



BSI Standards Publication

Road vehicles — Open Test sequence eXchange format (OTX)

Part 4: Expanded extensions interface definition

National foreword

This British Standard is the UK implementation of ISO 13209-4:2021.

The UK participation in its preparation was entrusted to Technical Committee AUE/16, Data Communication (Road Vehicles).

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2021
Published by BSI Standards Limited 2021

ISBN 978 0 580 51658 0

ICS 43.040.15; 43.180

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 May 2021.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

INTERNATIONAL STANDARD

ISO
13209-4

First edition
2021-05-20

Road vehicles — Open Test sequence eXchange format (OTX) —

Part 4: Expanded extensions interface definition

*Véhicules routiers — Format public d'échange de séquence-
tests (OTX) —*

Partie 4: Définition de l'interface des extensions étendues



Reference number
ISO 13209-4:2021(E)

AVA Documents
Document Sharing Hub
© ISO 2021



COPYRIGHT PROTECTED DOCUMENT

© ISO 2021, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

Foreword	xii
Introduction	xiii
1 Scope	1
2 Normative references	1
3 Terms, definitions and abbreviated terms	1
3.1 Terms and definitions	1
3.2 Abbreviated terms	1
4 Requirements	2
4.1 Basic principles for requirements definition	2
4.2 Requirement listing	2
4.2.1 Assertion requirements	2
4.2.2 BlackBox requirements	2
4.2.3 BusMonitoring requirements	2
4.2.4 ComInterface requirements	3
4.2.5 CommonDialogs requirements	3
4.2.6 DataType requirements	3
4.2.7 DiagComPlus requirements	4
4.2.8 DiagConfiguration requirements	4
4.2.9 DiagDataBrowsingPlus requirements	5
4.2.10 ECUConfiguration requirements	6
4.2.11 EventPlus requirements	6
4.2.12 ExternalServiceProvider requirements	7
4.2.13 File requirements	7
4.2.14 FlashPlus requirements	7
4.2.15 Persistence requirements	8
4.2.16 SQL requirements	8
4.2.17 StateMachineProcedure requirements	9
4.2.18 StateVariable requirements	9
4.2.19 TestResultHandling requirements	9
4.2.20 Util requirements	10
4.2.21 VehicleInfo requirements	10
4.2.22 XML requirements	11
4.2.23 ZipHandling requirements	12
5 Extension overview	13
5.1 General	13
5.2 Dependencies	14
5.3 OTX packaging (PTX)	18
6 OTX Assertion extension	18
6.1 General	18
6.2 Exceptions	18
6.2.1 Exceptions overview	18
6.2.2 Exceptions syntax	18
6.2.3 Exceptions semantics	19
6.3 Actions	19
6.3.1 Actions overview	19
6.3.2 Actions syntax	19
6.3.3 Actions general semantics	20
7 OTX BlackBox extension	20
7.1 General	20
7.2 Datatypes	20
7.2.1 Datatypes overview	20
7.2.2 Datatypes syntax	20

	7.2.3	Datatypes semantics.....	21
7.3		Variable access.....	21
	7.3.1	Variable-access overview.....	21
	7.3.2	Variable-access syntax.....	21
	7.3.3	Variable-access semantics.....	22
7.4		Terms.....	22
	7.4.1	Terms overview.....	22
	7.4.2	Terms syntax.....	22
	7.4.3	Terms general semantics.....	23
8		OTX BusMonitoring extension.....	24
8.1		General.....	24
8.2		Datatypes.....	24
	8.2.1	Datatypes overview.....	24
	8.2.2	Datatypes syntax.....	24
	8.2.3	Datatypes semantics.....	25
8.3		Enumerations.....	26
	8.3.1	Enumerations syntax.....	26
	8.3.2	Enumerations semantics.....	26
8.4		Exceptions.....	27
	8.4.1	Exceptions overview.....	27
	8.4.2	Exceptions syntax.....	27
	8.4.3	Exceptions semantics.....	27
8.5		Variable access.....	28
	8.5.1	Variable-access overview.....	28
	8.5.2	Variable-access syntax.....	28
	8.5.3	Variable-access semantics.....	28
8.6		Actions.....	28
	8.6.1	Actions overview.....	28
	8.6.2	Actions syntax.....	28
	8.6.3	Actions general semantics.....	29
8.7		Terms.....	31
	8.7.1	Terms overview.....	31
	8.7.2	Terms syntax.....	31
	8.7.3	Terms general semantics.....	31
9		OTX ComInterface extension.....	37
9.1		General.....	37
9.2		Datatypes.....	37
	9.2.1	Datatypes overview.....	37
	9.2.2	Datatypes syntax.....	37
	9.2.3	Datatypes semantics.....	38
9.3		Enumerations.....	38
	9.3.1	Enumerations syntax.....	38
	9.3.2	Enumerations semantics.....	39
9.4		Exceptions.....	39
	9.4.1	Exceptions overview.....	39
	9.4.2	Exceptions syntax.....	40
	9.4.3	Exceptions semantics.....	40
9.5		Variable access.....	41
	9.5.1	Variable-access overview.....	41
	9.5.2	Variable-access syntax.....	41
	9.5.3	Variable-access semantics.....	42
9.6		Actions.....	42
	9.6.1	Actions overview.....	42
	9.6.2	Actions syntax.....	42
	9.6.3	Actions general semantics.....	43
9.7		Terms.....	46
	9.7.1	Terms overview.....	46

9.7.2	Terms syntax.....	46
9.7.3	Semantics for ComInterface terms.....	47
9.7.4	Semantics for Enumeration terms.....	57
9.7.5	Terms general semantics.....	58
10	OTX CommonDialogs extension.....	58
10.1	General.....	58
10.2	Exceptions.....	59
10.2.1	Exceptions overview.....	59
10.2.2	Exceptions syntax.....	59
10.2.3	Exceptions semantics.....	59
10.3	Actions.....	59
10.3.1	Actions overview.....	59
10.3.2	Actions syntax.....	59
10.3.3	Actions general semantics.....	60
11	OTX DataType extension.....	63
11.1	General.....	63
11.2	Datatypes.....	63
11.2.1	Datatypes overview.....	63
11.2.2	Datatypes syntax.....	63
11.2.3	Datatypes semantics.....	64
11.3	Exceptions.....	66
11.3.1	Exceptions overview.....	66
11.3.2	Exceptions syntax.....	66
11.3.3	Exceptions semantics.....	67
11.4	Variable access.....	67
11.4.1	Variable-access overview.....	67
11.4.2	Variable-access syntax.....	67
11.4.3	Variable-access semantics.....	68
11.5	Declaration and arguments.....	68
11.5.1	Declaration and arguments syntax.....	68
11.5.2	Declaration and arguments semantics.....	69
11.6	Signatures.....	69
11.6.1	Signatures overview.....	69
11.6.2	Signatures syntax.....	69
11.6.3	Semantics for Enumerations.....	70
11.6.4	Semantics for Structures.....	71
11.7	Terms.....	72
11.7.1	Terms overview.....	72
11.7.2	Terms syntax.....	72
11.7.3	Semantics for EnumerationTerms.....	74
11.7.4	Semantics for ResourceLocationTerms.....	77
11.7.5	Semantics for StructureTerms.....	79
12	OTX DiagComPlus extension.....	80
12.1	General.....	80
12.2	Datatypes.....	81
12.2.1	Datatypes overview.....	81
12.2.2	Datatypes syntax.....	81
12.2.3	Datatypes semantics.....	81
12.3	Enumerations.....	82
12.3.1	Enumerations syntax.....	82
12.3.2	Enumerations semantics.....	82
12.4	Exceptions.....	83
12.4.1	Exceptions overview.....	83
12.4.2	Exceptions syntax.....	83
12.4.3	Exceptions semantics.....	83
12.5	Variable access.....	83
12.5.1	Variable-access overview.....	83

