpose, a structure `AvVehicleState` is defined with *Speed*, *EngineRotation* and *EngineTemperature*:

```
struct AvVehicleState
{
    unsigned char Speed;           /* unit: km/h */
    unsigned int EngineRotation;   /* unit: rpm */
    float EngineTemperature;       /* unit: degrees Celsius */
}
```

```
};
```

A constant `DefAvVehicleState` of the type `struct AvVehicleState` is defined, which provides the default values stored in ROM:

```
const struct AvVehicleState DefAvVehicleState={0,0,0};
```

A variable `CurAvVehicleState` of the type `struct AvVehicleState` is defined, which holds the current average values stored in RAM:

```
struct AvVehicleState CurAvVehicleState;
```

The consistency of the data stored in non-volatile memory shall be checked by an additional CRC32 value. No block-specific permanent RAM CRC block is required for the RAM block.

The variable `CurAvVehicleState` serves as permanent RAM data block for the NVRAM Manager. Therefore the configuration parameter `NvMRamBlockDataAddress` must be set to the address of `CurAvVehicleState`. To define the size of the RAM data block the parameter `NvMNvBlockLength` must be set to the size of `CurAvVehicleState`. This means that the user must determine the size of the `struct AvVehicleState`, which highly depends on the hardware and compiler used.

NOTE**The size of a structure can be different from the sum of the size of its members**

The actual size of a structure depends on the byte alignment of the `struct` members. This has to be determined and extra space included if necessary when configuring the block size.

For this example we assume that the size of an unsigned char is 1 byte, the size of an unsigned int is 2 bytes and the size of a float is 4 bytes. We also assume that the hardware allows only 2 byte aligned data. If we further assume that the struct members *Speed*, *EngineRotation* and *EngineTemperature* are located in this order within the memory, then there is a gap of 1 byte between *Speed* and *EngineRotation*. There is no gap between *EngineRotation* and *EngineTemperature*. Therefore the total size of `struct AvVehicleState` is 8 bytes. Thus `NvMNvBlockLength` must be set to 8. One method to determine the size of a variable is to pick up the value from the *MAP-file* if it is generated by the linker and if the linker provides the size information.

Since it is not required to assign a block specific permanent RAM CRC block to the RAM block, the configuration parameter `NvMCalcRamBlockCrc` is disabled.

However, since the CRC is to be checked when reading, the parameter `NvMBlockUseCrc` is enabled and the parameter `NvMBlockCRCType` is set to `NVM_CRC32` to enable 32-bit CRC calculation for this block.

The constant `DefAvVehicleState` serves as ROM data block for the NVRAM Manager. Therefore the configuration parameter `NvMRomBlockDataAddress` must be set to the address of `DefAvVehicleState`, because `DevAvVehicleState` is of the same data type as the RAM data block `CurAvVehicleState`. The size of the ROM data block is also `NvMNvBlockLength`. Since a native block is used, the ROM block must only contain one ROM data block, therefore `NvMRomBlockNum` must be set to one.

The `NvMNvBlockBaseNumber` for this NVRAM block is set to 3. The data index zero indicates that the first (and only) NV block is addressed. If the configuration parameter `NvMDatasetSelectionBits` is set to 4, then the logical address of the NV block is 0x30. Since the `CurAvVehicleState` data must be written to non-volatile memory each time the car is stopped, the configuration parameters `NvMBlockWriteProt` and `NvMWriteBlockOnce` must be set to false.

The memory layout of this example is illustrated in [Figure 4.4, “Example memory layout of native block”](#).

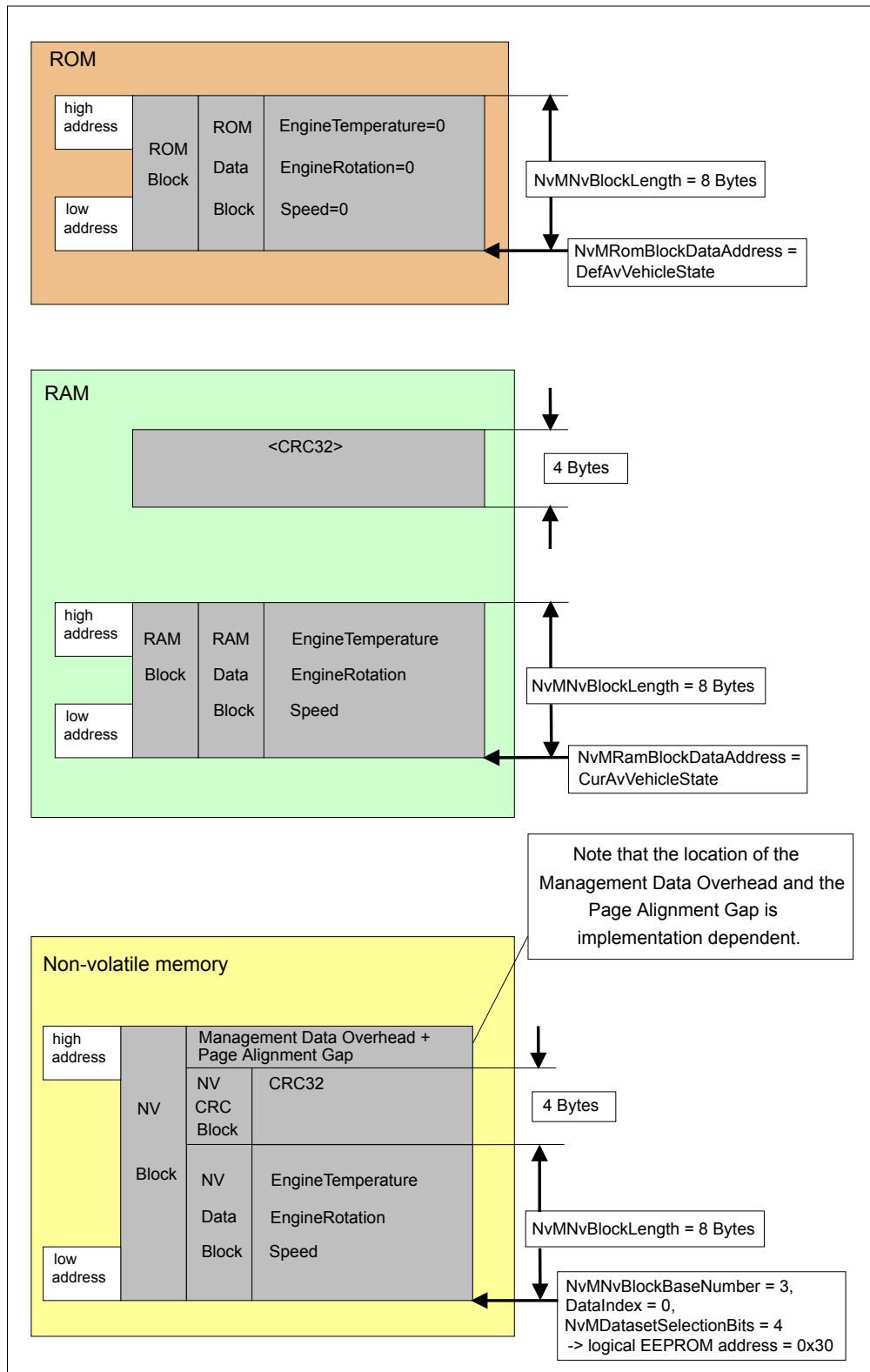


Figure 4.4. Example memory layout of native block

Since the NVRAM block shall be located in EEPROM, the configuration parameter `NvmTargetBlockReference` is set to `NvmEaRef` and the parameter `NameOfEaBlock` refers to the corresponding block in the `Ea` configuration. This means the block `EaBlockAvVehicleState` must be configured in the `EaBlockConfiguration` list.

4.2.6.2. Redundant block

To define a redundant block, set the configuration parameter `NvMBlockManagementType` to `NVM_BLOCK_REDUNDANT`. Optionally, a ROM block can be configured but it must contain exactly one ROM data block. The NVRAM block must contain exactly two NV blocks. This means: the parameter `NvMNvBlockNum` must be set to 2. The data in the second NV block is equal to the data in the first NV block and is used if an error occurs while reading the first one.

If reading one of the NV copies fails, or if reading both copies succeeds but a CRC mismatch occurs, the redundancy of the block will be lost and `NvM` will set the request result according to the value of the `NvMRedundantRecovery` configuration parameter, as follows:

- ▶ If `NvMRedundantRecovery` is set to `NVM_RECOVERY_ON_REQUEST`, the job result will be `NVM_REQ_REDUNDANCY_FAILED`. The user will be notified about the loss of redundancy and may call `NvM_WriteBlock()` in order to repair the block, or the block's redundancy will be restored automatically during the shut-down process.
- ▶ If `NvMRedundantRecovery` is set to `NVM_AUTOMATIC_RECOVERY`, the job result will be `NVM_REQ_OK`. The user will not be notified about the loss of redundancy and `NvM` will automatically restore the block's redundancy during the shut-down process.

The following scenarios describe `NvM`'s behaviour when restoring a block's redundancy:

- ▶ If the configured RAM block is marked as being `VALID/CHANGED`, both NV copies will be written with data contained in the RAM block during `NvM_WriteBlock()` or `NvM_WriteAll()`.
- ▶ If the user requested a `NvM_WriteBlock()` job, both NV copies will be written with data contained in the RAM block
- ▶ If a CRC mismatch occurred between the two NV copies, the corrupt copy will be written with data from the redundant NV copy
- ▶ If the RAM block is not marked as being `VALID`, the corrupt copy will be written with data from the correct NV copy. In this case, `NvM` will trigger a read operation in order to retrieve data from the correct copy.

4.2.6.2.1. Example: Instrument cluster extended

According to the example in [Section 4.2.6.1.1, "Example: Instrument cluster"](#), the average *Speed*, *EngineRotation* and *EngineTemperature* of a car must be calculated and stored in non-volatile memory. But now a re-

dundant copy of the data shall be available in non-volatile memory and the RAM CRC block shall be located immediately after the RAM data block. To use the redundant block the configuration parameter `NvMBlockManagementType` must be set to `NVM_BLOCK_REDUNDANT` and `NvMNvBlockNum` must be set to 2. To locate the RAM CRC block immediately after the RAM data block the configuration parameter `NvMUserProvidesSpaceForBlockAndCrc` must be enabled. The struct `AvVehicleStateCrc` must be defined to extend the original struct `AvVehicleState`, which contains the members `Speed`, `EngineRotation` and `EngineTemperature`. The extended struct contains a field large enough to hold the 32-bit CRC value. For example, add the member `unsigned long MyCrc32Buffer`.

```
struct AvVehicleState
{
    unsigned char Speed;                /* unit: km/h */
    unsigned int EngineRotation;        /* unit: rpm */
    float EngineTemperature;           /* unit: degrees Celsius */
};

struct AvVehicleStateCrc
{
    struct AvVehicleState MyRamData;
    unsigned long MyCrc32Buffer;
};
```

A variable `CurAvVehicleStateCrc` of the type `struct AvVehicleStateCrc` is defined, which holds the current average values stored in RAM and which serves as permanent RAM data block and RAM CRC block for the `NVRAM Manager`. The configuration parameter `NvMNvBlockLength` must still be set to 8 because the length of the user data still covers only the members `Speed`, `EngineRotation` and `EngineTemperature`. The size of the appended RAM CRC block is added automatically by the `NVRAM Manager` during the copy operations.

A constant `DefAvVehicleState` of the type `struct AvVehicleState` is defined, which provides the default values stored in ROM and which serves as ROM data block for the `NVRAM Manager`. Since it is of type `struct AvVehicleState`, it does not contain the member `MyCrc32Buffer`. A ROM block shall not have an associated CRC block.

The memory layout of this example is illustrated in [Figure 4.5, “Example memory layout of a redundant block”](#).

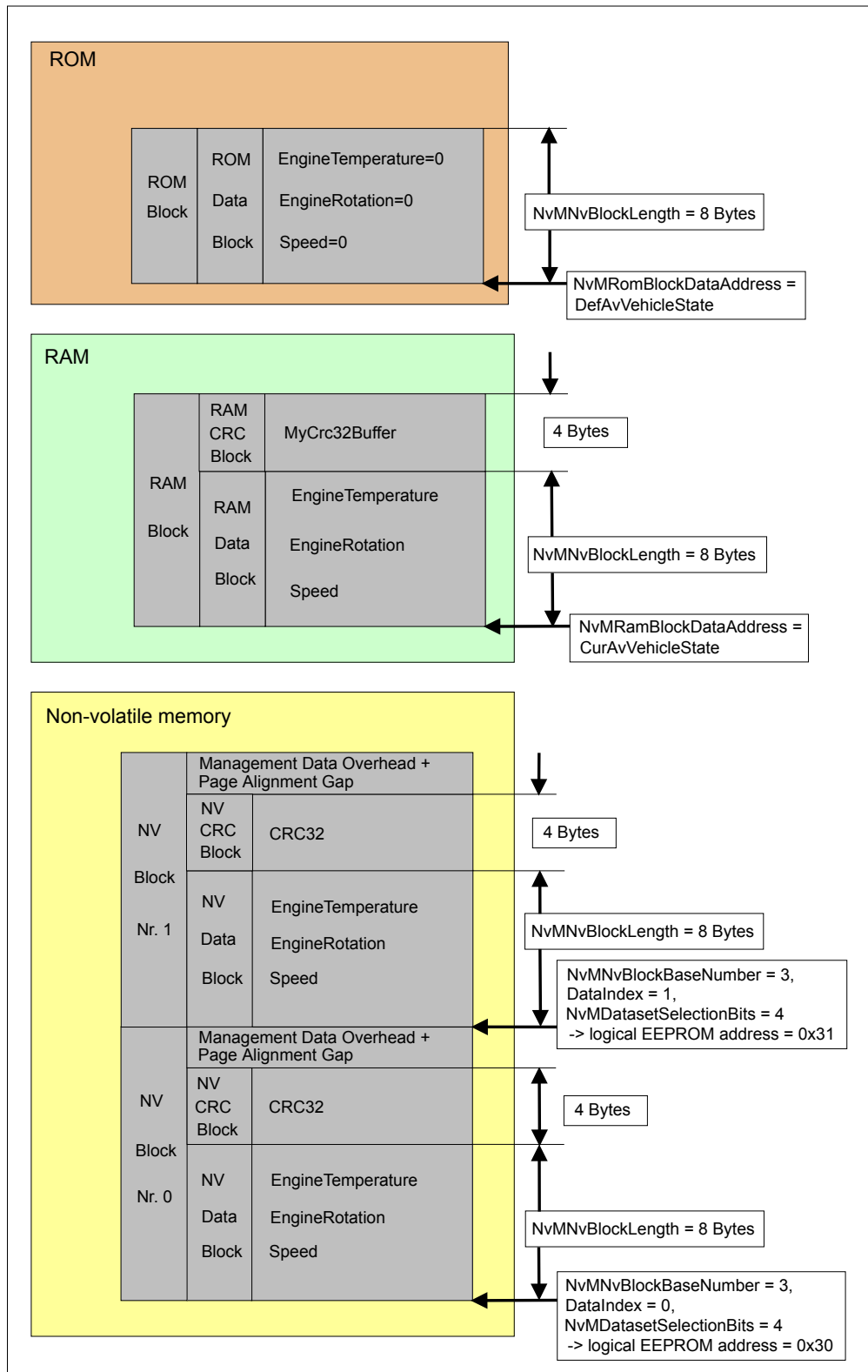


Figure 4.5. Example memory layout of a redundant block

4.2.6.3. Dataset block

To configure a dataset block, set the configuration parameter `NvMBlockManagementType` to `NVM_BLOCK_DATASET`. Optionally, a ROM block can be configured and it can consist of multiple ROM data blocks. The ROM data blocks are located consecutively in ROM.

The number of NV blocks is defined by the parameter `NvMNvBlockNum`. A specific NV block can be selected by the logical address which is combined from the `NvMNvBlockBaseNumber` and the data index. The first NV block is selected by data index zero and the last NV block is selected by data index `NvMNvBlockNum-1`.

4.2.6.3.1. Example

A car radio shall be able to store the frequency of ten radio stations in non-volatile memory. Six radio stations shall be reprogrammable by the user and four shall be permanent and implemented by the radio manufacturer. On the radio there are ten radio buttons labelled with numbers from 1 to 10. If button **k** is pressed, the radio should play the station transmitting at the stored frequency *k*. The buttons from 1 to 6 shall be used for the reprogrammable frequencies and the buttons from 7 to 10 for the permanent ones. A turning knob on the radio can be used to select an arbitrary frequency from 80.0 MHZ to 110.0 MHZ. Another button labelled **S** can be used to store the currently selected frequency to non-volatile memory and to assign it to radio button **k** if the radio button **k** is pressed immediately after the **S**. To store the ten frequencies in non-volatile memory a dataset block can be used. The consistency of the stored frequencies shall be ensured by a CRC16 calculation.

A frequency is represented by the data type *float* which is assumed to be 4 bytes in size. The currently selected frequency is implemented by the variable `CurFrequency`:

```
float CurFrequency;
```

The four permanent frequencies are implemented by the constant array `FixFrequencies`, which is initialized with the chosen frequencies:

```
const float FixFrequencies[4] = {82.5, 87.7, 93.3, 106.5};
```

- ▶ Set the configuration parameter `NvMBlockManagementType` to `NVM_BLOCK_DATASET`.
- ▶ Each reprogrammable frequency is stored in a separate NV data block. Therefore, set the parameter `NvMNvBlockNum` to 6.
- ▶ The four permanent frequencies are stored in ROM where each frequency is stored in a separate ROM data block. Therefore, set the parameter `NvMRomBlockNum` to 4.

Each field of array `FixFrequencies` serves as a single ROM data block. The variable `CurFrequency` serves also as a permanent RAM data block and the array `FixFrequencies` serves as multiple ROM data blocks.

- ▶ Set the parameter `NvMRamBlockDataAddress` to `&CurFrequency`.
- ▶ Set the parameter `NvMRomBlockDataAddress` to `&FixFrequencies[0]`. Set the parameter `NvMNvBlockLength` to 4 because the size of the data type of a frequency is 4 bytes.
- ▶ To use the CRC16 check for each NV block, enable the parameter `NvmBlockUseCrc`.
- ▶ Set the parameter `NvMBlockCRCType` to `NVM_CRC16`.
- ▶ Enable the parameter `NvmCalcRamBlockCrc` to assign a block specific RAM CRC block to this RAM block. The RAM CRC block is managed internally by the NVRAM Manager.

The data indices 0 to 5 are assigned to the six NV blocks and the data indices 6 to 9 are assigned to the four ROM Data blocks.

The memory layout of this example is illustrated in [Figure 4.6, “Example ROM and RAM memory layout of a dataset block”](#) and [Figure 4.7, “Example non-volatile memory layout of a dataset block”](#).

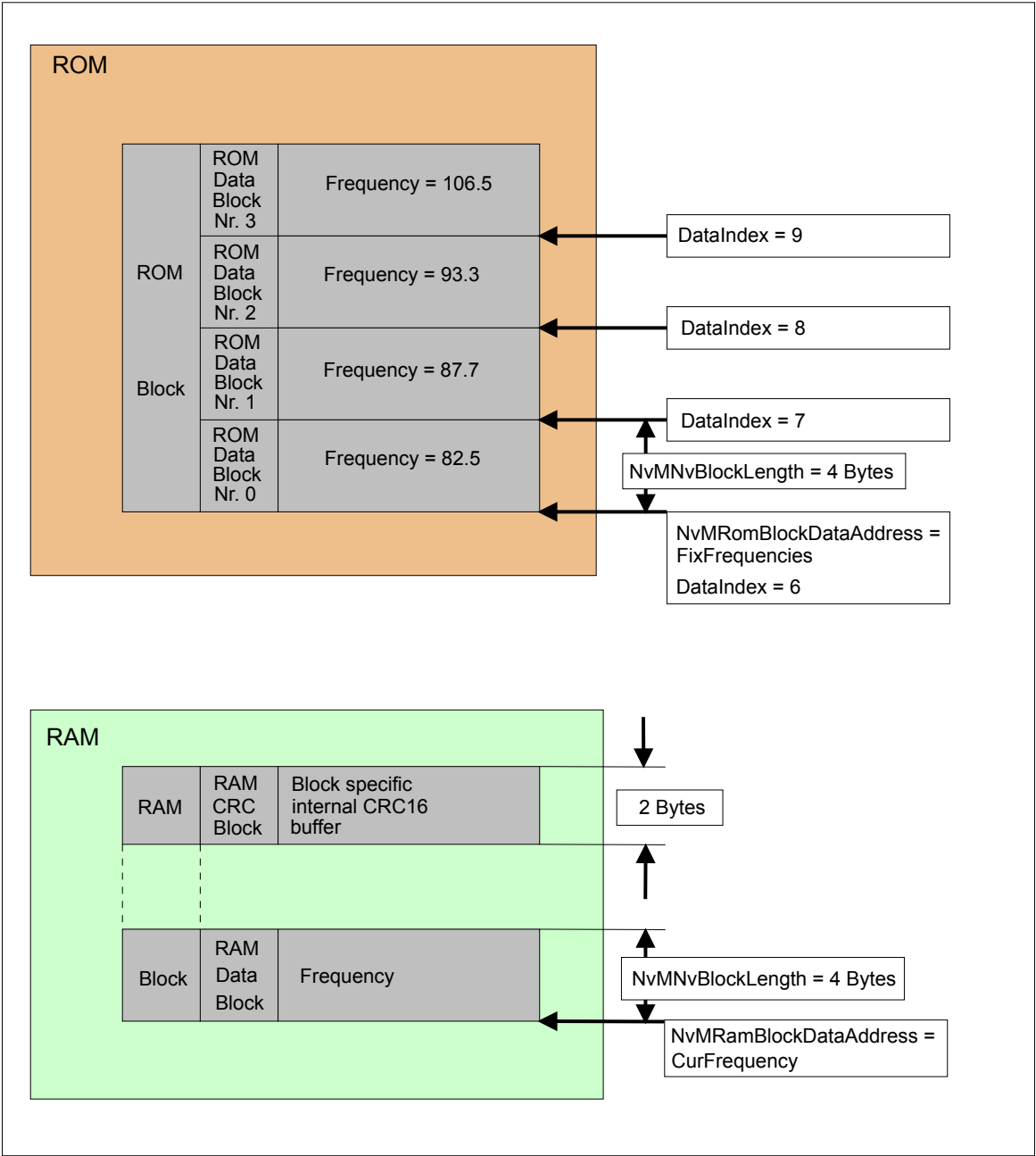


Figure 4.6. Example ROM and RAM memory layout of a dataset block

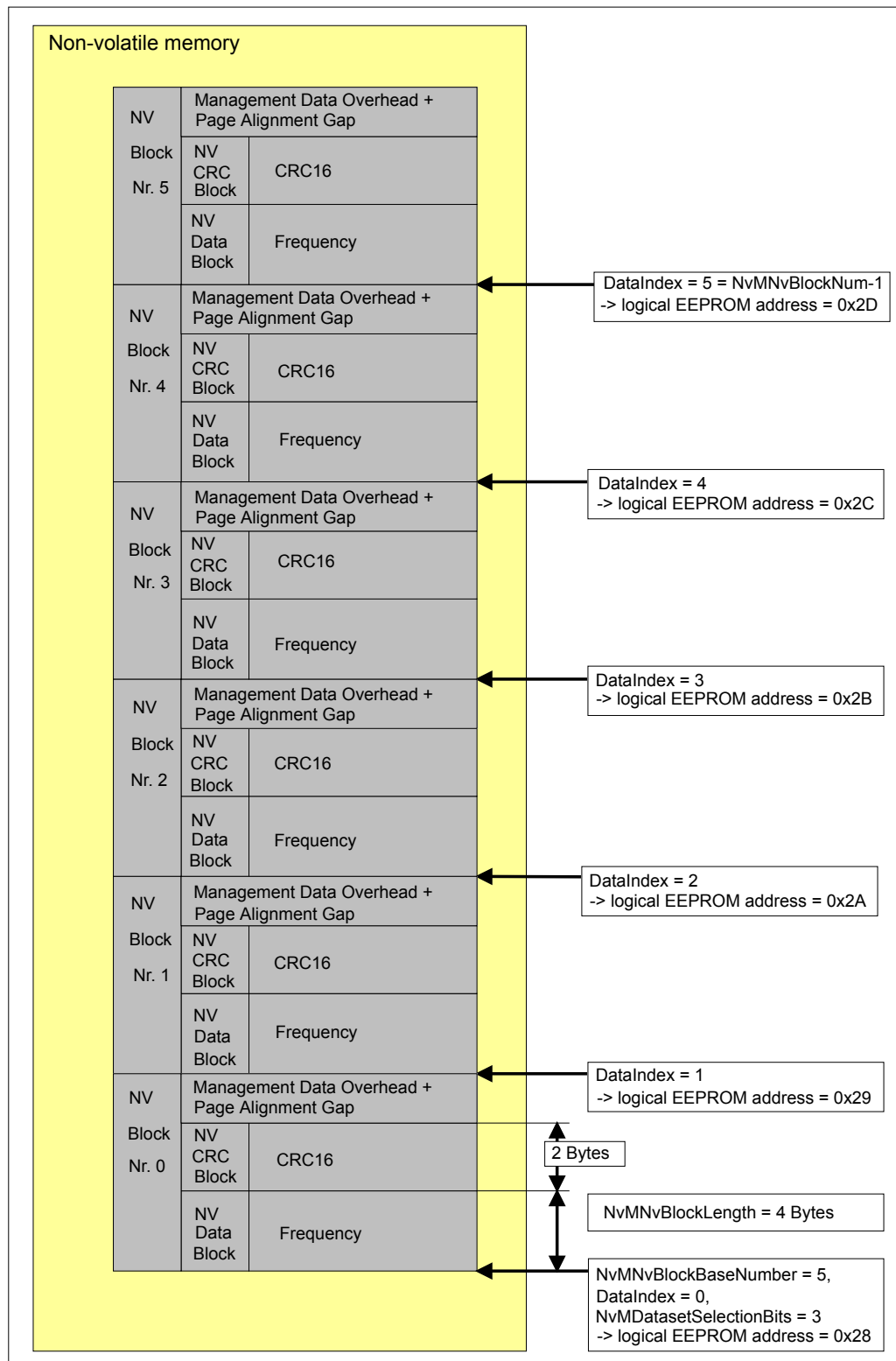


Figure 4.7. Example non-volatile memory layout of a dataset block

If the Name of the NVRAM block is set to `RadioStationFrequency`, then the symbolic name is `NvM-Conf_NvMBlockDescriptor_RadioStationFrequency`.

If the application software detects that the radio button `k` is pressed, the corresponding frequency can be copied from non-volatile memory to the variable `CurFrequency` by calling the NVRAM Manager services:

```
NvM_SetDataIndex(NvMConf_NvMBlockDescriptor_RadioStationFrequency, k-1);  
NvM_ReadBlock(NvMConf_NvMBlockDescriptor_RadioStationFrequency, NULL_PTR);
```

The second parameter `NvM_SrcPtr` is set to `NULL_PTR` because the data shall not be copied to a temporary RAM data block. Instead, the data shall be copied to the permanent RAM data block `CurFrequency` that is configured.

If `k` is lower than or equal to 6, then the frequency of NV block number `k-1` is read because the data index `k-1` is assigned to the NV block number `k-1`. If `k` is greater than 6 and lower or equal to 10, then the frequency of ROM data block `k-7` is read because the data index `k-1` is assigned to the ROM data block number `k-7`.

If the user selects a new frequency with the turning knob, this new frequency is stored in variable `CurFrequency`. If the user then presses the button `S` immediately followed by the radio button `k` the new frequency can be copied from `CurFrequency` to non-volatile memory by calling the NVRAM Manager services:

```
NvM_SetDataIndex(NvMConf_NvMBlockDescriptor_RadioStationFrequency, k-1);  
NvM_WriteBlock(NvMConf_NvMBlockDescriptor_RadioStationFrequency, NULL_PTR);
```

This only works if data index `k` is lower or equal to 6. Otherwise `E_NOT_OK` will be returned by `NvM_WriteBlock()`.

There are two possibilities to copy the data of the ROM data blocks to `CurFrequency`.

1.

```
NvM_SetDataIndex(  
NvMConf_NvMBlockDescriptor_RadioStationFrequency, i);  
NvM_ReadBlock(  
NvMConf_NvMBlockDescriptor_RadioStationFrequency, NULL_PTR);
```

The data index `i` must be in the range `6 <= i <= 9` to select a certain ROM data block.

2.

```
NvM_SetDataIndex(  
NvMConf_NvMBlockDescriptor_RadioStationFrequency, i);  
NvM_RestoreBlockDefaults(  
NvMConf_NvMBlockDescriptor_RadioStationFrequency, NULL_PTR);
```

The data index i must be in the range $6 \leq i \leq 9$ to select a certain ROM data block. If the data index i is in the range $0 \leq i \leq 5$, an NV block is selected and `NvM_RestoreBlockDefaults()` returns `E_NOT_OK` immediately.

4.2.7. Memory configuration identification

When you set up a configuration of blocks in the `NvM`, you also provide an identification number to be associated with your configuration. This value is known as the compiled configuration ID and you set this in the configuration parameter `NvMCompiledConfigId`. This value is stored as the ROM block for the first block in memory. This block is known as the configuration ID block and the first block in a configuration is reserved for this purpose.

NOTE



The `NvM` configuration identification relies on preconfigured values that shall not be changed

The `NvM` configuration identification implementation relies on the preconfigured values of the parameters `NvMRamBlockDataAddress` and `NvMRomBlockDataAddress` for the reserved `NvM` block 1. These two parameters are preconfigured with the values `&NvM_ConfigurationId` and `&NvM_CompiledConfigurationId` respectively. Do not change the preconfigured values.

During `NvM_ReadAll()`, the configuration ID is copied from the NV memory of block 1 to the associated RAM block. The value is then compared to the compiled configuration ID stored in ROM. The following list describes various scenarios that are handled by the `NvM` when the configuration ID block is read from NV memory.

- ▶ If the NV block 1 is invalid (see [Section 4.2.11.1.4, “Invalid block handling”](#)) or was never written (i.e. is empty), the new compiled configuration ID is copied to the associated RAM block, and the block is marked to be written to NV memory during `NvM_WriteAll()`. `NvM_ReadAll()` proceeds with copying data from the NV blocks to the corresponding RAM blocks.

The result of the multi-block operation is `NVM_REQ_NOT_OK`. The result of block 1 is `NVM_REQ_NV_INVALIDATED` or `NVM_REQ_INTEGRITY_FAILED`.

- ▶ If the NV block 1 cannot be read because of an error detected by the underlying layer or the `NvM` CRC check fails, the new compiled configuration ID is only copied to RAM if the parameter `NvMDynamicConfiguration` is set to `TRUE`. The `NvM` module then proceeds as in the case of a configuration ID mismatch described below.

The result of the multi-block operation is `NVM_REQ_NOT_OK`. The result of block 1 is `NVM_REQ_REDUNDANCY_FAILED`.

- ▶ If the NV block 1 is successfully read and a configuration ID mismatch occurs, the data layout has been changed, e.g. new blocks are added, some blocks are deleted, or the length of a block was changed. The subsequent processing is then dependent on the setting of the configuration parameter `NvMDynamicConfiguration`.

- ▶ If a mismatch occurs and the parameter `NvMDynamicConfiguration` is set to `TRUE`, the compiled configuration ID is loaded into the associated RAM block, and block 1 is marked to be written to NV memory during `NvM_WriteAll()`. In addition, default values are loaded from ROM for all blocks that are configured with the `NvMResistantToChangedSw` set to `FALSE`. For blocks with parameter `NvMResistantToChangedSw` set to `TRUE`, `NvM_ReadAll()` continues copying the NV data to RAM. Default values are only loaded from ROM if the block is found to be invalid.
- ▶ If a mismatch occurs and the parameter `NvMDynamicConfiguration` is set to `FALSE`, the new compiled configuration ID is not written automatically to NV memory during `NvM_WriteAll()`. `NvM_ReadAll()` continues copying the NV data to RAM. Default values are only loaded if a block is found to be invalid.

The result of the multi-block operation is set according to the individual block job results, and can have one of the following two values:

- ▶ `NVM_REQ_OK`: the status of all blocks is either `NVM_REQ_OK` or `NVM_REQ_BLOCK_SKIPPED`
- ▶ `NVM_REQ_NOT_OK`: the status of at least one block is neither `NVM_REQ_OK` nor `NVM_REQ_BLOCK_SKIPPED`

The result of block 1 is `NVM_REQ_NOT_OK`.

- ▶ If NV block 1 is successfully read and a configuration ID match occurs, `NvM_ReadAll()` proceeds with copying data from the NV blocks to the corresponding RAM blocks.

The result of the multi-block operation is set according to the individual block job results, and can have one of the following two values:

- ▶ `NVM_REQ_OK`: the status of all blocks is either `NVM_REQ_OK` or `NVM_REQ_BLOCK_SKIPPED`
- ▶ `NVM_REQ_NOT_OK`: the status of at least one block is neither `NVM_REQ_OK` nor `NVM_REQ_BLOCK_SKIPPED`

The result of block 1 is `NVM_REQ_OK`.

4.2.8. Implicit and explicit synchronization

The `NvM` module provides two types of synchronization mechanisms in order to ensure a reliable communication with applications: implicit synchronization and explicit synchronization.

The usage of either synchronization mechanism is configurable for each block.

4.2.8.1. Implicit synchronization

Implicit synchronization is the default mechanism used by the `NvM` module. The disadvantage of this mechanism is that synchronization between applications sharing the same resources (a RAM block or an NVRAM block) is difficult, as this is not handled by the `NvM` module.

A summary of the rules to which applications have to adhere, if implicit synchronization is used, is presented below:

- ▶ Single-block write requests (`NvM_WriteBlock()` and `NvM_WritePRAMBlock()`):

The application provides a RAM block with the data intended to be written to NV memory and then issues a write request to the `NvM` module.

After the write request is issued, the contents of the RAM block must not be modified until the request ends, but the RAM block can be read.

- ▶ Single-block read requests (`NvM_ReadBlock()`, `NvM_ReadPRAMBlock()`, `NvM_RestoreBlockDefaults()`, and `NvM_RestorePRAMBlockDefaults()`):

The application provides a RAM block that will be filled with data from NV memory and then issues a read/restore request to the `NvM` module.

After the read/restore request is issued, the contents of the RAM block must not be modified until the request ends.

- ▶ Multi-block write requests (`NvM_WriteAll()`):

Note: `NvM_WriteAll()` is issued only by `EcuM` during shut-down.

The application has to provide a RAM block with the data intended to be written to NV memory.

After the shut-down procedure is initiated, the contents of the RAM block must not be modified by the application anymore.

- ▶ Multi-block read requests (`NvM_ReadAll()`):

Note: `NvM_ReadAll()` is issued only by `EcuM` during start-up.

The application has to provide a RAM block that will be filled with data from NV memory.

After the start-up procedure is initiated, the contents of the RAM block must not be modified by the application until the processing of `NvM_ReadAll()` is completed.

Note: in addition to the conditions listed above for the multi-block operations `NvM_WriteAll()` and `NvM_ReadAll()`, when using the implicit synchronization mechanism the following two conditions must also be true:

- ▶ The block must be marked for being processed during start-up and/or shut-down (`NvMSelectBlockForReadAll` and `NvMSelectBlockForWriteAll` configuration parameters).

- ▶ A permanent RAM block must be configured (`NvMRamBlockDataAddress` configuration parameter).

4.2.8.2. Explicit synchronization

This synchronization mechanism has the advantage of not restricting the availability of RAM blocks during job processing. Depending on the specific configuration options, the RAM usage of the `NvM` module may increase, because the `NvM` module allocates an internal buffer with the size of the largest block configured for explicit synchronization. Data between the application and the internal buffer is transferred in both directions via callback routines, called by the `NvM` module.

The user-defined callback routines are called for both single-block and multi-block operations. `NvM` does not copy data internally in any case, even if a permanent RAM is configured. Data synchronization shall always be done in the user-defined callback functions.

A summary of the rules to which applications have to adhere, if explicit synchronization is used, is presented below:

- ▶ Single-block write requests (`NvM_WriteBlock()` and `NvM_WritePRAMBlock()`):

The application provides a RAM block with the data intended to be written and then issues a write request to the `NvM` module.

After the write request is issued, the RAM block can be modified until the callback routine is called.

The `NvM` module calls the routine configured via `NvMWriteRamBlockToNvCallback` in order to synchronize application data with the internal buffer of `NvM`.

The application can signal to the `NvM` module if the data was copied successfully or not by means of the return value of the callback function. If the return value of the user-defined callback function is `E_NOT_OK`, the `NvM` module will retry the operation `NvMRepeatMirrorOperations` times.

- ▶ Single-block read requests (`NvM_ReadBlock()`, `NvM_ReadPRAMBlock()`, `NvM_RestoreBlockDefaults()`, and `NvM_RestorePRAMBlockDefaults()`):

The application provides a RAM block that will be filled with NVRAM data and then issues a read/restore request to the `NvM` module.

After the read/restore request is issued, the RAM block can be modified until the callback routine is called.

The `NvM` module calls the routine configured via `NvMReadRamBlockFromNvCallback` in order to synchronize application data with the internal buffer of `NvM`.

The application can signal to the `NvM` module if the data was copied successfully or not by means of the return value of the callback function. If the return value of the user-defined callback function is `E_NOT_OK`, the `NvM` module will retry the operation `NvMRepeatMirrorOperations` times.

► Multi-block write requests (`NvM_WriteAll()`):

Note: `NvM_WriteAll()` is issued only by `EcuM` during shut-down.

The application has to provide a RAM block with the data intended to be written to the NV memory.

During the shut-down sequence, the application can modify the RAM block until the callback routine is called, when `NvM_WriteAll()` starts processing the block.

The `NvM` module calls the routine configured via `NvMWriteRamBlockToNvCallback` in order to synchronize application data with the internal buffer of `NvM`.

The application can signal to the `NvM` module if the data was copied successfully or not by means of the return value of the callback function. If the return value of the user-defined callback function is `E_NOT_OK`, the `NvM` module will retry the operation `NvMRepeatMirrorOperations` times.

► Multi-block read requests (`NvM_ReadAll()`):

Note: `NvM_ReadAll()` is issued only by `EcuM` during start-up.

The application has to provide a RAM block that will be filled with NVRAM data.

During the start-up sequence, the application can modify the RAM block until the callback routine is called, when `NvM_ReadAll()` starts processing the block.

The `NvM` module calls the routine configured via `NvMReadRamBlockFromNvCallback` in order to synchronize application data with the internal buffer of `NvM`.

The application can signal to the `NvM` module if the data was copied successfully or not by means of the return value of the callback function. If the return value of the user-defined callback function is `E_NOT_OK`, the `NvM` module will retry the operation `NvMRepeatMirrorOperations` times.

Note: in addition to the conditions listed above for the multi-block operations `NvM_WriteAll()` and `NvM_ReadAll()`, when using the explicit synchronization mechanism the following three conditions must also be true:

- The block must be marked for being processed during start-up and/or shut-down (`NvMSelectBlockForReadAll` and `NvMSelectBlockForWriteAll` configuration parameters).
- The explicit synchronization mechanism must be enabled for the block (`NvMBlockUseSyncMechanism` configuration parameter).
- The user must define and implement the functionality of the explicit synchronization callback functions (`NvMReadRamBlockFromNvCallback` and `NvMWriteRamBlockToNvCallback` configuration parameters).

4.2.9. Prioritization of RAM buffers

When processing asynchronous single block requests the user has the possibility to use three different types of RAM blocks:

- ▶ Permanent RAM block - configured with parameter `NvMRamBlockDataAddress`
- ▶ Temporary RAM block - provided at run-time within a single block request
- ▶ Explicit synchronization callbacks - configured with parameter `NvMBlockUseSyncMechanism`

You have the possibility to use a temporary RAM block even if the NVRAM block is configured with a permanent RAM block or explicit synchronization callbacks. To do this, call the `NvM_ReadBlock()`, `NvM_WriteBlock()`, or `NvM_RestoreBlockDefaults()` interface and provide a pointer to the temporary RAM location within the single block request (`NvM_SrcPtr` or `NvM_DstPtr` parameters depending on the request type). This means that a temporary RAM buffer provided within an asynchronous request is always used with highest priority.

If a block is configured with both permanent RAM and explicit synchronization callbacks, the explicit synchronization is used with higher priority than the permanent RAM blocks. This means that NvM does not copy data internally to the permanent RAM block. Explicit synchronization callbacks are called to copy the data to the user RAM.

In summary, NvM uses the following ordering when all options may be used:

1. Temporary RAM block
2. Explicit synchronization callbacks
3. Permanent RAM block

4.2.10. Data recovery for blocks with reconfigured length

Blocks can be read from the NV memory even if their configured length has changed.

If a block is detected with the length changed, the following applies:

- ▶ If the block is found smaller in the NV memory, the NvM gets the result `MEMIF_JOB_OK_SIZE_INCREASED` from the lower layer. The NvM then sends to the application the recovered data plus padding bytes with the negated value of the last byte of data. The application gets as request result the value `NVM_REQ_OK_BLK_INCREASED`.
- ▶ If the block is found bigger in the NV memory, the NvM gets the result `MEMIF_JOB_OK_SIZE_DECREASED` from the lower layer. The NvM then sends to the application the recovered data truncated to the current configured block length. The application gets as request result the value `NVM_REQ_OK_BLK_DECREASED`.
- ▶ If the reconfigured block has CRC configured, the CRC check is considered passed.
- ▶ The run of the block check mechanism on this block is considered as failed.

- ▶ The pre-write verification mechanism is skipped for this block.
- ▶ The post-write verification mechanism is skipped for this block.
- ▶ Write protection is not set for this block.

It is not recommended to configure the crypto hook mechanism for the blocks with reconfigured length. But if the crypto hook mechanism is configured, the hook must be implemented to take into consideration the data recovery of the reconfigured block.

4.2.10.1. Configuration options

The feature is at Memory Stack level. It can be activated in `Fee` with the parameter `FeeDynamicBlockLength`.

4.2.11. Working with the abstraction modules `Ea` and `Fee`

4.2.11.1. Functional description

Basic functions of the abstraction modules include:

- ▶ Access to the underlying memory using the drivers
- ▶ Memory life time management
- ▶ Simple data consistency checking
- ▶ Garbage collection
- ▶ Initialization and internal operations
- ▶ Reporting of production errors

4.2.11.1.1. Underlying memory access

The abstraction layer supports reading, writing, erasing and invalidation of data that is stored in the NV memory using the underlying driver i.e. either the `Eep` module or the `Fls` module. In addition, job status information can be queried and job completion or failure can be notified to the upper layer i.e. the `NvM`.

4.2.11.1.2. Memory lifetime management

Memory lifetime management is handled differently between the `Ea` and the `Fee` due to the different physical characteristics of the two underlying memory types.

The `Ea` uses a specific writing algorithm to distribute the data over the physical EEPROM. This is based on the use of a fixed set of memory locations. This method is possible since the erase function can target small page sizes and so memory can be overwritten on a per block basis. A data block typically occupies more than one page in the physical memory.

In order to manage the lifetime of the EEPROM, the `Ea` maintains a number of copies for each data block i.e. the `Ea` reserves consecutive space in the NV memory to allow writing the block to different physical locations. These copies are managed internally by the `Ea` by the use of a counter. This counter is referred to as the Virtual Lifetime Extension Counter (VLEC) and the value is stored in the NV memory along with the data when a block is written.

The number of copies used to write a particular data block depends on the setting of the parameter `EaNumberOfWriteCycles` in the `Ea` and the parameter `EepAllowedWriteCycles` in the underlying `Eep` module. This is calculated as $\text{copies} = \text{ceil}(x)$ where $x = \text{EaNumberOfWriteCycles} / \text{EepAllowedWriteCycles}$. The NV memory reserved for a block is then sufficient to store this number of copies of the data block.

In the Flash EEPROM however, the page sizes that can be erased are much larger. Therefore, in the `Fee`, one physical page is used to write several data blocks. It is not possible to erase a single data block at a time and so, typically, a `Fee` does not explicitly manage write cycles but continues to write evenly over the memory space available. Data and memory management in the `Fee` are described below in [Section 4.2.11.1.6, “Garbage collection”](#).

4.2.11.1.3. Data consistency checking

Simple data consistency checking is achieved by writing management information to the physical memory together with the user data. In the `Ea` this management information is located physically at the start and end of the actual block data. In the `Fee` however, the management information is partially located with the block and partially in a separate memory area designated for management information purposes only. The physical layout of memory used by the abstraction modules is described in section [Section 4.2.11.2, “Memory layout”](#)

The consistency of the data itself is not managed by the abstraction module. This must be done by the use of CRC calculation and is handled by the NVRAM manager depending on the block configuration. The abstraction modules can simply report on block inconsistency if a block is incompletely written to NV memory due to ECU reset during the block write process.

To avoid recovery of out-of-date data after an ECU reset, use the explicit block invalidation API to mark the block as invalid.

4.2.11.1.4. Invalid block handling

A block is marked as an invalid block if the API functions `Ea_InvalidateBlock()` or `Fee_InvalidateBlock()` are explicitly called.

After the attempt to read the block, the job result is set to `MEMIF_BLOCK_INVALID`.

4.2.11.1.5. Inconsistent block handling

A block is considered to be inconsistent if the block was not yet written to NV memory. That means the underlying NV memory is erased (it may still be empty).

After the attempt to read the inconsistent block, the job result is set to `MEMIF_BLOCK_INCONSISTENT`.

In `Fee`, if a block with a length other than the configured length is found, it is ignored. The attempt to read the block returns the job result `MEMIF_BLOCK_INCONSISTENT`. Because of this job result, the `NvM` knows that it must load default data.

4.2.11.1.6. Garbage collection

Garbage collection refers to the management of data particular to the `Fee`. Since blocks cannot be erased as single entities, the `Fee` continues to write over the next available memory locations. If a particular block is written several times, then the `Fee` uses management information to retrieve the most recent copy that has been written when requested.

The whole memory that is configured in the `Fls` is managed in logical pages (sections) in the `Fee`. When a logical page is full, it is necessary to collect all valid data, transfer this data to an empty logical page and then erase the copied logical page completely.

NOTE



Garbage collection can delay ECU shutdown

When working with flash EEPROM, it is important to note that garbage collection may be necessary in order to complete the job of writing data during ECU shutdown. Since this may be undesirable, the worst case scenario should be investigated during project integration.

4.2.11.1.7. Initialization and internal operations

This chapter gives an overview of the initialization and internal operations in the `Fee`. During the initialization phase, the section status is read for all sections from the flash memory. For details regarding the section management object and their layout, see section [Section 4.2.11.2.2, “Fee memory layout”](#). Based on the information read, the detection of the section to be used (called the `ACTIVE` section) takes place. The following conditions are taken into consideration:

- ▶ The transfer of data from one section to the other can be interrupted while user data is copied, while the section status fields are written or while a section is erased.

- ▶ Faults can occur while the section status fields are read or written. This leads to inconsistent information in the section management object.

If the `Fee` cannot determine the current `ACTIVE` section, one section will be marked `ACTIVE` and *internal recovery* is started in order for the `Fee` to become operational.

WARNING **Out-of-date data may be recovered during the initialization phase**



`Fee` section management object information may be corrupted, e.g. due to unexpected ECU resets or voltage drops during memory access. If errors are detected, the `Fee` first tries to recover user data before erasing a section. This may lead to recovery of out-of-date data.

4.2.11.1.8. Production errors

Reporting of production errors can be enabled in `Fee` configuration. There are two production errors that can be reported to `Dem`:

- ▶ `FEE_E_FLASH_ACCESSIBLE_DEM_EVENT_ID`: A read/write/erase access during internal operation (reading/writing of section management object or erase of one section) failed. A retry mechanism is implemented internally in `Fee`. If an internal read/write/erase request during internal operations fails for three times, the production error `FEE_E_FLASH_ACCESSIBLE_DEM_EVENT_ID` is reported to `Dem`. `Fee` goes to `IDLE` state.
- ▶ `FEE_E_DATA_RECOVERED_DEM_EVENT_ID`: Internal recovery was necessary during start-up. User data that may include out-of-date data is internally recovered in the `Fee`.

4.2.11.2. Memory layout

In this section, the physical layout of the NV memory as managed by the EB abstraction modules is described.

In both the `Ea` and the `Fee`, management information is written to the NV memory when the data block is written. This management information is used for two purposes:

- ▶ to manage the location of the most recent copy of data written to the NV memory
- ▶ to detect inconsistent blocks

4.2.11.2.1. `Ea` memory layout

In the `Ea`, the management information is located physically with the data block. This includes the Virtual Lifetime Extension Counter (VLEC) and block validation information (IF).

The VLEC is used in two ways:

- ▶ to keep track of the current copy of the block, in particular when multiple copies are to be written.
- ▶ to indicate that the block was completely written. In this case, the VLEC is written at the start of the data and then again at the end of the data area.

In the first case, the VLEC is incremented and written to memory each time a data block is written. This information can be used to determine how many times a block was written. In the case that multiple copies are used, the VLEC is used to determine the location of the most recent copy as this has the highest VLEC value.

In the second case, the `Ea` checks that the VLEC was written at both the start and end of the data area. If a block is only partially written, then this inconsistency is reported to the upper layer.

Block validation information is also stored with the data block. As this must be separately written by the `Ea`, for example if the block is explicitly invalidated using the API function `Ea_InvalidateBlock()`, the validation information is written to the next physical memory location adjacent to the data block, i.e. in the next virtual page. This is referred to as the Invalidation Flag (IF). When a block is valid, this value is erased. When the block was invalidated explicitly, this flag has the value `0xAA`.

The layout of a block as written to NV memory is depicted in [Figure 4.8, “Ea block layout”](#) below. Note that the second VLEC written is aligned to the end of the last virtual page that is needed to write the data block. This means that between the end of the user data and the second VLEC there may be some bytes of padding (unwritten bytes).

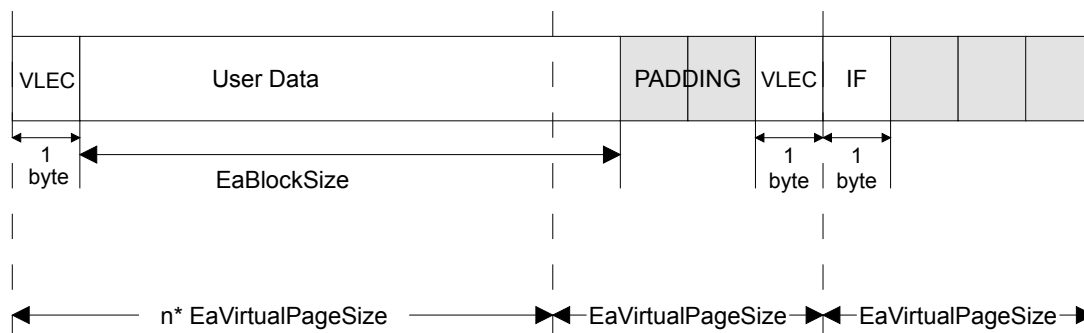


Figure 4.8. Ea block layout

Example

The following example demonstrates the layout of the NV memory when a block is configured with multiple copies. Assuming the parameter `EaNumberOfWriteCycles` in the `Ea` is configured to be 600000, and the parameter `EepAllowedWriteCycles` in the underlying `Eep` module is configured to be 200000. This means that the `Ea` maintains three copies of the data block in NV memory. [Figure 4.9, “Ea block with 3 copies, written 2 times”](#) shows the layout and the value of the VLEC when the block was written two times.