12.5.2	Variable-access syntax	84
12.5.3	Variable-access semantics	84
12.6	Actions	84
12.6.1	Actions overview	84
12.6.2	Actions syntax	84
12.6.3	Actions general semantics	85
12.7	Terms	87
12.7.1	Terms overview	87
12.7.2	Terms syntax	87
12.7.3	Terms general semantics	87
13	OTX DiagConfiguration extension	91
13.1	General	91
13.2	Exceptions	91
13.2.1	Exceptions overview	91
13.2.2	Exceptions syntax	91
13.2.3	Exceptions semantics	92
13.3	Actions	92
13.3.1	Actions overview	92
13.3.2	Actions syntax	92
13.3.3	Actions general semantics	93
13.4	Terms	94
13.4.1	Terms overview	94
13.4.2	Terms syntax	94
13.4.3	Terms general semantics	95
14	OTX DiagDataBrowsingPlus extension	97
14.1	General	97
14.2	Datatypes	97
14.2.1	Datatypes overview	97
14.2.2	Datatypes syntax	97
14.2.3	Datatypes semantics	99
14.3	Enumerations	102
14.3.1	Enumerations syntax	102
14.3.2	Enumerations semantics	103
14.4	Exceptions	106
14.4.1	Exceptions overview	106
14.4.2	Exceptions syntax	106
14.4.3	Exceptions semantics	106
14.5	Variable access	107
14.5.1	Variable-access overview	107
14.5.2	Variable-access syntax	107
14.5.3	Variable-access semantics	108
14.6	Terms	108
14.6.1	Terms overview	108
14.6.2	Semantics for DbComChannel terms	108
14.6.3	Semantics for DbDiagService terms	111
14.6.4	Semantics for DbDiagTroubleCode terms	117
14.6.5	Semantics for DbEcuVariant terms	122
14.6.6	Semantics for DbEnvDataDesc terms	122
14.6.7	Semantics for DbFaultMemory terms	125
14.6.8	Semantics for DbMatchingParameter terms	127
14.6.9	Semantics for DbObject terms	128
14.6.10	Semantics for DbParameter terms	132
14.6.11	Semantics for DbRequest terms	146
14.6.12	Semantics for DbResponse terms	150
14.6.13	Semantics for DbSpecialDataElement terms	154
14.6.14	Semantics for DbSpecialDataGroup terms	156
14.6.15	Semantics for DbSubComponent terms	160

	14.6.16	Semantics for DbTable terms	166
	14.6.17	Semantics for interval terms	171
	14.6.18	Semantics for McdDataType terms	174
	14.6.19	Semantics for McdParameter terms	175
	14.6.20	Semantics for McdResponseType terms	177
	14.6.21	Semantics for parameter terms	179
	14.6.22	Semantics for TextTableElement terms	183
15		OTX EcuConfiguration extension	185
	15.1	General	185
	15.2	Datatypes	185
	15.2.1	Datatypes overview	185
	15.2.2	Datatypes syntax	186
	15.2.3	Datatypes semantics	186
	15.3	Exceptions	187
	15.3.1	Exceptions overview	187
	15.3.2	Exceptions syntax	187
	15.3.3	Exceptions semantics	187
	15.4	Variable access	187
	15.4.1	Variable-access overview	187
	15.4.2	Variable-access syntax	187
	15.4.3	Variable-access semantics	188
	15.5	Terms	188
	15.5.1	Terms overview	188
	15.5.2	Semantics for DbConfigurationData terms	189
	15.5.3	Semantics for DbConfigurationRecord terms	190
	15.5.4	Semantics for DbDataRecord terms	192
	15.5.5	Semantics for DbItemValue terms	198
	15.5.6	Semantics for DbOptionItem terms	201
16		OTX EventPlus extension	208
	16.1	General	208
	16.2	Exceptions	209
	16.2.1	Exceptions overview	209
	16.2.2	Exceptions syntax	209
	16.2.3	Exceptions semantics	209
	16.3	Terms	210
	16.3.1	Terms overview	210
	16.3.2	Terms syntax	210
	16.3.3	Terms general semantics	210
17		OTX ExternalServiceProvider extension	211
	17.1	General	211
	17.2	Datatypes	212
	17.2.1	Datatypes overview	212
	17.2.2	Datatypes syntax	212
	17.2.3	Datatypes semantics	213
	17.3	Enumerations	214
	17.3.1	Enumerations syntax	214
	17.3.2	Enumerations semantics	214
	17.4	Exceptions	214
	17.4.1	Exceptions overview	214
	17.4.2	Exceptions syntax	214
	17.4.3	Exceptions semantics	215
	17.5	Variable access	216
	17.5.1	Variable-access overview	216
	17.5.2	Variable-access syntax	216
	17.5.3	Variable-access semantics	217
	17.6	Declaration and arguments	217
	17.6.1	Declaration and arguments syntax	217

17.6.2	Declaration and arguments semantics	218
17.7	Signatures	219
17.7.1	Signatures overview	219
17.7.2	Signatures syntax	219
17.7.3	Signatures general semantics	219
17.8	Actions	226
17.8.1	Actions overview	226
17.8.2	Actions syntax	226
17.8.3	Actions general semantics	228
17.9	Terms	233
17.9.1	Terms overview	233
17.9.2	Terms syntax	234
17.9.3	Semantics for Enumeration terms	235
17.9.4	Semantics for External Service provider event query terms	236
17.9.5	Semantics for External service provider event source terms	237
17.9.6	General semantics	238
18	OTX File extension	241
18.1	General	241
18.2	Datatypes	242
18.2.1	Datatypes overview	242
18.2.2	Datatypes syntax	242
18.2.3	Datatypes semantics	242
18.3	Exceptions	243
18.3.1	Exceptions overview	243
18.3.2	Exceptions syntax	243
18.3.3	Exceptions semantics	244
18.4	Variable access	245
18.4.1	Variable-access overview	245
18.4.2	Variable-access syntax	245
18.4.3	Variable-access semantics	246
18.5	Actions	246
18.5.1	Actions overview	246
18.5.2	Actions syntax	246
18.5.3	Actions general semantics	248
18.6	Terms	251
18.6.1	Terms overview	251
18.6.2	Terms syntax	251
18.6.3	Terms general semantics	253
19	OTX FlashPlus extension	262
19.1	General	262
19.2	Exceptions	263
19.2.1	Exceptions overview	263
19.2.2	Exceptions syntax	263
19.2.3	Exceptions semantics	263
19.3	Actions	263
19.3.1	Actions overview	263
19.3.2	Actions syntax	263
19.3.3	Actions general semantics	264
19.4	Terms	265
19.4.1	Terms overview	265
19.4.2	Terms syntax	265
19.4.3	Terms general semantics	265
20	OTX Persistence extension	267
20.1	General	267
20.2	Exceptions	268
20.2.1	Exceptions overview	268
20.2.2	Exceptions syntax	268

20.2.3	Exceptions semantics.....	268
20.3	Actions.....	269
20.3.1	Actions overview.....	269
20.3.2	Actions syntax.....	269
20.3.3	Actions general semantics.....	269
21	OTX SQL extension.....	270
21.1	General.....	270
21.2	Datatypes.....	271
21.2.1	Datatypes overview.....	271
21.2.2	Datatypes syntax.....	271
21.2.3	Datatypes semantics.....	271
21.3	Exceptions.....	272
21.3.1	Exceptions overview.....	272
21.3.2	Exceptions syntax.....	272
21.3.3	Exceptions semantics.....	272
21.4	Variable access.....	273
21.4.1	Variable-access overview.....	273
21.4.2	Variable-access syntax.....	273
21.4.3	Variable-access semantics.....	273
21.5	Actions.....	274
21.5.1	Actions overview.....	274
21.5.2	Actions syntax.....	274
21.5.3	Actions general semantics.....	274
21.6	Terms.....	276
21.6.1	Terms overview.....	276
21.6.2	Terms syntax.....	276
21.6.3	Terms general semantics.....	277
22	OTX StateMachineProcedure extension.....	283
22.1	General.....	283
22.2	Datatypes.....	285
22.2.1	Datatypes overview.....	285
22.2.2	Datatypes syntax.....	285
22.2.3	Datatypes semantics.....	286
22.3	Procedures.....	288
22.3.1	Procedures overview.....	288
22.3.2	Procedures syntax.....	288
22.3.3	Procedures general semantics.....	289
23	OTX StateVariable extension.....	292
23.1	General.....	292
23.2	Declaration and arguments.....	293
23.2.1	Declaration and arguments syntax.....	293
23.2.2	Declaration and arguments semantics.....	293
24	OTX TestResultHandling extension.....	293
24.1	General.....	293
24.2	Datatypes.....	294
24.2.1	Datatypes overview.....	294
24.2.2	Datatypes syntax.....	294
24.2.3	Datatypes semantics.....	295
24.3	Enumerations.....	297
24.3.1	Enumerations syntax.....	297
24.3.2	Enumerations semantics.....	298
24.4	Exceptions.....	299
24.4.1	Exceptions overview.....	299
24.4.2	Exceptions syntax.....	299
24.4.3	Exceptions semantics.....	300
24.5	Variable access.....	300

24.5.1	Variable-access overview	300
24.5.2	Variable-access syntax	300
24.5.3	Variable-access semantics	301
24.6	Declaration and arguments	301
24.6.1	Declaration and arguments syntax	301
24.6.2	Declaration and arguments semantics	302
24.7	Actions	302
24.7.1	Actions overview	302
24.7.2	Actions syntax	302
24.7.3	Actions general semantics	304
24.8	Terms	312
24.8.1	Terms overview	312
24.8.2	Terms syntax	313
24.8.3	Semantics for Enumeration terms	314
24.8.4	Terms general semantics	317
25	OTX Util extension	319
25.1	General	319
25.2	Exceptions	319
25.2.1	Exceptions overview	319
25.2.2	Exceptions syntax	320
25.2.3	Exceptions semantics	320
25.3	Terms	321
25.3.1	Terms overview	321
25.3.2	Terms syntax	321
25.3.3	Semantics for Util terms	323
26	OTX VehicleInfo extension	329
26.1	General	329
26.2	Datatypes	330
26.2.1	Datatypes overview	330
26.2.2	Datatypes syntax	330
26.2.3	Datatypes semantics	330
26.3	Enumerations	330
26.3.1	Enumerations syntax	330
26.3.2	Enumerations semantics	331
26.4	Exceptions	331
26.4.1	Exceptions overview	331
26.4.2	Exceptions syntax	331
26.4.3	Exceptions semantics	332
26.5	Variable access	332
26.5.1	Variable-access overview	332
26.5.2	Variable-access syntax	333
26.5.3	Variable-access semantics	333
26.6	Terms	333
26.6.1	Terms overview	333
26.6.2	Terms syntax	333
26.6.3	Terms general semantics	334
27	OTX XML extension	336
27.1	General	336
27.2	Datatypes	337
27.2.1	Datatypes overview	337
27.2.2	Datatypes syntax	337
27.2.3	Datatypes semantics	337
27.3	Exceptions	337
27.3.1	Exceptions overview	337
27.3.2	Exceptions syntax	338
27.3.3	Exceptions semantics	338
27.4	Variable access	339

27.4.1	Variable-access overview	339
27.4.2	Variable-access syntax	339
27.4.3	Variable-access semantics	339
27.5	Actions	339
27.5.1	Actions overview	339
27.5.2	Actions syntax	339
27.5.3	Actions general semantics	341
27.6	Terms	345
27.6.1	Terms overview	345
27.6.2	Terms syntax	345
27.6.3	Terms general semantics	347
28	OTX ZipHandling extension	353
28.1	General	353
28.2	Exceptions	353
28.2.1	Exceptions overview	353
28.2.2	Exceptions syntax	353
28.2.3	Exceptions semantics	354
28.3	Actions	355
28.3.1	Actions overview	355
28.3.2	Actions syntax	355
28.3.3	Actions general semantics	356
Annex A (normative) Comprehensive checker rule listing		360
Annex B (normative) PTX file		368
Bibliography		378

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 31, *Data communication*.

A list of all parts in the ISO 13209 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Diagnostic test sequences are utilized whenever automotive components or functions with diagnostic abilities are being diagnosed, tested, reprogrammed or initialised by off-board test equipment. Test sequences define the succession of interactions between the user (i.e. workshop or assembly line staff), the diagnostic application (the test equipment) and the vehicle communication interface as well as any calculations and decisions that have to be carried out. Test sequences provide a means to define interactive, guided diagnostics or similar test logic.

Today, the automotive industry mainly relies on paper documentation and/or proprietary authoring environments to document and to implement such test sequences for a specific test application. An author who is setting up engineering, assembly line or service diagnostic test applications implements the required test sequences manually, supported by non-uniform test sequence documentation, most likely using different authoring applications and formats for each specific test application. This redundant effort can be greatly reduced if processes and tools support the OTX concept.

The ISO 13209 series proposes an open and standardized format for the human- and machine-readable description of diagnostic test sequences. The format supports the requirements of transferring diagnostic test sequence logic uniformly between electronic system suppliers, vehicle manufacturers and service dealerships/repair shops.

This document extends the core of the ISO 13209 series with a set of additional, generally usable OTX extensions which are not part of [ISO 13209-3](#), using the extension mechanism rules described in [ISO 13209-2](#).

[Figure 1](#) gives an overview of the additional extensions related to the existing extensions in [ISO 13209-3](#).

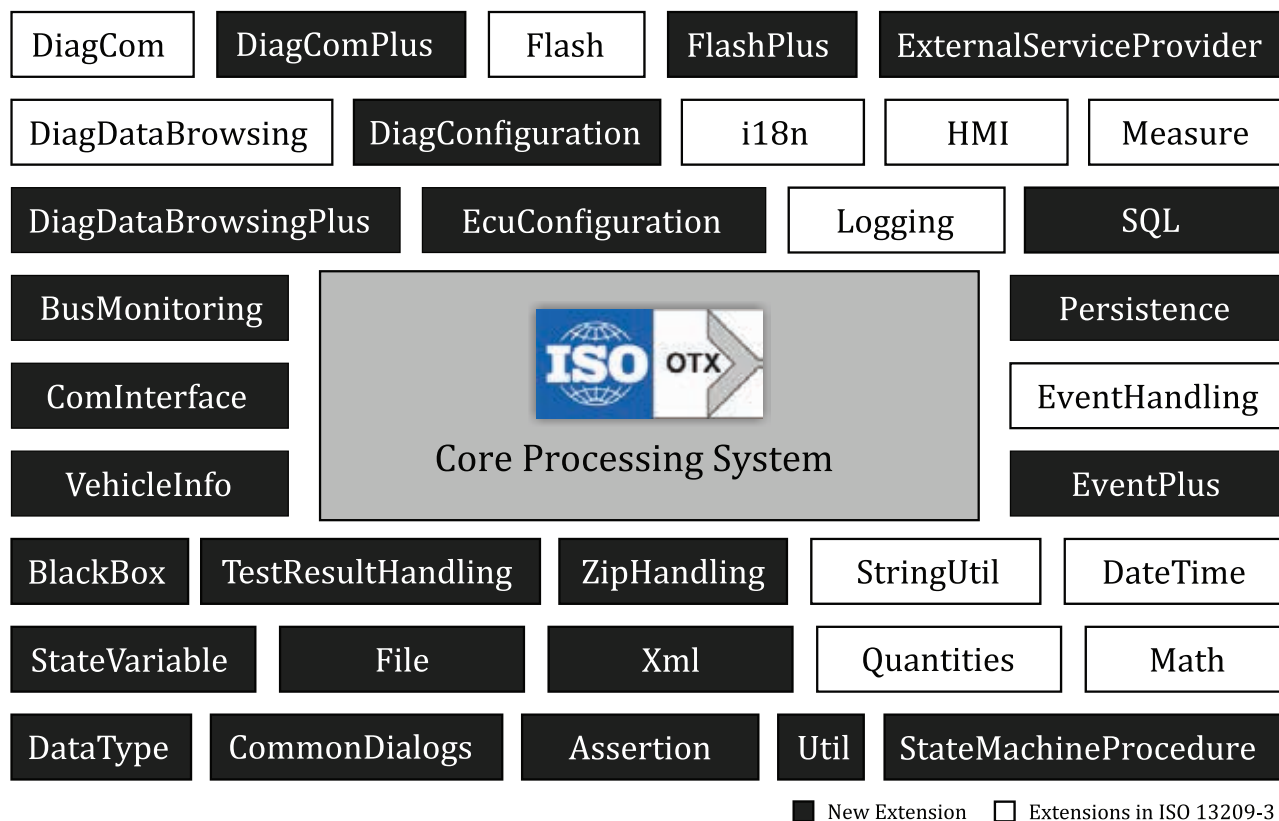


Figure 1 — OTX extensions overview

Besides these extensions a new container format, called PTX, is introduced (see [Annex B](#)).