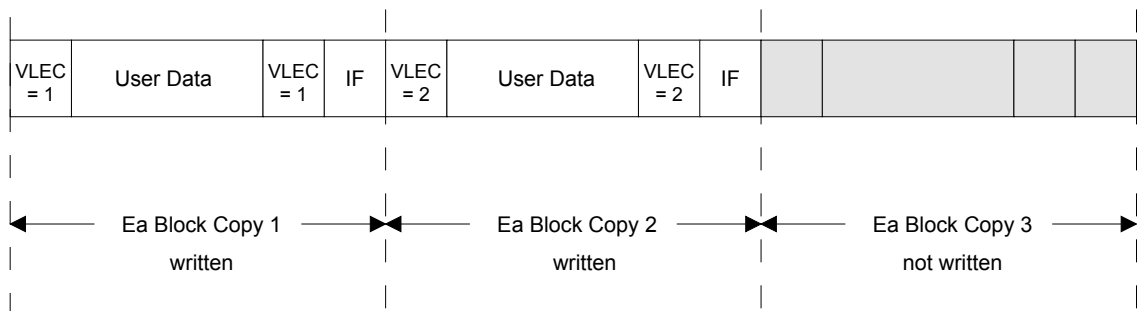


Figure 4.9. Ea block with 3 copies, written 2 times

If the block is then written a further 2 times, the first copy will be overwritten on the fourth write. [Figure 4.10, “Ea block with 3 copies, written 4 times”](#) shows the layout and the value of the VLEC when the block was written four times.

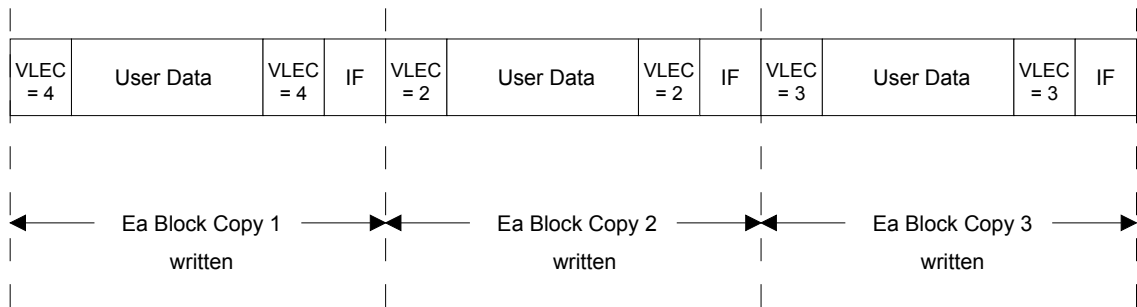


Figure 4.10. Ea block with 3 copies, written 4 times

4.2.11.2.2. Fee memory layout

In the Fee, the entire memory that is configured in the Fls is managed in logical pages. [Figure 4.11, “Fee logical pages”](#) shows the logical structure of the NV memory as managed by the Fee.

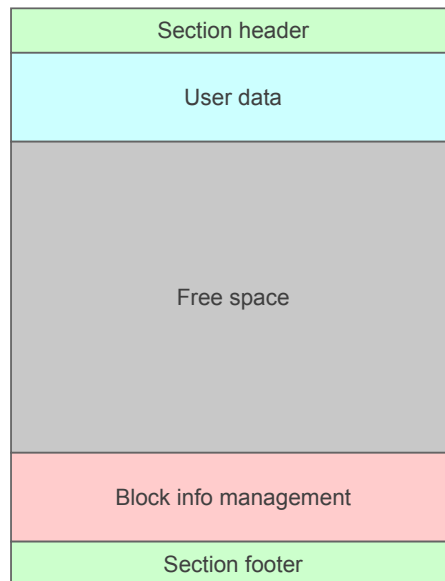


Figure 4.11. Fee logical pages

These logical pages (sections) are used alternately for writing to the NV memory. Each section contains three types of information:

- ▶ section management information
- ▶ block management information
- ▶ user data

Section management information consists of seven fields that contain relevant information related to the status of each section. These fields are independently read or written. The fields and the data stored can be described as follows:

- ▶ `Erasable marker`: value `0xDD`, indicates that all valid data was transferred to the alternate section and therefore this section can be erased. This field is written after all relevant user data was copied to the other section. Field length is the virtual page size or 2 bytes if the virtual page size is 1 byte.
- ▶ `Erase counter`: represents a counter that stores the approximate number of times the Fee sections were erased. It does not store a section-specific value. This counter is always written immediately after a section is completely erased. Field length is 4 bytes.
- ▶ `Erase counter CRC`: represents a CRC calculated over the erase counter value. Field length is 1 byte.
- ▶ `Section and configuration identifier CRC`: represents a CRC calculated over the section counter and configuration identifier values. Field length is 1 byte.
- ▶ `Configuration identifier`: is a checksum calculated over Fee section configuration parameters: start address and size of each configured/computed section, virtual page size and number of sections. Field length is 1 byte.

- ▶ **Section counter:** is a 1-byte circular counter that maintains the correct order of sections. It is incremented circularly when the section is validated after a section erase.
- ▶ **Active Marker:** value 0xAA, indicates that a section is the current working section. This field is written to a section before any user data is written to that section. Field length is virtual page size or 2 bytes if the virtual page size is 1 byte.

Figure 4.12, “Fee section management object” shows the layout of the section management object in NV memory.

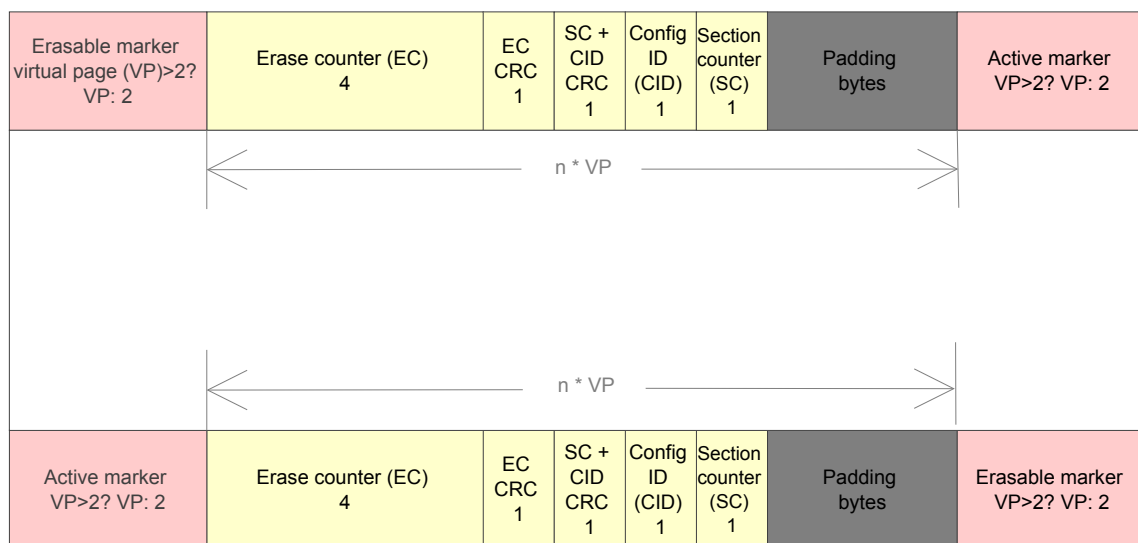


Figure 4.12. Fee section management object

The Fee block management information (block info) comprises an area of data that is written separately from the data block itself. This includes the FeeBlockSize, FeeBlockNumber, and the FeeBlockDataAddress. In addition, the Written marker is written to the NV memory next to the block info, but only after block data is completely written.

Blocks are located using the size information and the address stored in the management block during start-up. The address of the most recent copy of each block is then stored and maintained in an internal cache during run-time.

Block consistency is determined by checking that the Written marker is written, even partially. If a block was only partially written, this marker never gets to be written and the block info is ignored.

Block validation information is stored in the management bytes. If a block is explicitly invalidated using the Fee API function Fee_InvalidateBlock(), then the management information for the block is written to the NV memory with a block size of zero.

Figure 4.13, “Fee block management information” shows the layout of block management information in NV memory.

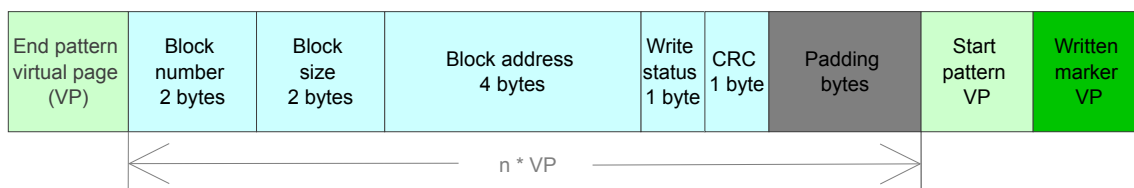


Figure 4.13. Fee block management information

The user data is written starting at the low memory address in the NV memory. When a Fee block is written, the block management information is first written at the next available location in high memory. The data is then written at the next available location in low memory. If a block is written more than once, the information is simply written to the next available location. The memory location first used cannot be overwritten as in the Ea, due to the hardware properties of the flash memory.

The Fee buffer is used only internally for initialization and for switching. The user jobs are performed using directly the user's buffer.

If a block has multiple copies, the block is still written to the next available location in NV memory. There is no space reserved for block copies as in the Ea. In effect, there is no explicit copy management. Blocks are written sequentially to the NV memory in the order requested by the block write API.

The chosen memory layout ensures that old data written with another non-volatile layout (another configuration) is not accidentally read for a different block ID. So, if the Fee is started with some invalid data present inside the flash memory, the Fee reads only the valid data. The invalid data is ignored and reported to NVMe as inconsistent.

4.2.11.3. Patterns for data consistency

With this feature, the reliability of read/write/erase operations can be increased. It is an additional feature to the general data consistency checking that is described in [Section 4.2.11.1.3, "Data consistency checking"](#). The feature is based on, but not limited to, the use of Renesas RH850 devices.

4.2.11.3.1. Functional description

The principle of the algorithm is to validate a successful write operation in flash memory by writing an additional pattern. This pattern is written regardless of whether the write/erase operation was triggered by a user request or as part of internal operations performed by the Fee module.

When data is read from the flash memory, the value of the consistency patterns is verified if the **pattern for data consistency** feature is enabled. If the patterns are empty, i.e. they are equal to the flash erase value, the write request is considered incomplete and data is treated as inconsistent.

The use of the **pattern for data consistency** feature increases the reliability of the data. However, it also has a negative impact on the processing time and space required in the physical flash memory. This is due to additional read/write operations for handling the consistency patterns.

4.2.11.3.2. Memory layout with consistency patterns

At the level of the sections, the structure of the memory layout remains the same. However, more space is reserved by `Fee` within a section for writing the section management object's patterns to the physical memory as depicted in the picture below:

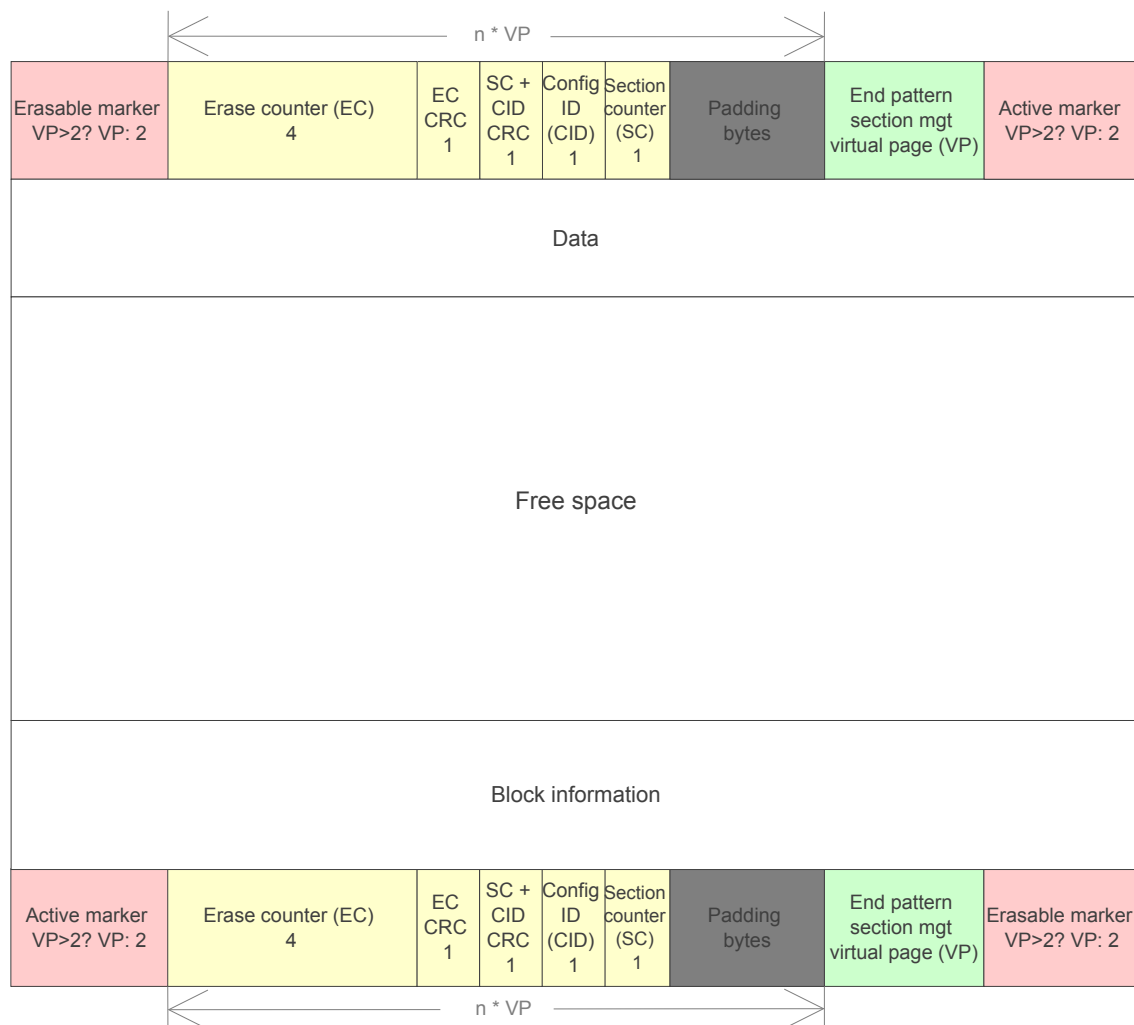


Figure 4.14. `Fee` section management object layout with consistency patterns enabled

In the physical memory, the consistency patterns are written for both internal management information and user data.

For the section management object, consistency patterns are written to flash memory after the section management object's space but before the start of any block management information.

For the block management information, the start pattern is written to flash memory *after* the actual block management information. The end pattern is written to flash memory *before* the actual block management information.

For the block data, the start pattern is written to flash memory *before* the actual block data. The end pattern is written to flash memory *after* the actual block data.

Figure 4.15, “Free logical page layout with consistency patterns enabled” depicts the memory layout of the block management information and block data along with the data consistency patterns.

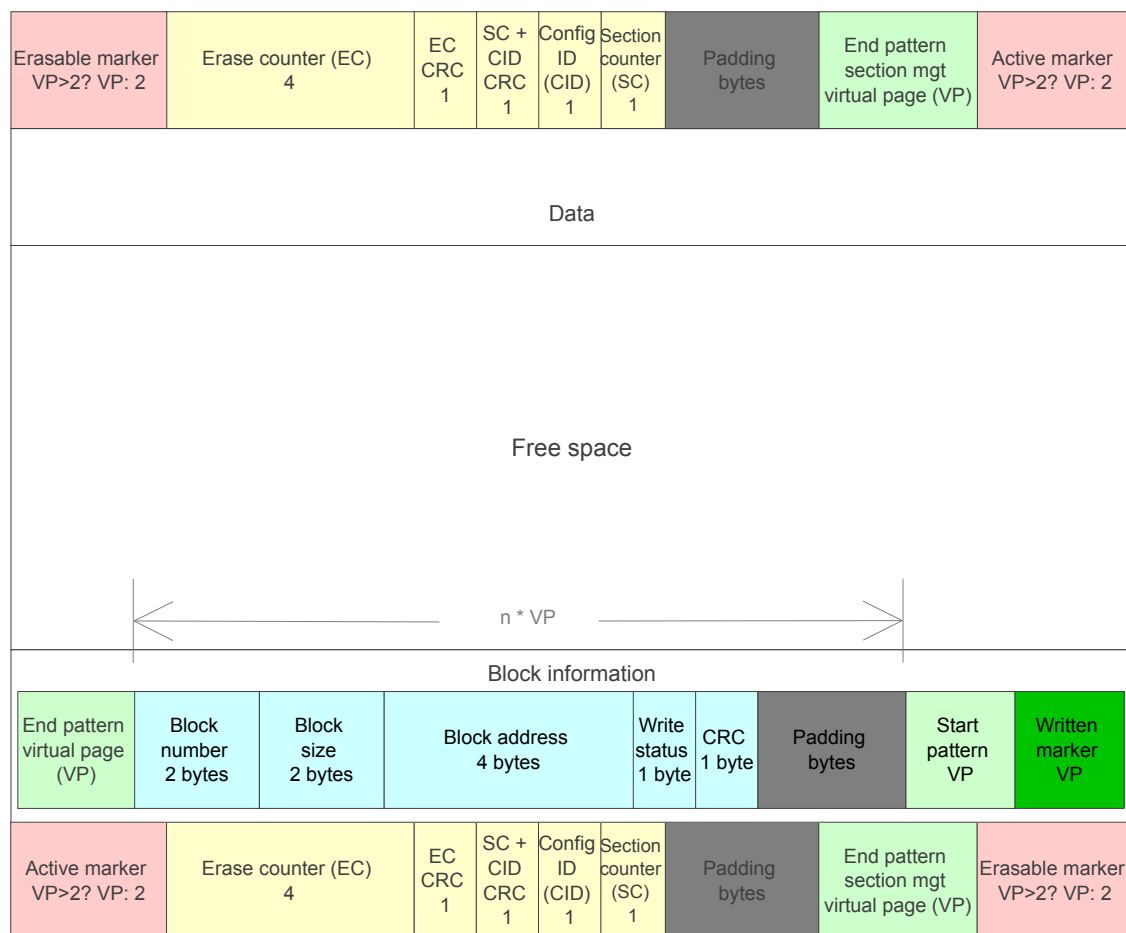


Figure 4.15. Free logical page layout with consistency patterns enabled

Note that if the use of consistency start patterns is disabled in your configuration, i.e. `FeeConsistencyStartPattern = false`, only the end patterns are actually written to the memory.

WARNING



Memory layouts with and without consistency patterns are incompatible

If you use the **pattern for data consistency** feature, it affects the memory layout of Fee. Fee memory layouts for configurations that have the feature enabled are incompatible with configurations that have the feature disabled.

Do not activate/deactivate the **pattern for data consistency** feature if data was already saved to NV memory.

4.2.11.3.3. Configuration options

To use the **pattern for data consistency** feature, enable the configuration parameters `FeeConsistencyPattern` and `FeeConsistencyStartPattern`. If `FeeConsistencyPattern/FeeConsistencyStartPattern` is active, the user data is written with an offset of 1 virtual page relative to the block data start address. In contrast, if the feature is not active, the user data is written with offset 0.

You can choose between two modes of operation of the algorithm, as described in the following table:

<code>FeeConsistencyPattern</code>	<code>FeeConsistencyStartPattern</code>	Behavior
false	false	The behavior of Fee is not changed. Default data consistency checking is used.
true	false	Fee validates each successful write operation by writing an additional end pattern. Fee validates each successful read operation by checking the value of the end pattern.
true	true	Fee marks the start of a write operation by writing an additional start pattern. Fee validates each successful write operation by writing an additional end pattern. Fee validates each successful read operation by checking the value of the start and end patterns.
false	true	Invalid configuration. An error is reported at configuration time.

FeeConsistencyPattern	FeeConsistencyStartPattern	Behavior
		The use of consistency start patterns is only allowed if FeeConsistencyPattern is enabled.

Table 4.1. Configuration options for consistency patterns

4.2.11.4. Configurable number of sections

To make Fee more flexible, you can configure the number of sections used by Fee for storing the data.

4.2.11.4.1. Functional description

You can configure a maximum of 254 independent sections. The minimum number of sections is two. Fee uses a mechanism to ensure that no data loss occurs during a section switch, provided all blocks to be switched can be read successfully.

4.2.11.4.2. Configuration options

To configure the number of sections, set the configuration parameter FeeNumberOfSections.

4.2.11.5. Secured section switch

This feature ensures that blocks are not lost during a section switch.

4.2.11.5.1. Functional description

In normal conditions, any job request can interrupt a section switch. The section switch continues only after the job is completed.

With the secured section switch, the section switch operation can be prioritized under certain conditions. In this case, any incoming or already memorized job stays in pending state and is performed only after the section switch is completed.

The secured section switch is applied if the space becomes limited. Every store job, such as write, invalidate, or erase-immediate, reduces the space in the flash memory. Without the secured section switch, if a certain limit is passed, Fee cannot guarantee that all written blocks can still be switched. With the secured section switch,

`Fee` keeps an empty section ready for a high-priority section switch. Thus, if only two sections are configured, any job that implies a section change puts `Fee` in high-priority section switch mode.

The feature includes a restart section switch functionality. If an error occurs during a section switch, and the space gets too small in the last available section, `Fee` restarts the section switch and ensures that all blocks are copied successfully so that block losses are avoided.

4.2.11.5.2. Configuration options

This feature is part of the basic implementation and is not configurable.

4.2.11.6. Section erase counter

The section erase counter provides information about the lifetime of the flash and the number of erase cycles for each section.

The section erase counter can be retrieved by using the [Fee_GetEraseCounterValue\(\)](#) API.

4.2.11.6.1. Configuration options

To enable the `Fee_GetEraseCounterValue()` API, set the configuration parameter `FeeEraseCounter-Api`.

4.2.11.7. Abort erase

The **Abort erase** feature ensures that immediate data is written as quickly as possible, by allowing immediate write requests to interrupt an ongoing erase section operation.

4.2.11.7.1. Functional description

Any write job for an immediate block that is requested during an ongoing section erase causes the abort of the section erase if the erase is not a high-priority internal operation. The write job is performed only after the highest internal operations are completed.

4.2.11.7.2. Configuration options

To use the feature, set the configuration parameter `FeeEnableAbortErase`.

4.2.11.8. Freeze activities

With this feature, you can stop the processing of any memory stack requests for blocks that are configured for `Fee`.

4.2.11.8.1. Functional description

You can freeze the `Fee` module for all activities. The current activity is finished and memorized, but its result is not processed. No other pending activity is triggered. `Fee` does not make any further changes or accesses to the non-volatile memory. Thus, the memory remains in the same state as if it were frozen. This could be useful in the following use cases:

If the erase of a section or the validation done as part of a section switch fails, the retry would most probably fail as well. This would lead to a premature wear of flash memory. Freezing saves the erase failing cycles.

If the power drops during a switch, `Fee` might fail to read the blocks to be switched and thus, might not switch them. When the section that was switched is erased, these blocks are lost. If the `Fee` is frozen for the duration of the power drop, then, at worst, only one block reading fails while one read retry remains.

If it is a better solution to keep the non-volatile memory contents as they are, rather than risking erroneous behavior of the `Fee` module, freezing the `Fee` activities might be the best option.

The `Fee` activities can be frozen and unfrozen by using the [Fee_FreezeActivities\(\)](#) API.

4.2.11.8.2. Configuration options

To use the **Freeze activities** feature, set the configuration parameter `FeeFreezeActivitiesApi`.

4.2.11.9. Switch non-configured blocks

With this feature, `Fee` can detect and switch valid blocks that are not configured. The configurations of the memory stack in application and bootloader do not have to be identical. An update on application side does not require an update in the bootloader.

4.2.11.9.1. Functional description

`Fee` can handle a given number of non-configured blocks during a section switch or during `Fee` initialization. This means that `Fee` identifies these blocks based on block information objects and caches them. It stores the

configuration attributes of each block in order to have all information required for switching the blocks, even though they are not part of the `Fee` configuration.

Due to memory consumption constraints, the number of blocks that `Fee` can handle is limited by configuration. Once the configured number of independent blocks is reached, the rest of the non-configured blocks is not taken into account during the section switch or during initialization.

`Fee` cannot handle unlimited amounts of data due to flash memory size constraints. Therefore you should try to estimate the total size of non-configured blocks as best as possible when configuring. If `Fee` detects during initialization that the size of non-configured blocks exceeds the total amount of data that `Fee` can handle, `Fee` does no longer memorize any non-configured blocks, even though the expected number of non-configured blocks was not reached. This means that the estimated total size of non-configured blocks was not configured correctly.

4.2.11.9.2. Configuration options

To use the feature, set the configuration parameters `FeeNumberOfNotConfigBlocks` and `Fee-DataSizeNotConfiguredBlocks`.

4.2.11.10. Write custom

The **Write custom** feature allows you to request a write job for a non-configured block.

4.2.11.10.1. Functional description

The write custom request is handled via the [Fee_WriteCustom\(\)](#) API.

You must pass the following parameters to this API:

- ▶ the number of non-configured blocks,
- ▶ the pointer to the data buffer,
- ▶ the size of the non-configured block.

The job is performed internally like a regular write job. The written non-configured block is copied by a section switch. You can request a write custom job with a different length for an already written non-configured block.

As `Fee` often receives jobs from `NvM`, you must make sure that `Fee` is not busy when you request a write custom job. Otherwise the write custom job is rejected.

NOTE



Do not request a write custom job for a configured block

It is possible to request a write custom job for a configured block if the given size is correct. But this is not recommended because it may cause desynchronization with the `NvM` block due to CRC or `StaticId` mismatch.

4.2.11.10.2. Configuration options

To use the feature, set the configuration parameter `FeeWriteCustomApi`, and configure at least one non-configured block in `FeeNumberOfNotConfigBlocks`.

4.2.11.11. Read custom

The **Read custom** feature allows you to request a read for a non-configured block.

4.2.11.11.1. Functional description

The read custom request is handled via the [Fee_ReadCustom\(\)](#) API.

You must pass the following parameters to this API:

- ▶ the number of non-configured blocks,
- ▶ the read address offset inside the block,
- ▶ the pointer to the data buffer,
- ▶ the number of bytes to be read.

The job is performed internally like a regular read job.

As `Fee` often receives jobs from `NvM`, you must make sure that `Fee` is not busy when you request a read custom job. Otherwise the read custom job is rejected.

You can also request a read custom job for a configured block. The functionality is the same as with `Fee_Read()`.

4.2.11.11.2. Configuration options

To use the feature, set the configuration parameter `FeeReadCustomApi`, and configure at least one non-configured block in `FeeNumberOfNotConfigBlocks`.

4.2.11.12. Cancel section erase

This interface allows you to cancel an ongoing section erase.

4.2.11.12.1. Functional description

Via the [Fee_CancelSectionErase\(\)](#) API, you can cancel a section erase if this is an ongoing internal operation. But you cannot cancel the erase if it has the highest internal priority. In that case, the cancellation request is rejected.

The `Fee_CancelSectionErase()` API acts synchronously, and it cannot interrupt `Fee_MainFunction()`. If the API is called during the execution of `Fee_MainFunction()`, the action is rejected. `Fee` does not perform any retry for the request. Such retries have to be configured at the caller side.

4.2.11.12.2. Configuration options

To use the **Cancel section erase** feature, set the configuration parameter `FeeCancelSectionEraseApi`.

4.2.11.13. Emergency block

The feature ensures that the block that is configured as emergency block is written with the highest possible priority.

4.2.11.13.1. Functional description

As prerequisite to emergency block writing, `Fee` must have finalized the first part of the initialization, in which the status of all blocks is determined. If the emergency block writing is requested earlier, the job fails. Emergency block writing is also not possible if the flash memory is mass-erased, corrupted, or recovering after a previous emergency block writing with highest internal priority. If requested in these cases, the job also fails.

The emergency block can be written at least once per section switch cycle. It can be written many times if the space allows.

Any pending or ongoing job from `NvM` is reported as failed if emergency block writing is requested. Any internal `Fee` operation is suspended when the emergency block is requested to be written. Once an emergency block writing is requested, it can only be temporarily interrupted with the **Freeze activities** feature.

`Fee` freezes internally after a successful writing of the emergency block or if it determines after completion of the first part of the initialization that the emergency block was written to the flash memory. The frozen state after

the emergency block writing is different from the frozen state after executing the **Freeze activities** feature. In the emergency frozen state, `Fee` accepts and processes only a read job for the emergency block, or an erase-immediate job for the emergency block. The emergency frozen state is left when the erase-immediate job for the emergency block is successfully finished. Afterwards `Fee` resumes any internal activity.

NOTE



Call `Fee_GetJobResult()` to keep NvM synchronized

After the erasing of the emergency block is finished, you must call the API `Fee_GetJobResult()`. If you do not call `Fee_GetJobResult()`, the synchronization with NvM is lost. The following sequence is recommended:

1. `Fee_Write()` for the emergency block
 2. `Fee_GetJobResult()`
 3. `Fee_Read()`
 4. `Fee_GetJobResult()`
 5. `Fee_EraseImmediate()`
 6. `Fee_GetJobResult()` - This call is mandatory.
-

4.2.11.13.2. Configuration options

To use the **Emergency block** feature, set the configuration parameter `FeeCriticalBlock`. You can configure only one block as emergency block. This block must be configured as immediate block.

4.2.11.14. Fls alignment compatibility

The **Fls alignment compatibility** feature ensures the compatibility with `Fls` drivers that require 32-bit alignment for addresses that are passed as API parameters.

4.2.11.14.1. Functional description

Any block to be written from an address that is not aligned to 4 bytes is first copied to the internal buffer and then sent to `Fls`. Any block to be read to an address that is not aligned to 4 bytes is first read to the internal buffer and then copied to the user address. The `Fee` buffer size must be greater or equal to the largest block.

4.2.11.14.2. Configuration options

To use the feature, set the configuration parameter `FeeUseBufferForJobs`.

4.2.11.15. Small section support

When you enable the **Small section support**, the total block size can be larger than the smallest configured section if the following is met:

- ▶ There are at least 4 configured sections.
- ▶ The total block size can fit in half of the configured sections.

4.2.11.15.1. Functional description

The **Small section support** enables the configuration of sections with a size smaller than the total size of the configured `Fee` blocks when 4 or more sections are configured.

- ▶ If set to true: `Fee` allows you to configure sections with a size smaller than the total size of the blocks.
- ▶ If set to false: `Fee` does not allow you to configure sections with a size smaller than the total size of the blocks.

A warning instead of an error is issued at generation phase if the following applies:

- ▶ the number of configured sections is at least 4, and
- ▶ the size of all configured blocks and their overhead fit in half the size of the total sections, rounded down.

This could lead to an overload of both the run-time of `Fee`'s internal behavior and the flash memory. It is your responsibility to consider if the drawbacks are acceptable for the project.

4.2.11.15.2. Configuration options

To use the feature, set the configuration parameter `FeeEnableSmallSectionSize`.

4.2.11.16. Drivers with `vendorId` and `vendorApilnfix` support

The **Drivers with `vendorId` and `vendorApilnfix` support** feature allows you to use flash drivers that use `vendorId` and `vendorApilnfix`.

4.2.11.16.1. Functional description

At generation phase, if `Fee` finds a `Fls` module in the project that has `vendorId` and `vendorApilnfix` configured, `Fee` maps the standard APIs to the vendor `Fls` APIs.

4.2.11.16.2. Configuration options

To use the feature, set the configuration parameters `VendorId` and `VendorApiInfix` in `Fls`.

4.2.11.17. Initialize in loop

The **Initialize in loop** configuration parameter allows you to configure the initialization buffer to be generated with the smallest possible value to give the best performance if the initialization is done in a loop without using the OS.

4.2.11.17.1. Functional description

Initialization is done in a loop with the OS disabled. Otherwise, the initialization performance is reduced.

- ▶ If `FeeInitializeInLoop` is set to `true`, `Fee` generates the smallest size for the init (start-up) buffer, leading to better efficiency.
- ▶ If `FeeInitializeInLoop` is set to `false`, `Fee` generates the optimal size for the init (start-up) buffer when the start-up runs on scheduled tasks.

You do not need to configure the init buffer size. The generation algorithm sets it to the optimal size, depending on the way you run the `Fee` start-up phase. You may still set the preferred value for the main `Fee` buffer. The amount of it needed at start-up is internally set.

4.2.11.17.2. Configuration options

To use the feature, set the configuration parameter `FeeInitializeInLoop`.

4.2.12. Multi-core support for NvM

For a multi-core ECU, `NvM` can provide a subset of its AUTOSAR-compliant functionality on other cores. A platform-specific inter-core communication mechanism must be integrated for this feature.

4.2.12.1. Functional description

`NvM` allows you to configure a wrapper for its interfaces that shall be placed on all the cores that require `NvM` services. The `NvM` wrapper checks that conditions are met and calls the multi-core mechanism that shall call the actual `NvM` API.

The following picture illustrates the `NvM_ReadBlock()` usage with multi-core wrappers.

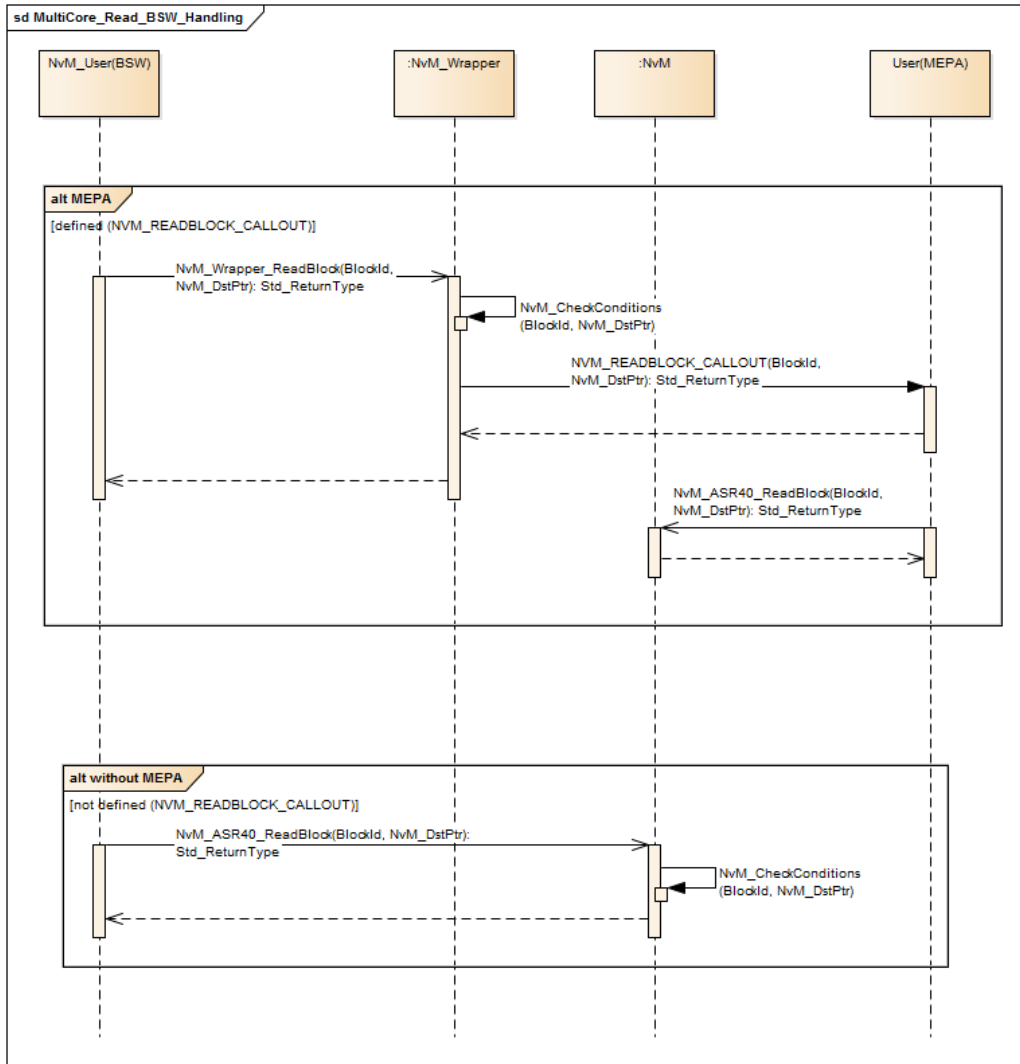


Figure 4.16. NVRAM multi-core call example

The following `NvM` services are provided on other cores:

- ▶ `NvM_ReadBlock()`
- ▶ `NvM_WriteBlock()`
- ▶ `NvM_RestoreBlockDefaults()`
- ▶ `NvM_EraseNvBlock()`
- ▶ `NvM_InvalidateNvBlock()`
- ▶ `NvM_ReadPRAMBlock()`
- ▶ `NvM_WritePRAMBlock()`
- ▶ `NvM_RestorePRAMBlockDefaults()`

► `NvM_CancelJob()`

4.2.12.2. Configuration options

To use the feature, each callback may be enabled and the function name must be set in the corresponding field.

For example, in order to enable the `NvM_ReadBlock()` callout, `NvMReadBlockCallout` must be set:

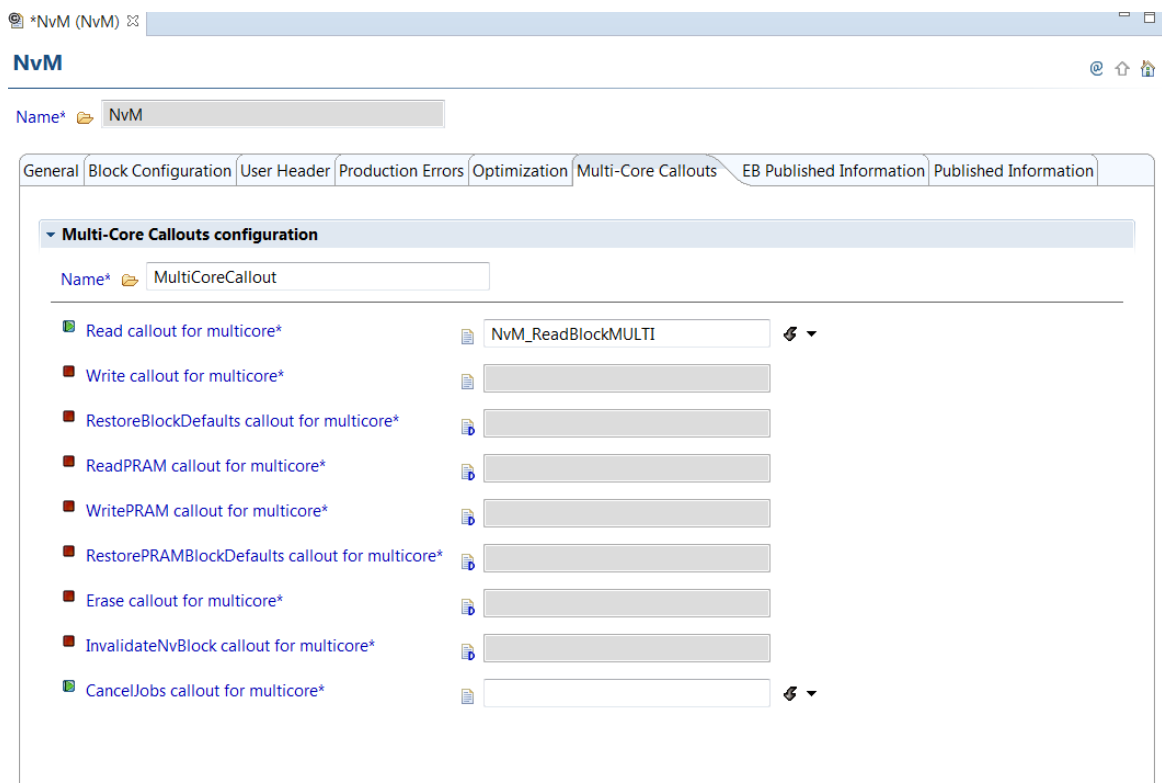


Figure 4.17. NVRAM multi-core callout configuration

4.2.13. RAM block state after soft reset

In `NvM`, you can configure whether the RAM blocks state persists after a warm reset or is initialized to INVALID/UNCHANGED.

4.2.13.1. Functional description

The `NvM_AdminBlockTable` is located in a different memory section based on the configuration parameter `NvMResetRamBlockAfterReset` as follows:

- ▶ `NvMResetRamBlockAfterReset` set to false: The **NvM_AdminBlockTable** is allocated to `VAR_POWER_ON_INIT`. The RAM state persists after a warm reset and is not read from the NV memory with `NvM_ReadAll()`, even though the block does not have the `NvMCalcRamBlockCrc` configured.
- ▶ `NvMResetRamBlockAfterReset` set to true: **NvM_AdminBlockTable** is allocated to `VAR_CLEARED`. The state of the blocks is reinitialized after a warm reset.

4.2.13.2. Configuration options

To use the feature, set the configuration parameter `NvMResetRamBlockAfterReset`.

4.3. Configuring the memory stack

This chapter provides step-by-step instructions for configuring the memory stack. To achieve the best results, proceed through these steps in the order suggested.

4.3.1. Background information

To configure the memory stack, you must understand the details of the parameters of each module. Read the extended descriptions provided in the module reference sections for each module as referenced in the configuration steps below. These extended descriptions give details about the purpose of each parameter and, where applicable, the range of the parameter, the dependencies on other parameters and additional notes to help you to configure the parameter correctly. Also, using EB tresos Studio, these are to be found in the online help. They are also displayed in the **Element Description** view in the user interface.

4.3.2. Configuration steps

It is recommended to use a bottom-up approach to configuring the stack.

1. Configure the `Eep` or `Fls` driver module as required for your project. Follow instructions given in the suppliers reference documentation for these modules. The reference documentation is located in the folder `$TRESOS_BASE\doc\5.0_MCAL_modules` where `$TRESOS_BASE` is the root directory of your EB tresos Studio installation.
2. Configure the `Ea` or `Fee` modules. For detailed help, use the `Ea` module reference or the `Fee` module reference in the document EB tresos AutoCore module references.

3. Configure the `MemIf` module. For detailed help, use the `MemIf` module reference in the document EB tresos AutoCore module references.
4. Configure the `Crc` module. For detailed help, use the `Crc` module reference in the document EB tresos AutoCore module references.
5. Configure the `NvM` module. For detailed help, use the `NvM` module reference in the document EB tresos AutoCore module references.

You can use the `Memory Stack Editor` for assistance in the configuration steps 2 and 5 above. If you use the `Memory Stack Editor` for this purpose, you configure the `NvM` before the `Ea` or `Fee`. Refer to [Section 4.4, “The Memory Stack Editor”](#) for further instructions.

4.4. The Memory Stack Editor

4.4.1. Background information

The `Memory Stack Editor` is designed to assist with the configuration of NV blocks in the modules `NvM`, `Ea` and/or `Fee`. To use the editor, it is necessary to include these modules, together with the `Eep` and/or `Fee` in your EB tresos Studio project.

Based on the blocks configured for the `NvM`, the blocks for the `Ea` and the `Fee` can be automatically configured using the editor. It is not necessary to manually configure the `Ea` or `Fee` thus you can save time and effort when you configure the memory stack.

Using the `Memory Stack Editor` you can:

- ▶ manually add, remove or duplicate blocks in the `NvM` configuration using the list manipulation buttons
- ▶ automatically add `NvM` blocks which are specified in the service needs of your application software components using the **Evaluate NV block needs** button
- ▶ automatically update the blocks configured for the `Ea` and `Fee` so that these are consistent with the `NvM` configuration using the **Memory Stack Configurator** button

When blocks are added to the `NvM` configuration using the `Memory Stack Editor`, some block specific parameters can be set. The parameters that can be set are those that are needed in order to be able to create the `Ea` or `Fee` blocks automatically. Additional block specific parameters must be configured using the `NvM` Generic Editor.

When blocks are added to the `NvM` configuration using the `Memory Stack Editor`, default values are assigned to the block specific parameters. The editor can be used to adjust these if necessary. The edited values will then be stored in the `NvM` configuration when the editor is closed.

When blocks have been configured using the `NvM` Generic Editor, these are automatically loaded when the `Memory Stack Editor` is opened. The parameter values already configured for these blocks will not be changed by the `Memory Stack Editor` unless the user specifically does so.

When an existing configuration for the `NvM`, the `Ea` and the `Fee` is loaded into the `Memory Stack Editor`, the module configurations are checked for consistency. An error is displayed if these are not consistent. The user can then adjust the configurations automatically with the editor by using the **Memory Stack Configurator** button.

WARNING



The Memory Stack Configurator button always configures the `Ea` and the `Fee` to be consistent with the `NvM`

If your existing configuration is loaded into the `Memory Stack Editor` and this is reported to be inconsistent, use the **Memory Stack Configurator** button to make the configurations consistent only if the `NvM` configuration is correct.

4.4.2. Using the Memory Stack Editor

In order to use the `Memory Stack Editor`, first add memory stack modules to your project in EB tresos Studio. For a configuration using EEPROM, the `Memory Stack Editor` requires the modules `NvM`, `Ea` and `Eep` in your project. For a configuration using Flash EEPROM, the `Memory Stack Editor` requires the modules `NvM`, `Fee` and `Fls` in your project.

Before running the editor, configure the `Eep` module and/or the `Fls` as required for your project.

To automatically configure any `NvM` blocks that are needed by the basic software modules you may also run the `Service Needs Calculator`. For example, use the `Service Needs Calculator` to automatically configure the `NvM` blocks that are needed to store `Dem` events. Instructions on using the `Service Needs Calculator` can be found in the EB tresos AutoCore Generic 8 Base documentation.

When the memory stack modules have been added to your project, you see the `Memory Stack Editor` in the **Sidebar** view in EB tresos Studio. The **sidebar** view is organized in categories; the categories you see depend on which EB tresos products you have installed. The `Memory Stack Editor` appears in the **ECU** category:

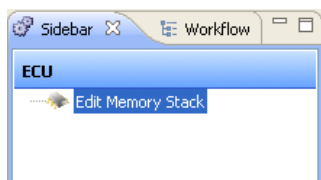


Figure 4.18. The `Memory Stack Editor` in the **ECU** category of the **sidebar** view

Double-click on the editor to open it.

The Memory Stack Editor appears in a new window:

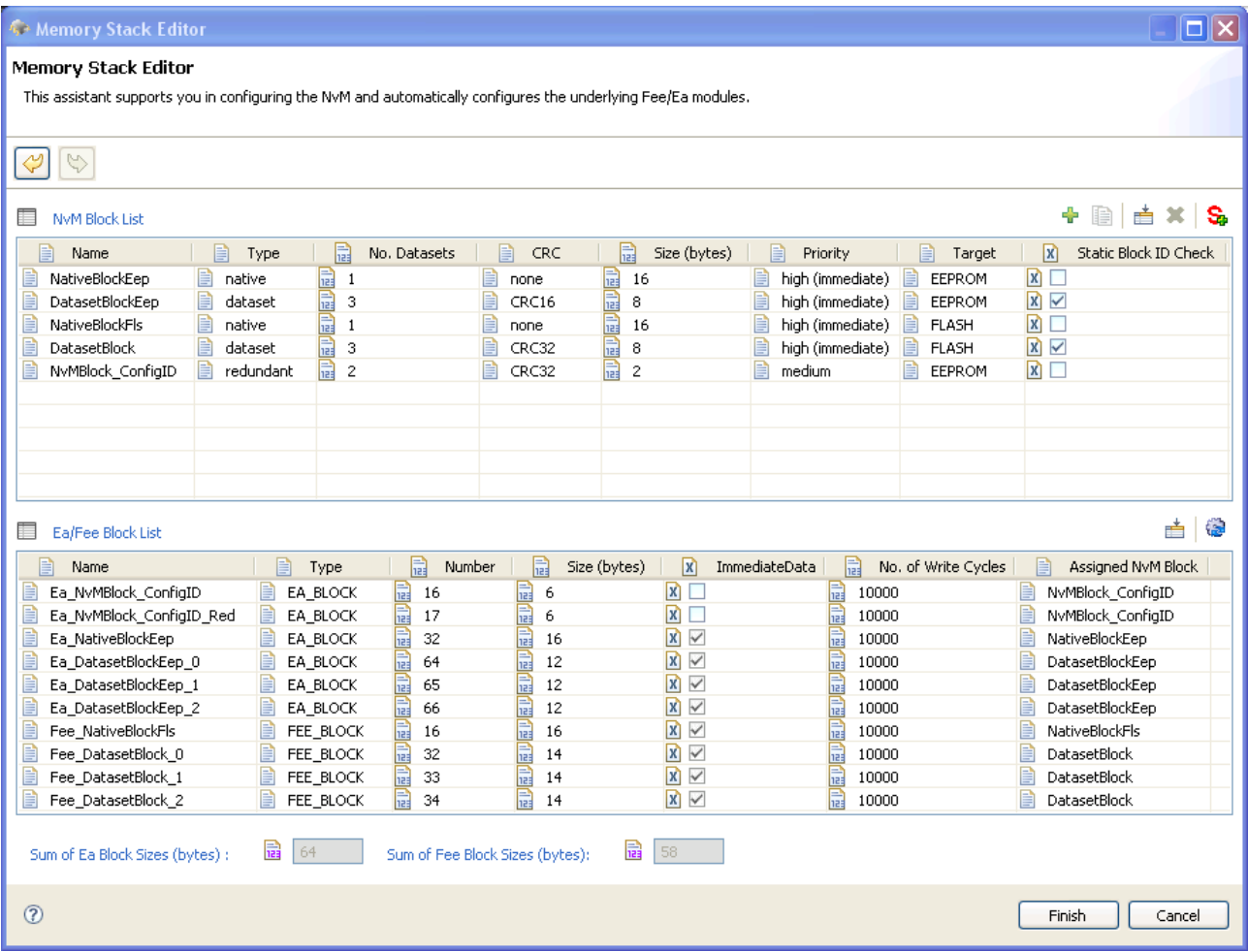


Figure 4.19. The Memory Stack Editor window

NOTE







The Memory Stack Editor fails to open if there are fatal errors in the configuration


Some configuration errors cannot be safely interpreted by the Memory Stack Editor. In such cases an error dialog appears and the editor cannot be opened. Resolve the error reported in the dialog using the Generic Editor before using the Memory Stack Editor again.

4.4.2.1. Memory Stack Editor window layout

The Memory Stack Editor displays two tables:

- ▶ the **NvM Block List** table
- ▶ the **Ea/Fee Block List** table

Above the **NvM Block List** table you find the buttons to add , remove  and duplicate  blocks in the list. In addition, you find the button to automatically calculate the NvM blocks that are needed by your software components .

Above the **Ea/Fee Block List** table you find the button to automatically update the list of blocks that have to be configured in the Ea or Fee modules based on the current entries in the NvM configuration .

Below the **Ea/Fee Block List** table, the sums of the sizes of the blocks that are configured are shown. These show the sum of the sizes of the blocks that are configured for the Ea or Fee respectively. The size for a block comprises the NV data size, the CRC size if this is used and also the block ID size if the static block ID check is activated.

NOTE





The Sum of Ea Block sizes / Sum of Fee Block sizes only show the sum of the configured data sizes

The **Sum of Ea Block sizes / Sum of Fee Block sizes** only show the sum of the configured data sizes. They do not show the memory size requirements needed in order to store these data blocks.

4.4.2.2. Configuring the memory stack with the Memory Stack Editor

Once you open the editor, the blocks that have already been configured for the NvM are displayed in the **NvM Block List** table. If the corresponding blocks have already been configured for the underlying Ea or Fee modules, these are displayed in the **Ea/Fee Block List** table.

- ▶ If the **Ea/Fee Block List** is not up to date with the NvM configuration, click the **Memory Stack Configurator** button  to resolve the differences.
- ▶ To add the blocks that are specified in the service needs of your application software components, press the **Evaluate NV block needs** button . If additional blocks are added in this way, use the **Memory Stack Configurator** button to update the **Ea/Fee Block List** again.
- ▶ Add additional blocks to the NvM configuration as required. Use the **Memory Stack Configurator** button to update the **Ea/Fee Block List** again.
- ▶ Use the **Finish** button to save changes and close the editor.