The main goals are:

- enlargement of the field of application,
- ensuring exchangeability between different tools and suppliers,
- improvement of OTX code quality,
- improvement of standard acceptance.

The following common requirements are covered:

- general read-write access to files,
- XML processing,
- declare data, whose internal properties are unknown,
- deep change monitoring in lists or maps,
- store arbitrary information on the current runtime,
- handling of user-defined data types: structure and enumeration,
- ensure correct implementation of OTX test sequences,
- support of DoIP,
- configuration tasks,
- browsing of ODX database,
- extend OTX to functionality that is encapsulated in external services, part of a system, a device, a database or a simple library,
- functional access to ECUs,
- capturing, evaluating and persisting results of test sequences,
- transport status information from inside a sequence to the environment,
- advanced convenience functionality,
- collection of communication data at runtime,
- select and modify project and vehicle information's for MVCI systems,
- support of variant coding,
- enable late bound flashing of ECUs,
- access to SQL databases,
- evaluating test sequence results in a structured way,
- access to information about vehicle network architecture,
- state machine,
- compressed data exchange.

Road vehicles — Open Test sequence eXchange format (OTX) —

Part 4: Expanded extensions interface definition

1 Scope

This document defines the Open Test sequence eXchange (OTX) additional extension requirements and data model specifications.

The requirements are derived from the use cases described in [ISO 13209-1](#). They are listed in [Clause 4](#).

The data model specification aims at an exhaustive definition of all features of the OTX extensions which have been implemented to satisfy the requirements. This document establishes rules for the syntactical entities of each extension. Each of these syntactical entities is accompanied by semantic rules which determine how OTX documents containing extension features are to be interpreted. The syntax rules are provided by UML class diagrams and XML schemas, whereas the semantics are given by UML activity diagrams and prose definitions.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

[ISO 13209-1:2011](#), *Road vehicles — Open Test sequence eXchange format (OTX) — Part 1: General information and use cases*

ISO 13209-2:-¹⁾, *Road vehicles — Open Test sequence eXchange format (OTX) — Part 2: Core data model specification and requirements*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in [ISO 13209-1](#), [ISO 13209-2](#) apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <http://www.electropedia.org/>

3.2 Abbreviated terms

API Application Programmers Interface

ECU Electronic Control Unit

1) Second edition under preparation. Stage at the time of publication: ISO/DIS 13209-2:2021

MVCI	Modular Vehicle Communication Interface
OTX	Open Test sequence eXchange
PDU	Protocol Data Unit
URI	Uniform Resource Identifier
XSD	XML-Schema Definition
XML	Extensible Markup Language

4 Requirements

4.1 Basic principles for requirements definition

Some basic principles have been established as a guideline to define the OTX requirements.

- a) OTX requirements specify the conditions that the OTX data model and format shall satisfy.
- b) All stakeholders (system suppliers, OEMs, tool suppliers), which offer test procedures are expected to implement and follow the requirements of this document.

The content of OTX documents and the quality of the information is the responsibility of the originator.

4.2 Requirement listing

4.2.1 Assertion requirements

4.2.1.1 A01 – Add assertions to OTX procedures to verify correctness

Rationale: It shall be possible to include actions which verify that preconditions and/or postconditions are true.

Description: During development time, a failing assertion indicates that a programming mistake has been made.

4.2.2 BlackBox requirements

4.2.2.1 BB01 – Transport unknown data types

Rationale: It shall be possible to include actions which verify that preconditions and/or postconditions are true.

Description: It can be necessary to transport values of unknown data types in OTX. For example a user-specific extension needs to handle such values or a method of an external service provider that has properties of unknown complex data types.

4.2.3 BusMonitoring requirements

4.2.3.1 BM01 – Bus monitoring

Rationale: It shall be possible to monitor a bus system like CAN.

Description: Frames on a connected bus shall be accessible for using signal data on the monitored bus.

4.2.3.2 BM02 – Bus tracing

Rationale: It shall be possible to trace a bus system like CAN.

Description: Frames on a connected bus shall be traceable for saving traces in test context and analyse them after a test.

4.2.4 ComInterface requirements

4.2.4.1 CI01 – Get communication interfaces and their resources at runtime and use them

Rationale: It shall be possible to get communication interfaces at runtime and use them for example to get a new **diag:ComChannel** on this interface.

Description: Functionality is required to get available communication interfaces and their resources at runtime and use them for diagnostics.

4.2.4.2 CI02 – Using of communication interfaces methods

Rationale: It shall be possible to use communication interfaces and their methods (e.g. get the current battery voltage or ignition state).

Description: Functionality is required to use communication interface methods.

4.2.4.3 CI03 – Activate and deactivate Ethernet

Rationale: It shall be possible to activate and deactivate the Ethernet on a VCI.

Description: Functionality is required to activate and deactivate Ethernet on a VCI.

4.2.5 CommonDialogs requirements

4.2.5.1 CD01 – It shall be possible to select a file system resource for the purposes of loading or saving file(s)

Rationale: There are occasions during the execution of an OTX sequence when a file system resource shall be selected, either to read or write information. The runtime system should provide a standard mechanism to request such resource(s).

Description: Dialogs are required to satisfy certain use cases (such as file system resource selection). Access to these resources and how information is returned is standardized to ensure exchangeability is preserved.

4.2.6 DataType requirements

4.2.6.1 DT01 – Efficient access to named collections (enumeration)

Rationale: It shall be possible to support well known and often used enumerations.

Description: An enumeration provides an efficient way to define a set of named integral constants that may be assigned to a variable.

4.2.6.2 DT02 – Support of structures

Rationale: It shall be possible to support well known and often used structures.

Description: A structure is a value type that is typically used to encapsulate small groups of related variables, such as the coordinates of a rectangle or the characteristics of an item in an inventory.

4.2.7 DiagComPlus requirements

4.2.7.1 DCP01 – Creating diag:ComChannel without starting communication

Rationale: It shall be possible to create a **diag:ComChannel** without starting communication.

Description: A functionality that creates a **diag:ComChannel** without starting communication is required.

4.2.7.2 DCP02 – Getting the state of an initialised diag:ComChannel

Rationale: It shall be possible to get the current state of a **diag:ComChannel**.

Description: A functionality that gets the current state of a **diag:ComChannel** is required.

4.2.7.3 DCP03 – Getting all ComParameters as otx:List

Rationale: It shall be possible to get all ComParameters of a **diag:ComChannel** in an **otx:List** of **otx:String**.

Description: A functionality that gets all ComParameters of a **diag:ComChannel** in an **otx:List** of **otx:String** is required.

4.2.7.4 DCP04 – Executing a functional hex service and getting responses of each ECU that responded

Rationale: It shall be possible to execute a functional hexadecimal diagnostic service and get all available responses.

Description: A functionality that executes a functional hexadecimal diagnostic service and ensures that all responses of this service can be retrieved is required.

4.2.7.5 DCP05 – Start and stop communication

Rationale: It shall be possible to explicitly execute a start communication and stop communication on a **diag:ComChannel**.

Description: Functionality is required with which an execution of a start communication and a stop communication on a **diag:ComChannel** is enabled.

4.2.8 DiagConfiguration requirements

4.2.8.1 DC01 – Selecting the diagnostic project that contains the diagnostic database

Rationale: It shall be possible to select and change the active diagnostic project that contains the diagnostic database while executing an OTX test sequence.

Description: In some cases there is a requirement to change the active database project.