NOTE



The Finish button can only be used when the configuration is error free

If your configuration contains errors, refer to the error message displayed above the **NvM Block List** table. Adjust the configuration as indicated in the error message. Then use the **Finish** button to exit.

4.5. The Memory Stack Automation Unattended Wizard

4.5.1. Background information

All requirements in the [Section 4.2, “Background information”](#) which are related to the Memory Stack Editor, the **Memory Stack Configurator**, and the **Evaluate NV block needs assistant** are also applicable for the Memory Stack Automation Unattended Wizard.

If you want to configure the Memory Stack Automation Unattended Wizard, you need to open it from the **Unattended Wizard Dialog**. To open the **Unattended Wizard Dialog** you need the same requirements as they are used for the Memory Stack Editor.

4.5.2. Using the Memory Stack Automation Unattended Wizard

Ensure that the requirements are covered.

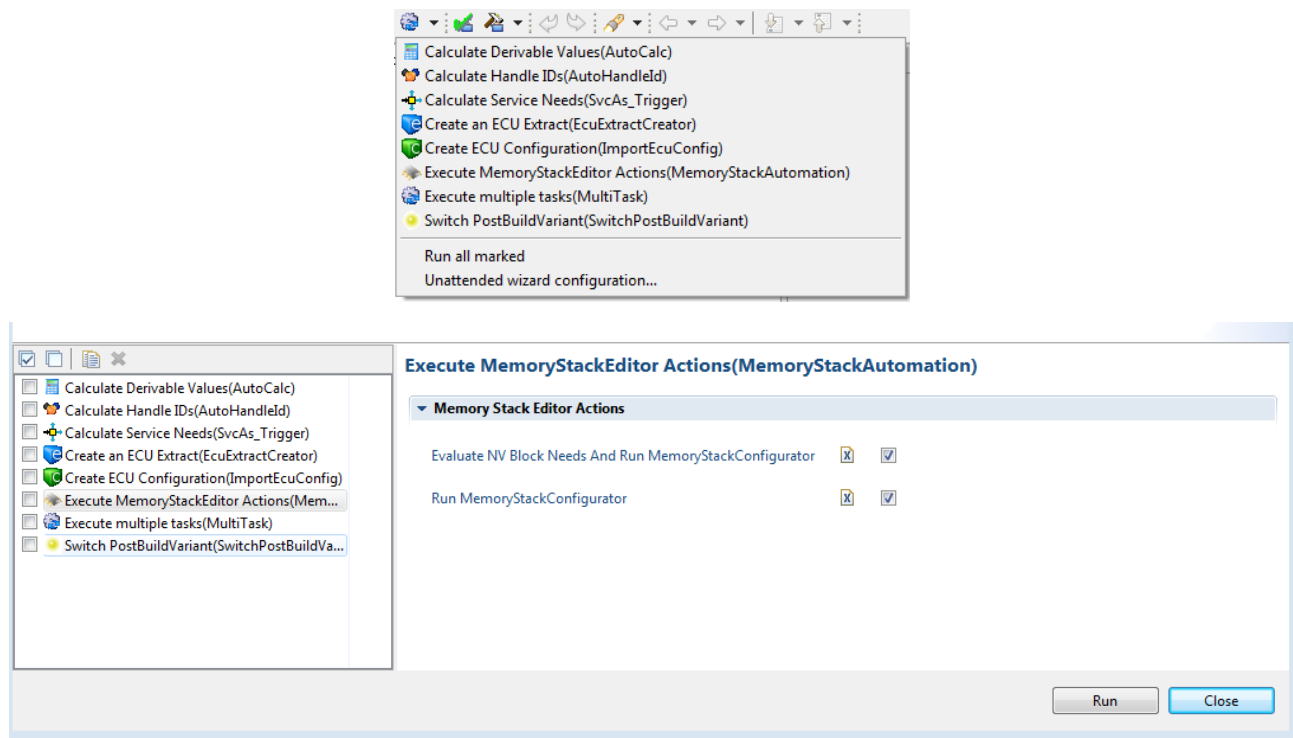


Step 1

Open the Memory Stack Automation Unattended Wizard inside the **Unattended Wizard Dialog**.

Step 2

Choose between the two **Memory Stack Editor Actions**.



Once the `Memory Stack Automation` is configured, you can execute from the command line, exactly as all of the other `Unattended Wizards`. In this case `Memory Stack Automation` can perform precisely the previously configured actions from UI.

If you execute from the command line and there are any configuration errors regarding the needed modules for the `Memory Stack Automation`, the performed changes are not saved.

In this case, fix the configuration problems in EB tresos Studio and perform the desired action again.

4.5.3. Configuring the memory stack with the Memory Stack Automation

The two `Memory Stack Editor Actions` of the **Memory Stack Automation**, the **Memory Stack Configurator**, and the **Evaluate NV block needs** work exactly as in the case of the **Memory Stack Editor**.

5. ACG8 Memory Stack module references

5.1. Overview

This chapter provides module references for the ACG8 Memory Stack product modules. These include a detailed description of all configuration parameters. Furthermore this chapter lists the application programming interface with all data types, constants and functions.

The content of the sections is sorted alphabetically according the EB tresos AutoCore Generic module names.

For further information on the functional behavior of these modules, refer to the chapter ACG8 Memory Stack user's guide.

5.1.1. Notation in EB module references

EB notation may differ from the AUTOSAR standard notation in the software specification documents (SWS). This section describes the notation of *default value* and *range* fields in the EB module references.

5.1.1.1. Default value of configuration parameters

If there is no default value specified for a parameter, the default value field is omitted to prevent ambiguity with parameters that have -- as default values.

Example: The parameter `BswMCompuConstText` of the `BswM` module of EB tresos AutoCore Generic 8 Mode Management has no default value field, therefore it is omitted.

5.1.1.2. Range information of configuration parameters

The range of a configuration parameter contains an upper and a lower boundary. However, in special cases the range of allowed values can be computed by means of an XPath function that is evaluated at configuration time. An XPath function can either be a standard `xpath:<function>()` or a custom `cxpath:<function>()` function. The range of a configuration parameter may be computed based on other configuration parameters that are referenced from the XPath function. For more information on custom XPath functions, see section *Custom XPath Functions API* of the EB tresos Studio developer's guide.

Example: The parameter `BswMCompuConstText` of the `BswM` module of EB tresos AutoCore Generic 8 Mode Management has the custom XPath function `cxpath:getCompuMethodsVT()` in the range field which provides the allowed values.

5.2. Crc

5.2.1. Configuration parameters

Containers included		
Container name	Multiplicity	Description
CommonPublishedInformation	1..1	Label: Common Published Information Common container, aggregated by all modules. It contains published information about vendor and versions.
CrcGeneral	1..1	General configuration of CRC module
PublishedInformation	1..1	Label: EB Published Information Additional published parameters not covered by CommonPublishedInformation container.

Parameters included	
Parameter name	Multiplicity
IMPLEMENTATION_CONFIG_VARIANT	1..1

Parameter Name	IMPLEMENTATION_CONFIG_VARIANT
Label	Config Variant
Multiplicity	1..1
Type	ENUMERATION
Default value	VariantPreCompile
Range	VariantPreCompile

5.2.1.1. CommonPublishedInformation

Parameters included	
Parameter name	Multiplicity
ArMajorVersion	1..1
ArMinorVersion	1..1

Parameters included	
ArPatchVersion	1..1
SwMajorVersion	1..1
SwMinorVersion	1..1
SwPatchVersion	1..1
ModuleId	1..1
VendorId	1..1
Release	1..1

Parameter Name	ArMajorVersion	
Label	AUTOSAR Major Version	
Description	Major version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	4	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	ArMinorVersion	
Label	AUTOSAR Minor Version	
Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	2	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	ArPatchVersion	
Label	AUTOSAR Patch Version	
Description	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	

Type	INTEGER_LABEL
Default value	0
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	SwMajorVersion
Label	Software Major Version
Description	Major version number of the vendor specific implementation of the module.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	6
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	SwMinorVersion
Label	Software Minor Version
Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	11
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	SwPatchVersion
Label	Software Patch Version
Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	13
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	ModuleId
Label	Numeric Module ID
Description	Module ID of this module from Module List
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	201
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	VendorId
Label	Vendor ID
Description	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	1
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	Release
Label	Release Information
Multiplicity	1..1
Type	STRING_LABEL
Default value	
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

5.2.1.2. CrcGeneral

Parameters included	
Parameter name	Multiplicity
Crc16Mode	0..1
Crc16TableSize	1..1

Parameters included	
Crc32Mode	0..1
Crc32TableSize	1..1
Crc64Mode	0..1
Crc64TableSize	1..1
Crc8H2FMode	0..1
Crc8H2FTableSize	1..1
Crc8Mode	0..1
Crc8TableSize	1..1
CrcDevErrorDetect	1..1

Parameter Name	Crc16Mode	
Label	CRC16 Mode	
Description	<p>Switch to select the CRC16 calculation method.</p> <p>Range:</p> <ul style="list-style-type: none"> ► CRC_16_RUNTIME: calculate CRC at runtime ► CRC_16_TABLE: use a lookup table for CRC values <p>The calculation is based on the CCITT CRC16 Standard.</p>	
Multiplicity	0..1	
Type	ENUMERATION	
Default value	CRC_16_TABLE	
Range	<div>CRC_16_RUNTIME</div> <div>CRC_16_TABLE</div>	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	Crc16TableSize	
Label	CRC16 Table Size	
Description	<p>Number of elements in the lookup table for the table based CRC16 calculation methods.</p> <p>Range:</p> <ul style="list-style-type: none"> ► CRC_TABLE_16_ELEMENTS: table size of 16 elements 	

	<p>► CRC_TABLE_256_ELEMENTS: table size of 256 elements</p> <p>Note: The table based algorithm with 256 entries is roughly twice as fast as the one for using 16 entries. However this uses 16 times more ROM to store the tables.</p> <p>This is an EB specific enhancement as AUTOSAR does not specify table based algorithms with different table sizes.</p>	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	CRC_TABLE_16_ELEMENTS	
Range	CRC_TABLE_16_ELEMENTS	
	CRC_TABLE_256_ELEMENTS	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	Crc32Mode	
Label	CRC32 Mode	
Description	<p>Switch to select the CRC32 calculation method.</p> <p>Range:</p> <p>► CRC_32_RUNTIME: calculate CRC at runtime</p> <p>► CRC_32_TABLE: use a lookup table for CRC values</p> <p>The calculation is based on the IEEE-802.3 CRC32 Ethernet Standard.</p> <p>Note: In this implementation, an optimized algorithm is used where the polynomial is reflected rather than the input. Therefore the reflected polynomial 0xEDB88320 is used rather than the polynomial 0x04C11DB7 defined by the standard.</p>	
Multiplicity	0..1	
Type	ENUMERATION	
Default value	CRC_32_TABLE	
Range	CRC_32_RUNTIME	
	CRC_32_TABLE	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	Crc32TableSize
Label	CRC32 Table Size
Description	<p>Number of elements in the lookup table for the table based CRC32 calculation methods.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ CRC_TABLE_16_ELEMENTS: table size of 16 elements ▶ CRC_TABLE_256_ELEMENTS: table size of 256 elements <p>Note: The table based algorithm with 256 entries is roughly twice as fast as the one for using 16 entries. However this uses 16 times more ROM to store the tables.</p> <p>This is an EB specific enhancement as AUTOSAR does not specify table based algorithms with different table sizes.</p>
Multiplicity	1..1
Type	ENUMERATION
Default value	CRC_TABLE_16_ELEMENTS
Range	<div>CRC_TABLE_16_ELEMENTS</div> <div>CRC_TABLE_256_ELEMENTS</div>
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	Crc64Mode
Label	CRC64 Mode
Description	<p>Switch to select the CRC64 calculation method.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ CRC_64_RUNTIME: calculate CRC at runtime ▶ CRC_64_TABLE: use a lookup table for CRC values <p>The calculation is based on the CRC-64-ECMA Standard (ECMA-182 / ISO/IEC 13421)</p> <p>In order to enable CRC64, first enable generation of base types for 64bit in Base/BaseTypes/BaseTypes64bit</p> <p>Note: In this implementation, an optimized algorithm is used where the polynomial is reflected rather than the input. Therefore the reflected polynomi-</p>

	al 0xC96C5795D7870F42 is used rather than the polynomial 0x42F0E1E-BA9EA3693 defined by the standard.	
Multiplicity	0..1	
Type	ENUMERATION	
Default value	CRC_64_TABLE	
Range	CRC_64_RUNTIME	
	CRC_64_TABLE	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	Crc64TableSize	
Label	CRC64 Table Size	
Description	<p>Number of elements in the lookup table for the table based CRC64 calculation methods.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ CRC_TABLE_16_ELEMENTS: table size of 16 elements ▶ CRC_TABLE_256_ELEMENTS: table size of 256 elements <p>Note: The table based algorithm with 256 entries is roughly twice as fast as the one for using 16 entries. However this uses 16 times more ROM to store the tables.</p> <p>This is an EB specific enhancement as AUTOSAR does not specify table based algorithms with different table sizes.</p>	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	CRC_TABLE_16_ELEMENTS	
Range	CRC_TABLE_16_ELEMENTS	
	CRC_TABLE_256_ELEMENTS	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	Crc8H2FMode	
Label	CRC8 H2F Mode	
Description	Switch to select the CRC8 (0x2F polynomial) calculation method.	

	Range: <ul style="list-style-type: none"> ▶ CRC_8_RUNTIME: calculate CRC at runtime ▶ CRC_8_TABLE: use a lookup table for CRC values 	
Multiplicity	0..1	
Type	ENUMERATION	
Default value	CRC_8H2F_TABLE	
Range	CRC_8H2F_RUNTIME	
	CRC_8H2F_TABLE	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	Crc8H2FTableSize	
Label	CRC8 H2F Table Size	
Description	<p>Number of elements in the lookup table for the table based CRC8 (0x2F polynomial) calculation methods.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ CRC_TABLE_16_ELEMENTS: table size of 16 elements ▶ CRC_TABLE_256_ELEMENTS: table size of 256 elements <p>Note: The table based algorithm with 256 entries is roughly twice as fast as the one for using 16 entries. However this uses 16 times more ROM to store the tables.</p> <p>This is an EB specific enhancement as AUTOSAR does not differentiate between table based algorithms with different table sizes.</p>	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	CRC_TABLE_16_ELEMENTS	
Range	CRC_TABLE_16_ELEMENTS	
	CRC_TABLE_256_ELEMENTS	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	Crc8Mode
Label	CRC8 Mode

Description	<p>Switch to select the CRC8 calculation method.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ <code>CRC_8_RUNTIME</code>: calculate CRC at runtime ▶ <code>CRC_8_TABLE</code>: use a lookup table for CRC values <p>The calculation is based on the SAE-J1850 CRC8 Standard.</p>	
Multiplicity	0..1	
Type	ENUMERATION	
Default value	<code>CRC_8_TABLE</code>	
Range	<code>CRC_8_RUNTIME</code>	
	<code>CRC_8_TABLE</code>	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	Crc8TableSize	
Label	CRC8 Table Size	
Description	<p>Number of elements in the lookup table for the table based CRC8 calculation methods.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ <code>CRC_TABLE_16_ELEMENTS</code>: table size of 16 elements ▶ <code>CRC_TABLE_256_ELEMENTS</code>: table size of 256 elements <p>Note: The table based algorithm with 256 entries is roughly twice as fast as the one for using 16 entries. However this uses 16 times more ROM to store the tables.</p> <p>This is an EB specific enhancement as AUTOSAR does not specify table based algorithms with different table sizes.</p>	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	<code>CRC_TABLE_16_ELEMENTS</code>	
Range	<code>CRC_TABLE_16_ELEMENTS</code>	
	<code>CRC_TABLE_256_ELEMENTS</code>	
Configuration class	VariantPreCompile:	VariantPreCompile

Origin	Elektrobit Automotive GmbH
---------------	----------------------------

Parameter Name	CrcDevErrorDetect	
Label	Enable Development Error Detection	
Description	<p>Enables the use of development error detection.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Development error detection enabled ▶ <code>false</code> = Development error detection disabled 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.2.1.3. PublishedInformation

Parameters included	
Parameter name	Multiplicity
PbcfgMSupport	1..1

Parameter Name	PbcfgMSupport	
Label	PbcfgM support	
Description	Specifies whether or not the Crc can use the PbcfgM module for post-build support.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

5.2.2. Recommended configurations

5.2.2.1. CrcRecConfigurationCrc16Crc32

Containers included	
Container name	Container definition
CrcGeneral	CrcGeneral

Parameters included	
Parameter name	Value

5.2.2.1.1. CrcGeneral

Parameters included	
Parameter name	Value
Crc16Mode	CRC_16_TABLE
Crc32Mode	CRC_32_TABLE

5.2.3. Application programming interface (API)

5.2.3.1. Macro constants

5.2.3.1.1. CRC_AR_RELEASE_MAJOR_VERSION

Purpose	AUTOSAR release major version.
Value	4U

5.2.3.1.2. CRC_AR_RELEASE_MINOR_VERSION

Purpose	AUTOSAR release minor version.
----------------	--------------------------------

Value	0U
--------------	----

5.2.3.1.3. CRC_AR_RELEASE_REVISION_VERSION

Purpose	AUTOSAR release revision version.
Value	3U

5.2.3.1.4. CRC_E_PARAM_DATA

Purpose	DET Error: Invalid function parameter.
Value	1U

5.2.3.1.5. CRC_GET_VERSION_INFO_API_ID

Purpose	Service ID for Crc_GetVersionInfo.
Value	0x04U

5.2.3.1.6. CRC_INSTANCE_ID

Purpose	Module's instance ID.
Value	0U

5.2.3.1.7. CRC_MODULE_ID

Purpose	AUTOSAR module identification.
Value	201U

5.2.3.1.8. CRC_SW_MAJOR_VERSION

Purpose	AUTOSAR module major version.
----------------	-------------------------------

Value	6U
--------------	----

5.2.3.1.9. CRC_SW_MINOR_VERSION

Purpose	AUTOSAR module minor version.
Value	11U

5.2.3.1.10. CRC_SW_PATCH_VERSION

Purpose	AUTOSAR module patch version.
Value	13U

5.2.3.1.11. CRC_VENDOR_ID

Purpose	AUTOSAR vendor identification: Elektrobit Automotive GmbH.
Value	1U

5.2.3.1.12. DBG_CRC_CALCULATECRC16_ENTRY

Purpose	Entry point of function Crc_CalculateCRC16() .
Value	

5.2.3.1.13. DBG_CRC_CALCULATECRC16_EXIT

Purpose	Exit point of function Crc_CalculateCRC16() .
Value	

5.2.3.1.14. DBG_CRC_CALCULATECRC32_ENTRY

Purpose	Entry point of function Crc_CalculateCRC32() .
----------------	--

Value	
--------------	--

5.2.3.1.15. DBG_CRC_CALCULATECRC32_EXIT

Purpose	Exit point of function Crc_CalculateCRC32() .
Value	

5.2.3.1.16. DBG_CRC_CALCULATECRC64_ENTRY

Purpose	Entry point of function Crc_CalculateCRC64() .
Value	

5.2.3.1.17. DBG_CRC_CALCULATECRC64_EXIT

Purpose	Exit point of function Crc_CalculateCRC64() .
Value	

5.2.3.1.18. DBG_CRC_CALCULATECRC8H2F_ENTRY

Purpose	Entry point of function Crc_CalculateCRC8H2F() .
Value	

5.2.3.1.19. DBG_CRC_CALCULATECRC8H2F_EXIT

Purpose	Exit point of function Crc_CalculateCRC8H2F() .
Value	

5.2.3.1.20. DBG_CRC_CALCULATECRC8_ENTRY

Purpose	Entry point of function Crc_CalculateCRC8() .
----------------	---

Value	
-------	--

5.2.3.1.21. DBG_CRC_CALCULATECRC8_EXIT

Purpose	Exit point of function Crc_CalculateCRC8() .
Value	

5.2.3.1.22. DBG_CRC_GETVERSIONINFO_ENTRY

Purpose	Entry point of function Crc_GetVersionInfo() .
Value	

5.2.3.1.23. DBG_CRC_GETVERSIONINFO_EXIT

Purpose	Exit point of function Crc_GetVersionInfo() .
Value	

5.2.3.2. Functions

5.2.3.2.1. Crc_CalculateCRC16

Purpose	Calculation of CRC16.	
Synopsis	<pre>uint16 Crc_CalculateCRC16 (const uint8 * Crc_DataPtr , uint32 Crc_Length , uint16 Crc_StartValue16 , boolean Crc_IsFirstCall);</pre>	
Service ID	2	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Crc_DataPtr	Valid pointer to start address of data block
	Crc_Length	Length of data block in bytes
	Crc_StartValue16	Initial Value
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation; start from initial

	value, ignore Crc_StartValue16. FALSE: Subsequent call in a call sequence; Crc_StartValue16 is interpreted to be the return value of the previous function call.
Return Value	16 bit result of CRC calculation
Description	This function performs the calculation of a CRC16 value (runtime or table variant, depending on Configuration Parameter Crc16Mode) over the memory block referenced by Crc_DataPtr of byte length Crc_Length.

5.2.3.2.2. Crc_CalculateCRC32

Purpose	Calculation of CRC32.	
Synopsis	<pre>uint32 Crc_CalculateCRC32 (const uint8 * Crc_DataPtr , uint32 Crc_Length , uint32 Crc_StartValue32 , boolean Crc_IsFirstCall);</pre>	
Service ID	3	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Crc_DataPtr	Valid pointer to start address of data block
	Crc_Length	Length of data block in bytes
	Crc_StartValue32	Initial Value
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue32. FALSE: Subsequent call in a call sequence; Crc_StartValue32 is interpreted to be the return value of the previous function call.
Return Value	calculated CRC32 value	
Description	This function performs the calculation of a CRC32 value (runtime or table variant, depending on Configuration Parameter Crc32Mode) over the memory block referenced by Crc_DataPtr of byte length Crc_Length.	

5.2.3.2.3. Crc_CalculateCRC64

Purpose	Calculation of CRC64.
----------------	-----------------------

Synopsis	<pre>uint64 Crc_CalculateCRC64 (const uint8 * Crc_DataPtr , uint32 Crc_Length , uint64 Crc_StartValue64 , boolean Crc_IsFirstCall);</pre>	
Service ID	0x07	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Crc_DataPtr	Valid pointer to start address of data block
	Crc_Length	Length of data block in bytes
	Crc_StartValue64	Initial Value
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue64. FALSE: Subsequent call in a call sequence; Crc_StartValue64 is interpreted to be the return value of the previous function call.
Return Value	calculated CRC64 value	
Description	<p>This function performs the calculation of a CRC64 value based on CRC-64-EC-MA standard (runtime or table variant, depending on Configuration Parameter Cr-c64Mode) over the memory block referenced by Crc_DataPtr of byte length Crc_Length.</p>	

5.2.3.2.4. Crc_CalculateCRC8

Purpose	Calculation of CRC8.	
Synopsis	<pre>uint8 Crc_CalculateCRC8 (const uint8 * Crc_DataPtr , uint32 Crc_Length , uint8 Crc_StartValue8 , boolean Crc_IsFirstCall);</pre>	
Service ID	1	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Crc_DataPtr	Valid pointer to start address of data block
	Crc_Length	Length of data block in bytes
	Crc_StartValue8	Initial Value
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue8. FALSE: Subsequent call in a call sequence; Crc_Start-

		Value8 is interpreted to be the return value of the previous function call.
Return Value	8 bit result of CRC calculation.	
Description	This function performs the calculation of a 8-bit SAE J1850 CRC value (runtime or table variant, depending on Configuration Parameter Crc8Mode) over the memory block referenced by Crc_DataPtr of byte length Crc_Length.	

5.2.3.2.5. Crc_CalculateCRC8H2F

Purpose	Calculation of CRC8H2F.	
Synopsis	<pre>uint8 Crc_CalculateCRC8H2F (const uint8 * Crc_DataPtr , uint32 Crc_Length , uint8 Crc_StartValue8H2F , boolean Crc_IsFirstCall);</pre>	
Service ID	5	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Crc_DataPtr	Valid pointer to start address of data block
	Crc_Length	Length of data block in bytes
	Crc_StartValue8H2F	Initial Value
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue8H2F. FALSE: Subsequent call in a call sequence; Crc_StartValue8H2F is interpreted to be the return value of the previous function call.
Return Value	8 bit result of CRC calculation.	
Description	This function performs the calculation of a 8-bit CRC with polynom 0x2F (runtime or table variant, depending on Configuration Parameter Crc8H2FMode) over the memory block referenced by Crc_DataPtr of byte length Crc_Length.	

5.2.3.2.6. Crc_GetVersionInfo

Purpose	Return the modules version information.
Synopsis	<pre>void Crc_GetVersionInfo (Std_VersionInfoType * VersionInfoPtr);</pre>

Service ID	4	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (out)	<code>VersionInfoPtr</code>	Pointer to struct to be filled with the version information
Description	This function provides the information to module vendor ID, module ID and software version major.minor.patch	

5.2.4. Integration notes

5.2.4.1. Exclusive areas

Exclusive areas are not used by the `Crc` module.

5.2.4.2. Production errors

Production errors are not reported by the `Crc` module.

5.2.4.3. Memory mapping

General information about memory mapping is provided in the EB tresos AutoCore Generic documentation. Refer to the section `Memory mapping and compiler abstraction` in the `Integration notes` section for details.

The following table provides the list of sections that may be mapped for this module:

Memory section
<code>CONST_8</code>
<code>CONST_16</code>
<code>CONST_32</code>
<code>CONST_64</code>
<code>CODE</code>

5.2.4.4. Integration requirements

WARNING



Integration requirements list is not exhaustive

The following list of integration requirements helps you to integrate your product. However, this list is not exhaustive. You also require information from the user guide, release notes, and EB tresos AutoCore known issues to successfully integrate your product.

Integration requirements are not listed for the Crc module.

5.3. Ea

5.3.1. Configuration parameters

Containers included		
Container name	Multiplicity	Description
CommonPublishedInformation	1..1	Label: Common Published Information Common container, aggregated by all modules. It contains published information about vendor and versions.
EaBlockConfiguration	1..n	Configuration of block specific parameters for the EEPROM abstraction module.
EaGeneral	1..1	Label: General Configuration Parameters General configuration of the EEPROM abstraction module. This container lists block independent configuration parameters.
EaPublishedInformation	1..1	Label: Ea Published Information Additional published parameters not covered by Common-PublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.
PublishedInformation	1..1	Label: EB Published Information Additional published parameters not covered by Common-PublishedInformation container.

Parameters included	
Parameter name	Multiplicity
IMPLEMENTATION_CONFIG_VARIANT	1..1

Parameter Name	IMPLEMENTATION_CONFIG_VARIANT
Label	Config Variant
Multiplicity	1..1
Type	ENUMERATION
Default value	VariantPreCompile
Range	VariantPreCompile

5.3.1.1. CommonPublishedInformation

Parameters included	
Parameter name	Multiplicity
ArMajorVersion	1..1
ArMinorVersion	1..1
ArPatchVersion	1..1
SwMajorVersion	1..1
SwMinorVersion	1..1
SwPatchVersion	1..1
ModuleId	1..1
VendorId	1..1
Release	1..1

Parameter Name	ArMajorVersion
Label	AUTOSAR Major Version
Description	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	2
Configuration class	PublishedInformation:

Origin	Elektrobit Automotive GmbH
---------------	----------------------------

Parameter Name	ArMinorVersion	
Label	AUTOSAR Minor Version	
Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	0	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	ArPatchVersion	
Label	AUTOSAR Patch Version	
Description	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	0	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	SwMajorVersion	
Label	Software Major Version	
Description	Major version number of the vendor specific implementation of the module.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	5	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	SwMinorVersion	
Label	Software Minor Version	
Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.	

Multiplicity	1..1
Type	INTEGER_LABEL
Default value	12
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	SwPatchVersion
Label	Software Patch Version
Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	14
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	ModuleId
Label	Numeric Module ID
Description	Module ID of this module from Module List
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	40
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	VendorId
Label	Vendor ID
Description	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	1
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	Release
Label	Release Information
Multiplicity	1..1
Type	STRING_LABEL
Default value	
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

5.3.1.2. EaBlockConfiguration

Parameters included	
Parameter name	Multiplicity
EaBlockNumber	1..1
EaBlockSize	1..1
EaImmediateData	1..1
EaNumberOfWriteCycles	1..1
EaDeviceIndex	1..1

Parameter Name	EaBlockNumber
Label	Block Number
Description	<p>Block identifier (handle).</p> <p>0x0000 and 0xFFFF shall not be used for block numbers.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ min = $2^{\text{NVM_DATA_SELECTION_BITS}}$ ▶ max = $0\text{xFFFF} - (2^{\text{NVM_DATA_SELECTION_BITS}} - 1)$ <p>Note: Depending on the number of bits set aside for dataset selection several other block numbers shall also be left out to ease implementation.</p>
Multiplicity	1..1
Type	INTEGER
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	EaBlockSize
----------------	-------------

Label	Block Size
Description	Size of the logical block in bytes. Note: If CRC is configured for the block in the NVRAM manager, the size configured here must be large enough for the number of bytes of block data and the number of bytes needed for the CRC type configured.
Multiplicity	1..1
Type	INTEGER
Default value	1
Range	<=65535 >=1
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	EaImmediateData
Label	Immediate Data
Description	Marker for high priority data. This must be set consistently with the block priority configured in the NVRAM manager. A block with priority value 0 shall be marked as containing immediate data here. ► <code>true</code> = Block contains immediate data. ► <code>false</code> = Block does not contain immediate data.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	EaNumberOfWriteCycles
Label	Number of write cycles
Description	Number of write cycles required for this block. Range: 1 . . 4294967295 Restriction: The number of write cycles divided by the number of write cycles supported by the EEPROM driver must be less than 256.

Multiplicity	1..1
Type	INTEGER
Default value	100000
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	EaDeviceIndex
Label	EEPROM Device
Description	EEPROM (Eep) device where the NVRAM block is located. This information is referenced by the NVRAM manager and the Memory Abstraction Interface to address this logical block.
Multiplicity	1..1
Type	SYMBOLIC-NAME-REFERENCE
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

5.3.1.3. EaGeneral

Parameters included	
Parameter name	Multiplicity
EaBuffer	0..1
EaDevErrorDetect	1..1
EaIndex	1..1
EaMainFunctionPeriod	1..1
EaNvmJobEndNotification	0..1
EaNvmJobErrorNotification	0..1
EaPollingMode	1..1
EaSetModeSupported	1..1
EaVersionInfoApi	1..1
EaVirtualPageSize	1..1

Parameter Name	EaBuffer
Label	Internal Buffer Size

Description	<p>The size in bytes to be used for the internal buffer.</p> <p>Optimal runtime performance can be achieved by setting this value to the maximum size of a block including its management overhead.</p>	
Multiplicity	0..1	
Type	INTEGER	
Range	<=65535	
	>=0	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	EaDevErrorDetect	
Label	Enable Development Error Detection	
Description	<p>Enables the use of development error detection.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Development error detection enabled. ▶ <code>false</code> = Development error detection disabled. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	EaIndex	
Label	Module Instance Index	
Description	<p>This index number specifies the instance of this module. If only one instance is present it shall have the index number 0.</p> <p><i>The current implementation only supports one instance of the Ea module.</i></p>	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<=254	
	>=0	
Configuration class	VariantPreCompile:	VariantPreCompile

Origin	AUTOSAR_ECUC
---------------	--------------

Parameter Name	EaMainFunctionPeriod	
Description	This parameter defines the periodic cycle time, in seconds, for calling Ea main function.	
Multiplicity	1..1	
Type	FLOAT	
Default value	0.01	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	EaNvmJobEndNotification	
Label	NvM Job End Notification Function	
Description	This is mapped to the job end notification routine provided by the upper layer module (NvM_JobEndNotification). The Ea will call this function on successful completion of a job request.	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Default value	NvM_JobEndNotification	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	EaNvmJobErrorNotification	
Label	NvM Job Error Notification Function	
Description	This is mapped to the job error notification routine provided by the upper layer module (NvM_JobErrorNotification). The Ea will call this function if an error occurs during the processing of a job request.	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Default value	NvM_JobErrorNotification	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	EaPollingMode	
Label	Enable Polling Mode	

Description	<p>Enables the polling mode for this module. If polling mode is enabled, the Ea will poll the underlying Eep module for job results. If polling mode is disabled, the Ea shall provide notification functions that can be called by the underlying Eep module when jobs have completed.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Polling mode enabled. ▶ <code>false</code> = Polling mode disabled. <p><i>The functionality related to this parameter is not fully supported by the current implementation. Regardless of this switch setting, the Ea will always use polling mode.</i></p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	EaSetModeSupported	
Label	Enable Set Mode API	
Description	<p>Enables the <code>Ea_SetMode()</code> API function.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = <code>Ea_SetMode()</code> enabled. ▶ <code>false</code> = <code>Ea_SetMode()</code> disabled. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	EaVersionInfoApi	
Label	Enable Version Info API	
Description	<p>Enables the <code>Ea_GetVersionInfo()</code> API function.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = <code>Ea_GetVersionInfo()</code> enabled. ▶ <code>false</code> = <code>Ea_GetVersionInfo()</code> disabled. 	
Multiplicity	1..1	
Type	BOOLEAN	

Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	EaVirtualPageSize
Label	Virtual Page Size
Description	<p>The size in bytes to which logical blocks shall be aligned.</p> <p>The value must be set to <code>EEP_WRITE_UNIT_SIZE</code> if <code>EEP_WRITE_UNIT_SIZE</code> is greater than or equal to <code>EEP_ERASE_UNIT_SIZE</code>, else it must be set to <code>EEP_ERASE_UNIT_SIZE</code>.</p> <p>The parameters <code>EEP_ERASE_UNIT_SIZE</code> and <code>EEP_WRITE_UNIT_SIZE</code> are published by the Eeprom Driver.</p>
Multiplicity	1..1
Type	INTEGER
Default value	1
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

5.3.1.4. EaPublishedInformation

Parameters included	
Parameter name	Multiplicity
EaBlockOverhead	1..1
EaMaximumBlockingTime	1..1
EaPageOverhead	1..1

Parameter Name	EaBlockOverhead
Label	Block Management Overhead
Description	<p>Management overhead per logical block in bytes.</p> <p>Note: The formula to calculate Block overhead is, the bytes required for aligning the configured size of block along with 2 bytes (for management information) to virtual page size + virtual page size.</p>
Multiplicity	1..1

Type	INTEGER_LABEL	
Default value	4	
Range	<=65535	
	>=0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	EaMaximumBlockingTime	
Label	Maximum Blocking Time	
Description	<p>The maximum time the EA module's API routines shall be blocked (delayed) by internal operations.(EA070)</p> <p>unit: [s]</p> <p>Note(1): Configuration generators need to convert this value into a hardware specific representation (ticks) when generating the modules configuration header file.</p> <p>Note(2): Internal operations in this case means operations that are not explicitly invoked from the upper layer module but need to be handled for proper operation of this module or the underlying memory driver.</p> <p><i>This parameter is not used by the implementation.</i></p>	
Multiplicity	1..1	
Type	FLOAT_LABEL	
Default value	0.0	
Range	<=Infinity	
	>=0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	EaPageOverhead	
Label	Page Management Overhead	
Description	<p>Management overhead per page in bytes.</p> <p>Note: There is no overhead per virtual page but only an overhead per block.</p> <p><i>This parameter is not used internally by the current implementation.</i></p>	

Multiplicity	1..1
Type	INTEGER_LABEL
Default value	0
Range	<div><=65535</div> <div>>=0</div>
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

5.3.1.5. PublishedInformation

Parameters included	
Parameter name	Multiplicity
PbcfgMSupport	1..1

Parameter Name	PbcfgMSupport
Label	PbcfgM support
Description	Specifies whether or not the Ea can use the PbcfgM module for post-build support.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

5.3.2. Recommended configurations

5.3.2.1. EaRecConfiguration

Containers included	
Container name	Container definition

Containers included	
EaBlockConfiguration_0	EaBlockConfiguration
EaBlockConfiguration_1	EaBlockConfiguration
Parameters included	
Parameter name	Value

5.3.2.1.1. EaBlockConfiguration_0

Parameters included	
Parameter name	Value
EaBlockNumber	16
EaImmediateData	true
EaBlockSize	4

5.3.2.1.2. EaBlockConfiguration_1

Parameters included	
Parameter name	Value
EaBlockNumber	17
EaImmediateData	true
EaBlockSize	4

5.3.3. Application programming interface (API)

5.3.3.1. Macro constants

5.3.3.1.1. DBG_EA_CANCEL_ENTRY

Purpose	Entry point of function Ea_Cancel() .
Value	

5.3.3.1.2. DBG_EA_CANCEL_EXIT

Purpose	Exit point of function Ea_Cancel() .
Value	

5.3.3.1.3. DBG_EA_ERASEIMMEDIATEBLOCK_ENTRY

Purpose	Entry point of function Ea_EraseImmediateBlock() .
Value	

5.3.3.1.4. DBG_EA_ERASEIMMEDIATEBLOCK_EXIT

Purpose	Exit point of function Ea_EraseImmediateBlock() .
Value	

5.3.3.1.5. DBG_EA_GETJOBRESULT_ENTRY

Purpose	Entry point of function Ea_GetJobResult() .
Value	

5.3.3.1.6. DBG_EA_GETJOBRESULT_EXIT

Purpose	Exit point of function Ea_GetJobResult() .
Value	

5.3.3.1.7. DBG_EA_GETSTATUS_ENTRY

Purpose	Entry point of function Ea_GetStatus() .
Value	

5.3.3.1.8. DBG_EA_GETSTATUS_EXIT

Purpose	Exit point of function Ea_GetStatus() .
----------------	---

Value	
--------------	--

5.3.3.1.9. DBG_EA_GETVERSIONINFO_ENTRY

Purpose	Entry point of function Ea_GetVersionInfo() .
Value	

5.3.3.1.10. DBG_EA_GETVERSIONINFO_EXIT

Purpose	Exit point of function Ea_GetVersionInfo() .
Value	

5.3.3.1.11. DBG_EA_INIT_ENTRY

Purpose	Entry point of function Ea_Init() .
Value	

5.3.3.1.12. DBG_EA_INIT_EXIT

Purpose	Exit point of function Ea_Init() .
Value	

5.3.3.1.13. DBG_EA_INVALIDATEBLOCK_ENTRY

Purpose	Entry point of function Ea_InvalidateBlock() .
Value	

5.3.3.1.14. DBG_EA_INVALIDATEBLOCK_EXIT

Purpose	Exit point of function Ea_InvalidateBlock() .
Value	

5.3.3.1.15. DBG_EA_JOBENDNOTIFICATION_ENTRY

Purpose	Entry point of function Ea_JobEndNotification() .
Value	

5.3.3.1.16. DBG_EA_JOBENDNOTIFICATION_EXIT

Purpose	Exit point of function Ea_JobEndNotification() .
Value	

5.3.3.1.17. DBG_EA_JOBERRORNOTIFICATION_ENTRY

Purpose	Entry point of function Ea_JobErrorNotification() .
Value	

5.3.3.1.18. DBG_EA_JOBERRORNOTIFICATION_EXIT

Purpose	Exit point of function Ea_JobErrorNotification() .
Value	

5.3.3.1.19. DBG_EA_JOBRESULT

Purpose	Change of Ea_JobResult.
Value	

5.3.3.1.20. DBG_EA_MAINFUNCTION_ENTRY

Purpose	Entry point of function Ea_MainFunction() .
Value	

5.3.3.1.21. DBG_EA_MAINFUNCTION_EXIT

Purpose	Exit point of function Ea_MainFunction() .
----------------	--

Value	
--------------	--

5.3.3.1.22. DBG_EA_MAINSTATE

Purpose	Change of Ea_MainState.
Value	

5.3.3.1.23. DBG_EA_READ_ENTRY

Purpose	Entry point of function Ea_Read() .
Value	

5.3.3.1.24. DBG_EA_READ_EXIT

Purpose	Exit point of function Ea_Read() .
Value	

5.3.3.1.25. DBG_EA_SETMODE_ENTRY

Purpose	Entry point of function Ea_SetMode() .
Value	

5.3.3.1.26. DBG_EA_SETMODE_EXIT

Purpose	Exit point of function Ea_SetMode() .
Value	

5.3.3.1.27. DBG_EA_STATUS

Purpose	Change of Ea_Status.
Value	

5.3.3.1.28. DBG_EA_WRITE_ENTRY

Purpose	Entry point of function Ea_Write() .
Value	

5.3.3.1.29. DBG_EA_WRITE_EXIT

Purpose	Exit point of function Ea_Write() .
Value	

5.3.3.1.30. EA_API_CANCEL

Purpose	Service ID for Ea_Cancel.
Value	0x04U

5.3.3.1.31. EA_API_ERASEIMMEDIATEBLOCK

Purpose	Service ID for Ea_EraseImmediateBlock.
Value	0x09U

5.3.3.1.32. EA_API_GETJOBRESULT

Purpose	Service ID for Ea_GetJobResult.
Value	0x06U

5.3.3.1.33. EA_API_GETSTATUS

Purpose	Service ID for Ea_GetStatus.
Value	0x05U

5.3.3.1.34. EA_API_GETVERSIONINFO

Purpose	Service ID for Ea_GetVersionInfo.
Value	0x08U

5.3.3.1.35. EA_API_INIT

Purpose	Service ID for Ea_Init.
Value	0x00U

5.3.3.1.36. EA_API_INVALIDATEBLOCK

Purpose	Service ID for Ea_InvalidateBlock.
Value	0x07U

5.3.3.1.37. EA_API_JOBENDNOTIFICATION

Purpose	Service ID for Ea_JobEndNotification.
Value	0x10U

5.3.3.1.38. EA_API_JOBERRORNOTIFICATION

Purpose	Service ID for Ea_JobErrorNotification.
Value	0x11U

5.3.3.1.39. EA_API_MAINFUNCTION

Purpose	Service ID for Ea_MainFunction.
Value	0x12U

5.3.3.1.40. EA_API_READ

Purpose	Service ID for Ea_Read.
Value	0x02U

5.3.3.1.41. EA_API_SETMODE

Purpose	Service ID for Ea_SetMode.
Value	0x01U

5.3.3.1.42. EA_API_WRITE

Purpose	Service ID for Ea_Write.
Value	0x03U

5.3.3.1.43. EA_AR_RELEASE_MAJOR_VERSION

Purpose	AUTOSAR release major version.
Value	4U

5.3.3.1.44. EA_AR_RELEASE_MINOR_VERSION

Purpose	AUTOSAR release minor version.
Value	0U

5.3.3.1.45. EA_AR_RELEASE_REVISION_VERSION

Purpose	AUTOSAR release revision version.
Value	3U

5.3.3.1.46. EA_E_BUSY

Purpose	Development error indicating that the status is BUSY.
Value	(uint8)0x06U

5.3.3.1.47. EA_E_BUSY_INTERNAL

Purpose	Development error indicating that the status is BUSY due to internal operation.
Value	(uint8)0x07U

5.3.3.1.48. EA_E_INVALID_BLOCK_LEN

Purpose	Development error indicating that the Block length is invalid.
Value	(uint8)0x05U

5.3.3.1.49. EA_E_INVALID_BLOCK_NO

Purpose	DET Error: invalid block number.
Value	2U

5.3.3.1.50. EA_E_INVALID_BLOCK_OFS

Purpose	Development error indicating that the block offset is invalid.
Value	(uint8)0x03U

5.3.3.1.51. EA_E_INVALID_CANCEL

Purpose	Development error indicating that the cancel request is invalid.
Value	(uint8)0x08U

5.3.3.1.52. EA_E_INVALID_DATA_PTR

Purpose	Development error indicating that a null pointer is passed as a parameter.
Value	(uint8)0x04U

5.3.3.1.53. EA_E_INVALID_MODE

Purpose	Development error indicating that the requested mode is invalid.
Value	(uint8)0x09U

5.3.3.1.54. EA_E_UNINIT

Purpose	DET Error: module not initialized.
Value	1U

5.3.3.1.55. EA_INSTANCE_ID

Purpose	Module's instance ID.
----------------	-----------------------

Value	0U
--------------	----

5.3.3.1.56. EA_MODULE_ID

Purpose	AUTOSAR module identification.
Value	40U

5.3.3.1.57. EA_SW_MAJOR_VERSION

Purpose	AUTOSAR module major version.
Value	5U

5.3.3.1.58. EA_SW_MINOR_VERSION

Purpose	AUTOSAR module minor version.
Value	12U

5.3.3.1.59. EA_SW_PATCH_VERSION

Purpose	AUTOSAR module patch version.
Value	14U

5.3.3.1.60. EA_VENDOR_ID

Purpose	AUTOSAR vendor identification: Elektrobit Automotive GmbH.
Value	1U

5.3.3.2. Functions

5.3.3.2.1. Ea_Cancel

Purpose	Calls the function Eep_Cancel.
----------------	--------------------------------

Synopsis	<code>void Ea_Cancel (void);</code>
Service ID	4
Sync/Async	asynchronous
Reentrancy	non reentrant
Description	This function calls the Eep_Cancel() to cancel ongoing jobs.

5.3.3.2.2. Ea_EraseImmediateBlock

Purpose	Pre-erase the data in the given block.	
Synopsis	<code>Std_ReturnType Ea_EraseImmediateBlock (uint16 BlockNumber);</code>	
Service ID	9	
Sync/Async	asynchronous	
Reentrancy	non reentrant	
Parameters (in)	BlockNumber	Identifier of the block (from the configuration).
Return Value	Standard Return Code	
	E_OK	Job accepted.
	E_NOT_OK	BlockNumber not found.
Description	This function erases the selected block so that data can be immediately written during the next write operation. Uses the EEPROM Driver function Eep_Erase. When multiple copies are present for a block, the job of the function first invalidates the latest copy for the block in order to ensure that old data will not be returned by a subsequent Ea_Read. It then erases the location where the next copy is to be written.	

5.3.3.2.3. Ea_GetJobResult

Purpose	Returns the current job result.	
Synopsis	<code>MemIf_JobResultType Ea_GetJobResult (void);</code>	
Service ID	6	
Sync/Async	synchronous	
Reentrancy	non reentrant	
Return Value	MemIf_JobResultType	
	MEMIF_JOB_OK	Job finished successfully.

	MEMIF_JOB_PENDING	Job is being performed.
	MEMIF_JOB_CANCELED	Job has been cancelled.
	MEMIF_JOB_FAILED	Job failed.
	MEMIF_BLOCK_INCONSISTENT	Block inconsistent, can't read data.
	MEMIF_BLOCK_INVALID	Block marked invalid, can't read data.
Description	This function returns the status of the current job or the last finished one if none is ongoing. Calls the function Eep_GetJobResult of the EEPROM Driver. If it returns MEMIF_JOB_OK, the result from the EEPROM Abstraction module is returned, otherwise, the result from the EEPROM Driver is returned.	

5.3.3.2.4. Ea_GetStatus

Purpose	Returns the status of the Ea.	
Synopsis	MemIf_StatusType Ea_GetStatus (void);	
Service ID	5	
Sync/Async	synchronous	
Reentrancy	non reentrant	
Return Value	MemIf_StatusType	
	MEMIF_UNINIT	The underlying driver has not been initialized.
	MEMIF_IDLE	The underlying driver and Ea module are idle.
	MEMIF_BUSY	The underlying driver is busy.
Description	Calls the Eep_GetStatus function of the underlying EEPROM Driver and passes the returned value back to the caller. If the underlying driver returned the status MEMIF_IDLE and an internal operation is currently ongoing in the EEPROM Abstraction module, this function returns MEMIF_BUSY_INTERNAL.	

5.3.3.2.5. Ea_GetVersionInfo

Purpose	Returns Ea's version information.	
Synopsis	void Ea_GetVersionInfo (Std_VersionInfoType * VersionInfoPtr);	
Service ID	8	

Sync/Async	synchronous	
Reentrancy	reentrant	
Parameters (out)	<code>VersionInfoPtr</code>	Pointer to standard version information structure.
Description	This service returns the version information of this module. The version information includes Module ID, Vendor ID and vendor specific version numbers. NOTE: This function cannot be disabled by configuration parameter <code>EaVersionInfoApi</code> but the function is only linked to the user application when it is invoked.	

5.3.3.2.6. Ea_Init

Purpose	Initializes the Ea module.	
Synopsis	<code>void Ea_Init (void);</code>	
Service ID	0	
Sync/Async	synchronous	
Reentrancy	non reentrant	
Description	Initialize the Ea module.	

5.3.3.2.7. Ea_InvalidateBlock

Purpose	Marks the given block as invalidated.	
Synopsis	<code>Std_ReturnType Ea_InvalidateBlock (uint16 BlockNumber);</code>	
Service ID	7	
Sync/Async	asynchronous	
Reentrancy	non reentrant	
Parameters (in)	<code>BlockNumber</code>	Identifier of the block (from the configuration).
Return Value	Standard Return Code	
	<code>E_OK</code>	Job accepted.
	<code>E_NOT_OK</code>	BlockNumber not found.
Description	This function marks the selected block as invalid. Uses the EEPROM Driver function <code>Eep_Write</code> .	

5.3.3.2.8. Ea_JobEndNotification

Purpose	Called by the EEPROM driver when a job is successfully finished.
Synopsis	<code>void Ea_JobEndNotification (void);</code>
Service ID	16
Sync/Async	synchronous
Reentrancy	non reentrant
Description	This callback function is called by the underlying EEPROM Driver to report the successful end of an asynchronous operation. When a job given to the EEPROM Abstraction module is finished, a callback function will be called in the upper layer.

5.3.3.2.9. Ea_JobErrorNotification

Purpose	Called by the EEPROM driver when a job fails.
Synopsis	<code>void Ea_JobErrorNotification (void);</code>
Service ID	17
Sync/Async	synchronous
Reentrancy	non reentrant
Description	This callback function is called by the underlying EEPROM Driver to report the failure of an asynchronous operation. When a job given to the EEPROM Abstraction module is finished, a callback function will be called in the upper layer.

5.3.3.2.10. Ea_MainFunction

Purpose	Handles the requested jobs asynchronously.
Synopsis	<code>void Ea_MainFunction (void);</code>
Service ID	18
Description	This function needs to be called periodically by the BSW scheduler.

5.3.3.2.11. Ea_Read

Purpose	Reads data from EEPROM.
----------------	-------------------------

Synopsis	Std_ReturnType Ea_Read (uint16 BlockNumber , uint16 BlockOffset , uint8 * DataBufferPtr , uint16 Length);	
Service ID	2	
Sync/Async	asynchronous	
Reentrancy	non reentrant	
Parameters (in)	BlockNumber	Identifier of the block (from the configuration)
	BlockOffset	Read address offset inside the block
	Length	Number of bytes to read
Parameters (out)	DataBufferPtr	Pointer to data buffer
Return Value	Standard Return Code E_OK Job accepted E_NOT_OK BlockNumber not found	
Description	Read Length bytes of data in the given block starting at BlockOffset and write them in DataBufferPtr.	

5.3.3.2.12. Ea_SetMode

Purpose	Sets EEPROM driver to FAST or SLOW mode.	
Synopsis	void Ea_SetMode (MemIf_ModeType Mode);	
Service ID	1	
Sync/Async	synchronous	
Reentrancy	non reentrant	
Parameters (in)	Mode	Desired mode for the underlying EEPROM driver
Description	Calls the EEPROM driver service Eep_SetMode to switch the EEPROM driver to fast or slow mode	

5.3.3.2.13. Ea_Write

Purpose	Writes data to EEPROM.	
Synopsis	Std_ReturnType Ea_Write (uint16 BlockNumber , const uint8 * DataBufferPtr);	

Service ID	3	
Sync/Async	asynchronous	
Reentrancy	non reentrant	
Parameters (in)	BlockNumber	Identifier of the block (from the configuration).
	DataBufferPtr	Pointer to data buffer
Return Value	Standard Return Code	
	E_OK	Job accepted.
	E_NOT_OK	BlockNumber not found.
Description	Write the data from DataBufferPtr to the EEPROM in the block identified by BlockNumber. The length is always the configured length for the given block.	

5.3.4. Integration notes

5.3.4.1. Exclusive areas

Exclusive areas are not used by the `Ea` module.

5.3.4.2. Production errors

Production errors are not reported by the `Ea` module.

5.3.4.3. Memory mapping

General information about memory mapping is provided in the EB tresos AutoCore Generic documentation. Refer to the section `Memory mapping and compiler abstraction` in the `Integration notes` section for details.

The following table provides the list of sections that may be mapped for this module:

Memory section
CONST_16
CONST_UNSPECIFIED

CODE
VAR_INIT_8
VAR_INIT_UNSPECIFIED
VAR_INIT_16

5.3.4.4. Integration requirements

WARNING



Integration requirements list is not exhaustive

The following list of integration requirements helps you to integrate your product. However, this list is not exhaustive. You also require information from the user's guide, release notes, and EB tresos AutoCore known issues to successfully integrate your product.

5.3.4.4.1. dev.Ea.IntegrationRestrictions

Description	Integration restriction and recommendation. The EB memory stack modules NvM, Ea and Fee make only limited use of the callback calls from their underlying modules. During the integration make sure that the NvM, Ea, and Fee main functions are only called from the same task context so that they cannot preempt each other. The configuration parameter EaPollingMode is only used for controlling the availability of callback functions. The module operates as if polling mode is always selected. The callback functions Ea_JobEndNotification() and Ea_JobErrorNotification() can be called but will have no effect.
Rationale	This approach enables a simple and lock-free implementation resulting in smaller code.

5.4. Fee

5.4.1. Configuration parameters

Containers included		
Container name	Multiplicity	Description



Containers included		
CommonPublishedInformation	1..1	Label: Common Published Information Common container, aggregated by all modules. It contains published information about vendor and versions.
FeeBlockConfiguration	1..n	Configuration of block specific parameters for the module.
FeeDefensiveProgramming	1..1	Label: Defensive Programming Options Parameters for defensive programming
FeeGeneral	1..1	Label: General Configuration Parameters Container for general parameters. These parameters are not specific to a block.
FeeDemEventParameterRefs	1..1	Label: Dem Event Parameter References Container for the references to DemEventParameter elements which shall be invoked using the API <code>Dem_ReportErrorStatus</code> API in case the corresponding error occurs. The <code>EventId</code> is taken from the referenced <code>DemEventParameter's DemEventId</code> value. The standardized errors are provided in the container and can be extended by vendor specific error references.
FeePublishedInformation	1..1	Label: Fee Published Information Additional published parameters not covered by Common-PublishedInformation container. Note that these parameters do not have any configuration class setting, since they are published information.
PublishedInformation	1..1	Label: EB Published Information Additional published parameters not covered by Common-PublishedInformation container.

Parameters included	
Parameter name	Multiplicity
IMPLEMENTATION_CONFIG_VARIANT	1..1

Parameter Name	IMPLEMENTATION_CONFIG_VARIANT
Label	Config Variant
Multiplicity	1..1
Type	ENUMERATION
Default value	VariantPreCompile

Range	VariantPreCompile
--------------	-------------------

5.4.1.1. CommonPublishedInformation

Parameters included	
Parameter name	Multiplicity
ArMajorVersion	1..1
ArMinorVersion	1..1
ArPatchVersion	1..1
SwMajorVersion	1..1
SwMinorVersion	1..1
SwPatchVersion	1..1
ModuleId	1..1
VendorId	1..1
Release	1..1

Parameter Name	ArMajorVersion	
Label	AUTOSAR Major Version	
Description	Major version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	2	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	ArMinorVersion	
Label	AUTOSAR Minor Version	
Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	0	

Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	ArPatchVersion	
Label	AUTOSAR Patch Version	
Description	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	0	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	SwMajorVersion	
Label	Software Major Version	
Description	Major version number of the vendor specific implementation of the module.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	6	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	SwMinorVersion	
Label	Software Minor Version	
Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	14	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	SwPatchVersion	
Label	Software Patch Version	

Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	13	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	ModuleId	
Label	Numeric Module ID	
Description	Module ID of this module from Module List	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	21	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	VendorId	
Label	Vendor ID	
Description	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	1	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	Release	
Label	Release Information	
Multiplicity	1..1	
Type	STRING_LABEL	
Default value		
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

5.4.1.2. FeeBlockConfiguration

Parameters included	
Parameter name	Multiplicity
FeeBlockNumber	1..1
FeeBlockSize	1..1
FeeImmediateData	1..1
FeeNumberOfWriteCycles	1..1
FeeDeviceIndex	1..1

Parameter Name	FeeBlockNumber	
Label	Block Number	
Description	<p>Defines the Fee block identifier (handle).</p> <p>0x0000 and 0xFFFF shall not be used for block numbers.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ min = $2^{\text{NVM_DATA_SELECTION_BITS}}$ ▶ max = $0xFFFF - (2^{\text{NVM_DATA_SELECTION_BITS}} - 1)$ <p>Note: Depending on the number of bits set aside for dataset selection several other block numbers shall also be left out to ease implementation.</p>	
Multiplicity	1..1	
Type	INTEGER	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeBlockSize	
Label	Block Size	
Description	<p>Defines the size of the logical block in bytes.</p> <p>Note: If CRC is configured for the block in the NVRAM manager, the size configured here must be large enough for the number of bytes of block data and the number of bytes needed for the CRC type configured.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ At least two flash sectors <code>FlsSector</code> must be available, in order for the check regarding the total size available in FLS to be enabled. 	

	<p>► The total flash space space required shall be less than the total flash size available, which is implemented as the sum of (Flash number of sectors * Flash sector size) for all configured flash sectors.</p>	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<=65535	
	>=1	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeelImmediateData	
Label	Immediate Data	
Description	<p>Defines if the block is an immediate block (usage of high priority data).</p> <p>► <code>true</code>: Block contains immediate data.</p> <p>► <code>false</code>: Block does not contain immediate data.</p> <p>Note: This must be set consistently with the block priority configured in the NVRAM manager. A block with priority value 0 shall be marked as containing immediate data here.</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeNumberOfWriteCycles	
Label	Number of Write Cycles	
Description	<p>Defines the number of write cycles required for this block.</p> <p>Note: This value must be set to at least 1.</p>	
Multiplicity	1..1	
Type	INTEGER	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeDeviceIndex	
Label	Flash Device	
Description	<p>Defines the flash (FIs) device where the NVRAM block is located.</p> <p>Note: This information is referenced by the NVRAM manager and the Memory Abstraction Interface to address this logical block.</p>	
Multiplicity	1..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.4.1.3. FeeDefensiveProgramming

Parameters included	
Parameter name	Multiplicity
FeeDefProgEnabled	1..1
FeePrecondAssertEnabled	1..1
FeePostcondAssertEnabled	1..1
FeeStaticAssertEnabled	1..1
FeeUnreachAssertEnabled	1..1
FeeInvariantAssertEnabled	1..1

Parameter Name	FeeDefProgEnabled
Label	Enable Defensive Programming
Description	<p>Enables or disables the defensive programming feature for the module Fee.</p> <p>Note: This feature is dependent on the use of the development error detection module. To use the defensive programming feature, proceed as follows:</p> <ol style="list-style-type: none"> 1. Enable development error detection 2. Enable defensive programming 3. Enable assertions as required
Multiplicity	1..1
Type	BOOLEAN
Default value	false

Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	FeePrecondAssertEnabled
Label	Enable Precondition Assertions
Description	<p>Enables handling of precondition assertion checks reported from the module Fee.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (FeeDevErrorDetect): must be enabled ▶ Enable Defensive Programming (FeeDefProgEnabled): must be enabled
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	FeePostcondAssertEnabled
Label	Enable Postcondition Assertions
Description	<p>Enables handling of postcondition assertion checks reported from the module Fee.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (FeeDevErrorDetect): must be enabled ▶ Enable Defensive Programming (FeeDefProgEnabled): must be enabled
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	FeeStaticAssertEnabled
Label	Enable Static Assertions
Description	Enables handling of static assertion checks reported from the module Fee.

	<p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (<code>FeeDevErrorDetect</code>): must be enabled ▶ Enable Defensive Programming (<code>FeeDefProgEnabled</code>): must be enabled
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	FeeUnreachAssertEnabled
Label	Enable Unreachable Code Assertions
Description	<p>Enables handling of unreachable code assertion checks reported from the module Fee.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (<code>FeeDevErrorDetect</code>): must be enabled ▶ Enable Defensive Programming (<code>FeeDefProgEnabled</code>): must be enabled
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	FeeInvariantAssertEnabled
Label	Enable Invariant Assertions
Description	<p>Enables handling of invariant assertion checks reported from functions of the module Fee.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (<code>FeeDevErrorDetect</code>): must be enabled ▶ Enable Defensive Programming (<code>FeeDefProgEnabled</code>): must be enabled
Multiplicity	1..1

Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.4.1.4. FeeGeneral

Containers included		
Container name	Multiplicity	Description
FeeSectionConfiguration	0..n	Configuration of FEE sections layout.

Parameters included	
Parameter name	Multiplicity
FeeBufferSize	0..1
FeeUseBufferForJobs	0..1
FeeInitializeInLoop	1..1
FeeDevErrorDetect	1..1
FeeIndex	1..1
FeeMainFunctionPeriod	1..1
FeeNvmJobEndNotification	0..1
FeeNvmJobErrorNotification	0..1
FeePollingMode	1..1
FeeSetModeSupported	1..1
FeeVersionInfoApi	1..1
FeeVirtualPageSize	1..1
FeeNumberOfSections	0..1
EnableAutoSectionGeneration	0..1
FeeProdErrorDetect	1..1
FeeEraseCounterApi	1..1
FeeFreezeActivitiesApi	1..1
FeeCancelSectionEraseApi	0..1
FeeConsistencyPattern	1..1
FeeConsistencyStartPattern	1..1

Parameters included	
FeeEnableAbortErase	1..1
FeeNumberOfNotConfigBlocks	0..1
FeeDataSizeNotConfiguredBlocks	1..1
FeeWriteCustomApi	1..1
FeeReadCustomApi	1..1
FeeCriticalBlock	0..1
FeeDynamicBlockLength	0..1
FeeEnableSmallSectionSize	1..1

Parameter Name	FeeBufferSize
Label	Internal Buffer Size
Description	<p>Defines the size (in bytes) of buffer in RAM, used internally by Fee module. The buffer is used to read the block management information during startup. The same buffer is also used to read and write memory block data from and to flash.</p> <p>By default a minimum buffer size is chosen so that it can just hold the largest configured memory block (including management data). For configurations with only many small memory blocks this leads to a small buffer size. In this case a manually increased buffer enhances the runtime performance during startup, as more block management information can be read from flash in each flash read call.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ The buffer size must be a multiple of virtual page size. ▶ The buffer size must be greater or equal to the largest block plus an overhead of 2 bytes. ▶ The buffer size must be at least 10 bytes or 5 times FeeVirtualPageSize, whichever is greater. <p>Note: This parameter is an EB specific extension of the Fee configuration enabling better runtime optimization.</p>
Multiplicity	0..1
Type	INTEGER
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	FeeUseBufferForJobs
----------------	---------------------

Label	Fls needs aligned addresses	
Description	<p>Assure compatibility with FLS drivers aligned at 4 bytes exclusively</p> <p>Setting this options shall cause the following:</p> <ul style="list-style-type: none"> ▶ The buffer size will be a multiple of virtual page size and 4. ▶ The buffer size must be greater or equal to the largest block. ▶ Any block to be written from an address that is not aligned to 4 bytes will first be copied to the internal buffer and then sent to FLS ▶ Any block to be read to an address that is not aligned to 4 bytes will first be copied to the internal buffer and then to NvM <p>Note: This parameter is an EB specific extension of the Fee configuration enabling compatibility addressing with specific Fls implementations.</p>	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeInitializeInLoop	
Label	Loop Initialization	
Description	Initialization will be done in a loop with the OS disabled otherwise the initialization performance will be reduced	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeDevErrorDetect	
Label	Enable Development Error Detection	
Description	<p>Enables the use of development error detection.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Development error detection enabled. ▶ <code>false</code> = Development error detection disabled. 	
Multiplicity	1..1	

Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeIndex	
Label	Module Instance Index	
Description	<p>Defines the instance number of this module. If only one instance is present it shall have the index number 0.</p> <p><i>The current implementation only supports one instance of the Fee module.</i></p>	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<div><=254</div> <div>>=0</div>	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeMainFunctionPeriod	
Description	Defines the periodic cycle time, in seconds, for calling Fee main function.	
Multiplicity	1..1	
Type	FLOAT	
Default value	0.01	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeNvmJobEndNotification	
Label	NvM Job End Notification Function	
Description	<p>Defines the job end notification routine provided by the upper layer module (NvM_JobEndNotification).</p> <p>Note: The Fee will call this function on successful completion of a job request.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	

Default value	NvM_JobEndNotification	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeNvmJobErrorNotification	
Label	NvM Job Error Notification Function	
Description	<p>Defines the job error notification routine provided by the upper layer module (NvM_JobErrorNotification).</p> <p>Note: The Fee will call this function if an error occurs during the processing of a job request.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Default value	NvM_JobErrorNotification	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeePollingMode	
Label	Enable Polling Mode	
Description	<p>Enables the polling mode for this module. If polling mode is enabled, the Fee will poll the underlying FIs module for job results. If polling mode is disabled, the Fee shall provide functions that can be called by the underlying FIs module when jobs have completed.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Polling mode enabled. ▶ <code>false</code> = Polling mode disabled. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeSetModeSupported	
Label	Enable SetMode Support	
Description	<p>Enables the SetMode functionality.</p> <ul style="list-style-type: none"> ▶ <code>true</code>: SetMode functionality supported / code present. 	

	<p>► <code>false</code>: SetMode functionality not supported / code not present.</p> <p>Note: This configuration setting has to be consistent with that of all underlying flash device drivers (configuration parameter <code>FlsSetModeApi</code>).</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeVersionInfoApi	
Label	Enable Version Info API	
Description	<p>Enables the <code>Fee_GetVersionInfo()</code> API function.</p> <p>► <code>true</code> = <code>Fee_GetVersionInfo()</code> enabled.</p> <p>► <code>false</code> = <code>Fee_GetVersionInfo()</code> disabled.</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeVirtualPageSize	
Label	Virtual Page Size	
Description	<p>Defines the size in bytes to which logical blocks shall be aligned.</p> <p>Note: The value shall be greater than 0.</p>	
Multiplicity	1..1	
Type	INTEGER	
Default value	4	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeNumberOfSections	
Label	Number of sections	
Description	Defines the number of sections to be used by Fee for storing data.	

	Note: The value shall be greater than 1.	
Multiplicity	0..1	
Type	INTEGER	
Default value	2	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	EnableAutoSectionGeneration	
Label	Enable Automatic Section Generation	
Description	<p>Enable the automatic generation of FEE section configuration</p> <p>Note: This can only be enabled for 3 or more sections.</p> <p>Note: If the automatic generation fail because of the FLS sector configuration then the user shall have to manually configure the section generation.</p>	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeProdErrorDetect	
Label	Enable Production Error Detection	
Description	<p>Enables the use of production error detection.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Production error detection enabled. ▶ <code>false</code> = Production error detection disabled. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeEraseCounterApi	
Label	Enable Erase Counter API	
Description	Enables the <code>Fee_GetEraseCounterValue()</code> API function.	

	<ul style="list-style-type: none"> ▶ <code>true = Fee_GetEraseCounterValue()</code> enabled. ▶ <code>false = Fee_GetEraseCounterValue()</code> disabled. <p>Dependencies:</p> <ul style="list-style-type: none"> ▶ The initial value of the flash memory erase counter (after flashing the ECU) depends on the flash memory erased value (<code>FlsErasedValue</code>). <p>Note: Valid values for the flash erase counter are in the range of 0x0000 to 0xFFFE. Value 0xFFFF is used to signal an invalid value of the counter.</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeFreezeActivitiesApi	
Label	Freeze All Activities API	
Description	<p>Enables the use of the Freeze Activities feature</p> <ul style="list-style-type: none"> ▶ <code>true = Freeze Activities API</code> can be used ▶ <code>false = Freeze Activities API</code> can not be used <p>Note: By using this feature the upper layer can stop all the Fee activities, therefore Fee will not pass any command to Fls until unfreeze.</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeCancelSectionEraseApi	
Label	Enable Cancel Section Erase Api	
Description	<p>Enables the <code>Fee_CancelSectionErase()</code> API function.</p> <ul style="list-style-type: none"> ▶ <code>true = Fee_CancelSectionErase()</code> enabled. ▶ <code>false = Fee_CancelSectionErase()</code> disabled. <p>Note: This feature may affect the flash memory life span if used excessively.</p>	

Multiplicity	0..1
Type	BOOLEAN
Default value	false
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	FeeConsistencyPattern
Label	Enable pattern for data consistency
Description	<p>Enables the use of a pattern for data consistency</p> <ul style="list-style-type: none"> ▶ <code>true</code> = pattern for data consistency enabled. ▶ <code>false</code> = pattern for data consistency disabled. <p>Note: This feature affects the memory layout.</p>
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	FeeConsistencyStartPattern
Label	Enable start pattern for data consistency
Description	<p>Enables the use of a pattern for data consistency</p> <ul style="list-style-type: none"> ▶ <code>true</code> = start pattern for data consistency enabled. ▶ <code>false</code> = start pattern for data consistency disabled. <p>Note: This feature affects the memory layout.</p>
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	FeeEnableAbortErase
Label	Enable abort erase.
Description	Enables the use of abort erase functionality.

	<p>Any write job for an immediate block will cause the abort of an erase operation, if in progress.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = abort erase enabled. ▶ <code>false</code> = abort erase disabled. <p>Dependencies:</p> <ul style="list-style-type: none"> ▶ This feature can only be enabled if <code>FIs_Cancel</code> API is available. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeNumberOfNotConfigBlocks	
Label	Number of not configured blocks to switch	
Description	<p>Defines maximum number of not configured blocks that Fee can identify and memorize at init and carry them during runtime through switch mechanism.</p> <p>Note: If this parameter is set to 0 only configured blocks are considered during the section switch.</p>	
Multiplicity	0..1	
Type	INTEGER	
Default value	0	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeDataSizeNotConfiguredBlocks	
Label	Total data size required for not configured blocks	
Description	<p>Defines the total data size to be considered by Fee for not configured blocks</p> <p>Note: When configuring this parameter, you shall consider the block size from upper layer including possible overhead like CRC or static block Id</p> <p>Additional flash for Fee block management (info and alignment) will be automatically reserved for the maximum number of not configured blocks. For an increased efficiency of allocation, in case less blocks than maximum are used, the size reserved for the management of the not used blocks can be used for data, but no block is allowed to be bigger than this parameter.</p>	

	For details about flash memory allocation, please refer to the user guide.	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeWriteCustomApi	
Label	Enable FeeWriteCustom API	
Description	<p>Enables the <code>Fee_WriteCustom()</code> API function.</p> <ul style="list-style-type: none"> ▶ <code>true = Fee_WriteCustom()</code> enabled. ▶ <code>false = Fee_WriteCustom()</code> disabled. <p>Dependencies:</p> <ul style="list-style-type: none"> ▶ <code>FeeNumberOfNotConfigBlocks</code> is different than 0. ▶ aligned addresses setting is off or there is one configured block with size greater or equal to the total size of not configured blocks 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeReadCustomApi	
Label	Enable FeeReadCustom API	
Description	<p>Enables the <code>Fee_ReadCustom()</code> API function.</p> <ul style="list-style-type: none"> ▶ <code>true = Fee_ReadCustom()</code> enabled. ▶ <code>false = Fee_ReadCustom()</code> disabled. <p>Dependencies:</p> <ul style="list-style-type: none"> ▶ <code>FeeNumberOfNotConfigBlocks</code> is different than 0. ▶ aligned addresses setting is off or there is one configured block with size greater or equal to the total size of not configured blocks 	
Multiplicity	1..1	

Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeCriticalBlock	
Label	Reference to Fee Emergency Block	
Description	Reference to the FEE block that shall contain critical data.	
Multiplicity	0..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeDynamicBlockLength	
Label	Enable DynamicBlockLength	
Description	<p>Enables the functionality of reading the blocks that have the length reconfigured.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Fee will read the block with the older length. ▶ <code>false</code> = Fee will not read the block with the older length. 	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FeeEnableSmallSectionSize	
Label	Enable Small Size Sections	
Description	<p>Enables configuring Sections with size less than the total size of the Fee blocks configured in case 4 or more sections are configured.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Fee will allow configuring sections with size less than the total size of the Blocks. ▶ <code>false</code> = Fee will not allow configuring sections with size less than the total size of the Blocks. 	
Multiplicity	1..1	
Type	BOOLEAN	

Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.4.1.5. FeeSectionConfiguration

Parameters included	
Parameter name	Multiplicity
FeeSectionNumber	1..1
FeeSectionStartAddress	1..1
FeeSectionSize	1..1

Parameter Name	FeeSectionNumber
Label	Section Number
Description	Defines the Fee Section identifier (handle). Range: <ul style="list-style-type: none"> ▶ min = 1 ▶ max = FeeNumberOfSections
Multiplicity	1..1
Type	INTEGER
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	FeeSectionStartAddress
Label	Section Start Address
Description	Defines the start address of the FEE section.
Multiplicity	1..1
Type	INTEGER
Default value	0
Range	<div><=4294967295</div> <div>>=0</div>
Configuration class	VariantPreCompile: VariantPreCompile

Origin	AUTOSAR_ECUC	
--------	--------------	--

Parameter Name	FeeSectionSize	
Label	Section Size	
Description	Defines the size in bytes of the FEE section.	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<=4294967295	
	>=0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.4.1.6. FeeDemEventParameterRefs

Parameters included	
Parameter name	Multiplicity
FEE_E_FLASH_ACCESSIBLE	0..1
FEE_E_DATA_RECOVERED	0..1

Parameter Name	FEE_E_FLASH_ACCESSIBLE
Label	Flash Accessible Event Parameter
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error <code>FEE_E_FLASH_ACCESSIBLE</code> has occurred.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Reporting of production errors shall be enabled by configuration (<code>FeeProdErrorDetect</code> shall be enabled). <p>Further notes:</p> <ul style="list-style-type: none"> ▶ Activation: Thrown, during initialization or during section switch if the section status cannot be read or written or a section cannot be erased. ▶ Healing: None. The error resides in memory until it is deleted. ▶ Trigger debounce: None. The error is reported on first occurrence.

	▶ Rate of diagnostic checks: Checked on each module initialization or during each section switch that takes place.	
Multiplicity	0..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	FEE_E_DATA_RECOVERED	
Label	Flash User Data Recovered	
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error <code>FEE_E_FLASH_ACCESSIBLE</code> has occurred.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Reporting of production errors shall be enabled by configuration (<code>FeeP-rodErrorDetect</code> shall be enabled). <p>Further notes:</p> <ul style="list-style-type: none"> ▶ Activation: Thrown, during startup if the user data from active section cannot be used or active section cannot be determined because of an unexpected section status combination and user data is recovered from the inactive section. ▶ Healing: None. The error resides in memory until it is deleted. ▶ Trigger debounce: None. The error is reported on first occurrence. ▶ Rate of diagnostic checks: Checked during each startup of the module. 	
Multiplicity	0..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.4.1.7. FeePublishedInformation

Parameters included	
Parameter name	Multiplicity
FeeBlockOverhead	1..1
FeeMaximumBlockingTime	1..1

Parameters included	
FeePageOverhead	1..1

Parameter Name	FeeBlockOverhead	
Label	Block Management Overhead	
Description	<p>Defines management overhead per logical block in bytes.</p> <p>Note: The formula to calculate Block overhead is, the bytes required for aligning the configured size of block along with 2 bytes (for management information) to virtual page size + 6 bytes(for management information) aligned to virtual page size.</p> <p><i>This parameter is not used by the implementation.</i></p>	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	0	
Range	<=65535	
	>=0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeeMaximumBlockingTime	
Label	Maximum Blocking Time	
Description	<p>Defines the maximum time the FEE module's API routines shall be blocked (delayed) by internal operations.</p> <p>Note: Internal operations in this case means operations that are not explicitly invoked from the upper layer module but need to be handled for proper operation of this module or the underlying memory driver.</p> <p>unit: milliseconds. Configuration generators may need to convert this value into a hardware specific representation (e.g. timer ticks).</p> <p><i>This parameter is not used by the implementation.</i></p>	
Multiplicity	1..1	
Type	FLOAT_LABEL	
Default value	0.0	
Range	<=Infinity	

	>=0.0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	FeePageOverhead	
Label	Page Management Overhead	
Description	<p>Defines the management overhead per page in bytes.</p> <p>Note: There is no overhead per virtual page but an overhead per block and per section.</p> <p><i>This parameter is not used internally by the current implementation.</i></p>	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	0	
Range	<=65535	
	>=0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.4.1.8. PublishedInformation

Parameters included		
Parameter name	Multiplicity	
PbcfgMSupport	1..1	

Parameter Name	PbcfgMSupport	
Label	PbcfgM support	
Description	Specifies whether or not the Fee can use the PbcfgM module for post-build support.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

5.4.2. Recommended configurations

5.4.2.1. FeeRecConfiguration

Containers included	
Container name	Container definition
FeeBlockConfiguration_0	FeeBlockConfiguration
FeeBlockConfiguration_1	FeeBlockConfiguration

Parameters included	
Parameter name	Value

5.4.2.1.1. FeeBlockConfiguration_0

Parameters included	
Parameter name	Value
FeeBlockNumber	16
FeeImmediateData	true
FeeBlockSize	4

5.4.2.1.2. FeeBlockConfiguration_1

Parameters included	
Parameter name	Value
FeeBlockNumber	17
FeeImmediateData	true
FeeBlockSize	4

5.4.3. Application programming interface (API)

5.4.3.1. Macro constants

5.4.3.1.1. DBG_FEE_CANCELSECTIONERASE_ENTRY

Purpose	Entry point of function Fee_CancelSectionErase() .
Value	

5.4.3.1.2. DBG_FEE_CANCELSECTIONERASE_EXIT

Purpose	Exit point of function Fee_CancelSectionErase() .
Value	

5.4.3.1.3. DBG_FEE_CANCEL_ENTRY

Purpose	Entry point of function Fee_Cancel() .
Value	

5.4.3.1.4. DBG_FEE_CANCEL_EXIT

Purpose	Exit point of function Fee_Cancel() .
Value	

5.4.3.1.5. DBG_FEE_ERASEIMMEDIATEBLOCK_ENTRY

Purpose	Entry point of function Fee_EraseImmediateBlock() .
Value	

5.4.3.1.6. DBG_FEE_ERASEIMMEDIATEBLOCK_EXIT

Purpose	Exit point of function Fee_EraseImmediateBlock() .
Value	

5.4.3.1.7. DBG_FEE_FREEZEACTIVITIES_ENTRY

Purpose	Entry point of function Fee_FreezeActivities() .
Value	

5.4.3.1.8. DBG_FEE_FREEZEACTIVITIES_EXIT

Purpose	Exit point of function Fee_FreezeActivities() .
Value	

5.4.3.1.9. DBG_FEE_GETERASECOUNTERVALUE_ENTRY

Purpose	Entry point of function Fee_GetEraseCounterValue() .
Value	

5.4.3.1.10. DBG_FEE_GETERASECOUNTERVALUE_EXIT

Purpose	Exit point of function Fee_GetEraseCounterValue() .
Value	

5.4.3.1.11. DBG_FEE_GETJOBRESULT_ENTRY

Purpose	Entry point of function Fee_GetJobResult() .
Value	

5.4.3.1.12. DBG_FEE_GETJOBRESULT_EXIT

Purpose	Exit point of function Fee_GetJobResult() .
Value	

5.4.3.1.13. DBG_FEE_GETSTATUS_ENTRY

Purpose	Entry point of function Fee_GetStatus() .
----------------	---

Value	
--------------	--

5.4.3.1.14. DBG_FEE_GETSTATUS_EXIT

Purpose	Exit point of function Fee_GetStatus() .
Value	

5.4.3.1.15. DBG_FEE_GETVERSIONINFO_ENTRY

Purpose	Entry point of function Fee_GetVersionInfo() .
Value	

5.4.3.1.16. DBG_FEE_GETVERSIONINFO_EXIT

Purpose	Exit point of function Fee_GetVersionInfo() .
Value	

5.4.3.1.17. DBG_FEE_INIT_ENTRY

Purpose	Entry point of function Fee_Init() .
Value	

5.4.3.1.18. DBG_FEE_INIT_EXIT

Purpose	Exit point of function Fee_Init() .
Value	

5.4.3.1.19. DBG_FEE_INVALIDATEBLOCK_ENTRY

Purpose	Entry point of function Fee_InvalidateBlock() .
----------------	---

Value	
--------------	--

5.4.3.1.20. DBG_FEE_INVALIDATEBLOCK_EXIT

Purpose	Exit point of function Fee_InvalidateBlock() .
Value	

5.4.3.1.21. DBG_FEE_JOBENDNOTIFICATION_ENTRY

Purpose	Entry point of function Fee_JobEndNotification() .
Value	

5.4.3.1.22. DBG_FEE_JOBENDNOTIFICATION_EXIT

Purpose	Exit point of function Fee_JobEndNotification() .
Value	

5.4.3.1.23. DBG_FEE_JOBERRORNOTIFICATION_ENTRY

Purpose	Entry point of function Fee_JobErrorNotification() .
Value	

5.4.3.1.24. DBG_FEE_JOBERRORNOTIFICATION_EXIT

Purpose	Exit point of function Fee_JobErrorNotification() .
Value	

5.4.3.1.25. DBG_FEE_MAINFUNCTION_ENTRY

Purpose	Entry point of function Fee_MainFunction() .
Value	

5.4.3.1.26. DBG_FEE_MAINFUNCTION_EXIT

Purpose	Exit point of function Fee_MainFunction() .
Value	

5.4.3.1.27. DBG_FEE_NEXTSTATE

Purpose	Change of Fee_NextState.
Value	

5.4.3.1.28. DBG_FEE_READ_CUSTOM_ENTRY

Purpose	Entry point of function Fee_ReadCustom() .
Value	

5.4.3.1.29. DBG_FEE_READ_CUSTOM_EXIT

Purpose	Exit point of function Fee_ReadCustom() .
Value	

5.4.3.1.30. DBG_FEE_READ_ENTRY

Purpose	Entry point of function Fee_Read() .
Value	

5.4.3.1.31. DBG_FEE_READ_EXIT

Purpose	Exit point of function Fee_Read() .
Value	

5.4.3.1.32. DBG_FEE_SETMODE_ENTRY

Purpose	Entry point of function Fee_SetMode() .
----------------	---



Value	
--------------	--

5.4.3.1.33. DBG_FEE_SETMODE_EXIT

Purpose	Exit point of function Fee_SetMode() .
Value	

5.4.3.1.34. DBG_FEE_STATE

Purpose	Change of Fee_State.
Value	

5.4.3.1.35. DBG_FEE_WRITE_CUSTOM_ENTRY

Purpose	Entry point of function Fee_WriteCustom() .
Value	

5.4.3.1.36. DBG_FEE_WRITE_CUSTOM_EXIT

Purpose	Exit point of function Fee_WriteCustom() .
Value	

5.4.3.1.37. DBG_FEE_WRITE_ENTRY

Purpose	Entry point of function Fee_Write() .
Value	

5.4.3.1.38. DBG_FEE_WRITE_EXIT

Purpose	Exit point of function Fee_Write() .
Value	

5.4.3.1.39. FEE_AR_RELEASE_MAJOR_VERSION

Purpose	AUTOSAR release major version.
Value	4U

5.4.3.1.40. FEE_AR_RELEASE_MINOR_VERSION

Purpose	AUTOSAR release minor version.
Value	0U

5.4.3.1.41. FEE_AR_RELEASE_REVISION_VERSION

Purpose	AUTOSAR release revision version.
Value	3U

5.4.3.1.42. FEE_CANCELSECTIONERASE_API_ID

Purpose	Service Id of Fee_CancelSectionErase.
Value	0x15U

5.4.3.1.43. FEE_CANCEL_API_ID

Purpose	Service Id of Fee_Cancel.
Value	0x04U

5.4.3.1.44. FEE_ERASEIMMEDIATEBLOCK_API_ID

Purpose	Service Id of Fee_EraseImmediateBlock.
Value	0x09U

5.4.3.1.45. FEE_FREEZEACTIVITIES_API_ID

Purpose	Service Id of Fee_FreezeActivities.
----------------	-------------------------------------

Value	0x13U
--------------	-------

5.4.3.1.46. FEE_GETJOBRESULT_API_ID

Purpose	Service Id of Fee_GetJobResult.
Value	0x06U

5.4.3.1.47. FEE_GETSTATUS_API_ID

Purpose	Service Id of Fee_GetStatus.
Value	0x05U

5.4.3.1.48. FEE_GET_ERASE_COUNTER_API_ID

Purpose	Service Id of Fee_GetEraseCounterValue.
Value	0x17U

5.4.3.1.49. FEE_GET_VERSION_INFO_API_ID

Purpose	Service Id of Fee_GetVersionInfo.
Value	0x08U

5.4.3.1.50. FEE_INIT_API_ID

Purpose	Service Id of Fee_Init.
Value	0x00U

5.4.3.1.51. FEE_INVALIDATEBLOCK_API_ID

Purpose	Service Id of Fee_InvalidateBlock.
Value	0x07U

5.4.3.1.52. FEE_JOBENDNOTIFICATION_API_ID

Purpose	Service Id of Fee_JobEndNotification.
Value	0x10U

5.4.3.1.53. FEE_JOBERRORNOTIFICATION_API_ID

Purpose	Service Id of Fee_JobErrorNotification.
Value	0x11U

5.4.3.1.54. FEE_MAINFUNCTION_API_ID

Purpose	Service Id of Fee_MainFunction.
Value	0x12U

5.4.3.1.55. FEE_MODULE_ID

Purpose	AUTOSAR module identification.
Value	21U

5.4.3.1.56. FEE_READ_API_ID

Purpose	Service Id of Fee_Read.
Value	0x02U

5.4.3.1.57. FEE_READ_CUSTOM_API_ID

Purpose	Service Id of Fee_ReadCustom.
Value	0x16U

5.4.3.1.58. FEE_SETMODE_API_ID

Purpose	Service Id of Fee_SetMode.
----------------	----------------------------

Value	0x01U
--------------	-------

5.4.3.1.59. FEE_SW_MAJOR_VERSION

Purpose	AUTOSAR module major version.
Value	6U

5.4.3.1.60. FEE_SW_MINOR_VERSION

Purpose	AUTOSAR module minor version.
Value	14U

5.4.3.1.61. FEE_SW_PATCH_VERSION

Purpose	AUTOSAR module patch version.
Value	13U

5.4.3.1.62. FEE_VENDOR_ID

Purpose	AUTOSAR vendor identification: Elektrobit Automotive GmbH.
Value	1U

5.4.3.1.63. FEE_WRITE_API_ID

Purpose	Service Id of Fee_Write.
Value	0x03U

5.4.3.1.64. FEE_WRITE_CUSTOM_API_ID

Purpose	Service Id of Fee_WriteCustom.
Value	0x14U

5.4.3.2. Functions

5.4.3.2.1. Fee_Cancel

Purpose	Service for cancelling an ongoing job.
Synopsis	<code>void Fee_Cancel (void);</code>
Service ID	0x04
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Description	This API cancels a previous job request. It also resets the module internal state machine so that a new job request can be accepted.

5.4.3.2.2. Fee_CancelSectionErase

Purpose	Vendor specific service to cancel a section erase operation.	
Synopsis	<code>Std_ReturnType Fee_CancelSectionErase (void);</code>	
Service ID	0x15	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Return Value	Std_ReturnType	
	E_OK	if an erase was ongoing and was canceled. E_NOT_OK no erase cancellation was done because either no erase is ongoing or the function call interrupted the Fee main function.
Description	This API aborts an ongoing section erase at request synchronously. Semantics : Should not be called while Fee main function is ongoing.	

5.4.3.2.3. Fee_EraseImmediateBlock

Purpose	Service to ensure that immediate data can be written.
Synopsis	<code>Std_ReturnType Fee_EraseImmediateBlock (uint16 BlockNumber);</code>

Service ID	0x09	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	BlockNumber	Number of logical block to be written
Return Value	Std_ReturnType	
	E_OK	Fee accepted the erase immediate job
	E_NOT_OK	Fee rejected the erase immediate job
Description	This function doesn't erase or invalidate the block. FEE makes sure that there is always space available for immediate block write and it is not sure that this API is called just before the write of an immediate block.	

5.4.3.2.4. Fee_FreezeActivities

Purpose	Vendor specific service to "freeze" or "unfreeze" the Fee module in terms of performing any activity.	
Synopsis	Std_ReturnType Fee_FreezeActivities (uint8 Freeze);	
Service ID	0x13	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Freeze	: TRUE - Fee will not perform any activity FALSE - Fee will behave normally
Return Value	Std_ReturnType	
	E_OK	request was accepted
	E_NOT_OK	request was not accepted
Description	This function causes Fee to allow the current operation at FIs level to complete but perform no other until the module is "unfrozen". Usage of this function to freeze the Fee module MUST BE DONE WITH CARE.	

5.4.3.2.5. Fee_GetEraseCounterValue

Purpose	Vendor specific service that returns the value of erase counter.	
Synopsis	uint32 Fee_GetEraseCounterValue (void);	
Service ID	0x17	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Return Value	uint32	
	CounterValue	Number of times Fee sections have been erased
Description	The Fee_GetEraseCounterValue() API returns the number of erase cycles for one section. This function provides means of estimating the wear of the flash memory device. Under unexpected conditions, if data cannot be recovered from flash, the value of the erase counter will also be lost.	

5.4.3.2.6. Fee_GetJobResult

Purpose	Service for getting the result of the last job.	
Synopsis	MemIf_JobResultType Fee_GetJobResult (void);	
Service ID	0x06	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Return Value	MemIf_JobResultType	
	MEMIF_JOB_PENDING	Requested job is in queue or being processed
	MEMIF_JOB_OK	Requested job finished successfully
	MEMIF_JOB_FAILED	Requested job failed with error
	MEMIF_JOB_CANCELED	Requested job got cancelled
	MEMIF_BLOCK_INCONSISTENT	Requested block is inconsistent in flash
	MEMIF_BLOCK_INVALID	Requested block is invalidated in flash or not found in flash
Description	This API returns the result of the last processed job.	

5.4.3.2.7. Fee_GetStatus

Purpose	Service for getting the status of the module.	
Synopsis	MemIf_StatusType Fee_GetStatus (void);	
Service ID	0x05	

Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Return Value	MemIf_StatusType	
	MEMIF_UNINIT	Fee has not been initialized.
	MEMIF_IDLE	Fee is Idle doing no job
	MEMIF_BUSY	Fee is busy processing previous request or request in queue
	MEMIF_BUSY_INTERNAL	Fee internal operation is in progress
Description	This API returns the status of the Fee module.	

5.4.3.2.8. Fee_GetVersionInfo

Purpose	Service for getting the version information.	
Synopsis	<pre>void Fee_GetVersionInfo (Std_VersionInfoType * VersionInfoPtr);</pre>	
Service ID	0x08	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (out)	VersionInfoPtr	Pointer to standard version information structure
Description	This API returns version information of Fee module. Normally, the application will invoke this API.	

5.4.3.2.9. Fee_Init

Purpose	Service for initialization of the Fee module.	
Synopsis	<pre>void Fee_Init (void);</pre>	
Service ID	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Description	This function initializes the FEE module. Normally, the ECU Manager invokes this API.	

5.4.3.2.10. Fee_InvalidateBlock

Purpose	Service for invalidating a logical block.	
Synopsis	<code>Std_ReturnType Fee_InvalidateBlock (uint16 BlockNumber);</code>	
Service ID	0x07	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	BlockNumber	Number of logical block to be invalidated
Return Value	Std_ReturnType	
	E_OK	Fee accepted the invalidate job
	E_NOT_OK	Fee rejected the invalidate job
Description	This API marks the contents of a logical block invalid. This is done by writing the block management info with an invalid block size.	

5.4.3.2.11. Fee_JobEndNotification

Purpose	Callback function to notify the successful completion of a flash job.
Synopsis	<code>void Fee_JobEndNotification (void);</code>
Service ID	0x10
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Description	This is called by the underlying flash driver to report the successful completion of an asynchronous operation.

5.4.3.2.12. Fee_JobErrorNotification

Purpose	Callback function to notify the failure of a flash job.
Synopsis	<code>void Fee_JobErrorNotification (void);</code>
Service ID	0x11
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Description	This is called by the underlying flash driver to report the failure of an asynchronous operation.

5.4.3.2.13. Fee_MainFunction

Purpose	Scheduled service to process all Fee jobs asynchronously.
Synopsis	<code>void Fee_MainFunction (void);</code>
Service ID	12
Sync/Async	Asynchronous
Reentrancy	Non Reentrant
Production Errors	<ul style="list-style-type: none"> ▶ FEE_E_FLASH_ACCESSIBLE: thrown, during initialization or during section switch if the section status cannot be read or written or a section cannot be erased. ▶ FEE_E_DATA_RECOVERED: thrown, during startup if the user data from active section cannot be used or active section cannot be determined because of an unexpected section status combination and user data is recovered from the inactive section. Outdated data could be recovered in this case.
Description	This function asynchronously handles the requested read/write/erase jobs and performs all internal management operations.

5.4.3.2.14. Fee_Read

Purpose	Service for reading the contents of a logical block.	
Synopsis	<code>Std_ReturnType Fee_Read (uint16 BlockNumber , uint16 BlockOffset , uint8 * DataBufferPtr , uint16 Length);</code>	
Service ID	0x02	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	BlockNumber	Number of logical block to be read
	BlockOffset	Read address offset inside the block
	DataBufferPtr	Pointer to data buffer
	Length	Number of bytes to read
Return Value	Std_ReturnType	
	E_OK	Fee accepted the read job

	E_NOT_OK	Fee rejected the read job
Description	This API starts a read operation. The read operation involves reading the contents of a logical block stored in flash into a user buffer. The actual read job is executed asynchronously within Fee_MainFunction.	

5.4.3.2.15. Fee_ReadCustom

Purpose	Vendor specific service for reading a not configured logical block.	
Synopsis	Std_ReturnType Fee_ReadCustom (uint16 BlockNumber , uint16 BlockOffset , uint8 * DataBufferPtr , uint16 Length);	
Service ID	0x16	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	BlockNumber	Number of not configured block
	BlockOffset	Read address offset inside the block
	DataBufferPtr	Pointer to data buffer
	Length	Number of bytes to read
Return Value	Std_ReturnType	
	E_OK	Fee accepted the read job
	E_NOT_OK	Fee rejected the read job
Description	This API starts a read operation for a not configured block. The read operation involves reading the contents of a logical block stored in flash into a user buffer. The actual read job is executed asynchronously within Fee_MainFunction.	

5.4.3.2.16. Fee_SetMode

Purpose	Service for setting the operation mode of the underlying flash driver.	
Synopsis	void Fee_SetMode (MemIf_ModeType Mode);	
Service ID	0x01	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Mode	The desired mode of the flash driver

Description	This function sets the operation mode of the underlying flash driver.
--------------------	---

5.4.3.2.17. Fee_Write

Purpose	Service for writing a logical block.	
Synopsis	Std_ReturnType Fee_Write (uint16 BlockNumber , const uint8 * DataBufferPtr);	
Service ID	3	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	BlockNumber	Number of logical block
Parameters (out)	DataBufferPtr	Pointer to data buffer
Return Value	Std_ReturnType	
	E_OK	Fee accepted the write job
	E_NOT_OK	Fee rejected the write job
Description	This API starts a write operation. This operation writes the contents of the user buffer to a logical block in flash. The actual write job is executed asynchronously within Fee_MainFunction.	

5.4.3.2.18. Fee_WriteCustom

Purpose	Vendor specific service for writing a not configured logical block.	
Synopsis	Std_ReturnType Fee_WriteCustom (uint16 BlockNumber , const uint8 * DataBufferPtr , uint16 BlockSize);	
Service ID	0x14	
Sync/Async	Asynchronous	
Reentrancy	Non Reentrant	
Parameters (in)	BlockNumber	Number of not configured block
	BlockSize	Size of not configured block
Parameters (out)	DataBufferPtr	Pointer to data buffer
Return Value	Std_ReturnType	
	E_OK	Fee accepted the write job

	E_NOT_OK	Fee rejected the write job
Description	This API starts a write operation for a not configured block. This operation writes the contents of the user buffer to a logical block in flash. The actual write job is executed asynchronously within Fee_MainFunction.	

5.4.4. Integration notes

5.4.4.1. Exclusive areas

Exclusive areas are not used by the `Fee` module.

5.4.4.2. Production errors

FEE_E_DATA_RECOVERED	► Fee_MainFunction
FEE_E_FLASH_ACCESSIBLE	► Fee_MainFunction

5.4.4.3. Memory mapping

General information about memory mapping is provided in the EB tresos AutoCore Generic documentation. Refer to the section `Memory mapping and compiler abstraction` in the `Integration notes` section for details.

The following table provides the list of sections that may be mapped for this module:

Memory section
CODE
VAR_CLEARED_UNSPECIFIED
VAR_CLEARED_32
CONFIG_DATA_UNSPECIFIED
CONFIG_DATA_32
VAR_INIT_UNSPECIFIED
CONST_UNSPECIFIED

5.4.4.4. Integration requirements

WARNING



Integration requirements list is not exhaustive

The following list of integration requirements helps you to integrate your product. However, this list is not exhaustive. You also require information from the user's guide, release notes, and EB tresos AutoCore known issues to successfully integrate your product.

5.4.4.4.1. lim.Fee.EB_INTREQ_Fee_0001

Description	Integration restriction and recommendation. The EB memory stack modules NvM, Ea, and Fee make only limited use of the callback calls from their underlying modules. This also means that callbacks from the FIs to the Fee are not synchronously forwarded to the NvM. During the integration one has to make sure that the NvM, Ea, and Fee main functions are only called from the same task context so that they cannot preempt each other.
Rationale	This approach enables a simple and lock-free implementation resulting in smaller code.

5.4.4.4.2. lim.Fee.EB_INTREQ_Fee_0002

Description	Contiguous and ascending Flash sectors. The Flash sectors configured in Fee_ - FIsSectorList of the flash driver configuration should be contiguous if more than one sector is configured and they shall be in the ascending order of their addresses.
Rationale	The logic used for calculating the Fee section size is dependent upon this configuration.

5.4.4.4.3. lim.Fee.EB_INTREQ_Fee_0003

Description	Flash virtual page size re-configuration. If the flash virtual page size is reconfigured, the Fee module cannot retrieve the data blocks which are present in the flash.
Rationale	Internal management information size and aligned block size are dependent on the virtual page size.

5.4.4.4.4. lim.Fee.EB_INTREQ_Fee_0004

Description	Flash erase value limitation. Only the first byte of the FIs published parameter FIsErasedValue is used by the Fee module.
--------------------	--

Rationale	The Fls erased value is not expected to be different from one byte to another.
------------------	--

5.4.4.4.5. lim.Fee.EB_INTREQ_Fee_0005

Description	When the user requests an erase-immediate job for emergency block it shall wait for job completion by verifying the result of the job returned by Fee_GetJobResult API.
Rationale	If other means are used the user shall still call at least once the API Fee_GetJobResult after termination of the erase immediate job. The reason for this is to be compatible with NvM behavior.

5.4.4.4.6. lim.Fee.EB_INTREQ_Fee_0006

Description	Once the emergency block is requested to be written, the user shall stop the NvM_MainFunction's task until the emergency block is erased, when the task can be resumed.
Rationale	User may read the emergency block any time, once the emergency block is written.

5.4.4.4.7. lim.Fee.EB_INTREQ_Fee_0007

Description	If the critical block is active in project, the user shall start the NvM_MainFunction's task at startup after the emergency block is erased.
Rationale	There is the possibility that the critical block to be already written in flash by the time the system starts. In this case Fee will remain frozen internally until the user requests an erase-immediate of this block. Meanwhile the user might read the emergency block any time.

5.4.4.4.8. lim.Fee.EB_INTREQ_Fee_0008

Description	If the user requested an erase-immediate for emergency block and immediately the environment requires the emergency block to be written again, the user shall not wait for the erase job to complete, but it shall request the emergency write job immediately as long as at least one Fee_MainFunction had been called after requesting the erase immediate job.
Rationale	The signal to unfreeze Fee from emergency situation is given by the request of the erase immediate. Therefore after the request Fee is not frozen any more, and in case of another emergency write it should behave consistently. Mind the fact that the code

	Fee_EraseImmediate(EBlock); Fee_Write(EBlock, Buff); will not work since by the time the write job comes Fee would still be frozen internally. It needs on cycle of Fee main function after the erase-immediate request in order to unfreeze.
--	---

5.4.4.4.9. lim.Fee.EB_INTREQ_Fee_0009

Description	The environment shall be aware that if an emergency write is called when NvM has a job already pending in Fee, the NvM job will be reported as failed.
Rationale	NvM is informed about the failure by either the notification callbacks or by NvM's polling mode. In this case the environment must take measures like retrying the jobs.

5.4.4.4.10. lim.Fee.EB_INTREQ_Fee_0010

Description	Any call of Fee_ReadCustom shall be made after Fee initialization is over (when Fee is Idle and not Busy Internal).
Rationale	The environment must be aware that Fee_WriteCustom depends on Fee initialization as the blocks that are not configured can only be detected at initialization.

5.4.4.4.11. lim.Fee.EB_INTREQ_Fee_0011

Description	The system shall be aware that SCHM_FEE_EXCLUSIVE_AREA_0 is used if Fee_WriteCustom or Fee_ReadCustom are configured.
Rationale	The interrupts will be disabled only for checking and setting a preemption flag, that prevents call collisions of the custom APIs with the standard APIs called by MemIf.

5.4.4.4.12. lim.Fee.EB_INTREQ_Fee_0012

Description	If a critical block is active in the project and a custom API (Fee_WriteCustom or Fee_ReadCustom) is used, the user shall consider the possibility of a critical block write request being declined if it preempts a custom API: a retry could be needed, or preemption to be prevented by the system.
Rationale	

5.5. MemIf

5.5.1. Configuration parameters

Containers included		
Container name	Multiplicity	Description
CommonPublishedInformation	1..1	Label: Common Published Information Common container, aggregated by all modules. It contains published information about vendor and versions.
MemIfGeneral	1..1	Label: General Configuration Parameters General configuration parameters for the MemIf module.
PublishedInformation	1..1	Label: EB Published Information Additional published parameters not covered by Common-PublishedInformation container.