4.2.8.2 DC02 – Selecting the vehicle information that contains the settings of the diagnostic database

- Rationale:** It shall be possible to select and change the active vehicle information that contains the diagnostic database while executing an OTX test sequence.
- Description:** If a change of the active database project is needed, the vehicle information shall be changeable as well.

4.2.9 DiagDataBrowsingPlus requirements

4.2.9.1 DDBP01 – Reading static information for MCDDbParameter

- Rationale:** It shall be possible to read static information for MCDDbParameter from the underlying ODX database.
- Description:** Detailed information of MCDDbParameters specified in the ODX database shall be read.

4.2.9.2 DDBP02 – Reading static information for MCDDbTable

- Rationale:** It shall be possible to read static information for MCDDbTable from the underlying ODX database.
- Description:** Detailed information of MCDDbTables specified in the ODX database shall be read.

4.2.9.3 DDBP03 – Reading static information for MCDDbEnvDataDesc

- Rationale:** It shall be possible to read static information for MCDDbEnvDataDesc from the underlying ODX database.
- Description:** Detailed information of MCDDbEnvDataDescs specified in the ODX database shall be read.

4.2.9.4 DDBP04 – Reading static information for MCDDbDiagTroubleCode

- Rationale:** It shall be possible to read static information for MCDDbDiagTroubleCode from the underlying ODX database.
- Description:** Detailed information of MCDDbDiagTroubleCodes specified in the ODX database shall be read.

4.2.9.5 DDBP05 – Reading static information for MCDDbFaultMemory

- Rationale:** It shall be possible to read static information for MCDDbFaultMemory from the underlying ODX database.
- Description:** Detailed information of MCDDbFaultMemories specified in the ODX database shall be read.

4.2.9.6 DDBP06 – Reading static information for MCDDbMatchingParameter

- Rationale:** It shall be possible to read static information for MCDDbMatchingParameter from the underlying ODX database.
- Description:** Detailed information of MCDDbMatchingParameters specified in the ODX database shall be read.

4.2.9.7 DDBP07 – Reading static information for MCDDbSpecialDataElement

Rationale: It shall be possible to read static information for MCDDbSpecialDataElement from the underlying ODX database.

Description: Detailed information of MCDDbSpecialDataElements specified in the ODX database shall be read.

4.2.9.8 DDBP08 – Reading static information for MCDDbSpecialDataGroup

Rationale: It shall be possible to read static information for MCDDbSpecialDataGroup from the underlying ODX database.

Description: Detailed information of MCDDbSpecialDataGroups specified in the ODX database shall be read.

4.2.9.9 DDBP09 – Reading static information for MCDDbTextTableElement

Rationale: It shall be possible to read static information for MCDDbTextTableElement from the underlying ODX database.

Description: Detailed information of MCDDbTextTableElements specified in the ODX database shall be read.

4.2.9.10 DDBP10 – Reading static information for MCDInterval

Rationale: It shall be possible to read static information for MCDInterval from the underlying ODX database.

Description: Detailed information of MCDIntervals specified in the ODX database shall be read.

4.2.9.11 DDBP11 – Reading static information for MCDDbSubComponent

Rationale: It shall be possible to read static information for MCDDbSubComponent from the underlying ODX database.

Description: Detailed information of MCDDbSubComponents specified in the ODX database shall be read.

4.2.10 ECUConfiguration requirements

4.2.10.1 EC01 – Configuration of an ECU by using a database (ODX-E file)

Rationale: In some cases ODX-E files are used to configure an ECU. Therefore configuration by using such files shall be enabled.

Description: OTX actions and terms are required to enable ECU configuration via a database.

4.2.11 EventPlus requirements

4.2.11.1 EP01 – Monitor changes on variables of complex types

Rationale: The OTX Core **EventHandling** extension shall be extended by a mechanism to monitor complex types.

Description: It shall be possible to react to changes deep inside the value of a variable which has a complex type, e.g. a List or Map variable.

4.2.12 ExternalServiceProvider requirements

4.2.12.1 ESP01 – Add method calls of encapsulated libraries into OTX test sequences

Rationale: Authors are able to add supplied functionality into OTX.

Description: Existing functionality of a third party company, that is encapsulated in a library, shall be added into an OTX test sequence (e.g. seed/key).

4.2.12.2 ESP02 – Add a proprietary test sequence into an OTX test sequence

Rationale: The author has the possibility to add small and simple pieces of OTX code into an already running proprietary test sequence. A step by step migration of existing proprietary test logic into OTX code is also possible.

Description: With the possibility to call a proprietary test sequence out of OTX, the author is able to exchange existing parts of proprietary test sequence into OTX (step by step migration). The author is also able to add new functionality written in OTX into proprietary test sequences. Accordingly the introduction of OTX test sequences is not complex and scalable.

4.2.12.3 ESP03 – Add any external functionality to an OTX test sequence

Rationale: Add any external functionality to an OTX test sequence.

Description: With the ExternalServiceProvider the author is able to connect custom specific systems to OTX test sequences.

4.2.13 File requirements

4.2.13.1 F01 – Read and write textual files

Rationale: It shall be possible to read and write textual files in the local file system.

Description: An OTX extension is required which allows access to textual files.

4.2.13.2 F02 – Read and write binary files

Rationale: It shall be possible to read and write binary files in the local file system.

Description: An OTX extension is required which allows access to binary files.

4.2.14 FlashPlus requirements

4.2.14.1 FP01 – Enable latebound flashing (select binary file while execution of a test)

Rationale: It shall be possible to get, set and change binary flash files for **flash:FlashSession** at runtime.

Description: In some cases there is a requirement to set a binary flash file while executing a script.

4.2.15 Persistence requirements

4.2.15.1 P01 – It shall be possible to save and load the value of all SimpleTypes and ByteField using persistent storage

Rationale: Authors should not be burdened with details of the conversion to file format – the storage format and location should be the responsibility of the runtime system.

Description: Simple types and ByteFields are literal data that can successfully be saved between sessions. These types are the building blocks of more complex structured data.

4.2.15.2 P02 – It shall be possible to save and load collection types (list and map) using persistent storage if the contents of the collections can also be persisted

Rationale: If the contents of lists and maps can be persisted, then lists and maps of these types can also be persisted.

Description: Extending persistence to collection types gives authors a powerful tool for saving and loading structured data with single actions.

4.2.15.3 P03 – It shall be possible for OTX Extensions to define additional data types that can persist

Rationale: There may be other data types defined by OTX extensions that need to persist.

Description: See the relevant extensions for details.

4.2.16 SQL requirements

4.2.16.1 SQL01 – Establishing database connections

Rationale: It shall be possible to create a database connection.

Description: A certain database connection can be established by the runtime system.

4.2.16.2 SQL02 – Executing of database commands

Rationale: It shall be possible to execute commands at a database connection.

Description: Depending of the database system, a command can be, e.g. an OTX SQL extension command.

4.2.16.3 SQL03 – Retrieving results of a database command

Rationale: It shall be possible to retrieve results from an executed command.

Description: The values of the result can be in an arbitrary structure.

4.2.16.4 SQL04 – Retrieving results shall be possible synchronously and asynchronously

Rationale: It shall be possible to retrieve results synchronously and asynchronously.

Description: If the data records read asynchronously the sequence shall not have to wait to retrieve a large amount of data.

4.2.17 StateMachineProcedure requirements

4.2.17.1 SMP01 – Description of state based workflows

Rationale: It shall be possible to describe state-based workflows.

Description: A state machine consists of 1 to n states and state transitions for the state change which will be realised by the occurrence of conditions supervised by triggers. The extension shall not contain layout information for the runtime system.

4.2.18 StateVariable requirements

4.2.18.1 SV01 – Support for state variables, which are the counterpart of context variables

Rationale: It shall be possible to easily transport status information from inside a sequence to the environment.

Description: State variables can only be evaluated by setting outcomes of the test application, e.g. to display the current execution progress.

4.2.19 TestResultHandling requirements

4.2.19.1 RH01 – Capturing of test result data

Rationale: Functionality shall be provided to capture test result data.

Description: Test result data (e.g. measured values) shall be captured in a structured way in relation to the environment (e.g. test object, test identifier, date/time). For the test result structure a data model shall be used.