Parameters included	
Parameter name	Multiplicity
IMPLEMENTATION_CONFIG_VARIANT	1..1

Parameter Name	IMPLEMENTATION_CONFIG_VARIANT
Label	Config Variant
Multiplicity	1..1
Type	ENUMERATION
Default value	VariantPreCompile
Range	VariantPreCompile

5.5.1.1. CommonPublishedInformation

Parameters included	
Parameter name	Multiplicity
ArMajorVersion	1..1
ArMinorVersion	1..1
ArPatchVersion	1..1
SwMajorVersion	1..1
SwMinorVersion	1..1
SwPatchVersion	1..1

Parameters included	
ModuleId	1..1
VendorId	1..1
Release	1..1

Parameter Name	ArMajorVersion	
Label	AUTOSAR Major Version	
Description	Major version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	1	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	ArMinorVersion	
Label	AUTOSAR Minor Version	
Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	4	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	ArPatchVersion	
Label	AUTOSAR Patch Version	
Description	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	0	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	SwMajorVersion
Label	Software Major Version
Description	Major version number of the vendor specific implementation of the module.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	5
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	SwMinorVersion
Label	Software Minor Version
Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	11
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	SwPatchVersion
Label	Software Patch Version
Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	11
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	ModuleId
Label	Numeric Module ID
Description	Module ID of this module from Module List
Multiplicity	1..1
Type	INTEGER_LABEL

Default value	22	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	VendorId	
Label	Vendor ID	
Description	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list	
Multiplicity	1..1	
Type	INTEGER_LABEL	
Default value	1	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

Parameter Name	Release	
Label	Release Information	
Multiplicity	1..1	
Type	STRING_LABEL	
Default value		
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

5.5.1.2. MemIfGeneral

Parameters included	
Parameter name	Multiplicity
MemIfDevErrorDetect	1..1
MemIfNumberOfDevices	1..1
MemIfVersionInfoApi	1..1

Parameter Name	MemIfDevErrorDetect
Label	Enable Development Error Detection
Description	Enable use of development error detection.

	<ul style="list-style-type: none"> ▶ <code>true</code>: development error detection enabled. ▶ <code>false</code>: development error detection disabled. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	MemIfNumberOfDevices	
Label	Number of Underlying Devices	
Description	<p>This specifies the number of underlying memory abstraction modules.</p> <p>Range: 1 .. 255</p>	
Multiplicity	1..1	
Type	INTEGER	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	MemIfVersionInfoApi	
Label	Enable Version Info API	
Description	<p>Enable the API to read out the module's version information.</p> <ul style="list-style-type: none"> ▶ <code>true</code>: Version Info API enabled. ▶ <code>false</code>: Version Info API disabled. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.5.1.3. PublishedInformation

Parameters included		
Parameter name		Multiplicity

Parameters included	
PbcfgMSupport	1..1

Parameter Name	PbcfgMSupport	
Label	PbcfgM support	
Description	Specifies whether or not the MemIf can use the PbcfgM module for post-build support.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

5.5.2. Application programming interface (API)

5.5.2.1. Type definitions

5.5.2.1.1. MemIf_CancelFctPtrType

Purpose	MemIf_Cancel function pointer type definition.
Type	<code>void(*) (void)</code>

5.5.2.1.2. MemIf_EraseImmedBlockFctPtrType

Purpose	MemIf_EraseImmediateBlock function pointer type definition.
Type	<code>Std_ReturnType(*) (uint16 BlockNumber)</code>

5.5.2.1.3. MemIf_GetJobResultFctPtrType

Purpose	MemIf_GetJobResult function pointer type definition.
---------	--

Type	MemIf_JobResultType (*) (void)
-------------	--

5.5.2.1.4. MemIf_GetStatusFctPtrType

Purpose	MemIf_GetStatus function pointer type definition.
Type	MemIf_StatusType (*) (void)

5.5.2.1.5. MemIf_InvalidateBlockFctPtrType

Purpose	MemIf_InvalidateBlock function pointer type definition.
Type	Std_ReturnType (*) (uint16 BlockNumber)

5.5.2.1.6. MemIf_JobResultType

Purpose	Type definition of the MemIf job result type.
Type	uint8
Description	This type denotes the result of the last job.

5.5.2.1.7. MemIf_ModeType

Purpose	Type definition for the operation mode.
Type	uint8
Description	The operation mode of the underlying abstraction modules and device drivers.

5.5.2.1.8. MemIf_ReadFctPtrType

Purpose	MemIf_Read function pointer type definition.
Type	Std_ReturnType (*) (uint16 BlockNumber, uint16 BlockOffset, uint8 *DataBufferPtr, uint16 Length)

5.5.2.1.9. MemIf_SetModeFctPtrType

Purpose	MemIf_SetMode function pointer type definition.
----------------	---

Type	<code>void(*) (MemIf_ModeType Mode)</code>
-------------	--

5.5.2.1.10. MemIf_StatusType

Purpose	Type definition of the MemIf status type.
Type	<code>uint8</code>
Description	This type denotes the current status of the underlying abstraction module and device driver. It shall be used as the return value of the corresponding driver's "GetStatus" function.

5.5.2.1.11. MemIf_WriteFctPtrType

Purpose	MemIf_Write function pointer type definition.
Type	<code>Std_ReturnType(*) (uint16 BlockNumber, const uint8 *DataBufferPtr)</code>

5.5.2.2. Macro constants

5.5.2.2.1. DBG_MEMIF_CANCEL_ENTRY

Purpose	Entry point of function MemIf_Cancel() .
Value	

5.5.2.2.2. DBG_MEMIF_CANCEL_EXIT

Purpose	Exit point of function MemIf_Cancel() .
Value	

5.5.2.2.3. DBG_MEMIF_ERASEIMMEDIATEBLOCK_ENTRY

Purpose	Entry point of function MemIf_EraseImmediateBlock() .
----------------	---

Value	
--------------	--

5.5.2.2.4. DBG_MEMIF_ERASEIMMEDIATEBLOCK_EXIT

Purpose	Exit point of function MemIf_EraseImmediateBlock() .
Value	

5.5.2.2.5. DBG_MEMIF_GETJOBRESULT_ENTRY

Purpose	Entry point of function MemIf_GetJobResult() .
Value	

5.5.2.2.6. DBG_MEMIF_GETJOBRESULT_EXIT

Purpose	Exit point of function MemIf_GetJobResult() .
Value	

5.5.2.2.7. DBG_MEMIF_GETSTATUS_ENTRY

Purpose	Entry point of function MemIf_GetStatus() .
Value	

5.5.2.2.8. DBG_MEMIF_GETSTATUS_EXIT

Purpose	Exit point of function MemIf_GetStatus() .
Value	

5.5.2.2.9. DBG_MEMIF_GETVERSIONINFO_ENTRY

Purpose	Entry point of function MemIf_GetVersionInfo() .
Value	

5.5.2.2.10. DBG_MEMIF_GETVERSIONINFO_EXIT

Purpose	Exit point of function MemIf_GetVersionInfo() .
Value	

5.5.2.2.11. DBG_MEMIF_INVALIDATEBLOCK_ENTRY

Purpose	Entry point of function MemIf_InvalidateBlock() .
Value	

5.5.2.2.12. DBG_MEMIF_INVALIDATEBLOCK_EXIT

Purpose	Exit point of function MemIf_InvalidateBlock() .
Value	

5.5.2.2.13. DBG_MEMIF_READ_ENTRY

Purpose	Entry point of function MemIf_Read() .
Value	

5.5.2.2.14. DBG_MEMIF_READ_EXIT

Purpose	Exit point of function MemIf_Read() .
Value	

5.5.2.2.15. DBG_MEMIF_SETMODE_ENTRY

Purpose	Entry point of function MemIf_SetMode() .
Value	

5.5.2.2.16. DBG_MEMIF_SETMODE_EXIT

Purpose	Exit point of function MemIf_SetMode() .
----------------	--

Value	
--------------	--

5.5.2.2.17. DBG_MEMIF_WRITE_ENTRY

Purpose	Entry point of function MemIf_Write() .
Value	

5.5.2.2.18. DBG_MEMIF_WRITE_EXIT

Purpose	Exit point of function MemIf_Write() .
Value	

5.5.2.2.19. MEMIF_AR_RELEASE_MAJOR_VERSION

Purpose	AUTOSAR release major version.
Value	4U

5.5.2.2.20. MEMIF_AR_RELEASE_MINOR_VERSION

Purpose	AUTOSAR release minor version.
Value	0U

5.5.2.2.21. MEMIF_AR_RELEASE_REVISION_VERSION

Purpose	AUTOSAR release revision version.
Value	3U

5.5.2.2.22. MEMIF_BLOCK_INCONSISTENT

Purpose	The requested block is inconsistent, it may contain corrupted data.
Value	((MemIf_JobResultType)4U)

5.5.2.2.23. MEMIF_BLOCK_INVALID

Purpose	The requested block has been marked as invalid, the requested operation can not be performed.
Value	((MemIf_JobResultType)5U)

5.5.2.2.24. MEMIF_BROADCAST_ID

Purpose	Device ID for "broadcast".
Value	255U
Description	In case the parameter given as device ID is MEMIF_BROADCAST_ID, the memory abstraction interface shall iterate over all underlying devices and return their combined status.

5.5.2.2.25. MEMIF_BUSY

Purpose	The underlying abstraction module or device driver is currently busy.
Value	((MemIf_StatusType)2U)

5.5.2.2.26. MEMIF_BUSY_INTERNAL

Purpose	The underlying abstraction module is busy with internal management operations.
Value	((MemIf_StatusType)3U)
Description	The underlying device driver can be busy or idle.

5.5.2.2.27. MEMIF_CANCEL_ID

Purpose	AUTOSAR API ID of MemIf_Cancel() function.
Value	4U

5.5.2.2.28. MEMIF_ERASEIMMEDIATEBLOCK_ID

Purpose	AUTOSAR API ID of MemIf_EraseImmediateBlock() function.
----------------	---

Value	9U
--------------	----

5.5.2.2.29. MEMIF_E_PARAM_DEVICE

Purpose	Development error codes.
Value	1U
Description	API service called with wrong device index parameter

5.5.2.2.30. MEMIF_E_PARAM_NULL_PTR

Purpose	Development error codes.
Value	2U
Description	API service called with invalid null pointer parameter

5.5.2.2.31. MEMIF_GETJOBRESULT_ID

Purpose	AUTOSAR API ID of MemIf_GetJobResult() function.
Value	6U

5.5.2.2.32. MEMIF_GETSTATUS_ID

Purpose	AUTOSAR API ID of MemIf_GetStatus() function.
Value	5U

5.5.2.2.33. MEMIF_GETVERSIONINFO_ID

Purpose	AUTOSAR API ID of MemIf_GetVersionInfo() function.
Value	8U

5.5.2.2.34. MEMIF_IDLE

Purpose	The underlying abstraction module or device driver is currently idle.
----------------	---

Value	((MemIf_StatusType)1U)
--------------	--

5.5.2.2.35. MEMIF_INSTANCE_ID

Purpose	MemIf instance ID.
Value	0U
Description	The identifier of the index based instance of the Memory Abstraction Interface, starting from 0. I.e. if the module is a single instance module, the instance ID is equal to 0.

5.5.2.2.36. MEMIF_INVALIDATEBLOCK_ID

Purpose	AUTOSAR API ID of MemIf_InvalidateBlock() function.
Value	7U

5.5.2.2.37. MEMIF_JOB_CANCELED

Purpose	The job has been canceled.
Value	((MemIf_JobResultType)3U)

5.5.2.2.38. MEMIF_JOB_FAILED

Purpose	The job has not been finished successfully.
Value	((MemIf_JobResultType)1U)

5.5.2.2.39. MEMIF_JOB_OK

Purpose	The job has been finished successfully.
Value	((MemIf_JobResultType)0U)

5.5.2.2.40. MEMIF_JOB_OK_SIZE_DECREASED

Purpose	The requested block is ok but the block length size is decreased compared to the one configured.
----------------	--

Value	((MemIf_JobResultType)7U)
--------------	---

5.5.2.2.41. MEMIF_JOB_OK_SIZE_INCREASED

Purpose	The requested block is ok but the block length size is increased compared to the one configured.
Value	((MemIf_JobResultType)6U)

5.5.2.2.42. MEMIF_JOB_PENDING

Purpose	The job has not yet been finished.
Value	((MemIf_JobResultType)2U)

5.5.2.2.43. MEMIF_MODE_FAST

Purpose	The underlying memory abstraction modules and drivers are working in fast mode.
Value	((MemIf_ModeType)1U)

5.5.2.2.44. MEMIF_MODE_SLOW

Purpose	The underlying memory abstraction modules and drivers are working in slow mode.
Value	((MemIf_ModeType)0U)

5.5.2.2.45. MEMIF_MODULE_ID

Purpose	AUTOSAR module identification.
Value	22U

5.5.2.2.46. MEMIF_READ_ID

Purpose	AUTOSAR API ID of MemIf_Read() function.
----------------	--

Value	2U
--------------	----

5.5.2.2.47. MEMIF_SETMODE_ID

Purpose	AUTOSAR API ID of MemIf_SetMode() function.
Value	1U

5.5.2.2.48. MEMIF_SW_MAJOR_VERSION

Purpose	AUTOSAR module major version.
Value	5U

5.5.2.2.49. MEMIF_SW_MINOR_VERSION

Purpose	AUTOSAR module minor version.
Value	11U

5.5.2.2.50. MEMIF_SW_PATCH_VERSION

Purpose	AUTOSAR module patch version.
Value	11U

5.5.2.2.51. MEMIF_UNINIT

Purpose	The underlying abstraction module or device driver has not been initialized (yet).
Value	((MemIf_StatusType)0U)

5.5.2.2.52. MEMIF_VENDOR_ID

Purpose	AUTOSAR vendor identification: Elektrobit Automotive GmbH.
----------------	--

Value	1U
--------------	----

5.5.2.2.53. MEMIF_WRITE_ID

Purpose	AUTOSAR API ID of MemIf_Write() function.
Value	3U

5.5.2.3. Objects

5.5.2.3.1. MemIf_CancelFctPtr

Purpose	Function pointer array with *_Cancel() functions.
Type	const MemIf_CancelFctPtrType

5.5.2.3.2. MemIf_EraseImmediateBlockFctPtr

Purpose	Function pointer array with *_EraseImmediateBlock() functions.
Type	const MemIf_EraseImmedBlockFctPtrType

5.5.2.3.3. MemIf_GetJobResultFctPtr

Purpose	Function pointer array with *_GetJobResult() functions.
Type	const MemIf_GetJobResultFctPtrType

5.5.2.3.4. MemIf_InvalidateBlockFctPtr

Purpose	Function pointer array with *_InvalidateBlock() functions.
Type	const MemIf_InvalidateBlockFctPtrType

5.5.2.3.5. MemIf_ReadFctPtr

Purpose	Function pointer array with *_Read() functions.
----------------	---

Type	const MemIf_ReadFctPtrType
------	--

5.5.2.3.6. MemIf_WriteFctPtr

Purpose	Function pointer array with *_Write() functions.
Type	const MemIf_WriteFctPtrType

5.5.2.4. Functions

5.5.2.4.1. MemIf_Cancel

Purpose	Calls the cancel function of the underlying modules: either Flash EEPROM Emulation or EEPROM Abstraction.	
Synopsis	void MemIf_Cancel (uint8 DeviceIndex);	
Service ID	4	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	DeviceIndex	is used for selection of memory abstraction modules (and thus memory devices). If only one memory abstraction module is configured, the parameter DeviceIndex is ignored.

5.5.2.4.2. MemIf_EraseImmediateBlock

Purpose	Mapped to service EraseImmediateBlock of corresponding EEPROM Abstraction and Flash EEPROM Emulation modules.	
Synopsis	Std_ReturnType MemIf_EraseImmediateBlock (uint8 DeviceIndex , uint16 BlockNumber);	
Service ID	9	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	DeviceIndex	is used for selection of memory abstraction modules (and thus memory devices).

		If only one memory abstraction module is configured, the parameter DeviceIndex is ignored.
	BlockNumber	Number of logical block in EA or FEE
Return Value	Std_ReturnType	
	E_OK	Block Erased successfully.
	E_NOT_OK	Block Erase Failed

5.5.2.4.3. MemIf_GetJobResult

Purpose	Mapped to service GetJobResult of corresponding EEPROM Abstraction and Flash EEPROM Emulation modules.	
Synopsis	<code>MemIf_JobResultType MemIf_GetJobResult (uint8 DeviceIndex);</code>	
Service ID	6	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	DeviceIndex	is used for selection of memory abstraction modules (and thus memory devices). If only one memory abstraction module is configured, the parameter DeviceIndex is ignored.
Return Value	job result	
	MEMIF_JOB_OK	Job finished successfully.
	MEMIF_JOB_FAILED	Job failed.
	MEMIF_JOB_PENDING	Job is being performed.
	MEMIF_JOB_CANCELLED	Job has been cancelled.
	MEMIF_BLOCK_INCONSISTENT	Block inconsistent, can't read data.
	MEMIF_BLOCK_INVALID	Block marked invalid, can't read data.
	MEMIF_COMPARE_UNEQUAL	Block Comparison Unequal

5.5.2.4.4. MemIf_GetStatus

Purpose	Mapped to service GetStatus of corresponding EEPROM Abstraction and Flash EEPROM Emulation modules.
----------------	---

Synopsis	<code>MemIf_StatusType MemIf_GetStatus (uint8 DeviceIndex);</code>	
Service ID	5	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	DeviceIndex	is used for selection of memory abstraction modules (and thus memory devices). If only one memory abstraction module is configured, the parameter DeviceIndex is ignored.
Return Value	Status of memory device	
	MEMIF_IDLE	if all underlying devices have returned this state
	MEMIF_UNINIT	if at least one device returned this state, all other returned states shall be ignored
	MEMIF_BUSY	if at least one configured device returned this state and no other device returned MEMIF_UNINIT
	MEMIF_BUSY_INTERNAL	if at least one configured device returned this state and no other device returned MEMIF_BUSY or MEMIF_UNINIT
Description	<p>This implementation is only used, if more than one memory abstraction module is configured or development error detection is defined.</p> <p>If the function MemIf_GetStatus is called with the device index denoting a broadcast to all configured devices (see MemIf036), this module shall call the "GetStatus" functions of all underlying devices in turn.</p> <p>The device index value MEMIF_BROADCAST_ID is used to identify all underlying devices within one call. This special "broadcast" device ID is only allowed in the call to MemIf_GetStatus to determine the status of all underlying abstraction modules and device drivers.</p>	

5.5.2.4.5. MemIf_GetVersionInfo

Purpose	Get version information.
Synopsis	<code>void MemIf_GetVersionInfo (Std_VersionInfoType * VersionInfoPtr);</code>
Service ID	8

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (out)	versioninfo	Pointer to standard version information structure.
Description	This function provides the information to module vendor ID, module ID and software version major.minor.patch	

5.5.2.4.6. MemIf_InvalidateBlock

Purpose	Mapped to service InvalidateBlock of corresponding EEPROM Abstraction and Flash EEPROM Emulation modules.	
Synopsis	Std_ReturnType MemIf_InvalidateBlock (uint8 DeviceIndex , uint16 BlockNumber);	
Service ID	7	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	DeviceIndex	is used for selection of memory abstraction modules (and thus memory devices). If only one memory abstraction module is configured, the parameter DeviceIndex is ignored.
	BlockNumber	Number of logical block in EA or FEE
Return Value	Std_ReturnType	
	E_OK	Invalidated Block successfully.
	E_NOT_OK	Block Invalidation Failed

5.5.2.4.7. MemIf_Read

Purpose	Mapped to read service of corresponding EEPROM Abstraction and Flash EEPROM Emulation modules.	
Synopsis	Std_ReturnType MemIf_Read (uint8 DeviceIndex , uint16 BlockNumber , uint16 BlockOffset , uint8 * DataBufferPtr , uint16 Length);	
Service ID	2	

Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	DeviceIndex	is used for selection of memory abstraction modules (and thus memory devices). If only one memory abstraction module is configured, the parameter DeviceIndex is ignored.
	BlockNumber	Number of logical block in EA or FEE
	BlockOffset	Read address offset inside the block
	Length	Number of bytes to read
Parameters (out)	DataBufferPtr	Pointer to data buffer
Return Value	Std_ReturnType	
	E_OK	Read finished successfully.
	E_NOT_OK	Read failed.

5.5.2.4.8. MemIf_SetMode

Purpose	Mapped to service SetMode of all EEPROM Abstraction and Flash EEPROM Emulation modules.	
Synopsis	<code>void MemIf_SetMode (MemIf_ModeType Mode);</code>	
Service ID	1	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	Mode	Desired mode for the underlying flash or EEPROM driver
Description	no error detection, because no DeviceIndex	

5.5.2.4.9. MemIf_Write

Purpose	Mapped to write service of corresponding EEPROM Abstraction and Flash EEPROM Emulation modules.	
Synopsis	<code>Std_ReturnType MemIf_Write (uint8 DeviceIndex , uint16 BlockNumber , const uint8 * DataBufferPtr);</code>	

Service ID	3	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	DeviceIndex	is used for selection of memory abstraction modules (and thus memory devices). If only one memory abstraction module is configured, the parameter DeviceIndex is ignored.
	BlockNumber	Number of logical block in EA or FEE
	DataBufferPtr	Pointer to data buffer
Return Value	Std_ReturnType	
	E_OK	Write finished successfully.
	E_NOT_OK	Write failed.

5.5.3. Integration notes

5.5.3.1. Exclusive areas

Exclusive areas are not used by the `MemIf` module.

5.5.3.2. Production errors

Production errors are not reported by the `MemIf` module.

5.5.3.3. Memory mapping

General information about memory mapping is provided in the EB tresos AutoCore Generic documentation. Refer to the section `Memory mapping and compiler abstraction` in the `Integration notes` section for details.

The following table provides the list of sections that may be mapped for this module:

Memory section



CODE
CONST_UNSPECIFIED

5.5.3.4. Integration requirements

WARNING **Integration requirements list is not exhaustive**



The following list of integration requirements helps you to integrate your product. However, this list is not exhaustive. You also require information from the user guide, release notes, and EB tresos AutoCore known issues to successfully integrate your product.

Integration requirements are not listed for the MemIf module.

5.6. NvM

5.6.1. Configuration parameters

Containers included		
Container name	Multiplicity	Description
CommonPublishedInformation	1..1	Label: Common Published Information Common container, aggregated by all modules. It contains published information about vendor and versions.
NvMBlockDescriptor	1..65536	Container for a management structure to configure the composition of a given NVRAM Block Management Type. Its multiplicity describes the number of configured NVRAM blocks, one block is required to be configured. The NVRAM block descriptors are condensed in the NVRAM block descriptor table. The start address of the table is configured in the <code>NVMCommon</code> parameters.
NvMDefensiveProgramming	1..1	Label: Defensive Programming Options Parameters for defensive programming
NvMCommon	1..1	Label: Common NvM Parameters

Containers included		
		Container for common configuration options.
NvmDemEventParameter-Refs	1..1	Label: Dem Event Parameter References Container for the references to DemEventParameter elements which shall be invoked using the API <code>Dem_ReportErrorStatus</code> API in case the corresponding error occurs. The <code>EventId</code> is taken from the referenced <code>DemEventParameter's DemEventId</code> value. The standardized errors are provided in the container and can be extended by vendor specific error references.
ReportToDem	1..1	Label: Production Error Handling Production error handling
MultiCoreCallout	1..1	Label: Multi-Core Callouts configuration Multi-Core Callouts configuration
PublishedInformation	1..1	Label: EB Published Information Additional published parameters not covered by Common-PublishedInformation container.

Parameters included	
Parameter name	Multiplicity
IMPLEMENTATION_CONFIG_VARIANT	1..1

Parameter Name	IMPLEMENTATION_CONFIG_VARIANT	
Label	Config Variant	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	VariantPreCompile	
Range	VariantPreCompile	
Configuration class	VariantPreCompile:	VariantPreCompile

5.6.1.1. CommonPublishedInformation

Parameters included	
Parameter name	Multiplicity
ArMajorVersion	1..1

Parameters included	
ArMinorVersion	1..1
ArPatchVersion	1..1
SwMajorVersion	1..1
SwMinorVersion	1..1
SwPatchVersion	1..1
ModuleId	1..1
VendorId	1..1
Release	1..1

Parameter Name	ArMajorVersion
Label	AUTOSAR Major Version
Description	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	3
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	ArMinorVersion
Label	AUTOSAR Minor Version
Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	2
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	ArPatchVersion
Label	AUTOSAR Patch Version
Description	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.

Multiplicity	1..1
Type	INTEGER_LABEL
Default value	0
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	SwMajorVersion
Label	Software Major Version
Description	Major version number of the vendor specific implementation of the module.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	6
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	SwMinorVersion
Label	Software Minor Version
Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	17
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	SwPatchVersion
Label	Software Patch Version
Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	22
Configuration class	PublishedInformation:

Origin	Elektrobit Automotive GmbH
---------------	----------------------------

Parameter Name	ModuleId
Label	Numeric Module ID
Description	Module ID of this module from Module List
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	20
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	VendorId
Label	Vendor ID
Description	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
Multiplicity	1..1
Type	INTEGER_LABEL
Default value	1
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

Parameter Name	Release
Label	Release Information
Multiplicity	1..1
Type	STRING_LABEL
Default value	
Configuration class	PublishedInformation:
Origin	Elektrobit Automotive GmbH

5.6.1.2. NvMBlockDescriptor

Containers included		
Container name	Multiplicity	Description

Containers included		
NvMTargetBlockReference	1..1	Label: Target Block Reference

Parameters included	
Parameter name	Multiplicity
NvMAdvancedRecovery	1..1
NvMBlockCrcType	0..1
NvMBlockJobPriority	1..1
NvMBlockManagementType	1..1
NvMBlockUseCrc	0..1
NvMBlockUseSyncMechanism	1..1
NvMBlockWriteProt	1..1
NvMBswMBlockStatusInformation	1..1
NvMCalcRamBlockCrc	0..1
NvMBlockUseCRCCompMechanism	0..1
NvMBlockUseSetRamBlockStatus	1..1
NvMExtraBlockChecks	1..1
NvMInitBlockCallback	0..1
NvMMaxNumOfReadRetries	1..1
NvMMaxNumOfWriteRetries	1..1
NvMNvBlockBaseNumber	1..1
NvMNvBlockLength	1..1
NvMNvBlockNum	1..1
NvMNvramBlockIdentifier	1..1
NvMNvramDeviceId	1..1
NvMProvideRteAdminPort	1..1
NvMProvideRteInitBlockPort	1..1
NvMProvideRteJobFinishedPort	1..1
NvMProvideRteMirrorPort	1..1
NvMProvideRteServicePort	1..1
NvMRPortInterfacesASRVersion	0..1
NvMRamBlockDataAddress	0..1
NvMReadRamBlockFromNvCallback	0..1

Parameters included	
NvMResistantToChangedSw	1..1
NvMRomBlockDataAddress	0..1
NvMRomBlockNum	1..1
NvMSelectBlockForReadAll	0..1
NvMSelectBlockForWriteAll	0..1
NvMSelectBlockForFirstInitAll	0..1
NvMSingleBlockCallback	0..1
NvMStaticBlockIDCheck	1..1
NvMBlockUseAutoValidation	0..1
NvMUserProvidesSpaceForBlockAndCrc	1..1
NvMEnBlockCheck	1..1
NvMEnableBlockCryptoSecurityHandling	1..1
NvMCryptoExtraInfoSize	1..1
NvMBcEnSetAPI	1..1
NvMBcEnAutoStart	1..1
NvMBcEnCrcComp	1..1
NvMBcEnRamComp	1..1
NvMBcEnReddCopiesComp	1..1
NvMBcEnAutoRepair	1..1
NvMBcDelayCounter	1..1
NvMWriteBlockOnce	1..1
NvMWriteRamBlockToNvCallback	0..1
NvMWriteVerification	1..1
NvMWriteVerificationDataSize	1..1
NvMPreWriteDataComp	0..1
NvMPreWriteDataCompDataSize	1..1

Parameter Name	NvMAdvancedRecovery
Label	Enable Advanced Recovery
Description	<p>Enables additional recovery mechanisms during NvM_ReadAll()/NvM_Read-Block() requests.</p> <p>► true = Additional recovery mechanism is enabled for this block.</p>

	<p>► <code>false</code> = Additional recovery mechanism is disabled for this block.</p> <p>Blocks can be recovered from the ROM block or via call to a user-defined initialization function.</p> <p>Blocks that are recovered will be marked for writing during <code>NvM_WriteAll()</code> (including blocks which are marked as write protected).</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► RAM Block Data Address (<code>NvMRamBlockDataAddress</code>): this feature is only supported for blocks with a permanent RAM block address configured. ► Rom Block Data Address (<code>NvMRomBlockDataAddress</code>) = if a ROM address is configured for the block, and the recovery from the redundant block was unsuccessful, the ROM copy will be used to try to recover the data ► Initialize NVRAM Block Data Function (<code>NvMInitBlockCallback</code>) = this specifies the user defined initialization function to be called by the NvM to recover the data, if the previous recovery attempts were unsuccessful. <p>Independent of the setting of this parameter, the implicit error recovery is performed for all unprotected blocks with a permanent RAM block.</p> <p>Independent of the setting of this parameter, explicit recovery can also be performed with <code>NvM_RestoreBlockDefaults</code>.</p>
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMBlockCrcType
Label	Block CRC Type
Description	<p>Defines CRC data width for the NVRAM block.</p> <p>Range:</p> <ul style="list-style-type: none"> ► Block CRC Type <code>NVM_CRC8</code> ► Block CRC Type <code>NVM_CRC16</code> ► Block CRC Type <code>NVM_CRC32</code> <p>Default: <code>NVM_CRC16</code></p>

	<p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► Enable Use of Block CRC (<code>NvMBlockUseCrc</code>): this must be enabled if CRC calculation for this block is required <p>Note: For CRC8 calculation, the NvM module supports SAE J1850 CRC Calculation only. Activate support for this CRC8 calculation by enabling the CRC8 Mode parameter in the Crc module (not the CRC8 H2F Mode)</p>	
Multiplicity	0..1	
Type	ENUMERATION	
Default value	NVM_CRC16	
Range	NVM_CRC8	
	NVM_CRC16	
	NVM_CRC32	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMBlockJobPriority	
Label	Job Priority for Block	
Description	<p>Defines the job priority for a block stored in non-volatile memory (NVRAM)</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► Enable Job Prioritization (<code>NvMJobPrioritization</code>): this must be enabled if job prioritization is required. Blocks will then be handled depending on their actual priority setting. ► Enable Use of Block CRC (<code>NvMBlockUseCrc</code>): This is expected to be disabled if the block is configured with immediate priority (<code>NvMBlockJobPriority = 0</code>). <p>Range:</p> <ul style="list-style-type: none"> ► 0 (highest) : use for blocks containing immediate data. These blocks shall be configured as immediate blocks in the underlying abstraction layer (Ea/Fee) ► 1 (high) .. 255 (low) : use to prioritize blocks other than those with immediate data. 	
Multiplicity	1..1	
Type	INTEGER	

Default value	0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMBlockManagementType	
Label	Block Management Type	
Description	<p>Defines the block management type for the NVRAM block.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ NVM_BLOCK_NATIVE (1 NV block) ▶ NVM_BLOCK_REDUNDANT (2 NV blocks) ▶ NVM_BLOCK_DATASET (1 .. 255 NV blocks, depending on number of data selection bits) <p>Note: the block management type selected affects the block numbering configuration in the underlying abstraction layer (Ea/Fee). For further details and examples, refer to the memory stack User's Guide.</p>	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	NVM_BLOCK_NATIVE	
Range	NVM_BLOCK_DATASET	
	NVM_BLOCK_NATIVE	
	NVM_BLOCK_REDUNDANT	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMBlockUseCrc	
Label	Enable Use of Block CRC	
Description	<p>Enables the use of CRC for the NVRAM block. The CRC will be written to NV memory together with the block data.</p> <p>Note: Memory space for the block data and its CRC is reserved internally unless otherwise specified by the user using parameter <code>NvMUserProvidesSpaceForBlockAndCrc</code> for permanent RAM blocks.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = CRC will be used for this NVRAM block ▶ <code>false</code> = CRC will not be used for this NVRAM block 	

Multiplicity	0..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMBlockUseSyncMechanism
Label	Enable Explicit Synchronization Mechanism
Description	<p>Enables explicit synchronization for this block.</p> <p>A RAM mirror is used as a buffer for reading and writing this block. The RAM block and the RAM mirror are synchronized explicitly when dedicated call-back routines are called from the NvM. The call-backs transfer the data between the RAM block and the RAM mirror and must be implemented by the application.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Explicit synchronization will be enabled for this block ▶ <code>false</code> = Explicit synchronization will be disabled for this block <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ NvM to RAM Copy Call-back Function (<code>NvMReadRamBlockFromNvCall-back</code>): call-back to transfer data from the NvM mirror to the RAM block ▶ RAM to NvM Copy Call-back Function (<code>NvMWriteRamBlockToNvCall-back</code>): call-back to transfer data from the RAM block to the NvM mirror
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMBlockWriteProt
Label	Enable Initial Write Protection for Block
Description	<p>Enables initial write protection for the NV block.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Initial block write protection is enabled ▶ <code>false</code> = Initial block write protection is disabled
Multiplicity	1..1
Type	BOOLEAN

Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMBswMBlockStatusInformation
Label	Enable BswM Block Status Information
Description	<p>Defines whether BswM is informed about the current status of the specified block.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Report current status of block to BswM ▶ <code>false</code> = Don't Report current status of block to BswM <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable <code>BswMNvMEnabled</code>: this must be selected to enable the NvM module related BswM API.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMCalcRamBlockCrc
Label	Enable Calculation of Permanent RAM Block CRC
Description	<p>Enables CRC (re)calculation for a permanent RAM block</p> <ul style="list-style-type: none"> ▶ <code>true</code> = CRC will be (re)calculated for this permanent RAM block ▶ <code>false</code> = CRC will not be (re)calculated for this permanent RAM block <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Use of Block CRC (<code>NvMBlockUseCrc</code>): this must be enabled to use CRC for a block and therefore also for configuring the CRC (re)calculation of a permanent RAM block.
Multiplicity	0..1
Type	BOOLEAN
Default value	false
Configuration class	PreCompile: VariantPreCompile

Origin	AUTOSAR_ECUC	
Parameter Name	NvMBlockUseCRCCompMechanism	
Label	Enable CRC Comparison Mechanism .	
Description	<p>Enables CRC comparison mechanism for a RAM block during a write job.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = The job of the function <code>NvM_WriteBlock</code> shall skip writing and consider the job as successfully finished if the RAM block CRC calculated by the write job is equal to the CRC calculated during the last successful read or write job. ▶ <code>false</code> = A "Write Request" job shall proceed with writing without taking in consideration any changes CRC value. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Use of Block CRC (<code>NvMBlockUseCrc</code>) : this must be enabled to use CRC for a block and therefore also for configuring the CRC comparison mechanism. 	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMBlockUseSetRamBlockStatus	
Label	Enable Service SetRamBlockStatus	
Description	<p>Defines if <code>NvMSetRamBlockStatusApi</code> shall be used for this block or not.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Calling of <code>NvMSetRamBlockStatus</code> for this RAM block shall set the status of the RAM block. ▶ <code>false</code> = Calling of <code>NvMSetRamBlockStatus</code> for this RAM block shall be ignored. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMExtraBlockChecks	
----------------	----------------------------	--

Label	Enable Block Size Check	
Description	<p>Enables extra compile-time checks for configured RAM (<code>NvMRamBlockDataAddress</code>) and ROM (<code>NvMRomBlockDataAddress</code>) block sizes.</p> <p>The size of the block data, calculated using the <code>sizeof()</code> C function, is compared to the data block size configured with the parameter <code>NvMNvBlockLength</code> and the length of the CRC (if CRC is configured for the block and <code>NvMUserProvidesSpaceForBlockAndCrc</code> configuration parameter is enabled).</p> <p>Note: The block size check is only performed if the configured value starts with "&". In case of using arrays, the block size check for the entire array will be performed only if the address of the array is given (e.g. <code>&My_RAMBlockTable</code>). The block size check for one element of the array is performed only if the address of one element is given (e.g. <code>&My_RAMBlockTable[0]</code>). Even if passing the name of the array might be a valid configuration, the block size check will not be performed in this case.</p> <p>Note: If the user enables the parameter <code>NvMUserProvidesSpaceForBlockAndCrc</code>, the space required for the CRC is considered along with the block length (<code>NvMNvBlockLength</code>) when the block size check is performed.</p> <p>Note: Due to the fact that Autosar does not specify the type of address for <code>NvMRamBlockDataAddress</code> and <code>NvMRomBlockDataAddress</code>, it is not possible to check for the correct sizes if the size of the associated RAM or ROM variable cannot be determined with the <code>sizeof()</code> C function. In such cases, disable the block size check.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► RAM Block Data Address <code>NvMRamBlockDataAddress</code> or Rom Block Data Address <code>NvMRomBlockDataAddress</code> needs to be configured for performing Extra Block Checks. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMInitBlockCallback
Label	Initialize NVRAM Block Data Function
Description	Defines the name of a block specific call-back function that will be called if no ROM data is available for initialization of the NVRAM block.

	Options:	
	<ul style="list-style-type: none"> ▶ name of call-back function: this function will be called to initialize NVRAM block data ▶ empty: no function will be called 	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMMaxNumOfReadRetries	
Label	Max. Number of Read Retries	
Description	<p>Defines the maximum number of read retries for this block.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 0 .. 7 	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<div><=7</div> <div>>=0</div>	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMMaxNumOfWriteRetries	
Label	Max. Number of Write Retries	
Description	<p>Defines the maximum number of write retries for an NVRAM block.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 0 .. 7 	
Multiplicity	1..1	
Type	INTEGER	
Default value	3	
Range	<div><=7</div> <div>>=0</div>	

Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMNvBlockBaseNumber	
Label	Block Base Number	
Description	<p>Defines the link between the Block Identifier (<code>NvMNvramBlockIdentifier</code>) used by SW-Cs and the <code>Fee/Ea Block Number</code> expected by the memory abstraction modules. The parameter is calculated from <code>FeeBlockNumber</code> or <code>EaBlockNumber</code> with all configured Data-set selection bits set to zero.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► <code>Fee/Ea Block Number</code> (<code>[Fee/Ea]BlockNumber</code>): the block base number is the value of <code>[Fee/Ea]BlockNumber</code> referenced in <code>NvMTargetBlock-Reference</code> shifted right by the number of data-set selection bits. ► <code>Number of Data-set Selection Bits</code> (<code>NvMDatasetSelectionBits</code>): See the above dependency. <p>Note: The calculated default value of the Block Base Number must not be changed. Whenever the Block Number of the associated Fee or Ea block is changed, the value of Block Base Number must be recalculated.</p>	
Multiplicity	1..1	
Type	INTEGER	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMNvBlockLength	
Label	Block Length	
Description	Defines the NV block data length in bytes	
Multiplicity	1..1	
Type	INTEGER	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMNvBlockNum	
Label	Number of NVRAM Copies for Block	
Description	<p>Defines the number of copies of NVRAM blocks in a contiguous memory area. This number depends on the block management type configured.</p>	

	<p>Range:</p> <ul style="list-style-type: none"> ▶ Block management type <code>NVM_BLOCK_DATASET</code> : 1 .. 255 Note: the number that can be configured here is limited. The total number of these copies plus the number of ROM copies must lie in this range. ▶ Block management type <code>NVM_BLOCK_NATIVE</code> : 1 ▶ Block management type <code>NVM_BLOCK_REDUNDANT</code> : 2 <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Block Management Type (<code>NvMBlockManagementType</code>): limits the range selection. 	
Multiplicity	1..1	
Type	INTEGER	
Default value	1	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMNvramBlockIdentifier	
Label	Block Identifier	
Description	<p>Defines a unique identifier to reference the NVRAM block.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ $1 \dots 2^{(16 - \text{NVM_DATASET_SELECTION_BITS})} - 1$ <p>Note: the following block identifiers are reserved</p> <ul style="list-style-type: none"> ▶ 0 : this is reserved to retrieve the results of multiblock requests with <code>NvM_GetErrorStatus</code> ▶ 1 : this is reserved for the redundant NVRAM block which holds the configuration ID 	
Multiplicity	1..1	
Type	INTEGER	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMNvramDeviceId	
Label	NVRAM Device Identifier	
Description	Defines the NVRAM device ID where the NVRAM block is located.	

	<p>The value configured here must match the device driver index (in Eep/FIs) that is referenced from the corresponding configuration for this block in the underlying abstraction layer (Ea/Fee).</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 0 .. 7 <p>Note: The range of values supported by the current NvM implementation is limited to 0 .. 7. The range supported by the underlying device(s) (Eep/FIs) may be larger. The configuration of the underlying driver device index is therefore restricted to the range supported here.</p>
Multiplicity	1..1
Type	INTEGER
Default value	0
Range	<div><=7</div> <div>>=0</div>
Configuration class	<div>VariantPreCompile:</div> <div>VariantPreCompile</div>
Origin	AUTOSAR_ECUC

Parameter Name	NvMProvideRteAdminPort
Label	Enable Rte Administration Port
Description	<p>Enables the generation of the <code>NvMAdmin</code> port for this block.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Use of Rte (<code>NvMRteUsage</code>): must be enabled. <p>This port includes the operation <code>SetBlockProtection()</code>.</p>
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	<div>VariantPreCompile:</div> <div>VariantPreCompile</div>
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMProvideRteInitBlockPort
Label	Enable Rte Init Block Port
Description	Enables the generation of the <code>NvMNotifyInitBlock</code> port for this block.

	<p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Use of Rte (NvMRteUsage): must be enabled. <p>This port includes the operation <code>InitBlock()</code>.</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMProvideRteJobFinishedPort	
Label	Enable Rte Job Finished Port	
Description	<p>Enables the generation of the <code>NvMNotifyJobFinished</code> port for this block.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Use of Rte (NvMRteUsage): must be enabled. <p>This port includes the operation <code>JobFinished()</code>.</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMProvideRteMirrorPort	
Label	Enable Rte Mirror Port	
Description	<p>Enables the generation of the <code>NvMMirror</code> port for this block.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Use of Rte (NvMRteUsage): must be enabled. ▶ Enable Explicit Synchronization Mechanism (NvMBlockUseSyncMechanism): This must be enabled to call the mirror functions. <p>This port includes the operations <code>ReadRamBlockFromNvm()</code> and <code>WriteRamBlockToNvm()</code>.</p>	
Multiplicity	1..1	

Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMPProvideRteServicePort	
Label	Enable Rte Service Port	
Description	<p>Enables the generation of the <code>NvMService</code> port for this block.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Use of Rte (<code>NvMRteUsage</code>): must be enabled. <p>This port includes the operations:</p> <ul style="list-style-type: none"> ▶ <code>GetErrorStatus()</code> ▶ <code>SetDataIndex()</code> ▶ <code>GetDataIndex()</code> ▶ <code>SetRamBlockStatus()</code> ▶ <code>ReadBlock()</code> ▶ <code>WriteBlock()</code> ▶ <code>RestoreBlockDefaults()</code> ▶ <code>EraseBlock()</code> ▶ <code>InvalidateNvBlock()</code> 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMRPortInterfacesASRVersion	
Label	R-Port Interfaces ASR Version	
Description	<p>Defines the AUTOSAR version of the R-Ports called by NvM.</p> <ul style="list-style-type: none"> ▶ <code>AUTOSAR_32</code> = NvM shall call the AUTOSAR 3.2 R-Port interfaces. ▶ <code>AUTOSAR_40</code> = NvM shall call the AUTOSAR 4.0 R-Port interfaces. ▶ <code>AUTOSAR_42</code> = NvM shall call the AUTOSAR 4.2 R-Port interfaces. 	

	<p>► DEFAULT = NvM shall call the R-Port interfaces selected by NvMDefaultASRServiceAPI configuration parameter.</p> <p>This configuration parameter refers to the following interfaces:</p> <ul style="list-style-type: none"> ► NvMNotifyInitBlock ► NvMNotifyJobFinished ► ReadRamBlockFromNvm ► WriteRamBlockToNvm <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► NvMRteUsage: Rte usage must be enabled in NvM. ► NvMProvideRteInitBlockPort ► NvMProvideRteJobFinishedPort ► NvMProvideRteMirrorPort ► NvMEnableASRxxServiceAPI or NvMDefaultASRServiceAPI: the value selected for the R-Port interfaces requires that the corresponding AUTOSAR Service API version is enabled by any of these two configuration parameters. 	
Multiplicity	0..1	
Type	ENUMERATION	
Range	AUTOSAR_32	
	AUTOSAR_40	
	AUTOSAR_42	
	DEFAULT	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMRamBlockDataAddress
Label	RAM Block Data Address
Description	<p>Defines the start address of the permanent RAM block data area for this block.</p> <p>If no permanent RAM data block is available for this block, leave this field empty. Otherwise, specify the address of a valid RAM location e.g. &My_RAMBlockTable[5] when referring to an element of an array, or &My_RAMBlockTable when referring the table</p> <p>Dependency on parameter(s):</p>

	► User Header File <code>NvMUserHeader</code> : if a permanent RAM block is configured, the user header file must be set to a valid header file including the declaration of the RAM block location (either directly or indirectly).	
Multiplicity	0..1	
Type	STRING	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMReadRamBlockFromNvCallback	
Label	NvM to RAM Copy Call-back Function	
Description	<p>Defines the name of a block specific call-back function which will be called by the NvM in order to let the application copy data from the NvM module's mirror to this block's RAM block.</p> <p>The call-back function shall have the following prototype:</p> <pre>► Std_ReturnType NvM_ReadRamBlockFromNvM(void* NvMBuffer)</pre> <p>The possible return values are:</p> <ul style="list-style-type: none"> ► E_OK : copy was successful ► E_NOT_OK : copy was not successful, call-back routine to be called again <p>Options:</p> <ul style="list-style-type: none"> ► name of call-back function: this function will be called to copy data from the NvM module's mirror to RAM block ► empty: no function will be called <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► Number of Repetitions for Mirror Operations (<code>NvMRepeatMirrorOperation</code>): number of times this call-back will be called if E_NOT_OK is returned. ► Enable Explicit Synchronization Mechanism (<code>NvMBlockUseSyncMechanism</code>): This must be enabled to call the NvM to RAM Copy Call-back Function. 	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMResistantToChangedSw
Label	Resistant to Changed Configuration
Description	<p>Defines whether or not an NVRAM block shall be handled during startup as being resistant to configuration changes.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = this block is not affected by the configuration change so its data can be loaded from NVRAM during startup ▶ <code>false</code> = this block is affected by the configuration change so default data for the block shall be loaded during the extended runtime preparation <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Dynamic Configuration (<code>NvMDynamicConfiguration</code>): must be enabled to activate the extended runtime preparation for those blocks with the parameter <code>NvMResistantToChangedSw</code> set to <code>false</code> here. <p>Note: If there is no default data available at configuration time then the application shall be responsible for providing the default initialization data. In this case the application has to use <code>NvM_GetErrorStatus()</code> to be able to distinguish between first initialization and corrupted data.</p>
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMRomBlockDataAddress
Label	ROM Block Data Address
Description	<p>Defines the start address of the ROM block data area for this block.</p> <p>If no ROM data block is available for this block, leave this field empty. Otherwise, specify the address of a valid ROM location e.g. <code>&My_ROMBlockTable[5]</code> when referring to an element of an array, or <code>&My_ROMBlockTable</code> when referring the table</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ User Header File <code>NvMUserHeader</code>: if a permanent ROM block is configured, the user header file must be set to a valid header file including the declaration of the RAM block location (either directly or indirectly).
Multiplicity	0..1

Type	STRING	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMRomBlockNum	
Label	Number of ROM Blocks	
Description	<p>Defines the number of ROM blocks in a contiguous area that are needed to store multiple instances of this block.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Block Management Type (NvMBlockManagementType): limits the range selection. <p>Range:</p> <ul style="list-style-type: none"> ▶ Block management type NVM_BLOCK_DATASET : 0 .. 255 ▶ Block management type NVM_BLOCK_NATIVE : 0 .. 1 ▶ Block management type NVM_BLOCK_REDUNDANT : 0 .. 1 	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMSelectBlockForReadAll	
Label	Select Block for ReadAll()	
Description	<p>Defines whether or not an NVRAM block shall be processed during NvM_ReadAll().</p> <ul style="list-style-type: none"> ▶ true = NVRAM block shall be processed during startup by NvM_ReadAll() ▶ false = NVRAM block shall not be processed during startup by NvM_ReadAll() <p>Dependency on parameter(s):</p> <p>At least one of the following conditions has to be fulfilled</p> <ul style="list-style-type: none"> ▶ RAM Block Data Address NvMRamBlockDataAddress: this must be set to a valid permanent RAM address, 	

	<ul style="list-style-type: none"> ▶ Explicit synchronization <code>NvMBlockUseSyncMechanism</code>: this must be enabled. ▶ Block management type <code>NvMBlockManagementType</code>: this must not be set to <code>NVM_BLOCK_DATASET</code>. 	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	true	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMSelectBlockForWriteAll	
Label	Select Block for WriteAll	
Description	<p>Defines whether or not the NVRAM block shall be processed during <code>NvM_WriteAll()</code>.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = NVRAM block shall be processed by <code>NvM_WriteAll</code> ▶ <code>false</code> = NVRAM block shall not be processed by <code>NvM_WriteAll</code> <p>Dependency on parameter(s):</p> <p>At least one of the following conditions has to be fulfilled:</p> <ul style="list-style-type: none"> ▶ RAM Block Data Address <code>NvMRamBlockDataAddress</code>: this must be set to a valid permanent RAM address, ▶ Explicit synchronization <code>NvMBlockUseSyncMechanism</code>: this must be enabled. 	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	true	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMSelectBlockForFirstInitAll	
Label	Select Block for FirstInitAll	
Description	<p>Defines whether or not the NVRAM block shall be processed during <code>NvM_FirstInitAll()</code>.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = NVRAM block shall be processed by <code>NvM_FirstInitAll</code> 	

	► <code>false</code> = NVRAM block shall not be processed by <code>NvM_FirstInitAll</code>	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMSingleBlockCallback	
Label	Single Block Call-back Function	
Description	<p>Defines the start address of a block specific function that shall be called on termination of each asynchronous single block request.</p> <p>Note: The call-back function is called inside a critical section. Hence, application must ensure that the run time of this function is reasonably short.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMStaticBlockIDCheck	
Label	Enable Static Block ID Check	
Description	<p>Enables the Static Block ID check for this block.</p> <p>If enabled, the Static Block ID is stored in the NV Block each time the NVRAM block is written to NV memory. The ID is read and compared to the requested block ID each time the block is read from NV memory.</p> <p>► <code>true</code> = Static Block ID check is enabled</p> <p>► <code>false</code> = Static Block ID check is disabled</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMBlockUseAutoValidation	
Label	Select Block for ValidateAll()	

Description	<p>This defines whether or not an NVRAM block shall be processed during <code>ValidateAll()</code>.</p> <p>Dependency on parameter(s):</p> <p>One of the following conditions has to be fulfilled</p> <ul style="list-style-type: none"> ▶ RAM Block Data Address <code>NvMRamBlockDataAddress</code>: this must be set to a valid permanent RAM address, ▶ Explicit synchronization <code>NvMBlockUseSyncMechanism</code>: this must be enabled. <p><code>NvMApiConfigClass</code> must be different than <code>NVM_API_CONFIG_CLASS_1</code>.</p> <p><code>NvMNvramBlockIdentifier</code> must be greater than 1</p>
Multiplicity	0..1
Type	BOOLEAN
Default value	false
Configuration class	PreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMUserProvidesSpaceForBlockAndCrc
Label	User Provides Space for Block and CRC
Description	<p>Enables storage of the block CRC in the user buffer when <code>NvMBlockUseCrc</code> is enabled.</p> <p>This configuration parameter shall be used only for permanent RAM blocks.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = user provides space for CRC storage directly after the block data, NvM module does not reserve internal buffer space for reading and writing this block. The configured permanent RAM buffer must be large enough to store both the NV data and the length of the block's CRC. ▶ <code>false</code> = user does not provide space for CRC, NvM module reserves internal buffer space large enough for block data and CRC <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Use of Block CRC (<code>NvMBlockUseCrc</code>): this must be enabled to use CRC for a block and therefore also for configuring the CRC storage in user space. ▶ RAM Block Data Address (<code>NvMRamBlockDataAddress</code>): this feature is only supported for blocks with a permanent RAM block address configured.

	<p>Note: the NvM creates an internal buffer which is large enough to store the largest block for which this parameter is set to false. This includes 1-4 bytes for the CRC checksum as needed. Therefore enabling <code>NvMUserProvidesSpaceForBlockAndCrc</code> for large blocks saves RAM as the the size of the internal buffer can be reduced to the size needed for the smaller blocks.</p> <p>Note: if <code>NvMUserProvidesSpaceForBlockAndCrc</code> is set to <code>true</code>, the application <i>must</i> ensure that the RAM block data is followed by reserved bytes of the size needed to store the associated CRC checksum. Otherwise, NvM will corrupt the application's data memory.</p> <p>To write the CRC, the NvM will use the RAM in the address range:</p> <ul style="list-style-type: none"> ▶ <code>NvMRamBlockDataAddress+NvMNvBlockLength</code> to <code>NvMRamBlockDataAddress+NvMNvBlockLength+0</code> to store a CRC8 value ▶ <code>NvMRamBlockDataAddress+NvMNvBlockLength</code> to <code>NvMRamBlockDataAddress+NvMNvBlockLength+1</code> to store a CRC16 value ▶ <code>NvMRamBlockDataAddress+NvMNvBlockLength</code> to <code>NvMRamBlockDataAddress+NvMNvBlockLength+3</code> to store a CRC32 value <p>Note: the application shall only provide this memory space. It <i>shall not</i> access the checksum bytes directly.</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMEnBlockCheck	
Label	Enable Background Block Check	
Description	Enables the Block Check mechanism for this block.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMEnableBlockCryptoSecurityHandling	
Label	Enable Crypto Security Handling	

Description	Enables crypto handling of NvM user data corresponding to this block using the Configured callback functions : 'NvMCryptoReadHook' and 'NvMCryptoWriteHook'.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMCryptoExtraInfoSize	
Label	Configure Block Crypto Extra Info Size	
Description	<p>Configuration of the particular's NvM Block Crypto Extra Info Size.</p> <p>The paramter shall be considered when calculating the actual block size when calling lower layers.</p> <p>This parameter means the NvM blocks will contain additional information needed for the Crypto Security Handling.</p> <p>This information can be either a MAC, a Hash or a Crypto Initialization Vector plus a MAC.</p>	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<div><=255</div> <div>>=0</div>	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMBcEnSetAPI	
Label	Enable usage of Set API for Block Check	
Description	<p>Defines if this user can control the Block Check mechanism through the NvM_ - EnableBc() API.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► Enable Block Check (NvMEnBlockCheck): this must be set to <code>true</code> 	
Multiplicity	1..1	

Type	BOOLEAN
Default value	true
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMBcEnAutoStart
Label	Enable the automatic start of Block Check mechanism.
Description	<p>Defines the automatic start of Block Check mechanism. If disabled then the BC mechanism has to be started using the set API <code>NvM_EnableBc()</code>.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Block Check (<code>NvMEnBlockCheck</code>): this must be set to <code>true</code>
Multiplicity	1..1
Type	BOOLEAN
Default value	true
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMBcEnCrcComp
Label	Enable CRC Comparison during Block Check
Description	<p>Defines if the CRC comparison will be done during a Block Check processing for the current block.</p> <p>The comparison will be done between the CRC done on the NV data and the CRC stored in NV.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Block Check (<code>NvMEnBlockCheck</code>): this must be set to <code>true</code> ▶ Enable Crc Storage (<code>NvMBlockUseCrc</code>): this must be set to <code>true</code>
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMBcEnRamComp
-----------------------	-----------------------

Label	Enable RAM Comparison during Block Check	
Description	<p>Defines if the RAM comparison will be done during a Block Check processing for the current block.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Block Check (NvMEnBlockCheck): this must be set to <code>true</code> ▶ Redundant Copies Comparison Disabled (NvMbCEnReddCopiesComp): this must be set to <code>false</code> ▶ Parameter NvMRamBlockDataAddress must be enabled and the field must not be empty. ▶ Parameter NvMBlockManagementType must not be set to <code>NVM_BLOCK_DATASET</code>. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMbCEnReddCopiesComp	
Label	Enable Redundant Copies Comparison during Block Check	
Description	<p>Defines if the data comparison between the copies of a redundant block will be done during a Block Check processing.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Block Check (NvMEnBlockCheck): this must be set to <code>true</code> ▶ RAM Comparison Disabled (NvMbCEnRamComp): this must be set to <code>false</code> ▶ Parameter (NvMBlockManagementType) must be set to <code>NVM_BLOCK_REDUNDANT</code> 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMbCEnAutoRepair	
Label	Enable Auto Repair feature	

Description	<p>Defines if the AutoRepair feature is enabled. If this is enabled and the block check verification fails then the NV memory will be restored either from RAM memory if configured as permanent and valid, either from ROM defaults if configured.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Block Check (NvMEnBlockCheck): this must be set to <code>true</code> ▶ Permanent RAM is present (NvMRamBlockDataAddress): this must be set ▶ or ROM defaults are present (NvMRomBlockDataAddress): this must be set ▶ or ROM defaults are present (NvMRomBlockDataAddress): this must be set 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMBcDelayCounter	
Label	Minimum delay between two consecutive Block Checks	
Description	<p>Defines the minimum number of NvM Main functions between the trigger of a Block Check mechanism for a NV Block.</p> <p>Value "0" means the Block Check will be done only once.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 0 .. 255 	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<div><=255</div> <div>>=0</div>	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMWriteBlockOnce
-----------------------	--------------------------

Label	Write Block Once
Description	<p>Enables a mechanism to write the NV block only once. The write protection of the block is automatically enabled after the block is written the first time.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = write protection after first write is set ▶ <code>false</code> = write protection after first write is not set <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Use of Block CRC (<code>NvMBlockUseCrc</code>): This must be enabled to let the NVRAM manager recalculate the CRC during startup. ▶ Initial Write Protection (<code>NvMBlockWriteProt</code>): This must be disabled to let the user write to the NVRAM block in case the CRC check fails.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMWriteRamBlockToNvCallback
Label	RAM to NvM Copy Call-back Function
Description	<p>Defines the name of a block specific call-back function which will be called by the NvM in order to let the application copy data from this block's RAM block to NvM module's mirror.</p> <p>The call-back function shall have the following prototype:</p> <ul style="list-style-type: none"> ▶ <code>Std_ReturnType NvM_WriteRamBlockToNvM(void* NvMBuffer)</code> <p>The possible return values are:</p> <ul style="list-style-type: none"> ▶ <code>E_OK</code> : copy was successful ▶ <code>E_NOT_OK</code> indicates copy was not successful, call-back routine to be called again <p>Options:</p> <ul style="list-style-type: none"> ▶ name of call-back function: this function will be called to copy data from the RAM block to NvM module's mirror ▶ empty: no function will be called <p>Dependency on parameter(s):</p>

	<ul style="list-style-type: none"> ▶ Number of Repetitions for Mirror Operations (<code>NvMRepeatMirrorOperation</code>): number of times this call-back will be called if <code>E_NOT_OK</code> is returned. ▶ Enable Explicit Synchronization Mechanism (<code>NvMBlockUseSyncMechanism</code>): This must be enabled to call the RAM to NvM Copy Call-back Function.
Multiplicity	0..1
Type	FUNCTION-NAME
Configuration class	PreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMWriteVerification
Label	Enable Write Verification
Description	<p>Enables write verification for this block.</p> <p>This defines whether or not the NV block is immediately read back and compared with the original content of the RAM Block each time the RAM Block is written to NV memory.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = NV block is read back and compared with the RAM Block ▶ <code>false</code> = NV block is not compared with the RAM Block
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMWriteVerificationDataSize
Label	Write Verification Data Size
Description	<p>Defines the number of bytes to be used in each step of write block verification. The verification takes place after writing the block to NV memory.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Write Verification (<code>NvMWriteVerification</code>): this must be set to <code>true</code>
Multiplicity	1..1
Type	INTEGER

Default value	1	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMPreWriteDataComp	
Label	Enable Data Comparison Mechanism.	
Description	<p>Enables pre write data comparison for this block.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Before Writing into NV, the NV block is read back and compared with the RAM Block ▶ <code>false</code> = Before Writing into NV, the NV block is not compared with the RAM Block <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ If <code>NvMBlockUseCrc</code> is enabled and <code>NvMBlockUseCRCCompMechanism</code> is also enabled, Data comparison is done only if CRC comparison is ok. 	
Multiplicity	0..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMPreWriteDataCompDataSize	
Label	Pre Write Data Comparison Data Size	
Description	<p>Defines the number of bytes to be used in each step of pre write block comparison. The comparison takes place before writing the block to NV memory.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Pre Write Data Comparison (<code>NvMPreWriteDataComp</code>): this must be set to <code>true</code> 	
Multiplicity	1..1	
Type	INTEGER	
Default value	1	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.6.1.3. NvMTargetBlockReference

Containers included		
Container name	Multiplicity	Description
NvMEaRef	1..1	Label: Ea Target Block Reference EEPROM Abstraction
NvMFeeRef	1..1	Label: Fee Target Block Reference Flash EEPROM Emulation

5.6.1.4. NvMEaRef

Parameters included	
Parameter name	Multiplicity
NvMNameOfEaBlock	1..1

Parameter Name	NvMNameOfEaBlock	
Label	Name of Ea Block	
Description	Reference to the EA block that is associated with this NV block.	
Multiplicity	1..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.6.1.5. NvMFeeRef

Parameters included	
Parameter name	Multiplicity
NvMNameOfFeeBlock	1..1

Parameter Name	NvMNameOfFeeBlock	
Label	Name of Fee Block	
Description	Reference to the FEE block that is associated with this NV block.	

Multiplicity	1..1
Type	SYMBOLIC-NAME-REFERENCE
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

5.6.1.6. NvMDefensiveProgramming

Parameters included	
Parameter name	Multiplicity
NvMDefProgEnabled	1..1
NvMPrecondAssertEnabled	1..1
NvMPostcondAssertEnabled	1..1
NvMStaticAssertEnabled	1..1
NvMUnreachAssertEnabled	1..1
NvMInvariantAssertEnabled	1..1

Parameter Name	NvMDefProgEnabled
Label	Enable Defensive Programming
Description	<p>Enables or disables the defensive programming feature for the module NvM.</p> <p>Note: This feature is dependent on the use of the development error detection module. To use the defensive programming feature, proceed as follows:</p> <ol style="list-style-type: none"> 1. Enable development error detection 2. Enable defensive programming 3. Enable assertions as required
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMPrecondAssertEnabled
Label	Enable Precondition Assertions

Description	<p>Enables handling of precondition assertion checks reported from the module NvM.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (NvMDevErrorDetect): must be enabled ▶ Enable Defensive Programming (NvMDefProgEnabled): must be enabled 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMPostcondAssertEnabled	
Label	Enable Postcondition Assertions	
Description	<p>Enables handling of postcondition assertion checks reported from the module NvM.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (NvMDevErrorDetect): must be enabled ▶ Enable Defensive Programming (NvMDefProgEnabled): must be enabled 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMStaticAssertEnabled	
Label	Enable Static Assertions	
Description	<p>Enables handling of static assertion checks reported from the module NvM.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (NvMDevErrorDetect): must be enabled ▶ Enable Defensive Programming (NvMDefProgEnabled): must be enabled 	

Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMUnreachAssertEnabled
Label	Enable Unreachable Code Assertions
Description	<p>Enables handling of unreachable code assertion checks reported from the module NvM.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (NvMDevErrorDetect): must be enabled ▶ Enable Defensive Programming (NvMDefProgEnabled): must be enabled
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMInvariantAssertEnabled
Label	Enable Invariant Assertions
Description	<p>Enables handling of invariant assertion checks reported from functions of the module NvM.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection (NvMDevErrorDetect): must be enabled ▶ Enable Defensive Programming (NvMDefProgEnabled): must be enabled
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

5.6.1.7. NvMCommon

Containers included		
Container name	Multiplicity	Description
NvMCommonCryptoSecurity-Parameters	1..1	Label: Common Crypto Security Parameters Defines the configuration of the cryptography parameters of NvM.
NvMServiceAPI	1..1	Label: Service API Parameters Defines the configuration of the service API of NvM.

Parameters included	
Parameter name	Multiplicity
NvMApiConfigClass	1..1
NvMBswMMultiBlockJobStatusInformation	1..1
NvMCompiledConfigId	1..1
NvMSoftwareChangeCallout	0..1
NvMCrcNumOfBytes	1..1
NvMDatasetSelectionBits	1..1
NvMDevErrorDetect	1..1
NvMDrvModeSwitch	1..1
NvMDynamicConfiguration	1..1
NvMCancelInternalOperations	1..1
NvMJobPrioritization	1..1
NvMMainFunctionCycleTime	1..1
NvMMultiBlockCallback	0..1
NvMPollingMode	1..1
NvMReadBlockHook	1..1
NvMRepeatMirrorOperations	1..1
NvMRteUsage	1..1
NvMSetRamBlockStatusApi	1..1
NvMSizeImmediateJobQueue	0..1
NvMSizeStandardJobQueue	1..1
NvMUserHeader	0..255
NvMVersionInfoApi	1..1

Parameters included	
NvMWriteBlockHook	1..1
NvMRedundantRecovery	1..1
NvMExportBlockLengths	1..1
NvMResultErasedBlocks	1..1
NvMEnableLegacySymbolicNames	1..1
NvMResetRamBlockAfterReset	1..1

Parameter Name	NvMApiConfigClass	
Label	API Class Configuration	
Description	<p>Enables specific API calls depending on the NvM API configuration class selected.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ NVM_API_CONFIG_CLASS_3 - all API functions are available ▶ NVM_API_CONFIG_CLASS_2 - an intermediate set API functions is available ▶ NVM_API_CONFIG_CLASS_1 - a minimum set of API functions is available 	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	NVM_API_CONFIG_CLASS_3	
Range	NVM_API_CONFIG_CLASS_1	
	NVM_API_CONFIG_CLASS_2	
	NVM_API_CONFIG_CLASS_3	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMBswMMultiBlockJobStatusInformation	
Label	Enable BSWM Multi Block Job Status Information	
Description	<p>Enables BSWM Multi Block Job Status Information.</p> <p>This parameter specifies whether BswM is informed about the current status of the multiblock job.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Report current status of multiblock job to BswM ▶ <code>false</code> = Do not inform BswM at all 	

	Dependency on parameter(s):	
	<ul style="list-style-type: none"> ► Enable <code>BswMNVmEnabled</code>: this must be selected to enable the NVm module related BswM API. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMCompiledConfigId	
Label	Compiled Configuration ID	
Description	<p>Defines the configuration ID regarding the NV memory layout.</p> <p>This configuration ID shall be published as e.g. a SW-C shall have the possibility to write it to NV memory.</p>	
Multiplicity	1..1	
Type	INTEGER	
Default value	1	
Range	<p>≤ 9223372036854775807</p> <p>≥ 0</p>	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMSoftwareChangeCallout	
Label	Software Change Callout	
Description	<p>Defines the name of a callout function which Nvm shall call in order to obtaining the new Configuration Id value that shall be compared to the content of Block1 for detecting software change.</p> <p>Options:</p> <ul style="list-style-type: none"> ► name of call-back function: this function will be called to get the new Configuration Id ► empty: no function will be called; Block 1 content will be compared with the compiled Configuration Id ► if disabled: no function will be called; Block 1 content will be compared with the compiled Configuration Id 	

Multiplicity	0..1
Type	FUNCTION-NAME
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMCrcNumOfBytes
Label	Max Number of Bytes in one CRC Calculation Cycle
Description	<p>Defines the maximum number of bytes which shall be processed within one cycle of job processing.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► Usage of Block CRC (NvMBlockUseCrc): this must be enabled for at least one block.
Multiplicity	1..1
Type	INTEGER
Default value	65535
Range	<p><=65535</p> <p>>=1</p>
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMDatasetSelectionBits
Label	Number of Data Selection Bits
Description	<p>Defines the number of least significant bits which shall be used to address a certain dataset of a NVRAM block within the interface to the memory hardware abstraction.</p> <p>0..8: Number of bits which are used for dataset or redundant block addressing.</p> <p>Range:</p> <ul style="list-style-type: none"> ► 0: No dataset or redundant NVRAM blocks are configured at all, no selection bits required. ► 1: In case of redundant NVRAM blocks are configured, but no dataset NVRAM blocks.
Multiplicity	1..1
Type	INTEGER

Default value	4
Range	<=8 >=0
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMDevErrorDetect
Label	Enable Development Error Detection
Description	Enables the use of development error detection. <ul style="list-style-type: none"> ▶ true = Development error detection enabled ▶ false = Development error detection disabled
Multiplicity	1..1
Type	BOOLEAN
Default value	true
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMDrvModeSwitch
Label	Enable Driver Mode Switch
Description	Enables the switching of memory drivers to fast mode during the requests <code>NvM_ReadAll</code> and <code>NvM_WriteAll</code> . <ul style="list-style-type: none"> ▶ true = mode switching enabled ▶ false = mode switching disabled <p>Note: Driver mode switching must be supported by the Nv memory drivers Eep or Fls. The value of <code>NvMDrvModeSwitch</code> configuration parameter must be consistent with the configuration of all underlying modules:</p> <ul style="list-style-type: none"> ▶ <code>EaSetModeSupported</code> for EEPROM devices ▶ <code>FeeSetModeSupported</code> and <code>FlsSetModeApi</code> for flash devices
Multiplicity	1..1
Type	BOOLEAN
Default value	true
Configuration class	VariantPreCompile: VariantPreCompile

Origin	AUTOSAR_ECUC
--------	--------------

Parameter Name	NvMDynamicConfiguration	
Label	Enable Dynamic Configuration	
Description	<p>Enables the dynamic configuration management handling for processing during the <code>NvM_ReadAll()</code> request</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Dynamic configuration management handling enabled ▶ <code>false</code> = Dynamic configuration management handling disabled 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMCancelInternalOperations	
Label	Enable Cancel Internal Operations	
Description	<p>Enables the option to cancel internal operations in the underlying modules for all devices which have the status different then <code>MEMIF_IDLE</code>.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Cancel internal operations enabled. ▶ <code>false</code> = Cancel internal operations disabled. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMJobPrioritization	
Label	Enable Job Prioritization	
Description	<p>Enables job prioritization handling</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Job prioritization handling enabled ▶ <code>false</code> = Job prioritization handling disabled 	
Multiplicity	1..1	
Type	BOOLEAN	

Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMMainFunctionCycleTime	
Label	NvM Main Function Cycle Time	
Description	Defines the periodic cycle time, in seconds, for calling NvM main function.	
Multiplicity	1..1	
Type	FLOAT	
Default value	0.01	
Range	<=255	
	>=0.001	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMMultiBlockCallback	
Label	Multiblock Request Call-back Function	
Description	<p>Defines the start address of a common call-back function that shall be called on termination of each asynchronous multi block request.</p> <p>Note: This call-back function is called inside a critical section. Hence, application must ensure that the run time of this function is reasonably short.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► This parameter shall be editable only if NvM/NvMCommon/BswMMulti-BlockJobStatusInformation is set to FALSE 	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Default value	EcuM_CB_NfyNvMJobEnd	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMPollingMode	
Label	Enable Polling Mode	
Description	Enables the polling mode in the NVRAM Manager and at the same time disable/enable the call-back functions usable by lower layers	

	<ul style="list-style-type: none"> ▶ <code>true</code> = Polling mode enabled, call-back function usage disabled ▶ <code>false</code> = Polling mode disabled, call-back function usage enabled <p>Note: The current implementation of the NvM supports polling mode only. This parameter setting is ignored.</p>
Multiplicity	1..1
Type	BOOLEAN
Default value	true
Configuration class	VariantPreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NvMReadBlockHook
Label	Enable Read Block Hook
Description	<p>Enables the use of a hook function for <code>NvM_ReadBlock()</code> / <code>NvM_ReadAll()</code>.</p> <p>This hook function, to be implemented by the user, will be called by the NvM at the end of a request to read a block i.e. when the read job has completed.</p> <p>The read hook feature is supported only for blocks configured with a permanent RAM.</p> <p>Only the data part (no header, no CRC) of the block can be updated by the read block hook function. For blocks configured with a CRC, the CRC value will be calculated over the changed data.</p>
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMRepeatMirrorOperations
Label	Number of Repetitions for Mirror Operations
Description	<p>Defines the number of retries to let the application copy data to or from the NvM module's mirror before postponing the current job.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 0 .. 7 <p>Dependency on parameter(s):</p>

	► Enable Explicit Synchronization Mechanism (<code>NvMBlockUseSyncMechanism</code>): This must be enabled for at least one NVM block.	
Multiplicity	1..1	
Type	INTEGER	
Default value	0	
Range	<=7	
	>=0	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMRteUsage	
Label	Enable Rte Usage	
Description	<p>Enables the use of the RTE for this module.</p> <p>► <code>true</code> = Rte enabled.</p> <p>► <code>false</code> = Rte disabled.</p> <p>Note: For easy integration, it is recommended that this parameter be disabled at the beginning of the integration work.</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMSetRamBlockStatusApi	
Label	Enable Set RAM Block Status API	
Description	<p>Enables the <code>NvM_SetRamBlockStatus()</code> API function.</p> <p>► <code>true</code> = <code>NvM_SetRamBlockStatus()</code> enabled</p> <p>► <code>false</code> = <code>NvM_SetRamBlockStatus()</code> disabled</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMSizeImmediateJobQueue	
Label	Size of Queue for Immediate Requests	
Description	<p>Defines the number of entries for the immediate priority job queue.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 1 .. 65535 <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Job Prioritization (NvMJobPrioritization): must be enabled. 	
Multiplicity	0..1	
Type	INTEGER	
Default value	1	
Range	<=65535	
	>=1	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMSizeStandardJobQueue	
Label	Size of Queue for Standard Requests	
Description	<p>Defines the number of entries for the standard priority job queue.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 1 .. 65535 	
Multiplicity	1..1	
Type	INTEGER	
Default value	3	
Range	<=65535	
	>=1	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMUserHeader	
Label	User Header File	
Description	<p>Defines user specific header files for user-defined symbols</p> <p>.</p>	

	<p>Note: <code>NvMUserHeader</code> shall be configured in the following scenarios.</p> <ul style="list-style-type: none"> ▶ If a permanent RAM block is configured with parameter <code>NvMRamBlock-DataAddress</code>, user header must be set to a valid header file including the declaration of the RAM block location. ▶ If a permanent ROM block is configured with parameter <code>NvMRomBlock-DataAddress</code> user header must be set to a valid header file including the declaration of the ROM block location. ▶ If a multi block call-back function is configured using the parameter <code>NvM-MultiBlockCallback</code>, user header must be set to a valid header file including the declaration of the call-back notification function. ▶ If a single block call-back function is configured using the parameter <code>NvMSingleBlockCallback</code>, user header must be set to a valid header file including the declaration of the call-back notification function. ▶ If call-back functions are configured using the parameters <code>NvMReadRam-BlockFromNvCallback/NvMWriteRamBlockToNvCallback</code>, user header must be set to a valid header file including the prototype of the mirror call-back functions. 	
Multiplicity	0..255	
Type	STRING	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMVersionInfoApi	
Label	Enable Version Info API	
Description	<p>Enables the <code>NvM_GetVersionInfo()</code> API function.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = <code>NvM_GetVersionInfo()</code> enabled ▶ <code>false</code> = <code>NvM_GetVersionInfo()</code> disabled 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMWriteBlockHook	
Label	Enable Write Block Hook	

Description	<p>Enables the use of a hook function for <code>NvM_WriteBlock()</code> / <code>NvM_WriteAll()</code>.</p> <p>Note: The write hook feature is supported only for blocks configured with a permanent RAM.</p> <p>This hook function, to be implemented by the user, will be called by the NvM at the start of a request to write a block i.e. before the write job is started.</p> <p>Only the data part (no header, no CRC) of the block can be updated by the write block hook function. For blocks configured with a CRC, the CRC value will be calculated over the changed data</p>	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMRedundantRecovery
Label	Recovery of Redundant Blocks
Description	<p>Defines the behavior of NvM in case loss of recovery is detected for a redundant block.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ <code>NVM_RECOVERY_ON_REQUEST</code> ▶ <code>NVM_AUTOMATIC_RECOVERY</code> <p>Note: The user has the possibility to configure the reaction in case the redundancy of a redundant block is lost:</p> <ul style="list-style-type: none"> ▶ <code>NVM_RECOVERY_ON_REQUEST</code> - when loss of redundancy is detected NvM will report <code>NVM_REQ_REDUNDANCY_FAILED</code> as result of the request. In this way the user is notified and can restore the redundant block immediately by triggering a <code>NvM_WriteBlock</code> request. If no write request is triggered the block will be automatically restored during <code>NvM_WriteAll</code>. ▶ <code>NVM_AUTOMATIC_RECOVERY</code> - loss of redundancy is transparent to the user. In case at least one copy of a redundant block is successfully read NvM will report <code>NVM_REQ_OK</code> as result of the operation. The block will be restored during shut-down (<code>NvM_WriteAll</code>).
Multiplicity	1..1

Type	ENUMERATION
Default value	NVM_AUTOMATIC_RECOVERY
Range	NVM_RECOVERY_ON_REQUEST NVM_AUTOMATIC_RECOVERY
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMExportBlockLengths
Label	Export block lengths
Description	<p>Enables or disables the exporting of block lengths for all configured NVRAM blocks to the NvM user.</p> <ul style="list-style-type: none"> ▶ <code>true</code> = NvM shall export block lengths ▶ <code>false</code> = NvM shall not export block lengths <p>Note: The value of the exported block lengths correspond to the value of the <code>NvMNvBlockLength</code> configuration parameter.</p> <p>The exported block lengths have the following prototype:</p> <ul style="list-style-type: none"> ▶ <code>NvMConf_BlockName_Length</code>
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMResultErasedBlocks
Label	Reported value for erased blocks
Description	<p>Defines the reported value of the lower layer in case of erased blocks or blocks not found in the memory.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ <code>MEMIF_BLOCK_INCONSISTENT</code>: lower layer reports block as inconsistent. ▶ <code>MEMIF_BLOCK_INVALID</code>: lower layer reports block as invalid.
Multiplicity	1..1
Type	ENUMERATION

Default value	MEMIF_BLOCK_INCONSISTENT
Range	MEMIF_BLOCK_INCONSISTENT
	MEMIF_BLOCK_INVALID
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMEnableLegacySymbolicNames	
Label	Enable Legacy Symbolic Name	
Description	Enables or disables exporting legacy symbolic names for AUTOSAR version older than 3.1 rev4 . That is, if enabled it exports the NvM block names without NvM_ prefix according to AUTOSAR 3.X	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	true	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMResetRamBlockAfterReset	
Label	Reset Ram Block after Reset	
Description	Reset the validity of Ram blocks that does not have CRC configured after a warm reset	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.6.1.8. NvMCommonCryptoSecurityParameters

Parameters included	
Parameter name	Multiplicity
NvMEnableCryptoSecurityHooks	1..1
NvMCryptoReadHook	1..1
NvMCryptoWriteHook	1..1

Parameter Name	NvMEnableCryptoSecurityHooks	
Label	Enable Read/Write Crypto Security Callbacks	
Description	<p>Enables the usage of crypto security hooks for handling of NvM Blocks's data</p> <ul style="list-style-type: none"> ▶ <code>true</code> = Enables Hooks Configuration. ▶ <code>false</code> = Disables Hooks Configuration. 	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMCryptoReadHook
Label	Configure Crypto Read Hook
Description	<p>Configures the callback API to apply the corresponding crypto algorithm for a NvM Block after Read.</p> <p>Function prototype : Parameters : NvM_ASR40_BlockIdType NvMBlockId, uint8 *pBuffer, uint16 UserDataSize, uint8 CryptoExtraInfoSize, boolean FirstCall</p> <p>Return : MemIf_JobResultType</p> <p>Sync/Async : Async</p> <p>Reentrancy : Non Reentrant</p> <p>Description : The function will take the data from "pBuffer", will process it according to the crypto algorithm and it and shall copy it back into "pBuffer". Data is considered not only user data but the entire NvM block data, comprised, depending on the configuration, by Static Block ID, User Data and CRC.</p> <p>The API shall be called with FirstCall set to TRUE only the first time is triggered for a block. If the API returns MEMIF_JOB_PENDING then the subsequent calls will be done with FirstCall set to FALSE and the call will function as a GetJobResult. The API will be called each NvM Main cycle with FirstCall set to FALSE until the function</p>

	<p>returns a value different then MEMIF_JOB_PENDING.</p> <p>The function shall return MEMIF_JOB_OK if the crypto processing was succesfull.</p> <p>The function shall return MEMIF_JOB_PENDING if the crypto processing could not be completed syncrounous and</p> <p>will be pending to be completed in the next main cycle.</p> <p>The function shall return MEMIF_JOB_FAILED if the crypto processing was not succesfull.</p> <p>Any other value returned shall be considered as MEMIF_JOB_FAILED.</p>	
Multiplicity	1..1	
Type	FUNCTION-NAME	
Default value		
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	
Parameter Name	NvMCryptoWriteHook	
Label	Configure Crypto Write Hook	
Description	<p>Configures the callback API to apply the corresponding crypto algorithm for a NvM Block before Write.</p> <p>Function prototype : Parameters : NvM_ASR40_BlockIdType NvMBlockId, uint8 *pBuffer, uint16 UserDataSize, uint8 CryptoExtraInfoSize, boolean FirstCall</p> <p>Return : MemIf_JobResultType</p> <p>Sync/Async : Async</p> <p>Reentrancy : Non Reentrant</p> <p>Description : The function will take the plain data from "pBuffer", will crypto process it and shall copy</p> <p>it back into "pBuffer". Data is considered not only user data but the entire NvM block data,</p> <p>comprised, depending on the configuration, by Static Block ID, User Data and CRC.</p> <p>The API shall be called with FirstCall set to TRUE only the first time is triggered for a block. If the API returns</p>	

	<p>MEMIF_JOB_PENDING then the subsequent calls will be done with FirstCall set to FALSE and the call will</p> <p>function as a GetJobResult. The API will be called each NvM Main cycle with FirstCall set to FALSE until the function</p> <p>returns a value different than MEMIF_JOB_PENDING.</p> <p>The function shall return MEMIF_JOB_OK if the crypto processing was successful.</p> <p>The function shall return MEMIF_JOB_PENDING if the crypto processing could not be completed synchronously and</p> <p>will be pending to be completed in the next main cycle.</p> <p>The function shall return MEMIF_JOB_FAILED if the crypto processing was not successful.</p> <p>Any other value returned shall be considered as MEMIF_JOB_FAILED.</p>	
Multiplicity	1..1	
Type	FUNCTION-NAME	
Default value		
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.6.1.9. NvMServiceAPI

Parameters included	
Parameter name	Multiplicity
NvMEnableASR32ServiceAPI	1..1
NvMEnableASR40ServiceAPI	1..1
NvMEnableASR42ServiceAPI	1..1
NvMDefaultASRServiceAPI	1..1

Parameter Name	NvMEnableASR32ServiceAPI
Label	Enable AUTOSAR 3.2 service API
Description	<p>Enables usage of the AUTOSAR 3.2 service API.</p> <p>► true = Enables AUTOSAR 3.2 service API.</p>

	► <code>false</code> = Disables AUTOSAR 3.2 service API.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMEnableASR40ServiceAPI
Label	Enable AUTOSAR 4.0 service API
Description	Enables usage of the AUTOSAR 4.0 service API. ► <code>true</code> = Enables AUTOSAR 4.0 service API. ► <code>false</code> = Disables AUTOSAR 4.0 service API.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMEnableASR42ServiceAPI
Label	Enable AUTOSAR 4.2 service API
Description	Enables usage of the AUTOSAR 4.2 service API. ► <code>true</code> = Enables AUTOSAR 4.2 service API. ► <code>false</code> = Disables AUTOSAR 4.2 service API.
Multiplicity	1..1
Type	BOOLEAN
Default value	false
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMDefaultASRServiceAPI
Label	Default AUTOSAR service API
Description	Defines the default AUTOSAR service API. Range:

	<ul style="list-style-type: none"> ▶ AUTOSAR_32 = AUTOSAR 3.2 service API is the default one. ▶ AUTOSAR_40 = AUTOSAR 4.0 service API is the default one. ▶ AUTOSAR_42 = AUTOSAR 4.2 service API is the default one. ▶ NONE = No default AUTOSAR service API is provided. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ If NvMDefaultASRServiceAPI is selected as NONE any of the service API parameters NvMEnableASR40ServiceAPI, NvMEnableASR32ServiceAPI or NvMEnableASR42ServiceAPI needs to be selected. 	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	AUTOSAR_40	
Range	AUTOSAR_32	
	AUTOSAR_40	
	AUTOSAR_42	
	NONE	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

5.6.1.10. NvmDemEventParameterRefs

Parameters included	
Parameter name	Multiplicity
NVM_E_INTEGRITY_FAILED	0..1
NVM_E_LOSS_OF_REDUNDANCY	0..1
NVM_E_QUEUE_OVERFLOW	0..1
NVM_E_REQ_FAILED	0..1
NVM_E_VERIFY_FAILED	0..1
NVM_E_WRITE_PROTECTED	0..1
NVM_E_WRONG_BLOCK_ID	0..1
NVM_E_BLOCK_CHECK	0..1

Parameter Name	NVM_E_INTEGRITY_FAILED
-----------------------	-------------------------------

Label	Integrity Failed Event Parameter	
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error <code>NVM_E_INTEGRITY_FAILED</code> has occurred.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Request Integrity Failure (<code>NvMIntegrityFailedReportToDem</code>): Select DEM to enable <code>NVM_E_INTEGRITY_FAILED</code>. <p>Further notes:</p> <ul style="list-style-type: none"> ▶ Activation: Thrown, if a CRC mismatch occurs for a block configured with CRC or <code>MEMIF_BLOCK_INCONSISTENT</code> is reported by the MemIf module during an attempt to read a NV block. ▶ Healing: None. The error resides in memory until it is deleted. ▶ Trigger debounce: None. The error is reported on first occurrence. ▶ Rate of diagnostic checks: Checked on every call of the service that reports this error. A list of API functions that report this error can be found in the table of production errors in the <i>Integration notes</i> section of the module references. 	
Multiplicity	0..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NVM_E_LOSS_OF_REDUNDANCY	
Label	Loss of Redundancy Event Parameter	
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error <code>NVM_E_LOSS_OF_REDUNDANCY</code> has occurred.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Loss of Redundancy (<code>NvMLossOfRedundancyReportToDem</code>): Select DEM to enable <code>NVM_E_LOSS_OF_REDUNDANCY</code>. <p>Further notes:</p> <ul style="list-style-type: none"> ▶ Activation: Thrown, if a redundant NV block is detected as damaged (the second copy is different than the first one), during a read request. ▶ Healing: None. The error resides in memory until it is deleted. ▶ Trigger debounce: None. The error is reported on first occurrence. 	

	<ul style="list-style-type: none"> ▶ Rate of diagnostic checks: Checked on every call of the service that reports this error. A list of API functions that report this error can be found in the table of production errors in the <i>Integration notes</i> section of the module references.
Multiplicity	0..1
Type	SYMBOLIC-NAME-REFERENCE
Configuration class	PreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NVM_E_QUEUE_OVERFLOW
Label	Queue Overflow Event Parameter
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error <code>NVM_E_QUEUE_OVERFLOW</code> has occurred.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Request Queue Overflow (<code>NvMQueueOverflowReportToDem</code>): Select DEM to enable <code>NVM_E_QUEUE_OVERFLOW</code>. <p>Further notes:</p> <ul style="list-style-type: none"> ▶ Activation: Thrown, if a new asynchronous NvM request cannot be processed because the NvM queue is full. ▶ Healing: None. The error resides in memory until it is deleted. ▶ Trigger debounce: None. The error is reported on first occurrence. ▶ Rate of diagnostic checks: Checked on every call of the service that reports this error. A list of API functions that report this error can be found in the table of production errors in the <i>Integration notes</i> section of the module references.
Multiplicity	0..1
Type	SYMBOLIC-NAME-REFERENCE
Configuration class	PreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NVM_E_REQ_FAILED
Label	Request Failed Event Parameter
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error <code>NVM_E_REQ_FAILED</code> has occurred.</p>

	<p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ NvM Request Failure (<code>NvMRequestFailedReportToDem</code>): Select DEM to enable <code>NVM_E_REQ_FAILED</code>. <p>Further notes:</p> <ul style="list-style-type: none"> ▶ Activation: Thrown, if a single block request fails or the lower layer module reports failure. ▶ Healing: None. The error resides in memory until it is deleted. ▶ Trigger debounce: <code>NVM_E_REQ_FAILED</code> shall be thrown only after a maximum number of retries (<code>NvMMaxNumOfReadRetries</code>, <code>NvMMaxNumOfWriteRetries</code>) is exceeded for requests configured with a retry counter. ▶ Rate of diagnostic checks: Checked on every call of the service that reports this error. A list of API functions that report this error can be found in the table of production errors in the <i>Integration notes</i> section of the module references.
Multiplicity	0..1
Type	SYMBOLIC-NAME-REFERENCE
Configuration class	PreCompile: VariantPreCompile
Origin	AUTOSAR_ECUC

Parameter Name	NVM_E_VERIFY_FAILED
Label	Verification Failed Event Parameter
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error <code>NVM_E_VERIFY_FAILED</code> has occurred.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Request Verify Failed (<code>NvMVerifyFailedReportToDem</code>): Select DEM to enable <code>NVM_E_VERIFY_FAILED</code>. <p>Further notes:</p> <ul style="list-style-type: none"> ▶ Activation: Thrown, if the content of the RAM block is not the same as the read back data. ▶ Healing: None. The error resides in memory until it is deleted. ▶ Trigger debounce: None. The error is reported on first occurrence. ▶ Rate of diagnostic checks: Checked on every call of the service that reports this error. A list of API functions that report this error can be found in the ta-

	ble of production errors in the <i>Integration notes</i> section of the module references.	
Multiplicity	0..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NVM_E_WRITE_PROTECTED	
Label	Block Write Protected Event Parameter	
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error <code>NVM_E_WRITE_PROTECTED</code> i.e. a write attempt to an NVRAM block with write protection has occurred.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Request Write Protected (<code>NvMWriteProtectedReportToDem</code>): Select DEM to enable <code>NVM_E_WRITE_PROTECTED</code>. <p>Further notes:</p> <ul style="list-style-type: none"> ▶ Activation: Thrown, if a write, erase or invalidate operation is requested for a write protected block. ▶ Healing: None. The error resides in memory until it is deleted. ▶ Trigger debounce: None. The error is reported on first occurrence. ▶ Rate of diagnostic checks: Checked on every call of the service that reports this error. A list of API functions that report this error can be found in the table of production errors in the <i>Integration notes</i> section of the module references. 	
Multiplicity	0..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NVM_E_WRONG_BLOCK_ID	
Label	Wrong Block Id Event Parameter	
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error <code>NVM_E_WRONG_BLOCK_ID</code> has occurred.</p> <p>Dependency on parameter(s):</p>	

	<p>► Wrong Block Id (NmWrongBlockIdReportToDem): Select DEM to enable NVM_E_WRONG_BLOCK_ID.</p> <p>Further notes:</p> <ul style="list-style-type: none"> ► Activation: Thrown, if the static block ID stored in the NV block header is different than the requested block ID. ► Healing: None. The error resides in memory until it is deleted. ► Trigger debounce: None. The error is reported on first occurrence. ► Rate of diagnostic checks: Checked on every call of the service that reports this error. A list of API functions that report this error can be found in the table of production errors in the <i>Integration notes</i> section of the module references. 	
Multiplicity	0..1	
Type	SYMBOLIC-NAME-REFERENCE	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NVM_E_BLOCK_CHECK	
Label	Block Check Event Parameter	
Description	<p>Reference to the <code>DemEventParameter</code> which shall be issued when the error NVM_E_BLOCK_CHECK has occurred.</p> <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► Wrong Block Id (NmBlockCheckReportProdError): Select DEM to enable NVM_E_BLOCK_CHECK. <p>Further notes:</p> <ul style="list-style-type: none"> ► Activation: Thrown, if the Block Check mechanism detects a corrupted block in NV memory. ► Healing: None. The error resides in memory until it is deleted. ► Trigger debounce: None. The error is reported on first occurrence. ► Rate of diagnostic checks: Checked on every call of the service that reports this error. A list of API functions that report this error can be found in the table of production errors in the <i>Integration notes</i> section of the module references. 	
Multiplicity	0..1	
Type	SYMBOLIC-NAME-REFERENCE	

Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.6.1.11. ReportToDem

Parameters included	
Parameter name	Multiplicity
NvMUserCalloutFunctionProductionErrors	0..1
NvMUserCalloutFunctionPassedProductionErrors	0..1
NvMIntegrityFailedReportToDem	1..1
NvMIntegrityFailedReportToDemDetErrorId	1..1
NvMRequestFailedReportToDem	1..1
NvMRequestFailedReportToDemDetErrorId	1..1
NvMWrongBlockIdReportToDem	1..1
NvMWrongBlockIdReportToDemDetErrorId	1..1
NvMLossOfRedundancyReportToDem	1..1
NvMLossOfRedundancyReportToDemDetErrorId	1..1
NvMQueueOverflowReportToDem	1..1
NvMQueueOverflowReportToDemDetErrorId	1..1
NvMVerifyFailedReportToDem	1..1
NvMVerifyFailedReportToDemDetErrorId	1..1
NvMWriteProtectedReportToDem	1..1
NvMWriteProtectedReportToDemDetErrorId	1..1
NvMBlockCheckReportProdError	1..1
NvMBlockCheckReportProdErrorId	1..1

Parameter Name	NvMUserCalloutFunctionProductionErrors
Label	User Callout Function for Failed Production Errors
Description	Defines the name of the user callout for production errors: NVM_E_USER_CALLOUT_LOSS_OF_REDUNDANCY, NVM_E_USER_CALLOUT_QUEUE_OVERFLOW, NVM_E_USER_CALLOUT_REQ_FAILED, NVM_E_USER_CALLOUT_VERIFY_FAILED, NVM_E_USER_CALLOUT_WRITE_PROTECTED, NVM_E_USER_CALLOUT_WRONG_BLOCK_ID, NVM_E_USER_CALLOUT_INTEGRITY_FAILED.

Multiplicity	0..1
Type	FUNCTION-NAME
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMUserCalloutFunctionPassedProductionErrors
Label	User Callout Function for Passed Production Errors
Description	Defines the name of the user callout for the case a production error has a passed status. Function prototype is " UserSWC_FunctionName(BlockId, Apild, ErrorId)"
Multiplicity	0..1
Type	FUNCTION-NAME
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMIntegrityFailedReportToDem
Label	Request Integrity Failure
Description	<p>Defines the handling of the production error: NVM_E_INTEGRITY_FAILED</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ DEM: Request Integrity Failure errors are reported to the Diagnostics Event Manager (Dem). ▶ DET: Request Integrity Failure errors are reported to the Development Error Tracer (Det) if enabled. ▶ UserCallout: Request Integrity Failure errors are reported with user configured callout. ▶ DISABLE: These errors are not reported at all. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Integrity Failed Event Parameter NVM_E_INTEGRITY_FAILED: this reference must be set if DEM is selected here. ▶ Enable Development Error Detection NvMDevErrorDetect: must be enabled if DET is selected here
Multiplicity	1..1
Type	ENUMERATION
Default value	DISABLE

Range	DEM
	DET
	UserCallout
	DISABLE
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMIntegrityFailedReportToDemDetErrorId
Label	Integrity Failure DemToDet Error Id
Description	<p>If the <code>NVM_E_INTEGRITY_FAILED</code> production error is reported to the Det, this parameter defines the unique error identifier which is reported there.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 25 .. 255, must be unique
Multiplicity	1..1
Type	INTEGER
Default value	25
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMRequestFailedReportToDem
Label	NvM Request Failure
Description	<p>Defines the handling of the production error: <code>NVM_E_REQ_FAILED</code></p> <p>Range:</p> <ul style="list-style-type: none"> ▶ DEM: NvM Request Failure errors are reported to the Diagnostics Event Manager (Dem). ▶ DET: NvM Request Failure errors are reported to the Development Error Tracer (Det) if enabled. ▶ UserCallout: NvM Request Failure errors are reported from user configured callout. ▶ DISABLE: These errors are not reported at all. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Request Failed Event Parameter <code>NVM_E_REQ_FAILED</code>: this reference must be set if DEM is selected here.

	► Enable Development Error Detection <code>NvMDevErrorDetect</code> : must be enabled if DET is selected here	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	DISABLE	
Range	DEM	
	DET	
	UserCallout	
	DISABLE	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMRequestFailedReportToDemDetErrorId	
Label	Request Failure DemToDet Error Id	
Description	<p>If the <code>NVM_E_REQ_FAILED</code> production error is reported to the Det, this parameter defines the unique error identifier which is reported there.</p> <p>Range:</p> <p>► 25 .. 255, must be unique</p>	
Multiplicity	1..1	
Type	INTEGER	
Default value	26	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMWrongBlockIdReportToDem	
Label	Wrong Block Id	
Description	<p>Defines the handling of the production error: <code>NVM_E_WRONG_BLOCK_ID</code></p> <p>Range:</p> <p>► DEM: Wrong Block Id errors are reported to the Diagnostics Event Manager (Dem).</p> <p>► DET: Wrong Block Id errors are reported to the Development Error Tracer (Det) if enabled.</p>	

	<ul style="list-style-type: none"> ► UserCallout: Wrong Block Id errors are reported with user configured callout. ► DISABLE: These errors are not reported at all. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► Wrong Block Id Event Parameter <code>NVM_E_WRONG_BLOCK_ID</code>: this reference must be set if DEM is selected here. ► Enable Development Error Detection <code>NvMDevErrorDetect</code>: must be enabled if DET is selected here
Multiplicity	1..1
Type	ENUMERATION
Default value	DISABLE
Range	<div>DEM</div> <div>DET</div> <div>UserCallout</div> <div>DISABLE</div>
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMWrongBlockIdReportToDemDetErrorId
Label	Wrong Block Id DemToDet Error Id
Description	<p>If the <code>NVM_E_WRONG_BLOCK_ID</code> production error is reported to the Det, this parameter defines the unique error identifier which is reported there.</p> <p>Range:</p> <ul style="list-style-type: none"> ► 25 .. 255, must be unique
Multiplicity	1..1
Type	INTEGER
Default value	27
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMLossOfRedundancyReportToDem
Label	Loss of Redundancy
Description	Defines the handling of the production error: <code>NVM_E_LOSS_OF_REDUNDANCY</code>

	<p>Range:</p> <ul style="list-style-type: none"> ▶ DEM: Loss of Redundancy errors are reported to the Diagnostics Event Manager (Dem). ▶ DET: Loss of Redundancy errors are reported to the Development Error Tracer (Det) if enabled. ▶ UserCallout: Loss of Redundancy errors are reported from user configured callout. ▶ DISABLE: These errors are not reported at all. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Loss of Redundancy Event Parameter <code>NVM_E_LOSS_OF_REDUNDANCY</code>: this reference must be set if DEM is selected here. ▶ Enable Development Error Detection <code>NvMDevErrorDetect</code>: must be enabled if DET is selected here 	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	DISABLE	
Range	DEM	
	DET	
	UserCallout	
	DISABLE	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMLossOfRedundancyReportToDemDetErrorId
Label	Loss of Redundancy DemToDet Error Id
Description	<p>If the <code>NVM_E_LOSS_OF_REDUNDANCY</code> production error is reported to the Det, this parameter defines the unique error identifier which is reported there.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 25 .. 255, must be unique
Multiplicity	1..1
Type	INTEGER
Default value	28

Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMQueueOverflowReportToDem	
Label	Request Queue Overflow	
Description	<p>Defines the handling of the production error: <code>NVM_E_QUEUE_OVERFLOW</code></p> <p>Range:</p> <ul style="list-style-type: none"> ▶ <code>DEM</code>: Request Queue Overflow errors are reported to the Diagnostics Event Manager (Dem). ▶ <code>DET</code>: Request Queue Overflow errors are reported to the DevelopmentError Tracer (Det) if enabled. ▶ <code>UserCallout</code>: Request Queue Overflow errors are reported from user configured callout. ▶ <code>DISABLE</code>: These errors are not reported at all. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Queue Overflow Event Parameter <code>NVM_E_QUEUE_OVERFLOW</code>: this reference must be set if <code>DEM</code> is selected here. ▶ Enable Development Error Detection <code>NvMDevErrorDetect</code>: must be enabled if <code>DET</code> is selected here 	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	DISABLE	
Range	<div>DEM</div> <div>DET</div> <div>UserCallout</div> <div>DISABLE</div>	
Configuration class	VariantPreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMQueueOverflowReportToDemDetErrorId
Label	Queue Overflow DemToDet Error Id
Description	If the <code>NVM_E_QUEUE_OVERFLOW</code> production error is reported to the Det, this parameter defines the unique error identifier which is reported there.