4.2.19.2 RH02 – Evaluation of test result data

Rationale: It shall be possible to evaluate captured test result data.

Description: For each test result, it shall be possible to define individual conditions to evaluate if the test result is OK or NOT OK.

4.2.19.3 RH03 – Summarising of test results in groups

Rationale: It shall be possible to summarise test results in groups.

Description: According to the test result data model, test results can be summarised in groups. It shall be possible to evaluate if the test result group is OK or NOT OK.

4.2.19.4 RH04 – Defining different test result levels

Rationale: It shall be possible to define different test result levels.

Description: It is necessary to have different test result levels according to the development process of OTX sequences, e.g. test results exclusively for debugging tasks.

4.2.19.5 RH05 – Multiple test instances

Rationale: Simultaneous operation with multiple test instances shall be possible.

Description: It shall be possible to capture and evaluate test result data of different test instances (e.g. several vehicles) at the same time.

4.2.20 Util requirements

4.2.20.1 U01 – Copying **otx:ByteFields**

Rationale: It shall be possible to copy an **otx:ByteField** to simplify assigning a value and not a reference to another **otx:ByteField**.

Description: **otx:ByteField** shall be copiable because it is needed to assign **otx:Bytefield** value of **otx:Bytefield** variable to another **otx:Bytefield** variable.

4.2.20.2 U02 – Comparing values of complex types

Rationale: It shall be possible to compare the values of complex types (not their references like **otx:IsEqual** does) in a simple way.

Description: An easy way to compare complex type values is needed (e.g. comparing of two hexadecimal responses as **otx:ByteField** Variable of an ECU).

4.2.20.3 U03 – Getting a random number

Rationale: It shall be possible to get a random number for special test cases.

Description: Some test sequences need to use a random numbers. Therefore, an easy way to generate a random number is needed.

4.2.20.4 U04 – Checking a variable's status

Rationale: It shall be possible to check whether a variable of a complex type is already initialised or not.

Description: In some cases it is important to check whether a complex variable is initialised or not. Therefore, an easy way to check the status of a complex variable is needed.

4.2.20.5 U05 – Simplify analysing the content of a list

Rationale: It shall be possible to simply analyse the content of **otx:List** instances.

Description: Analysis of an **otx:List** shall be simplified, because in some test sequences its analysis is necessary.

4.2.20.6 U06 – More possibilities to format a string

Rationale: It shall be possible to simply format an **otx:String**.

Description: In some cases strings have to be formatted in a special way.

4.2.21 VehicleInfo requirements

4.2.21.1 VI01 – Getting the gateway mode of an ECU

Rationale: It shall be possible to get the gateway mode of an ECU.

Description: An OTX term that gets the gateway mode of an ECU is required.

4.2.21.2 VI02 – Getting names of gateways that accesses an ECU

Rationale: It shall be possible to get the names of gateways that access an ECU.

Description: An OTX term that gets the names of gateways that access an ECU is required.

4.2.21.3 VI03 – Checking if ECU is accessed via gateways

Rationale: It shall be possible to check if an ECU is accessed by a gateway.

Description: An OTX term that checks if an ECU is accessed via one or more gateways is required.

4.2.22 XML requirements

4.2.22.1 X01 – The user shall be able to set XML processing Instructions

Rationale: The functionality shall be provided to store processing instructions.

Description: Processing instructions at document level define additional information outside the XML root tag. Therefore, an extra functionality is needed to add this information to the XML document.

4.2.22.2 X02 – The user shall be able to edit attributes, nodes and node text values

Rationale: The functionality shall be provided to modify tag attributes, nodes and containing text values.

Description: To create a complete XML structure, the functionality to edit attributes, nodes and the text value between the start tag and end tag is mandatory.

4.2.22.3 X03 – The user shall be able to add comments to an XML document

Rationale: The functionality shall be provided to add comments to an XML document.

Description: There shall be an OTX extension, which allows the appending of comment tags to a document root node.

4.2.22.4 X04 – The user shall be able to remove attributes and nodes

Rationale: The functionality shall be provided to remove attribute entries as well as complete nodes.

Description: There shall be an OTX extension, which allows deleting an attribute of an XML node. It should be possible to delete a node containing all children as well.

4.2.22.5 X05 – Validation of an XML document

Rationale: The functionality shall be provided to validate an XML document.

Description: There shall be an OTX extension, which allows validation of an XML document against a referenced schema.

4.2.22.6 X06 – The user shall be able to define the document encoding and version

- Rationale:** The functionality shall be provided to define the document encoding and version.
- Description:** There shall be an OTX extension, which allows the addition of the encoding and version as process instructions at the head of the XML document.

4.2.22.7 X07 – The user shall be able to parse XML content from file or ByteField

- Rationale:** The functionality shall be provided to parse an XML document tree from an addressed file or in-memory ByteField.
- Description:** An extension to parse an XML document from a file which is addressed by a URI should be provided. An in-memory ByteField as input shall be supported as well.

4.2.22.8 X08 – The user shall be able to query XML elements by XPath

- Rationale:** The functionality shall be provided to address an XML element using an XPath query.
- Description:** An extension to collect all matching XML elements starting from the document node shall be provided.

4.2.23 ZipHandling requirements

4.2.23.1 Z01 – Functionality shall be provided to extract a zip file

- Rationale:** The user shall be able to extract a zip file.
- Description:** An extension for zip file handling shall provide the possibility to extract a zip file to a target directory.

4.2.23.2 Z02 – Filter functionality for the file compress and decompress process shall be provided

- Rationale:** To reduce the amount of files, a wildcard filter for the file consistent extension shall be used.
- Description:** A wildcard filter reduces the amount of extracted or compressed files by applying a filter on the file consistent extension.

4.2.23.3 Z03 – The user shall decide whether existing files should be overwritten by the decompress process.

- Rationale:** The decompress functionality shall control whether existing files should be overwritten.
- Description:** The target directory of the extraction process can contain files already. The overwrite action of conflicting files shall be configured by the user.

4.2.23.4 Z04 – The user shall decide whether the target zip file should be overwritten by the compress process

- Rationale:** The compress functionality shall control how to handle an existing target zip file.
- Description:** If the addressed target zip file already exists, files can be added to the existing container or it will be overwritten. The functionality has to be configured by the user.

4.2.23.5 Z05 – The zip file shall be a URI

Rationale: The addressing of the zip file shall be performed using the URI definition.

Description: The URI path defines the address of the zip file resource.

5 Extension overview

5.1 General

OTX (Open Test sequence Exchange) is a platform and tester-independent exchange format for formal description of executable test sequences published in the ISO 13209 series. OTX is a standard for the reliable storage of test knowledge within vehicle diagnostics and vehicle test automation. The XML-based flow language provides the opportunity to share test sequences across departmental, tool and process boundaries. The expertise stored in these flows is not lost, but can be used even after many years. OTX is targeted primarily at the automotive industry, although the core features of the standard are applicable in any industry. Such test sequences are used in development, production and after sales and can be exchanged between vehicle manufacturers, tier 1 suppliers and tool providers.

OTX is the logical evolution of standardization in vehicle diagnostics. [Figure 2](#) shows the relation of OTX to other diagnostic standards.