	Range: ► 25 .. 255, must be unique
Multiplicity	1..1
Type	INTEGER
Default value	29
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMVerifyFailedReportToDem
Label	Request Verify Failed
Description	<p>Defines the handling of the production error: <code>NVM_E_VERIFY_FAILED</code></p> <p>Range:</p> <ul style="list-style-type: none"> ► <code>DEM</code>: Request Verify Failed errors are reported to the Diagnostics Event Manager (Dem). ► <code>DET</code>: Request Verify Failed errors are reported to the Development Error Tracer (Det) if enabled. ► <code>UserCallout</code>: Request Verify Failed errors are reported from user configured callout. ► <code>DISABLE</code>: These errors are not reported at all. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ► Verification Failed Event Parameter <code>NVM_E_VERIFY_FAILED</code>: this reference must be set ► Enable Development Error Detection <code>NvMDevErrorDetect</code>: must be enabled if DET is selected here
Multiplicity	1..1
Type	ENUMERATION
Default value	DISABLE
Range	<div>DEM</div> <div>DET</div> <div>UserCallout</div> <div>DISABLE</div>
Configuration class	VariantPreCompile: VariantPreCompile

Origin	Elektrobit Automotive GmbH
---------------	----------------------------

Parameter Name	NvMVerifyFailedReportToDemDetErrorId	
Label	Verify Failed DemToDet Error Id	
Description	<p>If the <code>NVM_E_VERIFY_FAILED</code> production error is reported to the Det, this parameter defines the unique error identifier which is reported there.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 25 .. 255, must be unique 	
Multiplicity	1..1	
Type	INTEGER	
Default value	30	
Configuration class	PreCompile:	VariantPreCompile
Origin	Elektrobit Automotive GmbH	

Parameter Name	NvMWriteProtectedReportToDem	
Label	Request Write Protected	
Description	<p>Defines the handling of the production error: <code>NVM_E_WRITE_PROTECTED</code></p> <p>Range:</p> <ul style="list-style-type: none"> ▶ DEM: Request Write Protected errors are reported to the Diagnostics Event Manager (Dem). ▶ DET: Request Write Protected errors are reported to the Development Error Tracer (Det) if enabled. ▶ UserCallout: Request Write Protected errors are reported from user configured callout. ▶ DISABLE: These errors are not reported at all. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Write Protected Event Parameter <code>NVM_E_WRITE_PROTECTED</code>: this reference must be set if DEM is selected here. ▶ Enable Development Error Detection <code>NvMDevErrorDetect</code>: must be enabled if DET is selected here 	
Multiplicity	1..1	
Type	ENUMERATION	
Default value	DISABLE	

Range	DEM
	DET
	UserCallout
	DISABLE
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMWriteProtectedReportToDemDetErrorId
Label	Write Protected DemToDet Error Id
Description	<p>If the <code>NVM_E_WRITE_PROTECTED</code> production error is reported to the Det, this parameter defines the unique error identifier which is reported there.</p> <p>Range:</p> <ul style="list-style-type: none"> ▶ 25 .. 255, must be unique
Multiplicity	1..1
Type	INTEGER
Default value	31
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMBlockCheckReportProdError
Label	Block Check Error
Description	<p>Defines the handling of the production error: <code>NVM_E_BLOCK_CHECK</code></p> <p>Range:</p> <ul style="list-style-type: none"> ▶ <code>DEM</code>: Block Check errors are reported to the Diagnostics Event Manager (Dem). ▶ <code>DET</code>: Block Check errors are reported to the Development Error Tracer (Det) if enabled. ▶ <code>UserCallout</code>: Block Check errors are reported from user configured call-out. ▶ <code>DISABLE</code>: These errors are not reported at all. <p>Dependency on parameter(s):</p> <ul style="list-style-type: none"> ▶ Enable Development Error Detection <code>NvMDevErrorDetect</code>: must be enabled if DET is selected here

Multiplicity	1..1
Type	ENUMERATION
Default value	DISABLE
Range	<div>DEM</div> <div>DET</div> <div>UserCallout</div> <div>DISABLE</div>
Configuration class	VariantPreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

Parameter Name	NvMBlockCheckReportProdErrorId
Label	Block Check Prod Error Id
Description	<p>If the NVM_E_BLOCK_CHECK production error is reported to the Det, this parameter defines the unique error identifier which is reported there.</p> <p>Range:</p> <p>► 25 .. 255, must be unique</p>
Multiplicity	1..1
Type	INTEGER
Default value	32
Configuration class	PreCompile: VariantPreCompile
Origin	Elektrobit Automotive GmbH

5.6.1.12. MultiCoreCallout

Parameters included	
Parameter name	Multiplicity
NvMReadBlockCallout	0..1
NvMWriteBlockCallout	0..1
NvMRestoreBlockDefaultsCallout	0..1
NvMReadPRAMBlockCallout	0..1
NvMWritePRAMBlockCallout	0..1
NvMRestorePRAMBlockDefaultsCallout	0..1

Parameters included	
NvMEraseNvBlockCallout	0..1
NvMInvalidateNvBlockCallout	0..1
NvMCancelJobsCallout	0..1

Parameter Name	NvMReadBlockCallout	
Label	Read callout for multicore	
Description	<p>Defines the name of the multi-core callout function to be used for NvM_Read-Block API.</p> <p>When the callout is configured a NvM_ReadBlock request evaluates first the condition for executing the request and then calls the callout function.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMWriteBlockCallout	
Label	Write callout for multicore	
Description	<p>Defines the name of the callout for the NvM_WriteBlock() API and all corespondent RTE APIs.</p> <p>When the callout is configured a NvM_WriteBlock request evaluates first the condition for executing the request and then calls the callout function.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMRestoreBlockDefaultsCallout	
Label	RestoreBlockDefaults callout for multicore	
Description	<p>Defines the name of the multi-core callout function to be used for NvM_Restore-BlockDefaults API.</p> <p>When the callout is configured a NvM_RestoreBlockDefaults request evaluates first the condition for executing the request and then calls the callout function.</p>	
Multiplicity	0..1	

Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMReadPRAMBlockCallout	
Label	ReadPRAM callout for multicore	
Description	<p>Defines the name of the multi-core callout function to be used for NvM_Read-PRAMBlock API.</p> <p>When the callout is configured a NvM_ReadPRAMBlock request evaluates first the condition for executing the request and then calls the callout function.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMWritePRAMBlockCallout	
Label	WritePRAM callout for multicore	
Description	<p>Defines the name of the multi-core callout function to be used for NvM_WritePRAMBlock API.</p> <p>When the callout is configured a NvM_WritePRAMBlock request evaluates first the condition for executing the request and then calls the callout function.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMRestorePRAMBlockDefaultsCallout	
Label	RestorePRAMBlockDefaults callout for multicore	
Description	<p>Defines the name of the multi-core callout function to be used for NvM_RestorePRAMBlockDefaults API.</p> <p>When the callout is configured a NvM_RestorePRAMBlockDefaults request evaluates first the condition for executing the request and then calls the callout function.</p>	
Multiplicity	0..1	

Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMEraseNvBlockCallout	
Label	Erase callout for multicore	
Description	<p>Defines the name of the multi-core callout function to be used for NvM_EraseNvBlock API.</p> <p>When the callout is configured a NvM_EraseNvBlock request evaluates first the condition for executing the request and then calls the callout function.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMInvalidateNvBlockCallout	
Label	InvalidateNvBlock callout for multicore	
Description	<p>Defines the name of the multi-core callout function to be used for NvM_InvalidateNvBlock API.</p> <p>When the callout is configured a NvM_InvalidateNvBlock request evaluates first the condition for executing the request and then calls the callout function.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	
Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

Parameter Name	NvMCancelJobsCallout	
Label	CancelJobs callout for multicore	
Description	<p>Defines the name of the multi-core callout function to be used for NvM_CancelJobs API.</p> <p>When the callout is configured a NvM_CancelJobs request evaluates first the condition for executing the request and then calls the callout function.</p>	
Multiplicity	0..1	
Type	FUNCTION-NAME	

Configuration class	PreCompile:	VariantPreCompile
Origin	AUTOSAR_ECUC	

5.6.1.13. PublishedInformation

Parameters included	
Parameter name	Multiplicity
PbcfgMSupport	1..1

Parameter Name	PbcfgMSupport	
Label	PbcfgM support	
Description	Specifies whether or not the NvM can use the PbcfgM module for post-build support.	
Multiplicity	1..1	
Type	BOOLEAN	
Default value	false	
Configuration class	PublishedInformation:	
Origin	Elektrobit Automotive GmbH	

5.6.2. Pre-configurations

5.6.2.1. NvMPreConfiguration

Containers included	
Container name	Container definition
NvMBlock_ConfigID	NvMBlockDescriptor

Parameters included	
Parameter name	Value

5.6.2.1.1. NvMBlock_ConfigID

Parameters included	
Parameter name	Value
NvMNvramBlockIdentifier	1
NvMNvBlockNum	2
NvMRomBlockNum	1
NvMBlockUseCrc	true
NvMCalcRamBlockCrc	true
NvMBlockManagementType	NVM_BLOCK_REDUNDANT
NvMNvBlockBaseNumber	1
NvMRamBlockDataAddress	&NvM_ConfigurationId
NvMRomBlockDataAddress	&NvM_CompiledConfigurationId

5.6.3. Application programming interface (API)

5.6.3.1. Type definitions

5.6.3.1.1. NvM_ASR32_BlockIdType

Purpose	Defines the NvM block ID which is equal to the number of the block in the configuration table. In case the Rte is available this type is defined in Rte_Type.h.
Type	uint16

5.6.3.1.2. NvM_ASR32_RequestResultType

Purpose	Defines the NvM error status e.g. NVM_REQ_OK. In case the Rte is available, this type is defined in Rte_Type.h.
Type	uint8

5.6.3.1.3. NvM_ASR40_BlockIdType

Purpose	Defines the NvM block ID which is equal to the number of the block in the configuration table. In case the Rte is available this type is defined in Rte_Type.h.
Type	uint16

5.6.3.1.4. NvM_ASR40_RequestResultType

Purpose	Defines the NvM error status e.g. NVM_REQ_OK. In case the Rte is available, this type is defined in Rte_Type.h.
Type	uint8

5.6.3.1.5. NvM_ASR42_BlockIdType

Purpose	Defines the NvM block ID which is equal to the number of the block in the configuration table. In case the Rte is available this type is defined in Rte_Type.h.
Type	uint16

5.6.3.1.6. NvM_ASR42_RequestResultType

Purpose	Defines the NvM error status e.g. NVM_REQ_OK. In case the Rte is available, this type is defined in Rte_Type.h.
Type	uint8

5.6.3.1.7. NvM_BlockIdType

Purpose	Provides AUTOSAR 4.0 NvM_BlockIdType as default to other BSW modules.
Type	NvM_ASR40_BlockIdType

5.6.3.1.8. NvM_RequestResultType

Purpose	Provides AUTOSAR 4.0 NvM_RequestResultType as default to other BSW modules.
----------------	---

Type	NvM_ASR40_RequestResultType
------	---

5.6.3.2. Macro constants

5.6.3.2.1. DBG_NVM_ASR40_CANCELJOBS_ENTRY

Purpose	Entry point of function NvM_CancelJobs .
Value	

5.6.3.2.2. DBG_NVM_ASR40_CANCELJOBS_EXIT

Purpose	Exit point of function NvM_CancelJobs .
Value	

5.6.3.2.3. DBG_NVM_ASR40_ERASENVBLOCK_ENTRY

Purpose	Entry point of function NvM_ASR40_EraseNvBlock() .
Value	

5.6.3.2.4. DBG_NVM_ASR40_ERASENVBLOCK_EXIT

Purpose	Exit point of function NvM_ASR40_EraseNvBlock() .
Value	

5.6.3.2.5. DBG_NVM_ASR40_GETDATAINDEX_ENTRY

Purpose	Entry point of function NvM_ASR40_GetDataIndex .
Value	

5.6.3.2.6. DBG_NVM_ASR40_GETDATAINDEX_EXIT

Purpose	Exit point of function NvM_ASR40_GetDataIndex .
---------	---

Value	
--------------	--

5.6.3.2.7. DBG_NVM_ASR40_GETERRORSTATUS_ENTRY

Purpose	Entry point of function NvM_ASR40_GetErrorStatus() .
Value	

5.6.3.2.8. DBG_NVM_ASR40_GETERRORSTATUS_EXIT

Purpose	Exit point of function NvM_ASR40_GetErrorStatus() .
Value	

5.6.3.2.9. DBG_NVM_ASR40_INVALIDATENVBLOCK_ENTRY

Purpose	Entry point of function NvM_ASR40_InvalidateNvBlock() .
Value	

5.6.3.2.10. DBG_NVM_ASR40_INVALIDATENVBLOCK_EXIT

Purpose	Exit point of function NvM_ASR40_InvalidateNvBlock() .
Value	

5.6.3.2.11. DBG_NVM_ASR40_READBLOCK_ENTRY

Purpose	Entry point of function NvM_ASR40_ReadBlock .
Value	

5.6.3.2.12. DBG_NVM_ASR40_READBLOCK_EXIT

Purpose	Exit point of function NvM_ASR40_ReadBlock .
----------------	--

Value	
--------------	--

5.6.3.2.13. DBG_NVM_ASR40_RESTOREBLOCKDEFAULTS_ENTRY

Purpose	Entry point of function NvM_ASR40_RestoreBlockDefaults() .
Value	

5.6.3.2.14. DBG_NVM_ASR40_RESTOREBLOCKDEFAULTS_EXIT

Purpose	Exit point of function NvM_ASR40_RestoreBlockDefaults() .
Value	

5.6.3.2.15. DBG_NVM_ASR40_SETBLOCKPROTECTION_ENTRY

Purpose	Entry point of function NvM_ASR40_SetBlockProtection() .
Value	

5.6.3.2.16. DBG_NVM_ASR40_SETBLOCKPROTECTION_EXIT

Purpose	Exit point of function NvM_ASR40_SetBlockProtection() .
Value	

5.6.3.2.17. DBG_NVM_ASR40_SETDATAINDEX_ENTRY

Purpose	Entry point of function NvM_ASR40_SetDataIndex .
Value	

5.6.3.2.18. DBG_NVM_ASR40_SETDATAINDEX_EXIT

Purpose	Exit point of function NvM_ASR40_SetDataIndex .
----------------	---

Value	
--------------	--

5.6.3.2.19. DBG_NVM_ASR40_SETRAMBLOCKSTATUS_ENTRY

Purpose	Entry point of function NvM_ASR40_SetRamBlockStatus() .
Value	

5.6.3.2.20. DBG_NVM_ASR40_SETRAMBLOCKSTATUS_EXIT

Purpose	Exit point of function NvM_ASR40_SetRamBlockStatus() .
Value	

5.6.3.2.21. DBG_NVM_ASR40_WRITEBLOCK_ENTRY

Purpose	Entry point of function NvM_ASR40_WriteBlock .
Value	

5.6.3.2.22. DBG_NVM_ASR40_WRITEBLOCK_EXIT

Purpose	Exit point of function NvM_ASR40_WriteBlock .
Value	

5.6.3.2.23. DBG_NVM_ASR42_READPRAMBLOCK_ENTRY

Purpose	Entry point of function NvM_ASR42_ReadPRAMBlock .
Value	

5.6.3.2.24. DBG_NVM_ASR42_READPRAMBLOCK_EXIT

Purpose	Exit point of function NvM_ASR42_ReadPRAMBlock .
----------------	--

Value	
--------------	--

5.6.3.2.25. DBG_NVM_ASR42_RESTOREPRAMBLOCKDEFAULTS_ENTRY

Purpose	Entry point of function NvM_ASR42_RestorePRAMBlockDefaults() .
Value	

5.6.3.2.26. DBG_NVM_ASR42_RESTOREPRAMBLOCKDEFAULTS_EXIT

Purpose	Exit point of function NvM_ASR42_RestorePRAMBlockDefaults() .
Value	

5.6.3.2.27. DBG_NVM_ASR42_WRITEPRAMBLOCK_ENTRY

Purpose	Entry point of function NvM_ASR42_WritePRAMBlock() .
Value	

5.6.3.2.28. DBG_NVM_ASR42_WRITEPRAMBLOCK_EXIT

Purpose	Exit point of function NvM_ASR42_WritePRAMBlock() .
Value	

5.6.3.2.29. DBG_NVM_CANCELWRITEALL_ENTRY

Purpose	Entry point of function NvM_CancelWriteAll() .
Value	

5.6.3.2.30. DBG_NVM_CANCELWRITEALL_EXIT

Purpose	Exit point of function NvM_CancelWriteAll() .
----------------	---

Value	
--------------	--

5.6.3.2.31. DBG_NVM_FIRSTINITALL_ENTRY

Purpose	Entry point of function NvM_FirstInitAll() .
Value	

5.6.3.2.32. DBG_NVM_FIRSTINITALL_EXIT

Purpose	Exit point of function NvM_FirstInitAll() .
Value	

5.6.3.2.33. DBG_NVM_GETVERSIONINFO_ENTRY

Purpose	Entry point of function NvM_GetVersionInfo() .
Value	

5.6.3.2.34. DBG_NVM_GETVERSIONINFO_EXIT

Purpose	Exit point of function NvM_GetVersionInfo() .
Value	

5.6.3.2.35. DBG_NVM_GLOBALCALLLEVEL

Purpose	Change of NvM_GlobalCallLevel.
Value	

5.6.3.2.36. DBG_NVM_GLOBALERRORSTATUS

Purpose	Change of NvM_GlobalErrorStatus.
----------------	----------------------------------

Value	
--------------	--

5.6.3.2.37. DBG_NVM_GLOBALGENERICSTATUS

Purpose	Change of NvM_GlobalErrorStatus.
Value	

5.6.3.2.38. DBG_NVM_GLOBALWORKINGSTATUS

Purpose	Change of NvM_GlobalWorkingStatus.
Value	

5.6.3.2.39. DBG_NVM_INIT_ENTRY

Purpose	Entry point of function NvM_Init() .
Value	

5.6.3.2.40. DBG_NVM_INIT_EXIT

Purpose	Exit point of function NvM_Init() .
Value	

5.6.3.2.41. DBG_NVM_JOBENDNOTIFICATION_ENTRY

Purpose	Entry point of function NvM_JobEndNotification() .
Value	

5.6.3.2.42. DBG_NVM_JOBENDNOTIFICATION_EXIT

Purpose	Exit point of function NvM_JobEndNotification() .
----------------	---

Value	
--------------	--

5.6.3.2.43. DBG_NVM_JOBERRORNOTIFICATION_ENTRY

Purpose	Entry point of function NvM_JobErrorNotification() .
Value	

5.6.3.2.44. DBG_NVM_JOBERRORNOTIFICATION_EXIT

Purpose	Exit point of function NvM_JobErrorNotification() .
Value	

5.6.3.2.45. DBG_NVM_MAINFUNCTION_ENTRY

Purpose	Entry point of function NvM_MainFunction() .
Value	

5.6.3.2.46. DBG_NVM_MAINFUNCTION_EXIT

Purpose	Exit point of function NvM_MainFunction() .
Value	

5.6.3.2.47. DBG_NVM_READALL_ENTRY

Purpose	Entry point of function NvM_ReadAll() .
Value	

5.6.3.2.48. DBG_NVM_READALL_EXIT

Purpose	Exit point of function NvM_ReadAll() .
----------------	--

Value	
--------------	--

5.6.3.2.49. DBG_NVM_SETBLOCKLOCKSTATUS_ENTRY

Purpose	Entry point of function NvM_SetBlockLockStatus() .
Value	

5.6.3.2.50. DBG_NVM_SETBLOCKLOCKSTATUS_EXIT

Purpose	Exit point of function NvM_SetBlockLockStatus() .
Value	

5.6.3.2.51. DBG_NVM_VALIDATEALL_ENTRY

Purpose	Entry point of function NvM_ValidateAll() .
Value	

5.6.3.2.52. DBG_NVM_VALIDATEALL_EXIT

Purpose	Exit point of function NvM_ValidateAll() .
Value	

5.6.3.2.53. DBG_NVM_WRITEALL_ENTRY

Purpose	Entry point of function NvM_WriteAll() .
Value	

5.6.3.2.54. DBG_NVM_WRITEALL_EXIT

Purpose	Exit point of function NvM_WriteAll() .
----------------	---

Value	
--------------	--

5.6.3.2.55. NVM_AR_RELEASE_MAJOR_VERSION

Purpose	AUTOSAR release major version.
Value	4U

5.6.3.2.56. NVM_AR_RELEASE_MINOR_VERSION

Purpose	AUTOSAR release minor version.
Value	0U

5.6.3.2.57. NVM_AR_RELEASE_REVISION_VERSION

Purpose	AUTOSAR release revision version.
Value	3U

5.6.3.2.58. NVM_BLOCK_INVALID

Purpose	Used to mark a block as invalid in the NvM_AdminBlockTable.
Value	0xFFU

5.6.3.2.59. NVM_BLOCK_VALID

Purpose	Used to mark a block as valid in the NvM_AdminBlockTable.
Value	0xA5U

5.6.3.2.60. NVM_CANCEL_JOBS_API_ID

Purpose	Defines API ID of function NvM_CancelJobs() .
----------------	---

Value	0x10U
--------------	-------

5.6.3.2.61. NVM_CANCEL_WRITE_ALL_API_ID

Purpose	Defines API ID of function NvM_CancelWriteAll() .
Value	0x0AU

5.6.3.2.62. NVM_CRYPT0_HOOKS_API_ID

Purpose	Defines API ID of crypto hooks operation...used for DET.
Value	0x16U

5.6.3.2.63. NVM_ENABLE_BC_API_ID

Purpose	Defines API ID of function NvM_EnableBc().
Value	0x14U

5.6.3.2.64. NVM_ERASE_NV_BLOCK_API_ID

Purpose	Defines API ID of function NvM_EraseNvBlock() .
Value	0x09U

5.6.3.2.65. NVM_E_BLOCK_CONFIG

Purpose	API read/write/control request failed.
Value	0x18U

5.6.3.2.66. NVM_E_BLOCK_LOCKED

Purpose	API write request failed.
----------------	---------------------------

Value	0x19U
--------------	-------

5.6.3.2.67. NVM_E_BLOCK_PENDING

Purpose	API read/write/control request failed.
Value	0x15U

5.6.3.2.68. NVM_E_BLOCK_WITHOUT_DEFAULTS

Purpose	Block doesn't have ROM default or init function configured.
Value	0x11U

5.6.3.2.69. NVM_E_CRYPTO_FAILED

Purpose	Crypto hook responded with fail operation or timeout occurred.
Value	0x1BU

5.6.3.2.70. NVM_E_CRYPTO_WRITE_FAILED

Purpose	Crypto write hook responded with fail operation or timeout occurred.
Value	0x1CU

5.6.3.2.71. NVM_E_NOT_INITIALIZED

Purpose	NVRAM Manager is still uninitialized.
Value	0x14U

5.6.3.2.72. NVM_E_PARAM_ADDRESS

Purpose	API requests called with wrong parameter.
----------------	---

Value	0x0DU
--------------	-------

5.6.3.2.73. NVM_E_PARAM_BLOCK_DATA_IDX

Purpose	API requests called with wrong parameter.
Value	0x0CU

5.6.3.2.74. NVM_E_PARAM_BLOCK_ID

Purpose	API requests called with wrong parameter.
Value	0x0AU

5.6.3.2.75. NVM_E_PARAM_BLOCK_TYPE

Purpose	API requests called with wrong parameter.
Value	0x0BU

5.6.3.2.76. NVM_E_PARAM_DATA

Purpose	API requests called with wrong parameter.
Value	0x0EU

5.6.3.2.77. NVM_E_PARAM_POINTER

Purpose	API requests called with wrong parameter.
Value	0x0FU

5.6.3.2.78. NVM_E_WRITE_ONCE_STATUS_UNKNOWN

Purpose	API write/erase/invalidate request with no prior read request.
----------------	--

Value	0x1AU
--------------	-------

5.6.3.2.79. NVM_FIRST_INIT_ALL_API_ID

Purpose	Defines API ID of function NvM_FirstInitAll() .
Value	0x15U

5.6.3.2.80. NVM_GET_DATA_INDEX_API_ID

Purpose	Defines API ID of function NvM_GetDataIndex() .
Value	0x02U

5.6.3.2.81. NVM_GET_ERROR_STATUS_API_ID

Purpose	Defines API ID of function NvM_GetErrorStatus() .
Value	0x04U

5.6.3.2.82. NVM_GET_VERSION_INFO_API_ID

Purpose	Defines API ID of function NvM_GetVersionInfo() .
Value	0x0FU

5.6.3.2.83. NVM_INVALIDATE_NV_BLOCK_API_ID

Purpose	Defines API ID of function NvM_InvalidateNvBlock() .
Value	0x0BU

5.6.3.2.84. NVM_MODULE_ID

Purpose	AUTOSAR module identification.
----------------	--------------------------------

Value	20U
--------------	-----

5.6.3.2.85. NVM_READ_ALL_API_ID

Purpose	Defines API ID of function NvM_ReadAll() .
Value	0x0CU

5.6.3.2.86. NVM_READ_BLOCK_API_ID

Purpose	Defines API ID of function NvM_ReadBlock() .
Value	0x06U

5.6.3.2.87. NVM_READ_PRAM_BLOCK_API_ID

Purpose	Defines API ID of function NvM_ReadPRAMBlock() .
Value	0x16U

5.6.3.2.88. NVM_REQ_BLOCK_SKIPPED

Purpose	Value of type NvM_RequestResultType returned by NvM_GetErrorStatus.
Value	4U

5.6.3.2.89. NVM_REQ_CANCELED

Purpose	Value of type NvM_RequestResultType returned by NvM_GetErrorStatus if the request has been cancelled.
Value	6U

5.6.3.2.90. NVM_REQ_CANCELLED

Purpose	Value of type NvM_RequestResultType returned by NvM_GetErrorStatus if NvM_WriteAll() is cancelled by NvM_CancelWriteAll() .
----------------	---

Value	6U
--------------	----

5.6.3.2.91. NVM_REQ_INTEGRITY_FAILED

Purpose	Value of type NvM_RequestResultType returned by NvM_GetErrorStatus if for e.g. the CRC check of the last API request has finished.
Value	3U

5.6.3.2.92. NVM_REQ_NOT_OK

Purpose	value of type NvM_RequestResultType returned by NvM_GetErrorStatus if the last API request has finished with an error.
Value	1U

5.6.3.2.93. NVM_REQ_NV_INVALIDATED

Purpose	Value of type NvM_RequestResultType returned by NvM_GetErrorStatus if the requested NvM block is invalid.
Value	5U

5.6.3.2.94. NVM_REQ_OK

Purpose	value of type NvM_RequestResultType returned by NvM_GetErrorStatus if the last API request has finished without error.
Value	0U

5.6.3.2.95. NVM_REQ_PENDING

Purpose	Value of type NvM_RequestResultType returned by NvM_GetErrorStatus if the last API request has still not finished.
Value	2U

5.6.3.2.96. NVM_REQ_REDUNDANCY_FAILED

Purpose	Value of type <code>NvM_RequestResultType</code> returned by <code>NvM_GetErrorStatus</code> if the required redundancy of the referenced NV block is lost.
Value	7U

5.6.3.2.97. NVM_REQ_RESTORED_DEFAULTS

Purpose	Value of type <code>NvM_RequestResultType</code> returned by <code>NvM_GetErrorStatus</code> if the referenced NV block has been restored with default values.
Value	8U

5.6.3.2.98. NVM_REQ_RESTORED_FROM_ROM

Purpose	Value of type <code>NvM_RequestResultType</code> returned by <code>NvM_GetErrorStatus</code> if the referenced NV block has been restored from ROM.defined needed for backwards compatibility with BSW modules that use it.
Value	NVM_REQ_RESTORED_DEFAULTS

5.6.3.2.99. NVM_RESTORE_BLOCK_DEFAULTS_API_ID

Purpose	Defines API ID of function NvM_RestoreBlockDefaults() .
Value	0x08U

5.6.3.2.100. NVM_RESTORE_PRAM_BLOCK_DEFAULTS_API_ID

Purpose	Defines API ID of function NvM_RestorePRAMBlockDefaults() .
Value	0x18U

5.6.3.2.101. NVM_SET_BLOCK_LOCK_STATUS_API_ID

Purpose	Defines API ID of function NvM_SetBlockLockStatus() .
----------------	---

Value	0x13U
--------------	-------

5.6.3.2.102. NVM_SET_BLOCK_PROTECTION_API_ID

Purpose	Defines API ID of function NvM_SetBlockProtection() .
Value	0x03U

5.6.3.2.103. NVM_SET_DATA_INDEX_API_ID

Purpose	***** API IDs
Value	0x01U
Description	Defines API ID of function NvM_SetDataIndex() .

5.6.3.2.104. NVM_SET_RAM_BLOCK_STATUS_API_ID

Purpose	Defines API ID of function NvM_SetRamBlockStatus() .
Value	0x05U

5.6.3.2.105. NVM_SW_MAJOR_VERSION

Purpose	AUTOSAR module major version.
Value	6U

5.6.3.2.106. NVM_SW_MINOR_VERSION

Purpose	AUTOSAR module minor version.
Value	17U

5.6.3.2.107. NVM_SW_PATCH_VERSION

Purpose	AUTOSAR module patch version.
----------------	-------------------------------

Value	22U
--------------	-----

5.6.3.2.108. NVM_VALIDATE_ALL_API_ID

Purpose	Defines API ID of function NvM_ValidateAll() .
Value	0x19U

5.6.3.2.109. NVM_VENDOR_ID

Purpose	AUTOSAR vendor identification: Elektrobit Automotive GmbH.
Value	1U

5.6.3.2.110. NVM_WRITE_ALL_API_ID

Purpose	Defines API ID of function NvM_WriteAll() .
Value	0x0DU

5.6.3.2.111. NVM_WRITE_BLOCK_API_ID

Purpose	Defines API ID of function NvM_WriteBlock() .
Value	0x07U

5.6.3.2.112. NVM_WRITE_PRAM_BLOCK_API_ID

Purpose	Defines API ID of function NvM_WritePRAMBlock() .
Value	0x17U

5.6.3.2.113. NvM_CancelJobs

Purpose	Wrapping macro for multicore function to other BSW modules.
----------------	---

Value	NvM_ASR40_CancelJobs
--------------	----------------------

5.6.3.2.114. NvM_EraseNvBlock

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_EraseNvBlock

5.6.3.2.115. NvM_GetDataIndex

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_GetDataIndex

5.6.3.2.116. NvM_GetErrorStatus

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_GetErrorStatus

5.6.3.2.117. NvM_InvalidateNvBlock

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_InvalidateNvBlock

5.6.3.2.118. NvM_ReadBlock

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_ReadBlock

5.6.3.2.119. NvM_ReadPRAMBlock

Purpose	Wrapping macro to provide AUTOSAR 4.2 API as default to other BSW modules.
----------------	--

Value	NvM_ASR42_ReadPRAMBlock
--------------	-------------------------

5.6.3.2.120. NvM_RestoreBlockDefaults

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_RestoreBlockDefaults

5.6.3.2.121. NvM_RestorePRAMBlockDefaults

Purpose	Wrapping macro to provide AUTOSAR 4.2 API as default to other BSW modules.
Value	NvM_ASR42_RestorePRAMBlockDefaults

5.6.3.2.122. NvM_SetBlockProtection

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_SetBlockProtection

5.6.3.2.123. NvM_SetDataIndex

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_SetDataIndex

5.6.3.2.124. NvM_SetRamBlockStatus

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_SetRamBlockStatus

5.6.3.2.125. NvM_WriteBlock

Purpose	Wrapping macro to provide AUTOSAR 4.0 API as default to other BSW modules.
Value	NvM_ASR40_WriteBlock

5.6.3.2.126. NvM_WritePRAMBlock

Purpose	Wrapping macro to provide AUTOSAR 4.2 API as default to other BSW modules.
Value	NvM_ASR42_WritePRAMBlock

5.6.3.3. Functions

5.6.3.3.1. NvM_ASR32_EraseNvBlock

Purpose	Service to erase a NV block.	
Synopsis	Std_ReturnType NvM_ASR32_EraseNvBlock (NvM_ASR32_BlockIdType BlockId);	
Service ID	9	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	<p>This service initiates erasing the data of this block in NV memory. If this block has immediate priority the underlying services Ea_EraseIm-mediateBlock() resp. Fee_-EraseImmedateBlock() are used for erasing. If this block has not immediate priority then this service shall return E_NOT_OK. In case this block has immediate priority a subsequent NvM_ReadBlock() request for this block results in setting the error status to NVM_REQ_NV_INVALIDATED. In case of an immediate priority block a NvM_WriteBlock() request may be faster than for a standard priority block because</p>	

	the block is already pre erased. If the used NVRAM hardware does not support separate erase and write commands and automatically erases upon a write command then there is no significant time difference in writing the data of an immediate priority block and a standard priority block. Please see the User's Guide of the EEPROM driver to get more details.
--	---

5.6.3.3.2. NvM_ASR32_GetDataIndex

Purpose	Service for getting the currently set DataIndex of a dataset NVRAM block.	
Synopsis	<pre>void NvM_ASR32_GetDataIndex (NvM_ASR32_BlockIdType BlockId , uint8 * DataIndex);</pre>	
Service ID	2	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvM-NvBlockIdentifier is equal to BlockId. Range: 0..65535
Parameters (out)	DataIndex	Pointer to where to store the current dataset index Range 0 .. size of NVM_VAR.
Description	The service gets the current index (association) of a Dataset Block (with/without ROM blocks) with its corresponding RAM block. The usage in conjunction with all other block management types is possible but without any effect.	

5.6.3.3.3. NvM_ASR32_GetErrorStatus

Purpose	Service to read the block dependent error/status information.	
Synopsis	<pre>void NvM_ASR32_GetErrorStatus (NvM_ASR32_BlockIdType BlockId , NvM_ASR32_RequestResultType * RequestResultPtr);</pre>	
Service ID	4	
Sync/Async	Synchronous	
Reentrancy	Reentrant	

Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvM-NvBlockIdentifier is equal to BlockId. Range: 0..65535.
Parameters (out)	RequestResultPtr	Pointer to where to store the request result. The buffer contains the error status of block 'BlockId'. Range 0..255.
Description	The request reads the block dependent error/status information in the administrative part of a RAM block. The status was set by a former or current synchronous request.	

5.6.3.3.4. NvM_ASR32_InvalidateNvBlock

Purpose	Service to invalidate a NV block.	
Synopsis	Std_ReturnType NvM_ASR32_InvalidateNvBlock (NvM_ASR32_BlockIdType BlockId);	
Service ID	11	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	This service initiates invalidating the data of this block in NV memory permanently. For this purpose the underlying function Ea_InvalidateBlock() resp. Fee_InvalidateBlock() is used. A subsequent NvM_ReadBlock() request for this block results in setting the error status to NVM_REQ_NV_INVALIDATED. The RAM Block is not affected by this service. If the RAM Block is valid it is not invalidated.	

--	--

5.6.3.3.5. NvM_ASR32_ReadBlock

Purpose	Service to copy the data of the NV block to its corresponding RAM block.	
Synopsis	<pre>Std_ReturnType NvM_ASR32_ReadBlock (NvM_ASR32_BlockIdType BlockId , uint8 * NvM_DstPtr);</pre>	
Service ID	6	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_INTEGRITY_FAILED: thrown, if a CRC mismatch occurs for a block configured with CRC or MEMIF_BLOCK_INCONSISTENT is reported by the MemIf module during an attempt to read a NV block. ▶ NVM_E_LOSS_OF_REDUNDANCY: thrown, if a redundant NV block is detected as damaged (the second copy is different than the first one). ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRONG_BLOCK_ID: thrown, if the static block ID stored in the NV block header is different than the requested block ID. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	NvM_DstPtr	Pointer to a temporary RAM block to where the NvM data shall be copied.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	read list overflow, request has not been accepted
Description	<p>Service for copying the data of the NVM Block to its corresponding RAM Block. This call takes over the given parameters, queues the read request in the read request list and returns.</p> <p>ErrorStatus after service finished</p>	

- ▶ NVM_REQ_OK : The NVM Block was copied successfully or the ROM Block was copied successfully if a Dataset Block was requested and the DataIndex selects a ROM Block.
- ▶ NVM_REQ_INVALIDATED : The job result of an underlying abstraction module is MEMIF_BLOCK_INVALID.
- ▶ NVM_REQ_INTEGRITY_FAILED : If a CRC mismatch occurs or if the job result of the underlying memory abstraction module is MEMIF_BLOCK_INCONSISTENT.
- ▶ NVM_REQ_CANCELLED: An underlying memory abstraction module reports MEMIF_JOB_CANCELED.
- ▶ NVM_REQ_NOT_OK: The job result of the underlying memory abstraction module is MEMIF_JOB_FAILED or the ROM Block was not copied successfully if a Dataset Block was requested and the DataIndex selects a ROM Block.

If an error is detected during processing [NvM_ReadBlock\(\)](#) and the ROM default data must be copied by [NvM_RestoreBlockDefaults\(\)](#) the ErrorStatus set by [NvM_RestoreBlockDefaults\(\)](#) is ignored and the above ErrorStatus applies nevertheless.

RAMBlockStatus after service finished (applies only for permanent RAM Blocks)

- ▶ VALID / UNCHANGED: A NVM Block was copied successfully to the RAM Block.
- ▶ VALID / CHANGED: A ROM Block was copied successfully to the RAM Block.
- ▶ INVALID / UNCHANGED: An error was detected during copying the NVM Block to the RAM Block and no ROM Block is configured.
- ▶ INVALID / CHANGED: Can not occur.

A ROM Block is copied if a Dataset Block is requested and the DataIndex selects a ROM Block, or an error is detected during copying the NVM Block and the default ROM Block data is copied.

5.6.3.3.6. NvM_ASR32_RestoreBlockDefaults

Purpose	Service to restore the default data to its corresponding RAM block.
Synopsis	<pre>Std_ReturnType NvM_ASR32_RestoreBlockDefaults (NvM_ASR32_ BlockIdType BlockId , uint8 * NvM_DestPtr);</pre>
Service ID	8

Sync/Async	Asynchronous	
Reentrancy	Non-Reentrant	
Production Errors	► NVM_E_QUEUE_OVERFLOW : thrown, if a new NVM request cannot be processed because the NVM queue is full.	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	NvM_DestPtr	Pointer to the RAM Data Block
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	control list overflow, request has not been accepted. If a Dataset Block is requested which has at least one ROM Block assigned but the DataIndex selects a NVM Block
Description	<p>The services copies the ROM Block default data to its corresponding RAM Block. The function queues the request and returns. If a Dataset Block is requested the user application is responsible to set the DataIndex to a ROM Data Block previously to calling NvM_RestoreBlockDefaults(). The DataIndex must be set by NvM_SetDataIndex(). If the DataIndex selects a NVM Block E_NOT_OK is returned and no default data is copied.</p> <p>ErrorStatusafter service finished</p> <ul style="list-style-type: none"> ► NVM_REQ_OK: The default data was copied successfully from the ROM Block to the RAM Block. ► NVM_REQ_NOT_OK: The default data could not be copied successfully from the ROM Block to the RAM Block. <p>RAMBlockStatus after service finished (applies only for permanent RAM Blocks)</p> <ul style="list-style-type: none"> ► VALID / CHANGED : The ROM Block data is copied successfully to the RAM Block. ► INVALID / UNCHANGED: The ROM Block data is not copied successfully to the RAM Block. 	

5.6.3.3.7. NvM_ASR32_SetBlockProtection

Purpose	Service for setting/resetting the write protection for a NV block.	
Synopsis	<pre>void NvM_ASR32_SetBlockProtection (NvM_ASR32_BlockIdType BlockId , boolean ProtectionEnabled);</pre>	
Service ID	3	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	ProtectionEnabled	TRUE: Write protection shall be enabled, FALSE: Write protection shall be disabled
Description	<p>This function is used to set the protection flag of the given block. If ProtectionEnabled is set to TRUE then all subsequent NvM_WriteBlock() requests for this block will be rejected and NvM_WriteBlock() returns E_NOT_OK. If ProtectionEnabled is set to FALSE then all subsequent NvM_WriteBlock() requests for this block will be accepted and NvM_WriteBlock() returns E_OK. NvM_WriteAll() skips all blocks who are write protected and sets the error status for these blocks to NVM_REQ_BLOCK_SKIPPED. If the configuration parameter NvMWriteBlockOnce is enabled for this block and DET error detection is enabled then write protection can be neither enabled nor disabled for this block and the DET error NVM_E_BLOCK_CONFIG is reported. If NvMWriteBlockOnce is enabled then the write protection flag is completely managed internally by the NVRAM Manager.</p>	

5.6.3.3.8. NvM_ASR32_SetDataIndex

Purpose	Service for setting the DataIndex of a dataset NVRAM block.	
Synopsis	<pre>void NvM_ASR32_SetDataIndex (NvM_ASR32_BlockIdType BlockId , uint8 DataIndex);</pre>	
Service ID	1	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvM-NvBlockIdentifier is equal to BlockId. Range: 0..65535.

	DataIndex	Index position of a NVM Block or ROM Data Block within a Dataset NVRAM Block. Range: 0..255
Description	The request sets the specified index to associate a dataset NV Block (with/without ROM Blocks) with its corresponding RAM Block. The usage in conjunction with all other block management types is possible, but without any effect.	

5.6.3.3.9. NvM_ASR32_SetRamBlockStatus

Purpose	Service for setting the RAM block status of an NVRAM block.	
Synopsis	void NvM_ASR32_SetRamBlockStatus (NvM_ASR32_BlockIdType BlockId , boolean BlockChanged);	
Service ID	5	
Sync/Async	Asynchronous/Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	BlockChanged	TRUE: block will be written by NvM_WriteAll() , FALSE: block will not be written by NvM_WriteAll() .
Description	<p>NvM_SetRamBlockStatus() prepares the NVM Block BlockId for the next execution of NvM_WriteAll(). If the parameter BlockChanged is Description set to TRUE and if a Permanent RAM Data Block is defined for block BlockId then the Permanent RAM Data Block is marked to be valid and changed and therefore the data will be written by the next call of NvM_WriteAll(). If CRC calculation is enabled for this block then the CRC will be recalculated in background. The RAM Block is synchronously marked as valid and changed previously to the CRC calculation. If BlockChanged is set to FALSE and if a Permanent Data RAM Block is defined for block BlockId then the RAM Block is marked as invalid and unchanged and therefore the data will not be written by the next call of NvM_WriteAll(). If the block BlockId has no Permanent RAM Data Block assigned it is not marked as valid and changed and therefore no data will be written into the corresponding NVMBlock by the next call of NvM_WriteAll(). NvM_SetRamBlockStatus() has no side effects to function NvM_WriteBlock(). NvM_WriteBlock() always writes the data independently of the changed and valid marks.</p> <p>Note: This function can be synchronous or asynchronous, depending on CRC disabled or enabled for the block. The CRC calculation is only triggered when a Perma-</p>	

	<p>nent RAM CRC Block is assigned to the requested NVRAM Block. This means the configuration parameter <code>NvMBlockUseCrc</code> must be enabled for the requested NVRAM Block. The CRC calculation in background is performed by the NvM_MainFunction() which must be called cyclically. If CRC calculation finished the <code>ErrorStatus</code> is set and the Single Block Job End Notification callback function for the requested NVRAM Block is called if configured.</p> <p>ErrorStatusafter service finished</p> <ul style="list-style-type: none"> ▶ <code>NVM_REQ_OK</code> : CRC calculation was triggered and finished successfully. <p>Note: The <code>ErrorStatus</code> remains unmodified if CRC calculation was not triggered.</p> <p>RAMBlockStatus after service finished (applies only for permanent RAM Blocks)</p> <ul style="list-style-type: none"> ▶ <code>VALID / CHANGED</code> : The value <code>TRUE</code> is passed to parameter <code>BlockChanged</code>. ▶ <code>INVALID / UNCHANGED</code>: The value <code>FALSE</code> is passed to parameter <code>BlockChanged</code>.
--	---

5.6.3.3.10. NvM_ASR32_WriteBlock

Purpose	Service to copy the data of the RAM block to its corresponding NV block.	
Synopsis	<pre>Std_ReturnType NvM_ASR32_WriteBlock (NvM_ASR32_BlockIdType BlockId , const uint8 * NvM_SrcPtr);</pre>	
Service ID	7	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. <code>NVM_E_REQ_FAILED</code> shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_VERIFY_FAILED: thrown, if the content of the RAM block is not the same as the read back data. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter <code>NvMN-vBlockIdentifier</code> is equal to <code>BlockId</code> .

	NvM_SrcPtr	Pointer to the RAM data block which shall be copied to NVRAM.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	<p>Service for copying the data of the RAM Block to its corresponding NVM Block. First the write protection attribute of the NVRAM Block shall be tested in the administrative part of the corresponding RAM Block. In case of disabled write protection, the request shall be queued in the appropriate write list. The acceptance result of the request is returned synchronously. In case the requested NVRAM Block has immediate priority zero the request is inserted into the Immediate Queue. A currently processed standard priority job is terminated and resumed after all immediate priority jobs are finished. Multiple concurrent requests are enqueued.</p>	

5.6.3.3.11. NvM_ASR40_CancelJobs

Purpose	Service to cancel all pending job requests for an NVRAM Block.	
Synopsis	<pre>Std_ReturnType NvM_ASR40_CancelJobs (NvM_BlockIdType BlockId);</pre>	
Parameters (in)	BlockId	The Block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId
Return Value	Std_returnType	
	E_OK	Request has been accepted
	E_OK	Request has not been accepted
Description	<p>This service cancels all queued single block requests(Read, Write, Erase, Invalidate or Restore) for the NVRAM block referenced by BlockId. It does not cancel an ongoing job processing.</p> <p>ServiceID{16} Reentrancy{Reentrant} Synchronicity{Asynchronous}</p>	

5.6.3.3.12. NvM_ASR40_EraseNvBlock

Purpose	Service to erase a NV block.
----------------	------------------------------

Synopsis	Std_ReturnType NvM_ASR40_EraseNvBlock (NvM_ASR40_BlockIdType BlockId);	
Service ID	9	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	<p>This service initiates erasing the data of this block in NV memory. If this block has immediate priority the underlying services Ea_EraseIm- mediateBlock() resp. Fee_-EraseImmediateBlock() are used for erasing. If this block has not immediate priority then this block is only invalidated via the function NvM_InvalidateBlock(). In both cases a subsequent NvM_ReadBlock() request for this block results in setting the error status to NVM_REQ_NV_INVALIDATED. Erasing a standard priority block is normally faster than erasing an immediate priority block because only few management data needs to be manipulated in the NV memory by the underlying Ea/Fee module. However, a subsequent NvM_WriteBlock() request may take more time because the NVM data also must be erased by NvM_WriteBlock() before it can be written. In case of an immediate priority block a NvM_WriteBlock() request may be faster than for a standard priority block because the block is already pre erased. If the used NVRAM hardware does not support separate erase and write commands and automatically erases upon a write command then there is no significant time difference in writing the data of an immediate priority block and a standard priority block. Please see the User's Guide of the EEPROM driver to get more details.</p>	

5.6.3.3.13. NvM_ASR40_GetDataIndex

Purpose	Service for getting the currently set DataIndex of a dataset NVRAM block.	
Synopsis	Std_ReturnType NvM_ASR40_GetDataIndex (NvM_ASR40_BlockIdType BlockId , uint8 * DataIndex);	
Service ID	2	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvM-NvBlockIdentifier is equal to BlockId. Range: 0..65535
Parameters (out)	DataIndex	Pointer to where to store the current dataset index Range 0 .. size of NVM_VAR.
Return Value	Std_ReturnType	
	E_OK	The index position has been retrieved successfully.
	E_NOT_OK	An error occurred.
Description	The service gets the current index (association) of a Dataset Block (with/without ROM blocks) with its corresponding RAM block. The usage in conjunction with all other block management types is possible but without any effect.	

5.6.3.3.14. NvM_ASR40_GetErrorStatus

Purpose	Service to read the block dependent error/status information.	
Synopsis	Std_ReturnType NvM_ASR40_GetErrorStatus (NvM_ASR40_BlockIdType BlockId , NvM_ASR40_RequestResultType * RequestResultPtr);	
Service ID	4	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvM-NvBlockIdentifier is equal to BlockId. Range: 0..65535.

Parameters (out)	RequestResultPtr	Pointer to where to store the request result. The buffer contains the error status of block 'BlockId'. Range 0..255.
Return Value	Std_ReturnType	
	E_OK	The block dependent error/status information was read successfully.
	E_NOT_OK	An error occurred.
Description	The request reads the block dependent error/status information in the administrative part of a RAM block. The status was set by a former or current synchronous request.	

5.6.3.3.15. NvM_ASR40_InvalidateNvBlock

Purpose	Service to invalidate a NV block.	
Synopsis	Std_ReturnType NvM_ASR40_InvalidateNvBlock (NvM_ASR40_BlockIdType BlockId);	
Service ID	11	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	This service initiates invalidating the data of this block in NV memory permanently. For this purpose the underlying function Ea_InvalidateBlock() resp. Fee_InvalidateBlock() is used. A subsequent NvM_ReadBlock() request for this block results in setting the error status to NVM_REQ_NV_INVALIDATED. The RAM Block is not affected by this service. If the RAM Block is valid it is not invalidated.	

--	--

5.6.3.3.16. NvM_ASR40_ReadBlock

Purpose	Service to copy the data of the NV block to its corresponding RAM block.	
Synopsis	<pre>Std_ReturnType NvM_ASR40_ReadBlock (NvM_ASR40_BlockIdType BlockId , void * NvM_DstPtr);</pre>	
Service ID	6	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_INTEGRITY_FAILED: thrown, if a CRC mismatch occurs for a block configured with CRC or MEMIF_BLOCK_INCONSISTENT is reported by the MemIf module during an attempt to read a NV block. ▶ NVM_E_LOSS_OF_REDUNDANCY: thrown, if a redundant NV block is detected as damaged (the second copy is different than the first one). ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRONG_BLOCK_ID: thrown, if the static block ID stored in the NV block header is different than the requested block ID. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	NvM_DstPtr	Pointer to a temporary RAM block to where the NvM data shall be copied.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	read list overflow, request has not been accepted
Description	<p>Service for copying the data of the NVM Block to its corresponding RAM Block. This call takes over the given parameters, queues the read request in the read request list and returns.</p> <p>ErrorStatus after service finished</p>	

- ▶ NVM_REQ_OK : The NVM Block was copied successfully or the ROM Block was copied successfully if a Dataset Block was requested and the DataIndex selects a ROM Block.
- ▶ NVM_REQ_INVALIDATED : The job result of an underlying abstraction module is MEMIF_BLOCK_INVALID.
- ▶ NVM_REQ_INTEGRITY_FAILED : If a CRC mismatch occurs or if the job result of the underlying memory abstraction module is MEMIF_BLOCK_INCONSISTENT.
- ▶ NVM_REQ_CANCELLED: An underlying memory abstraction module reports MEMIF_JOB_CANCELED.
- ▶ NVM_REQ_NOT_OK: The job result of the underlying memory abstraction module is MEMIF_JOB_FAILED or the ROM Block was not copied successfully if a Dataset Block was requested and the DataIndex selects a ROM Block.

If an error is detected during processing [NvM_ReadBlock\(\)](#) and the ROM default data must be copied by [NvM_RestoreBlockDefaults\(\)](#) the ErrorStatus set by [NvM_RestoreBlockDefaults\(\)](#) is ignored and the above ErrorStatus applies nevertheless.

RAMBlockStatus after service finished (applies only for permanent RAM Blocks)

- ▶ VALID / UNCHANGED: A NVM Block was copied successfully to the RAM Block.
- ▶ VALID / CHANGED: A ROM Block was copied successfully to the RAM Block.
- ▶ INVALID / UNCHANGED: An error was detected during copying the NVM Block to the RAM Block and no ROM Block is configured.
- ▶ INVALID / CHANGED: Can not occur.

A ROM Block is copied if a Dataset Block is requested and the DataIndex selects a ROM Block, or an error is detected during copying the NVM Block and the default ROM Block data is copied.

5.6.3.3.17. NvM_ASR40_RestoreBlockDefaults

Purpose	Service to restore the default data to its corresponding RAM block.
Synopsis	<pre>Std_ReturnType NvM_ASR40_RestoreBlockDefaults (NvM_ASR40_ BlockIdType BlockId , void * NvM_DestPtr);</pre>
Service ID	8

Sync/Async	Asynchronous	
Reentrancy	Non-Reentrant	
Production Errors	► NVM_E_QUEUE_OVERFLOW : thrown, if a new NVM request cannot be processed because the NVM queue is full.	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	NvM_DestPtr	Pointer to the RAM Data Block
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	control list overflow, request has not been accepted. If a Dataset Block is requested which has at least one ROM Block assigned but the DataIndex selects a NVM Block
Description	<p>The services copies the ROM Block default data to its corresponding RAM Block. The function queues the request and returns. If a Dataset Block is requested the user application is responsible to set the DataIndex to a ROM Data Block previously to calling NvM_RestoreBlockDefaults(). The DataIndex must be set by NvM_SetDataIndex(). If the DataIndex selects a NVM Block E_NOT_OK is returned and no default data is copied.</p> <p>ErrorStatusafter service finished</p> <ul style="list-style-type: none"> ► NVM_REQ_OK: The default data was copied successfully from the ROM Block to the RAM Block. ► NVM_REQ_NOT_OK: The default data could not be copied successfully from the ROM Block to the RAM Block. <p>RAMBlockStatus after service finished (applies only for permanent RAM Blocks)</p> <ul style="list-style-type: none"> ► VALID / CHANGED : The ROM Block data is copied successfully to the RAM Block. ► INVALID / UNCHANGED: The ROM Block data is not copied successfully to the RAM Block. 	