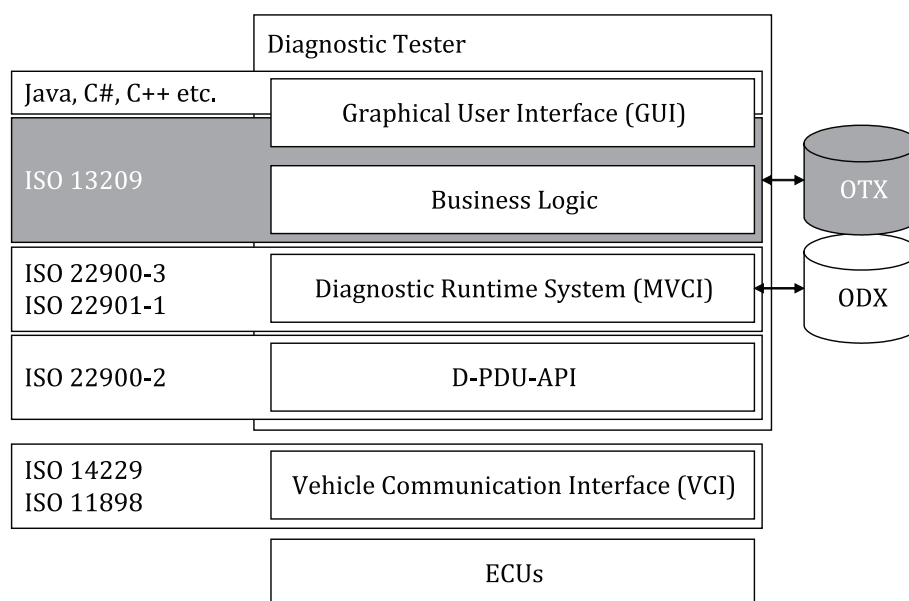


Figure 2 — Relation of OTX to standards in vehicle diagnostics

The ISO 13209 series consists of a core and an extension concept, which allows the integration of arbitrary interfaces. For example, extensions for diagnostic communication, flashing, event handling and logging are standardized in ISO standards. Based on these extensions the usage of an MVCI server ([ISO 22900-3](#) also known as ASAM MCD-3 D Server API) is simplified, because many implementation steps are hidden.

OTX is not restricted to diagnostic sequences. It can be also seen as a strategic migration platform for different, currently separated standards, e.g. ASAM GDI, ASAM XIL, ASAM AE MCD-3 MC.

This document represents the specification of the OTX standard extensions for data model version "1.0.0".

For the convenience of the user the schema definition files (xsd) are published together with the associated examples (otx) alongside this document, available at <https://standards.iso.org/iso/13209/-4/ed-1/en/>.

For the convenience of the user the schema definition files (xsd) for a possible, not normative TestResult data model are published together with the associated examples (xml) alongside this document, available at <https://standards.iso.org/iso/13209/-4/ed-1/en/>.

[Annex A](#) contains a comprehensive listing of all checker rules and shall be followed. The rules are needed because some constraints existing on OTX documents cannot be ensured by XSD validation alone. These constraints need to be checked by additional checker applications.

5.2 Dependencies

[Figure 3](#) shows a UML package diagram describing the full set of OTX extensions (together with the OTX Core) and the import dependencies in between them. OTX extensions use or extend types defined in the OTX Core. Therefore, all of the extensions are (directly or indirectly) based on the OTX Core data model, as specified by [ISO 13209-2](#).

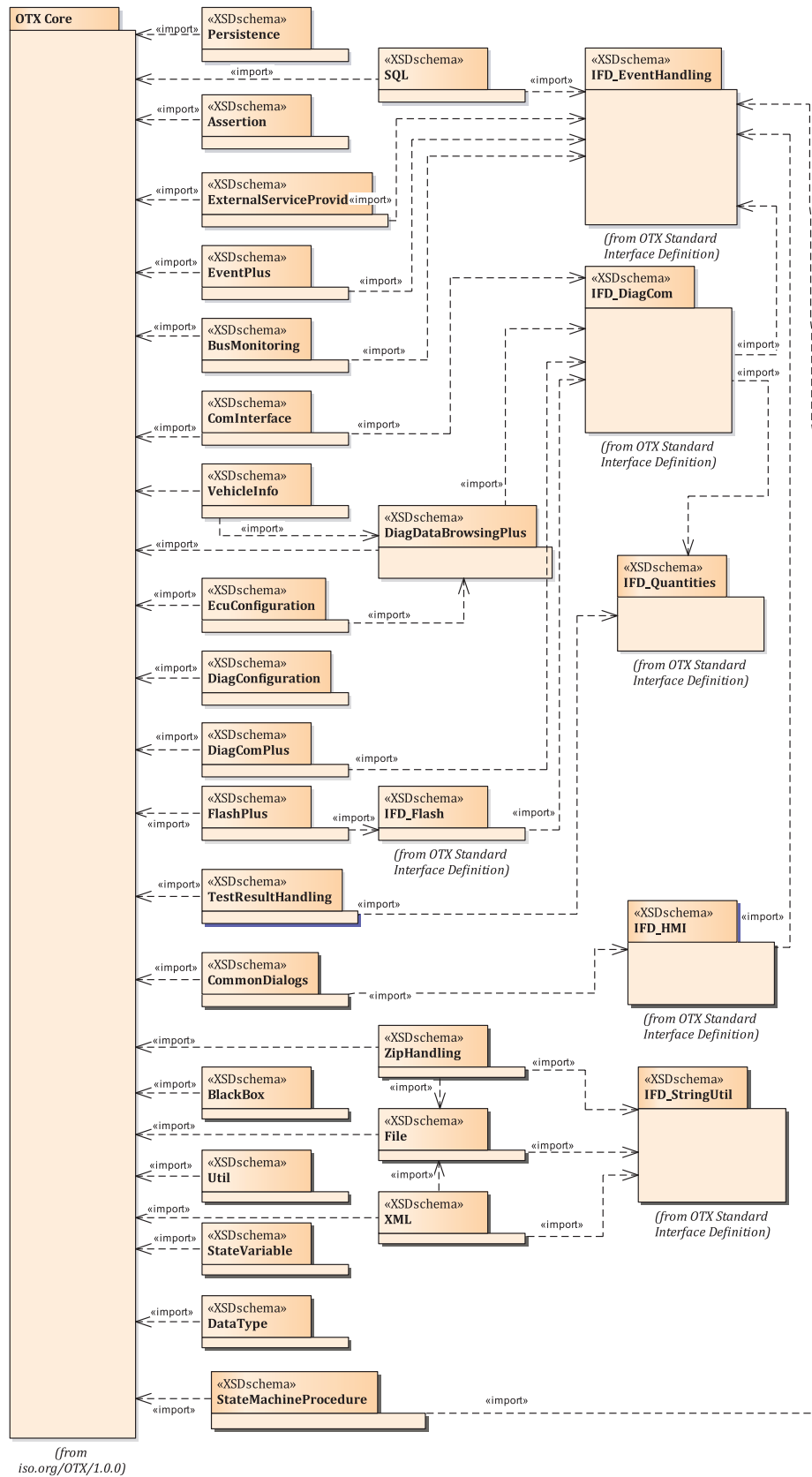


Figure 3 — Overview OTX schema dependencies

[Table 1](#) provides an overview of additional OTX extensions and their basic characteristics.

The OTX Core is a prerequisite for any OTX application and shall always be fully supported. By contrast, OTX applications are not required to support all of the extensions specified in this document. The set of supported extensions may vary depending on the field of application. However, an OTX application supporting an extension which imports other extensions shall support these extensions also. This guarantees that the set of supported extensions is consistent with regard to the dependencies.

Table 1 — OTX extension characteristics

Extension (schema file)	Summary	Dependencies
Assertion (otxIFD_Assertion.xsd)	Ensuring OTX sequence quality	Core
BlackBox (otxIFD_BlackBox.xsd)	Transport of customer specific data types	Core
BusMonitoring (otxIFD_BusMonitoring.xsd)	Collection of communication data at runtime	Core ComInterface EventHandling
ComInterface (otxIFD_ComInterface.xsd)	Handling of communication interfaces and diagnostic over IP	Core DiagCom
CommonDialogs (otxIFD_CommonDialogs.xsd)	Dialogs for common use cases	Core HMI
DataType (otxIFD_DataType.xsd)	Definition of user defined structure and enumeration data types	Core
DiagComPlus (otxIFD_DiagComPlus.xsd)	Support of different ECU responses for hex service and com channel controlling	Core DiagCom
DiagConfiguration (otxIFD_DiagConfiguration.xsd)	Configuration of project and vehicle information	Core DiagCom
DiagDataBrowsingPlus (otxIFD_DiagDataBrowsingPlus.xsd)	Browsing of ODX database properties	Core DiagCom
ECUConfiguration (otxIFD_ECUConfiguration.xsd)	ECU variant coding	Core DiagDataBrowsingPlus
EventPlus (otxIFD_EventPlus.xsd)	Support for deep change monitoring in a List or Map	Core EventHandling
ExternalServiceProvider (otxIFD_ExternalServiceProvider.xsd)	Provides actions, terms, events and data types for an access to external services	Core EventHandling
File (otxIFD_File.xsd)	General read write access to files	Core StringUtil
FlashPlus (otxIFD_FlashPlus.xsd)	Enable late binding of flash files	Core Flash DiagCom
Persistence (otxIFD_Persistence.xsd)	Store runtime information of an equal run-time system	Core
SQL (otxIFD_SQL.xsd)	Read and write access to a SQL database	Core EventHandling