5.6.3.3.18. NvM_ASR40_SetBlockProtection

Purpose	Service for setting/resetting the write protection for a NV block.	
Synopsis	Std_ReturnType NvM_ASR40_SetBlockProtection (NvM_ASR40_BlockIdType BlockId , boolean ProtectionEnabled);	
Service ID	3	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	ProtectionEnabled	TRUE: Write protection shall be enabled, FALSE: Write protection shall be disabled
Return Value	Std_ReturnType	
	E_OK	The block was enabled/disabled as requested.
	E_NOT_OK	An error occurred.
Description	<p>This function is used to set the protection flag of the given block. If ProtectionEnabled is set to TRUE then all subsequent NvM_WriteBlock() requests for this block will be rejected and NvM_WriteBlock() returns E_NOT_OK. If ProtectionEnabled is set to FALSE then all subsequent NvM_WriteBlock() requests for this block will be accepted and NvM_WriteBlock() returns E_OK. NvM_WriteAll() skips all blocks who are write protected and sets the error status for these blocks to NVM_REQ_BLOCK_SKIPPED. If the configuration parameter NvMWriteBlockOnce is enabled for this block and DET error detection is enabled then write protection can be neither enabled nor disabled for this block and the DET error NVM_E_BLOCK_CONFIG is reported. If NvMWriteBlockOnce is enabled then the write protection flag is completely managed internally by the NVRAM Manager.</p>	

5.6.3.3.19. NvM_ASR40_SetDataIndex

Purpose	Service for setting the DataIndex of a dataset NVRAM block.	
Synopsis	Std_ReturnType NvM_ASR40_SetDataIndex (NvM_ASR40_BlockIdType BlockId , uint8 DataIndex);	
Service ID	1	
Sync/Async	Synchronous	

Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvM-NvBlockIdentifier is equal to BlockId. Range: 0..65535.
	DataIndex	Index position of a NVM Block or ROM Data Block within a Dataset NVRAM Block. Range: 0..255
Return Value	Std_ReturnType	
	E_OK	The index position was set successfully.
	E_NOT_OK	An error occurred.
Description	The request sets the specified index to associate a dataset NV Block (with/without ROM Blocks) with its corresponding RAM Block. The usage in conjunction with all other block management types is possible, but without any effect.	

5.6.3.3.20. NvM_ASR40_SetRamBlockStatus

Purpose	Service for setting the RAM block status of an NVRAM block.	
Synopsis	Std_ReturnType NvM_ASR40_SetRamBlockStatus (NvM_ASR40_BlockId-Type BlockId , boolean BlockChanged);	
Service ID	5	
Sync/Async	Asynchronous/Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	BlockChanged	TRUE: block will be written by NvM_WriteAll() , FALSE: block will not be written by NvM_WriteAll() .
Return Value	Std_ReturnType	
	E_OK	The status of the RAM-Block was changed as requested.
	E_NOT_OK	An error occurred.
Description	NvM_SetRamBlockStatus() prepares the NVM Block BlockId for the next execution of NvM_WriteAll() . If the parameter BlockChanged is Description set to TRUE and if	

a Permanent RAM Data Block is defined for block BlockId then the Permanent RAM Data Block is marked to be valid and changed and therefore the data will be written by the next call of [NvM_WriteAll\(\)](#). If CRC calculation is enabled for this block then the CRC will be recalculated in background. The RAM Block is synchronously marked as valid and changed previously to the CRC calculation. If BlockChanged is set to FALSE and if a Permanent Data RAM Block is defined for block BlockId then the RAM Block is marked as invalid and unchanged and therefore the data will not be written by the next call of [NvM_WriteAll\(\)](#). If the block BlockId has no Permanent RAM Data Block assigned it is not marked as valid and changed and therefore no data will be written into the corresponding NVMBlock by the next call of [NvM_WriteAll\(\)](#). [NvM_SetRamBlockStatus\(\)](#) has no side effects to function [NvM_WriteBlock\(\)](#). [NvM_WriteBlock\(\)](#) always writes the data independently of the changed and valid marks.

Note: This function can be synchronous or asynchronous, depending on CRC disabled or enabled for the block. The CRC calculation is only triggered when a Permanent RAM CRC Block is assigned to the requested NVRAM Block. This means the configuration parameter NvMBlockUseCrc must be enabled for the requested NVRAM Block. The CRC calculation in background is performed by the [NvM_MainFunction\(\)](#) which must be called cyclically. If CRC calculation finished the ErrorStatus is set and the Single Block Job End Notification callback function for the requested NVRAM Block is called if configured.

ErrorStatusafter service finished

- NVM_REQ_OK : CRC calculation was triggered and finished successfully.

Note: The ErrorStatus remains unmodified if CRC calculation was not triggered.

RAMBlockStatus after service finished (applies only for permanent RAM Blocks)

- VALID / CHANGED : The value TRUE is passed to parameter BlockChanged.
- INVALID / UNCHANGED: The value FALSE is passed to parameter BlockChanged.

5.6.3.3.21. NvM_ASR40_WriteBlock

Purpose	Service to copy the data of the RAM block to its corresponding NV block.
Synopsis	<pre>Std_ReturnType NvM_ASR40_WriteBlock (NvM_ASR40_BlockIdType BlockId , const void * NvM_SrcPtr);</pre>
Service ID	7
Sync/Async	Asynchronous

Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_VERIFY_FAILED: thrown, if the content of the RAM block is not the same as the read back data. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	NvM_SrcPtr	Pointer to the RAM data block which shall be copied to NVRAM.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	<p>Service for copying the data of the RAM Block to its corresponding NVM Block. First the write protection attribute of the NVRAM Block shall be tested in the administrative part of the corresponding RAM Block. In case of disabled write protection, the request shall be queued in the appropriate write list. The acceptance result of the request is returned synchronously. In case the requested NVRAM Block has immediate priority zero the request is inserted into the Immediate Queue. A currently processed standard priority job is terminated and resumed after all immediate priority jobs are finished. Multiple concurrent requests are enqueued.</p>	

5.6.3.3.22. NvM_ASR42_EraseNvBlock

Purpose	Service to erase a NV block.
Synopsis	Std_ReturnType NvM_ASR42_EraseNvBlock (NvM_ASR42_BlockIdType BlockId);
Service ID	9

Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	<p>This service initiates erasing the data of this block in NV memory. If this block has immediate priority the underlying services Ea_EraseIm- mediateBlock() resp. Fee_- EraseImmediateBlock() are used for erasing. If this block has not immediate priority then this block is only invalidated via the function NvM_InvalidateBlock(). In both cases a subsequent NvM_ReadBlock() request for this block results in setting the error status to NVM_REQ_NV_INVALIDATED. Erasing a standard priority block is normally faster than erasing an immediate priority block because only few management data needs to be manipulated in the NV memory by the underlying Ea/Fee module. However, a subsequent NvM_WriteBlock() request may take more time because the NVM data also must be erased by NvM_WriteBlock() before it can be written. In case of an immediate priority block a NvM_WriteBlock() request may be faster than for a standard priority block because the block is already pre erased. If the used NVRAM hardware does not support separate erase and write commands and automatically erases upon a write command then there is no significant time difference in writing the data of an immediate priority block and a standard priority block. Please see the User's Guide of the EEPROM driver to get more details.</p>	

5.6.3.3.23. NvM_ASR42_GetDataIndex

Purpose	Service for getting the currently set DataIndex of a dataset NVRAM block.
----------------	---

Synopsis	Std_ReturnType NvM_ASR42_GetDataIndex (NvM_ASR42_BlockIdType BlockId , uint8 * dataIndex);	
Service ID	2	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvM-NvBlockIdentifier is equal to BlockId. Range: 0..65535
Parameters (out)	DataIndex	Pointer to where to store the current dataset index Range 0 .. size of NVM_VAR.
Return Value	Std_ReturnType	
	E_OK	The index position has been retrieved successfully.
	E_NOT_OK	An error occurred.
Description	The service gets the current index (association) of a Dataset Block (with/without ROM blocks) with its corresponding RAM block. The usage in conjunction with all other block management types is possible but without any effect.	

5.6.3.3.24. NvM_ASR42_GetErrorStatus

Purpose	Service to read the block dependent error/status information.	
Synopsis	Std_ReturnType NvM_ASR42_GetErrorStatus (NvM_ASR42_BlockIdType BlockId , NvM_ASR42_RequestResultType * RequestResultPtr);	
Service ID	4	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvM-NvBlockIdentifier is equal to BlockId. Range: 0..65535.
Parameters (out)	RequestResultPtr	Pointer to where to store the request result. The buffer contains the error status of block 'BlockId'. Range 0..255.

Return Value	Std_ReturnType	
	E_OK	The block dependent error/status information was read successfully.
	E_NOT_OK	An error occurred.
Description	The request reads the block dependent error/status information in the administrative part of a RAM block. The status was set by a former or current synchronous request.	

5.6.3.3.25. NvM_ASR42_InvalidateNvBlock

Purpose	Service to invalidate a NV block.	
Synopsis	Std_ReturnType NvM_ASR42_InvalidateNvBlock (NvM_ASR42_BlockIdType BlockId);	
Service ID	11	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMNvBlockIdentifier is equal to BlockId.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	This service initiates invalidating the data of this block in NV memory permanently. For this purpose the underlying function Ea_InvalidateBlock() resp. Fee_InvalidateBlock() is used. A subsequent NvM_ReadBlock() request for this block results in setting the error status to NVM_REQ_NV_INVALIDATED. The RAM Block is not affected by this service. If the RAM Block is valid it is not invalidated.	

5.6.3.3.26. NvM_ASR42_ReadBlock

Purpose	Service to copy the data of the NV block to its corresponding RAM block.	
Synopsis	Std_ReturnType NvM_ASR42_ReadBlock (NvM_ASR42_BlockIdType BlockId , void * NvM_DstPtr);	
Service ID	6	
Sync/Async	Asynchronous	
Reentrancy	Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_INTEGRITY_FAILED: thrown, if a CRC mismatch occurs for a block configured with CRC or MEMIF_BLOCK_INCONSISTENT is reported by the MemIf module during an attempt to read a NV block. ▶ NVM_E_LOSS_OF_REDUNDANCY: thrown, if a redundant NV block is detected as damaged (the second copy is different than the first one). ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRONG_BLOCK_ID: thrown, if the static block ID stored in the NV block header is different than the requested block ID. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	NvM_DstPtr	Pointer to a temporary RAM block to where the NvM data shall be copied.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	read list overflow, request has not been accepted
Description	<p>Service for copying the data of the NVM Block to its corresponding RAM Block. This call takes over the given parameters, queues the read request in the read request list and returns.</p> <p>ErrorStatus after service finished</p> <ul style="list-style-type: none"> ▶ NVM_REQ_OK : The NVM Block was copied successfully or the ROM Block was copied successfully if a Dataset Block was requested and the DataIndex selects a ROM Block. 	

- ▶ NVM_REQ_INVALIDATED : The job result of an underlying abstraction module is MEMIF_BLOCK_INVALID.
- ▶ NVM_REQ_INTEGRITY_FAILED : If a CRC mismatch occurs or if the job result of the underlying memory abstraction module is MEMIF_BLOCK_INCONSISTENT.
- ▶ NVM_REQ_CANCELLED: An underlying memory abstraction module reports MEMIF_JOB_CANCELED.
- ▶ NVM_REQ_NOT_OK: The job result of the underlying memory abstraction module is MEMIF_JOB_FAILED or the ROM Block was not copied successfully if a Dataset Block was requested and the DataIndex selects a ROM Block.

If an error is detected during processing [NvM_ReadBlock\(\)](#) and the ROM default data must be copied by [NvM_RestoreBlockDefaults\(\)](#) the ErrorStatus set by [NvM_RestoreBlockDefaults\(\)](#) is ignored and the above ErrorStatus applies nevertheless.

RAMBlockStatus after service finished (applies only for permanent RAM Blocks)

- ▶ VALID / UNCHANGED: A NVM Block was copied successfully to the RAM Block.
- ▶ VALID / CHANGED: A ROM Block was copied successfully to the RAM Block.
- ▶ INVALID / UNCHANGED: An error was detected during copying the NVM Block to the RAM Block and no ROM Block is configured.
- ▶ INVALID / CHANGED: Can not occur.

A ROM Block is copied if a Dataset Block is requested and the DataIndex selects a ROM Block, or an error is detected during copying the NVM Block and the default ROM Block data is copied.

5.6.3.3.27. NvM_ASR42_ReadPRAMBlock

Purpose	Service to copy the data of the NV block to its corresponding permanent RAM block.
Synopsis	<code>Std_ReturnType NvM_ASR42_ReadPRAMBlock (NvM_ASR42_BlockIdType BlockId);</code>
Service ID	22
Sync/Async	Asynchronous
Reentrancy	Reentrant

Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_INTEGRITY_FAILED: thrown, if a CRC mismatch occurs for a block configured with CRC or MEMIF_BLOCK_INCONSISTENT is reported by the MemIf module during an attempt to read a NV block. ▶ NVM_E_LOSS_OF_REDUNDANCY: thrown, if a redundant NV block is detected as damaged (the second copy is different than the first one). ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_WRONG_BLOCK_ID: thrown, if the static block ID stored in the NV block header is different than the requested block ID. 	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMNVBlockIdentifier is equal to BlockId.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	read list overflow, request has not been accepted
Description	<p>Service to copy the data of the NV block to its corresponding permanent RAM Block. This call takes over the given parameters, queues the read request in the read request list and returns.</p> <p>ErrorStatus after service finished</p> <ul style="list-style-type: none"> ▶ NVM_REQ_OK : The NVM Block was copied successfully or the ROM Block was copied successfully if a Dataset Block was requested and the DataIndex selects a ROM Block. ▶ NVM_REQ_INVALIDATED : The job result of an underlying abstraction module is MEMIF_BLOCK_INVALID. ▶ NVM_REQ_INTEGRITY_FAILED : If a CRC mismatch occurs or if the job result of the underlying memory abstraction module is MEMIF_BLOCK_INCONSISTENT. ▶ NVM_REQ_CANCELLED: An underlying memory abstraction module reports MEMIF_JOB_CANCELED. ▶ NVM_REQ_NOT_OK: The job result of the underlying memory abstraction module is MEMIF_JOB_FAILED or the ROM Block was not copied successfully if a Dataset Block was requested and the DataIndex selects a ROM Block. 	

	<p>If an error is detected during processing NvM_ReadPRAMBlock() and the ROM default data must be copied by NvM_RestorePRAMBlockDefaults() the ErrorStatus set by NvM_RestorePRAMBlockDefaults() is ignored and the above ErrorStatus applies nevertheless.</p> <p>RAMBlockStatus after service finished (applies only for permanent RAM Blocks)</p> <ul style="list-style-type: none"> ▶ VALID / UNCHANGED: A NVM Block was copied successfully to the RAM Block. ▶ VALID / CHANGED: A ROM Block was copied successfully to the RAM Block. ▶ INVALID / UNCHANGED: An error was detected during copying the NVM Block to the RAM Block and no ROM Block is configured. ▶ INVALID / CHANGED: Can not occur. <p>A ROM Block is copied if a Dataset Block is requested and the DataIndex selects a ROM Block, or an error is detected during copying the NVM Block and the default ROM Block data is copied.</p>
--	---

5.6.3.3.28. NvM_ASR42_RestoreBlockDefaults

Purpose	Service to restore the default data to its corresponding RAM block.	
Synopsis	<pre>Std_ReturnType NvM_ASR42_RestoreBlockDefaults (NvM_ASR42_ BlockIdType BlockId , void * NvM_DestPtr);</pre>	
Service ID	8	
Sync/Async	Asynchronous	
Reentrancy	Non-Reentrant	
Production Errors	▶ NVM_E_QUEUE_OVERFLOW : thrown, if a new NVM request cannot be processed because the NVM queue is full.	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	NvM_DestPtr	Pointer to the RAM Data Block
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	control list overflow, request has not been accepted. If a Dataset Block is request-

		ed which has at least one ROM Block assigned but the DataIndex selects a NVM Block
Description	<p>The services copies the ROM Block default data to its corresponding RAM Block. The function queues the request and returns. If a Dataset Block is requested the user application is responsible to set the DataIndex to a ROM Data Block previously to calling NvM_RestoreBlockDefaults(). The DataIndex must be set by NvM_SetDataIndex(). If the DataIndex selects a NVM Block E_NOT_OK is returned and no default data is copied.</p> <p>ErrorStatusafter service finished</p> <ul style="list-style-type: none"> ▶ NVM_REQ_OK: The default data was copied successfully from the ROM Block to the RAM Block. ▶ NVM_REQ_NOT_OK: The default data could not be copied successfully from the ROM Block to the RAM Block. <p>RAMBlockStatus after service finished (applies only for permanent RAM Blocks)</p> <ul style="list-style-type: none"> ▶ VALID / CHANGED : The ROM Block data is copied successfully to the RAM Block. ▶ INVALID / UNCHANGED: The ROM Block data is not copied successfully to the RAM Block. 	

5.6.3.3.29. NvM_ASR42_RestorePRAMBlockDefaults

Purpose	Service to restore the default data to its corresponding permanent RAM block.	
Synopsis	Std_ReturnType NvM_ASR42_RestorePRAMBlockDefaults (NvM_ASR42_-BlockIdType BlockId);	
Service ID	24	
Sync/Async	Asynchronous	
Reentrancy	Non-Reentrant	
Production Errors	▶ NVM_E_QUEUE_OVERFLOW : thrown, if a new NVM request cannot be processed because the NVM queue is full.	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
Return Value	Std_ReturnType	

	E_OK	request has been accepted
	E_NOT_OK	control list overflow, request has not been accepted. If a Dataset Block is requested which has at least one ROM Block assigned but the DataIndex selects a NVM Block
Description	<p>The services copies the ROM Block default data to its corresponding permanent RAM Block. The function queues the request and returns. If a Dataset Block is requested the user application is responsible to set the DataIndex to a ROM Data Block previously to calling NvM_RestoreBlockDefaults(). The DataIndex must be set by NvM_SetDataIndex(). If the DataIndex selects a NVM Block E_NOT_OK is returned and no default data is copied.</p> <p>ErrorStatusafter service finished</p> <ul style="list-style-type: none"> ▶ NVM_REQ_OK: The default data was copied successfully from the ROM Block to the RAM Block. ▶ NVM_REQ_NOT_OK: The default data could not be copied successfully from the ROM Block to the RAM Block. <p>RAMBlockStatus after service finished (applies only for permanent RAM Blocks)</p> <ul style="list-style-type: none"> ▶ VALID / CHANGED : The ROM Block data is copied successfully to the RAM Block. ▶ INVALID / UNCHANGED: The ROM Block data is not copied successfully to the RAM Block. 	

5.6.3.3.30. NvM_ASR42_SetBlockProtection

Purpose	Service for setting/resetting the write protection for a NV block.	
Synopsis	<pre>Std_ReturnType NvM_ASR42_SetBlockProtection (NvM_ASR42_Block-IdType BlockId , boolean ProtectionEnabled);</pre>	
Service ID	3	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.

	ProtectionEnabled	TRUE: Write protection shall be enabled, FALSE: Write protection shall be disabled
Return Value	Std_ReturnType	
	E_OK	The block was enabled/disabled as requested.
	E_NOT_OK	An error occurred.
Description	<p>This function is used to set the protection flag of the given block. If ProtectionEnabled is set to TRUE then all subsequent NvM_WriteBlock() requests for this block will be rejected and NvM_WriteBlock() returns E_NOT_OK. If ProtectionEnabled is set to FALSE then all subsequent NvM_WriteBlock() requests for this block will be accepted and NvM_WriteBlock() returns E_OK. NvM_WriteAll() skips all blocks who are write protected and sets the error status for these blocks to NVM_REQ_BLOCK_SKIPPED. If the configuration parameter NvMWriteBlockOnce is enabled for this block and DET error detection is enabled then write protection can be neither enabled nor disabled for this block and the DET error NVM_E_BLOCK_CONFIG is reported. If NvMWriteBlockOnce is enabled then the write protection flag is completely managed internally by the NVRAM Manager.</p>	

5.6.3.3.31. NvM_ASR42_SetDataIndex

Purpose	Service for setting the DataIndex of a dataset NVRAM block.	
Synopsis	Std_ReturnType NvM_ASR42_SetDataIndex (NvM_ASR42_BlockIdType BlockId , uint8 DataIndex);	
Service ID	1	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvM-NvBlockIdentifier is equal to BlockId. Range: 0..65535.
	DataIndex	Index position of a NVM Block or ROM Data Block within a Dataset NVRAM Block. Range: 0..255
Return Value	Std_ReturnType	
	E_OK	The index position was set successfully.
	E_NOT_OK	An error occurred.

Description	The request sets the specified index to associate a dataset NV Block (with/without ROM Blocks) with its corresponding RAM Block. The usage in conjunction with all other block management types is possible, but without any effect.
--------------------	--

5.6.3.3.32. NvM_ASR42_SetRamBlockStatus

Purpose	Service for setting the RAM block status of an NVRAM block.	
Synopsis	Std_ReturnType NvM_ASR42_SetRamBlockStatus (NvM_ASR42_BlockId-Type BlockId , boolean BlockChanged);	
Service ID	5	
Sync/Async	Asynchronous/Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	BlockChanged	TRUE: block will be written by NvM_WriteAll() , FALSE: block will not be written by NvM_WriteAll() .
Return Value	Std_ReturnType	
	E_OK	The status of the RAM-Block was changed as requested.
	E_NOT_OK	An error occurred.
Description	<p>NvM_SetRamBlockStatus() prepares the NVM Block BlockId for the next execution of NvM_WriteAll(). If the parameter BlockChanged is Description set to TRUE and if a Permanent RAM Data Block is defined for block BlockId then the Permanent RAM Data Block is marked to be valid and changed and therefore the data will be written by the next call of NvM_WriteAll(). If CRC calculation is enabled for this block then the CRC will be recalculated in background. The RAM Block is synchronously marked as valid and changed previously to the CRC calculation. If BlockChanged is set to FALSE and if a Permanent Data RAM Block is defined for block BlockId then the RAM Block is marked as invalid and unchanged and therefore the data will not be written by the next call of NvM_WriteAll(). If the block BlockId has no Permanent RAM Data Block assigned it is not marked as valid and changed and therefore no data will be written into the corresponding NVMBlock by the next call of NvM_WriteAll(). NvM_SetRamBlockStatus() has no side effects to function NvM_WriteBlock(). NvM_WriteBlock() always writes the data independently of the changed and valid marks.</p>	

	<p>Note: This function can be synchronous or asynchronous, depending on CRC disabled or enabled for the block. The CRC calculation is only triggered when a Permanent RAM CRC Block is assigned to the requested NVRAM Block. This means the configuration parameter <code>NvMBlockUseCrc</code> must be enabled for the requested NVRAM Block. The CRC calculation in background is performed by the NvM_MainFunction() which must be called cyclically. If CRC calculation finished the <code>ErrorStatus</code> is set and the Single Block Job End Notification callback function for the requested NVRAM Block is called if configured.</p> <p><code>ErrorStatus</code> after service finished</p> <ul style="list-style-type: none"> ▶ <code>NVM_REQ_OK</code> : CRC calculation was triggered and finished successfully. <p>Note: The <code>ErrorStatus</code> remains unmodified if CRC calculation was not triggered.</p> <p><code>RAMBlockStatus</code> after service finished (applies only for permanent RAM Blocks)</p> <ul style="list-style-type: none"> ▶ <code>VALID / CHANGED</code> : The value <code>TRUE</code> is passed to parameter <code>BlockChanged</code>. ▶ <code>INVALID / UNCHANGED</code>: The value <code>FALSE</code> is passed to parameter <code>BlockChanged</code>.
--	--

5.6.3.3.33. NvM_ASR42_WriteBlock

Purpose	Service to copy the data of the RAM block to its corresponding NV block.
Synopsis	<code>Std_ReturnType NvM_ASR42_WriteBlock (NvM_ASR42_BlockIdType BlockId , const void * NvM_SrcPtr);</code>
Service ID	7
Sync/Async	Asynchronous
Reentrancy	Reentrant
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. <code>NVM_E_REQ_FAILED</code> shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_VERIFY_FAILED: thrown, if the content of the RAM block is not the same as the read back data. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block.

Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	NvM_SrcPtr	Pointer to the RAM data block which shall be copied to NVRAM.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	Service for copying the data of the RAM Block to its corresponding NVM Block. First the write protection attribute of the NVRAM Block shall be tested in the administrative part of the corresponding RAM Block. In case of disabled write protection, the request shall be queued in the appropriate write list. The acceptance result of the request is returned synchronously. In case the requested NVRAM Block has immediate priority zero the request is inserted into the Immediate Queue. A currently processed standard priority job is terminated and resumed after all immediate priority jobs are finished. Multiple concurrent requests are enqueued.	

5.6.3.3.34. NvM_ASR42_WritePRAMBlock

Purpose	Service to copy the data of the NV block to its corresponding permanent RAM block.
Synopsis	Std_ReturnType NvM_ASR42_WritePRAMBlock (NvM_ASR42_BlockIdType BlockId);
Service ID	23
Sync/Async	Asynchronous
Reentrancy	Reentrant
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. ▶ NVM_E_REQ_FAILED: thrown, if a single block request fails or lower layer module reports failure. NVM_E_REQ_FAILED shall be reported only after a maximum number of retries is exceeded for a requests with a retry counter configured. ▶ NVM_E_VERIFY_FAILED: thrown, if the content of the RAM block is not the same as the read back data. ▶ NVM_E_WRITE_PROTECTED: thrown, if a write erase or invalidate operation is requested for a write protected block.

Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
Return Value	Std_ReturnType	
	E_OK	request has been accepted
	E_NOT_OK	request has not been accepted
Description	<p>Service for copying the data of the RAM Block to its corresponding permanent RAM Block. First the write protection attribute of the NVRAM Block shall be tested in the administrative part of the corresponding RAM Block. In case of disabled write protection, the request shall be queued in the appropriate write list. The acceptance result of the request is returned synchronously. In case the requested NVRAM Block has immediate priority zero the request is inserted into the Immediate Queue. A currently processed standard priority job is terminated and resumed after all immediate priority jobs are finished. Multiple concurrent requests are enqueued.</p>	

5.6.3.3.35. NvM_CancelWriteAll

Purpose	Service to cancel a running NvM_WriteAll request.
Synopsis	<code>void NvM_CancelWriteAll (void);</code>
Service ID	10
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	<p>This function cancels an ongoing NvM_WriteAll(). After the data of the currently processed NVRAM block has been written Description NvM_WriteAll() terminates and sets the multi request job result to NVM_REQ_CANCELLED. If NvM_WriteAll() is not running at the time NvM_CancelWriteAll() is called this request has no effect on the execution of the next NvM_WriteAll() request.</p> <p>Remarks: In contrast to the AUTOSAR NVRAM Manager specification the function NvM_CancelWriteAll is implemented as a synchronous function instead of an asynchronous one. Since NvM_WriteAll() must not be queued (requirement NVM420) NvM_CancelWriteAll() need not to be queued too and therefore it need not to be asynchronous.</p>

5.6.3.3.36. NvM_DisableBlockCheckMechanism

Purpose	Initialize the BlockCheck's internal variables.	
Synopsis	<code>void NvM_DisableBlockCheckMechanism (boolean BcDisable);</code>	
Parameters (in)	BcDisable	: If TRUE the Block Check mechanism is disabled, otherwise enabled.

5.6.3.3.37. NvM_EnableBlockCheck

Purpose	Enable/disable the Block Check mechanism for a given block.	
Synopsis	<code>Std_ReturnType NvM_EnableBlockCheck (NvM_BlockIdType BlockId , boolean BcEnable);</code>	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	Enable/Disable	value.
Return Value	:	void
:	Std_ReturnType	

5.6.3.3.38. NvM_FirstInitAll

Purpose	Initiates a multi block FirstInitAll request.	
Synopsis	<code>void NvM_FirstInitAll (void);</code>	
Service ID	21	
Sync/Async	Asynchronous	
Reentrancy	Non-Reentrant	
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full. 	
Description	<p>NvM_FirstInitAll() initializes every the configured blocks that correspond with the following conditions :</p> <ul style="list-style-type: none"> ▶ the NvM block is configured to be processed for NvM_FirstInitAll API. <p>Depending on the corresponding configuration the "Block Initialization" can means one of the following things :</p> <ul style="list-style-type: none"> ▶ Blocks of type DATASET, or blocks that do not have configured a ROM default, or no "InitCallback" configured will be invalidated. 	

	► Blocks that are type NATIVE or REDUNDANT and have either ROM defaults or "InitCallback" configured will have their default data copied into NV.
--	---

5.6.3.3.39. NvM_GetVersionInfo

Purpose	Service to get the version information of the NvM module.	
Synopsis	<pre>void NvM_GetVersionInfo (Std_VersionInfoType * versionInfoPtr);</pre>	
Service ID	15	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (out)	versionInfoPtr	Pointer to where to store the version information of this module.
Description	<p>This service returns the version information of this module. The version information includes:</p> <ul style="list-style-type: none"> ► Module ID ► Vendor ID ► Vendor specific version numbers <p>Remarks:</p> <ul style="list-style-type: none"> ► This function can be invoked by the user application although NvM_Init was not invoked yet. 	

5.6.3.3.40. NvM_Init

Purpose	Initializes the NvM module.	
Synopsis	<pre>void NvM_Init (void);</pre>	
Service ID	0	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Description	<p>This service initializes the NVRAM manager. The time consuming NVRAM block initialization and set-up according to the block descriptor is done by the NvM_ReadAll request.</p>	

5.6.3.3.41. NvM_JobEndNotification

Purpose	Function to be used by the underlying memory abstraction to signal end of job without error.
Synopsis	<code>void NvM_JobEndNotification (void);</code>
Service ID	17
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	<p>This callback function can be used by an underlying module to signal the NVRAM Manager that it finished a job successfully. The NVRAM Manager additionally calls MemIf_GetJobResult() to evaluate the job result of the underlying module and only if MEMIF_JOB_OK is returned it continues with normal operation. Therefore all underlying modules must also set the appropriate job result before calling NvM_JobEndNotification(). The NvM_MainFunction() must be called cyclically although NvM_JobEndNotification() is called by an underlying module because if the configuration parameter NvMCrcNumOf-Bytes is less than the size of the currently processed NVRAM Block then only the cyclic NvM_MainFunction() can finish the CRC calculation. This function might be called from interrupt level. The execution time depends mainly on the execution time of the CRC calculation of the current NVRAM Block. Therefore if it is called from interrupt level the configuration parameter NvMCrcNumOfBytes shall be set appropriately so that the CRC calculation is interrupted and finished by the NvM_MainFunction() which is called from Task level.</p>

5.6.3.3.42. NvM_JobErrorNotification

Purpose	Function to be used by the underlying memory abstraction to signal end of job with error.
Synopsis	<code>void NvM_JobErrorNotification (void);</code>
Service ID	18
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Description	<p>This callback function can be used by an underlying module to signal the NVRAM Manager that it finished a job with an error. The NVRAM Manager additionally calls MemIf_GetJobResult() to evaluate the job result of the underlying module. This is necessary because depending on the kind of error different error recovery is done. Therefore all underlying modules must also set the appropriate job result before calling NvM_JobErrorNotification(). The NvM_MainFunction() must be called cyclically although NvM_JobEndNotification() is called by an underlying module because if the</p>

	configuration parameter NvM- CrcNumOfBytes is less than the size of the currently processed NVRAM Block then only the cyclic NvM_MainFunction() can finish the CRC calculation. This function might be called from interrupt level. The execution time depends mainly on the execution time of the CRC calculation of the current NVRAM Block. Therefore if it is called from interrupt level the configuration parameter NvMCrcNumOfBytes shall be set appropriately so that the CRC calculation is interrupted and finished by the NvM_MainFunction() which is called from Task level.
--	--

5.6.3.3.43. NvM_MainFunction

Purpose	Service for performing the processing of the NvM jobs.
Synopsis	<code>void NvM_MainFunction (void);</code>
Service ID	14
Sync/Async	Synchronous
Reentrancy	Reentrant
Description	The NvM_MainFunction() processes the asynchronous requests and must be called cyclically. First NvM_MainFunction() checks if a asynchronous request was inserted in the queue. If the queue is empty it returns immediately else the first state of the state machine of the asynchronous function is called. If the asynchronous request calls a function of an underlying module which cannot be processed synchronously then this asynchronous request is interrupted until the NvM_MainFunction() is notified that the underlying function is finished. If a CRC is calculated and the configuration parameter NvMCrcNumOfBytes is less than the size of the current NVRAM Block then the CRC calculation is also interrupted each time NvM- CrcNumOfBytes bytes of the data are processed. If the asynchronous request is finished the NvM_MainFunction() removes this request from the queue and calls the NvM Single Block Job End Notification callback if it is configured for the current NVRAM Block.

5.6.3.3.44. NvM_ReadAll

Purpose	Initiates a multi block read request.
Synopsis	<code>void NvM_ReadAll (void);</code>
Service ID	12
Sync/Async	Asynchronous
Reentrancy	Non-Reentrant
Production Errors	► NVM_E_QUEUE_OVERFLOW : thrown, if a new NVM request cannot be processed because the NVM queue is full.

Description	<p>NvM_ReadAll() copies the data of all configured NVM Blocks from Non Volatile Memory to the corresponding RAM Data Blocks if the following conditions are achieved for a NVRAM Block</p> <ul style="list-style-type: none"> ▶ a permanent RAM block is configured ▶ the configuration parameter NvMSelectBlockForReadAll is enabled ▶ the RAM Block Status is a:) invalid b:) valid and inconsistent and CRC calculation is enabled <p>A RAM Block is consistent when a block specific RAM Crc Block is assigned to the RAM Block, and the CRC calculated from the RAM Data Block is equal to the CRC stored in the RAM Crc Block. This means if the RAM Block already contains valid and consistent data then it is not overwritten by the NVM Block data. To invalidate a RAM Block X the service NvM_SetRamBlockStatus(X,FALSE) must be used. The advantage of this approach is that after a hot reset the NVM data is not copied again when the RAM Block data is still valid and consistent. Also Dataset Blocks are processed by NvM_ReadAll(). The user is responsible that the DataIndex is set correctly previous to calling NvM_ReadAll(). By default the DataIndex is set to zero by NvM_Init(). Before the data is copied NvM_ReadAll() reads the ConfigurationId which is stored in NVRAM Block number 1 (NvMNvBlockIdentifier=1) and compares this value with the Compiled Configuration Id. Depending on the result of this comparison and other configuration parameters 4 different cases can occur:</p> <ul style="list-style-type: none"> ▶ If the Configuration IDs match then NvM_ReadAll() continues with copying the data from NVRAM to RAM. In this case it can be assumed that the NVRAM data layout has not changed and the data stored last in NVRAM can be copied successfully. ▶ If the Configuration IDs do not match and parameter NvMDynamicConfiguration is disabled then NvM_ReadAll() also continues with copying the data. In this case it can be assumed that the NVRAM data layout has changed (e.g. new NVRAM Blocks were added or blocks were deleted or the length of a block was changed) and that the data stored last in NVRAM can not be copied successfully for all NVRAM Blocks. First NvM_ReadAll() tries to copy the data from NVRAM to RAM for each block which must be processed by NvM_ReadAll(). If an error is detected the ROM default data is loaded or the error status is set according the specification of NvM_ReadBlock(). The new Compiled Configuration ID is not written automatically by the next invocation of NvM_WriteAll(). Note: it should be avoided that a configuration ID is changed when NvMDynamic- Configuration is disabled as there is the risk that incorrect data could be interpreted as valid. If the read data is interpreted as valid and if NvMWriteBlockOnce is enabled too then the write protection is set automatically and valid data can not be written at all ! Therefore if the configuration ID is changed, NvMDynamicConfiguration should be enabled.
--------------------	--

- ▶ If the Configuration IDs do not match and parameter `NvMDynamicConfiguration` is enabled then [NvM_ReadAll\(\)](#) also continues with copying the data. In this case also the block specific configuration parameter `NvMResistantToChangedSw` is evaluated for each block. If it is disabled then ROM default data is loaded immediately and the block is not set to write protected even if `NvMWriteBlockOnce` was enabled. This means the data can be written again. If `NvMResistantToChangedSw` is enabled `NvM_ReadAll()` first tries to copy the data from NVRAM to RAM assuming that the address of this block in NVRAM has not changed. The new Compiled Configuration ID is written automatically by the next invocation of [NvM_WriteAll\(\)](#). - If the Configuration IDs do not match because the NVRAM Block of the Configuration ID is invalid (this can happen for example when the Configuration ID is read for the first time from an uninitialized NVRAM device) then [NvM_ReadAll\(\)](#) continues with copying the data from NVRAM to RAM. If an error is detected the ROM default data is loaded or the error status is set according to the specification of [NvM_ReadBlock\(\)](#). The new Compiled Configuration ID is written automatically by the next invocation of [NvM_WriteAll\(\)](#).

NOTE: A mismatch of the Configuration IDs can be caused also by an error in reading the Configuration ID from NVRAM. This leads to the same behavior as that of [NvM_ReadAll\(\)](#) when a mismatch of Compiled Configuration ID occurs.

Block specific Error-Status after service finished

- ▶ `NVM_REQ_OK` : The NVM Block was copied successfully or the ROM Block was copied successfully if a Dataset Block was requested and the `DataIndex` selects a ROM Block. The ROM Block was copied successfully in case the Configuration ID's don't match and `NvMDynamicConfiguration` is enabled and `NvMResistantToChangedSw` is disabled.
- ▶ `NVM_REQ_INVALIDATED` : The job result of an underlying abstraction module is `MEMIF_BLOCK_INVALID`.
- ▶ `NVM_REQ_INTEGRITY_FAILED` : If a CRC mismatch occurs or if the job result of the underlying memory abstraction module is `MEMIF_BLOCK_INCONSISTENT`.
- ▶ `NVM_REQ_CANCELLED`: An underlying memory abstraction module reports `MEMIF_JOB_CANCELED`. In this case [NvM_ReadAll\(\)](#) is not terminated completely, the service proceeds copying the next block.
- ▶ `NVM_REQ_NOT_OK`: The job result of the underlying memory abstraction module is `MEMIF_JOB_FAILED` or the ROM Block was not copied successfully if a Dataset Block was requested and the `DataIndex` selects a ROM Block.
- ▶ `NVM_REQ_BLOCK_SKIPPED`: The block was not processed because the conditions listed above were not achieved.

	<p>If an error is detected during processing NvM_ReadBlock() and the ROM default data must be copied by NvM_RestoreBlockDefaults() the ErrorStatus set by NvM_RestoreBlockDefaults() is ignored and the above ErrorStatus applies nevertheless.</p> <p>Multi request ErrorStatus after service finished (stored in NVRAM Block 0)</p> <ul style="list-style-type: none"> ▶ NVM_REQ_OK: Each Block specific ErrorStatus is NVM_REQ_OK or NVM_REQ_SKIPPED. ▶ NVM_REQ_NOT_OK: The Block specific ErrorStatus of at least one NVRAM Block is neither NVM_REQ_OK nor NVM_REQ_BLOCK_SKIPPED. <p>Block specific RAMBlock-Status after service finished (applies only for permanent RAM Blocks)</p> <ul style="list-style-type: none"> ▶ VALID / UNCHANGED: A NVM Block was copied successfully to the RAM Block. ▶ VALID / CHANGED: A ROM Block was copied successfully to the RAM Block. ▶ INVALID / UNCHANGED: An error was detected during copying the NVM Block to the RAM Block and no ROM Block is configured. ▶ INVALID / CHANGED: Can not occur. <p>A ROM Block is copied if a Dataset Block is requested and the DataIndex selects a ROM Block, or an error is detected during copying the NVM Block and the default ROM Block data is copied, or the Configuration Id's don't match and NvMDynamic-Configuration is enabled and NvMResistantToChangedSw is disabled.</p>
--	---

5.6.3.3.45. NvM_ReadBlockHook

Purpose	Hook function for ReadBlock.	
Synopsis	<pre>void NvM_ReadBlockHook (uint16 BlockNum , uint8 * RamBlock-DataAddress , uint16 BlockLength);</pre>	
Sync/Async	Synchronous	
Parameters (in)	BlockNum	The block identifier of the currently processed block.
	RamBlockDataAddress	Pointer to a permanent RAM block. If no permanent RAM block is configured, it returns NULL_PTR
	BlockLength	Length of the currently processed block
Description	The Nvm_ReadBlockHook() provides a pre-read hook functionality. After reading the data from the underlying layer, the hook function is called.	

5.6.3.3.46. NvM_SetBlockLockStatus

Purpose	Service for setting the lock status of the NV block of an NVRAM block.	
Synopsis	<pre>void NvM_SetBlockLockStatus (NvM_ASR40_BlockIdType BlockId , boolean BlockLocked);</pre>	
Service ID	19	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	BlockId	The block identifier. Selects the block whose configuration parameter NvMN-vBlockIdentifier is equal to BlockId.
	BlockLocked	TRUE: Mark the NV block as locked. FALSE: Mark the NV block as unlocked.
Description	<p>If the API is called with parameter Locked as TRUE, the NV contents associated to the NVRAM block identified by BlockId cannot be modified by any other requests. The Block is skipped during NvM_WriteAll and other requests such as NvM_WriteBlock, NvM_InvalidateNvBlock and NvM_EraseNvBlock are rejected. At next start-up, during processing of NvM_ReadAll, this NVRAM block will be loaded from NV memory.</p> <p>If the API is called with parameter Locked as FALSE, this NVRAM block will be processed normally.</p> <p>Note 1: The setting made using this service cannot be changed by NvM_SetRam-BlockStatus or NvM_SetBlockProtection.</p> <p>Note 2: This service can only be used by BSW Components. It cannot be accessed via RTE.</p>	

5.6.3.3.47. NvM_ValidateAll

Purpose	Initiates a multi block validation request.
Synopsis	<pre>void NvM_ValidateAll (void);</pre>
Service ID	19
Sync/Async	Asynchronous
Reentrancy	Non-Reentrant

Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full.
Description	<p>NvM_ValidateAll() sets the RAM Block status to VALID /CHANGED and triggers a CRC recalculation if it is configured for the block, if the following conditions are achieved for a NVRAM Block:</p> <ul style="list-style-type: none"> ▶ a permanent RAM block, or explicit synchronization is configured ▶ the configuration parameter NvMBlockuseAutovalidation is enabled <p>Block specific Error-Status after service finished</p> <ul style="list-style-type: none"> ▶ NVM_REQ_OK : The NVM Block was validated successfully and CRC was recalculated if it was configured for block ▶ NVM_REQ_NOT_OK: The block was not validated, number of mirror retry operations is exceeded ▶ NVM_REQ_BLOCK_SKIPPED: The block was not processed because the conditions for ValidateAll were not achieved. <p>Multi request ErrorStatus after service finished (stored in NVRAM Block 0)</p> <ul style="list-style-type: none"> ▶ NVM_REQ_OK: Each Block specific ErrorStatus is NVM_REQ_OK or NVM_REQ_SKIPPED. ▶ NVM_REQ_NOT_OK: The Block specific ErrorStatus of at least one NVRAM Block is neither NVM_REQ_OK nor NVM_REQ_BLOCK_SKIPPED. <p>Block specific RAMBlock-Status after service finished (applies for permanent RAM and Explicit Synchronization Blocks)</p> <ul style="list-style-type: none"> ▶ VALID / UNCHANGED: A NvRAM Block is not validated. ▶ VALID / CHANGED: A NvRAM Block is validated. ▶ INVALID / UNCHANGED: A NvRAM Block is not validated. ▶ INVALID / CHANGED: Can not occur.

5.6.3.3.48. NvM_WriteAll

Purpose	Initiates a multi block write request.
Synopsis	<code>void NvM_WriteAll (void);</code>
Service ID	13

Sync/Async	Asynchronous
Reentrancy	Non-Reentrant
Production Errors	<ul style="list-style-type: none"> ▶ NVM_E_QUEUE_OVERFLOW: thrown, if a new NVM request cannot be processed because the NVM queue is full.
Description	<p>NvM_WriteAll() copies the data of all configured blocks except block 0 from the RAM-DataBlock to the corresponding NVM Block if the following conditions are true:</p> <ul style="list-style-type: none"> ▶ if a block has assigned a permanent RAM data block ▶ if the block was marked as changed by function NvM_SetRamBlockStatus() ▶ if the block is not write protected ▶ if the block is valid <p>If one of these conditions is false the block is skipped. The status of a skipped block is set to NVM_REQ_SKIPPED. If a block is configured with NvMWriteBlockOnce the block is marked as write protected after the data is written. If NvM_WriteAll() is cancelled by NvM_CancelWriteAll() the data of the currently processed NVRAM block will still be written and afterwards NvM_WriteAll() terminates and sets the multi request job result to NVM_REQ_CANCELLED. If one of the processed blocks can not be written successfully the multi request job result of NvM_WriteAll() is set to NVM_REQ_NOT_OK.</p>

5.6.3.3.49. NvM_WriteBlockHook

Purpose	Hook function for WriteBlock.	
Synopsis	<pre>void NvM_WriteBlockHook (uint16 BlockNum , uint8 * RamBlock-DataAddress , uint16 BlockLength);</pre>	
Sync/Async	Synchronous	
Parameters (in)	BlockNum	The block identifier of the currently processed block.
	RamBlockDataAddress	Pointer to a permanent RAM block. If no permanent RAM block is configured, it returns NULL_PTR
	BlockLength	Length of the currently processed block
Description	The Nvm_WriteBlockHook() provides a post-write hook functionality. Before writing the data to the underlying layer, the hook function is called.	

5.6.4. Integration notes

5.6.4.1. Exclusive areas

This section describes the exclusive areas used by the `NvM` module.

5.6.4.1.1. SCHM_NVM_EXCLUSIVE_AREA_0

Protected data structures	All shared data that shall be protected from mutual access.
Recommended locking mechanism	This exclusive area must always be protected by a locking mechanism. The options for locking are described in the <code>EB tresos AutoCore Generic</code> documentation. Refer to the section <code>Mapping exclusive areas in the basic software modules</code> in the <code>Integration notes</code> section for details.

5.6.4.2. Production errors

NVM_E_INTEGRITY_FAILED	<ul style="list-style-type: none"> ▶ NvM_ASR32_ReadBlock ▶ NvM_ASR40_ReadBlock ▶ NvM_ASR42_ReadBlock ▶ NvM_ASR42_ReadPRAMBlock
NVM_E_LOSS_OF_REDUNDANCY	<ul style="list-style-type: none"> ▶ NvM_ASR32_ReadBlock ▶ NvM_ASR40_ReadBlock ▶ NvM_ASR42_ReadBlock ▶ NvM_ASR42_ReadPRAMBlock
NVM_E_QUEUE_OVERFLOW	<ul style="list-style-type: none"> ▶ NvM_ASR32_EraseNvBlock ▶ NvM_ASR32_InvalidateNvBlock ▶ NvM_ASR32_ReadBlock ▶ NvM_ASR32_RestoreBlockDefaults ▶ NvM_ASR32_WriteBlock ▶ NvM_ASR40_EraseNvBlock ▶ NvM_ASR40_InvalidateNvBlock ▶ NvM_ASR40_ReadBlock

	<ul style="list-style-type: none"> ▶ NvM_ASR40_RestoreBlockDefaults ▶ NvM_ASR40_WriteBlock ▶ NvM_ASR42_EraseNvBlock ▶ NvM_ASR42_InvalidateNvBlock ▶ NvM_ASR42_ReadBlock ▶ NvM_ASR42_ReadPRAMBlock ▶ NvM_ASR42_RestoreBlockDefaults ▶ NvM_ASR42_RestorePRAMBlockDefaults ▶ NvM_ASR42_WriteBlock ▶ NvM_ASR42_WritePRAMBlock ▶ NvM_FirstInitAll ▶ NvM_ReadAll ▶ NvM_ValidateAll ▶ NvM_WriteAll
NVM_E_REQ_FAILED	<ul style="list-style-type: none"> ▶ NvM_ASR32_EraseNvBlock ▶ NvM_ASR32_InvalidateNvBlock ▶ NvM_ASR32_ReadBlock ▶ NvM_ASR32_WriteBlock ▶ NvM_ASR40_EraseNvBlock ▶ NvM_ASR40_InvalidateNvBlock ▶ NvM_ASR40_ReadBlock ▶ NvM_ASR40_WriteBlock ▶ NvM_ASR42_EraseNvBlock ▶ NvM_ASR42_InvalidateNvBlock ▶ NvM_ASR42_ReadBlock ▶ NvM_ASR42_ReadPRAMBlock ▶ NvM_ASR42_WriteBlock ▶ NvM_ASR42_WritePRAMBlock
NVM_E_VERIFY_FAILED	<ul style="list-style-type: none"> ▶ NvM_ASR32_WriteBlock ▶ NvM_ASR40_WriteBlock ▶ NvM_ASR42_WriteBlock ▶ NvM_ASR42_WritePRAMBlock

NVM_E_WRITE_PROTECTED	<ul style="list-style-type: none"> ▶ NvM_ASR32_EraseNvBlock ▶ NvM_ASR32_InvalidateNvBlock ▶ NvM_ASR32_WriteBlock ▶ NvM_ASR40_EraseNvBlock ▶ NvM_ASR40_InvalidateNvBlock ▶ NvM_ASR40_WriteBlock ▶ NvM_ASR42_EraseNvBlock ▶ NvM_ASR42_InvalidateNvBlock ▶ NvM_ASR42_WriteBlock ▶ NvM_ASR42_WritePRAMBlock
NVM_E_WRONG_BLOCK_ID	<ul style="list-style-type: none"> ▶ NvM_ASR32_ReadBlock ▶ NvM_ASR40_ReadBlock ▶ NvM_ASR42_ReadBlock ▶ NvM_ASR42_ReadPRAMBlock

5.6.4.3. Memory mapping

General information about memory mapping is provided in the EB tresos AutoCore Generic documentation. Refer to the section `Memory mapping and compiler abstraction` in the `Integration notes` section for details.

The following table provides the list of sections that may be mapped for this module:

Memory section
CODE
MC_SHARED_CODE
VAR_POWER_ON_INIT_8
ADMINBLOCK_UNSPECIFIED
MC_SHARED_VAR_POWER_ON_INIT_UNSPECIFIED
VAR_INIT_8
MC_SHARED_VAR_INIT_16
VAR_CLEARED_8
VAR_APPL_DATA
VAR_INIT_16

VAR_CLEARED_16
VAR_INIT_UNSPECIFIED
VAR_CLEARED_UNSPECIFIED
CONFIG_DATA_APPL_DATA
CONFIG_DATA_16
CONFIG_DATA_UNSPECIFIED
MC_SHARED_CONFIG_DATA_UNSPECIFIED
CONFIG_DATA_APPL_CONST
VAR_INIT_32
MC_SHARED_VAR_CLEARED_16

5.6.4.4. Integration requirements

WARNING



Integration requirements list is not exhaustive

The following list of integration requirements helps you to integrate your product. However, this list is not exhaustive. You also require information from the user's guide, release notes, and EB tresos AutoCore known issues to successfully integrate your product.

5.6.4.4.1. lim.NvM.EB_INTREQ_NvM_0001

Description	The EB memory stack modules NvM, Ea and Fee make only limited use of the call-back calls from their underlying modules. Make sure during the integration that the NvM, Ea, and Fee main functions are only called from the same task context so that they cannot preempt each other.
Rationale	This approach enables a simple and lock free implementation that results in smaller code.

5.6.4.4.2. lim.NvM.EB_INTREQ_NvM_0002

Description	If 'BlockCheck' mechanism is used and the API to disable it 'NvM_DisableBlockCheckMechanism' is used. This API shall never be called concurrent with the NvM_MainFunction.
Rationale	The purpose of 'NvM_DisableBlockCheckMechanism' is to optimize the NvM_MainFunction runtime during initialization. For this to work as expected the function shall be called to disable the 'BlockCheck' mechanism right after ReadAll is called and before

	the main function calls start. After ReadAll is completed the function shall be called to enable back the mechanism. The same usage can be done to speed up the WriteAll.
--	---

5.6.4.4.3. lim.NvM.EB_INTREQ_NvM_0003

Description	If legacy symbolic names (from AUTOSAR versions 3.x or lower than 4.0.3) are used then the macro NVM_PROVIDE_LEGACY_SYMBOLIC_NAMES shall be defined before including the NvM header file.
Rationale	The usage of legacy symbolic names is discouraged, but for backwards compatibility the usage of these symbolic names is provided. Recommendation is to go for AUTOSAR 4.0.3 symbolic names (See TPS_ECUC_02108 from AUTOSAR_TPS_ECU-Configuration.pdf) and not enable the macro.