Extension (schema file)	Summary	Dependencies
StateMachineProcedure (otxIFD_StateMachineProcedure.xsd)	Description of a behaviour model with states, transactions and actions	Core EventHandling
StateVariable (otxIFD_StateVariable.xsd)	Mechanism to transport status information from inside a sequence to the environment	Core
TestResultHandling (otxIFD_TestResultHandling.xsd)	Evaluated and structured storage of result Information (e.g. for statistics) from test sequences	Core Quantities
Util (otxIFD_Util.xsd)	Advanced convenience functionality	Core
VehicleInfo (otxIFD_VehicleInfo.xsd)	Access to information about the vehicle network architecture	Core DiagDataBrowsingPlus
Xml (otxIFD_Xml.xsd)	Read and interpret XML documents, modification of existing and creation of new documents	Core File StringUtil
ZipHandling (otxIFD_ZipHandling.xsd)	Compressed data exchange	Core

Table 2 shows the XSD namespace associations of additional OTX extensions. Each namespace has a prefix assigned to it. In the remainder of this document, the prefixes defined here are used to mark types which belong to extensions other than the one which is currently described. In contrast, the types defined by the currently described extension are not prefixed.

Table 2 — OTX extension namespace associations

Extension	Namespace	Prefix
Assertion	http://iso.org/OTX/1.0.0/Assertion	assert:
BlackBox	http://iso.org/OTX/1.0.0/BlackBox	blackBox:
BusMonitoring	http://iso.org/OTX/1.0.0/BusMonitoring	busMon:
ComInterface	http://iso.org/OTX/1.0.0/ComInterface	comIf:
CommonDialogs	http://iso.org/OTX/1.0.0/CommonDialogs	commDlg:
DataType	http://iso.org/OTX/1.0.0/DataType	dataType:
DiagComPlus	http://iso.org/OTX/1.0.0/DiagComPlus	diagComPlus:
DiagConfiguration	http://iso.org/OTX/1.0.0/DiagConfiguration	diagConf:
DiagDataBrowsingPlus	http://iso.org/OTX/1.0.0/DiagDataBrowsingPlus	diagDataPlus:
ECUConfiguration	http://iso.org/OTX/1.0.0/ECUConfiguration	ecuConf:
EventPlus	http://iso.org/OTX/1.0.0/EventPlus	eventPlus:
ExternalServiceProvider	http://iso.org/OTX/1.0.0/ExternalServiceProvider	esp:
File	http://iso.org/OTX/1.0.0/File	file:
FlashPlus	http://iso.org/OTX/1.0.0/FlashPlus	flashPlus:
Persistence	http://iso.org/OTX/1.0.0/Persistence	persist:
SQL	http://iso.org/OTX/1.0.0/SQL	sql:
StateMachineProcedure	http://iso.org/OTX/1.0.0/StateMachineProcedure	smp:
StateVariable	http://iso.org/OTX/1.0.0/StateVariable	stateVar:
TestResultHandling	http://iso.org/OTX/1.0.0/TestResultHandling	testResult:
Util	http://iso.org/OTX/1.0.0/Util	util:
VehicleInfo	http://iso.org/OTX/1.0.0/VehicleInfo	vehInf:

Extension	Namespace	Prefix
Xml	http://iso.org/OTX/1.0.0/XML	otxXml:
ZipHandling	http://iso.org/OTX/1.0.0/ZipHandling	zip:

5.3 OTX packaging (PTX)

The exchange of a whole OTX project can be realized with a file container called "PTX package". All files of an OTX project are packaged in one PTX container. A PTX container can handle OTX-files with the file extension *.otx as well as arbitrary files of any format that are necessary for the project. The file container format is described in [Annex B](#) and shall be followed.

6 OTX Assertion extension

6.1 General

Assertions are used to ensure the correct implementation of OTX test sequences. An assertion claims that a certain condition holds during OTX execution. It does not contribute to the test logic that is implemented in the OTX.

Assertions are usually evaluated during testing of the OTX test sequence itself. A failing assertion always indicates that the OTX test sequence contains a programming error and shall be changed.

During development of an OTX test sequence, assertions shall not be evaluated because it can be assumed that the sequence does not contain any programming errors. Runtime systems should, therefore, offer a configuration to enable or disable the evaluation of assertions. By default, the evaluation of assertions should be disabled. If the runtime system is configured not to evaluate assertion statements, it shall suppress any impact of assertions: it shall not evaluate their boolean conditions or error messages.

6.2 Exceptions

6.2.1 Exceptions overview

All exceptions introduced by this extension are derived from the **otx:Exception** type. They represent the full set of exceptions added by the **OTX Assertion extension**.

6.2.2 Exceptions syntax

The syntax of all **OTX Assertion extension** exception-type declarations is shown in [Figure 4](#).

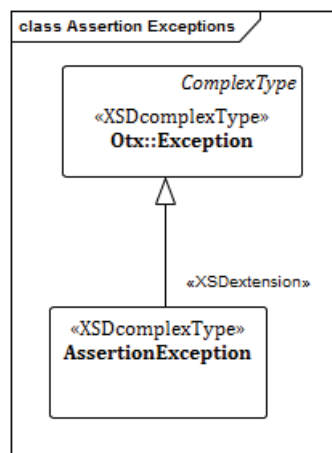


Figure 4 — Assertion exceptions

6.2.3 Exceptions semantics

6.2.3.1 General

Since all **OTX Assertion extension** exception types are implicit exceptions without initialisation parts, they cannot be declared constant.

6.2.3.2 AssertionException

IMPORTANT — All terms and action realisations in this extension may potentially throw this exception.

6.3 Actions

6.3.1 Actions overview

All actions introduced by this extension extend the **otx:ActionRealisation** extension interface.

6.3.2 Actions syntax

[Figure 5](#) shows the syntax of all actions in the **OTX Assertion extension**.

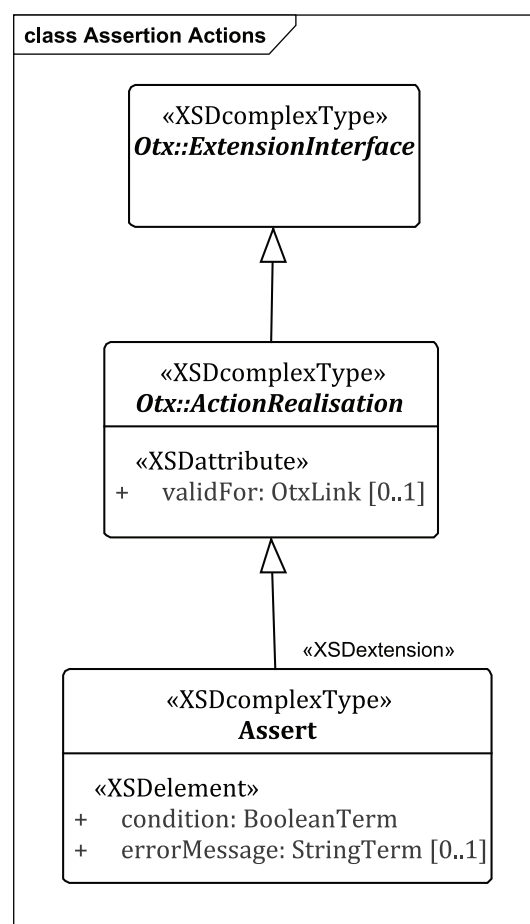


Figure 5 — Assertion actions

6.3.3 Actions general semantics

6.3.3.1 Assert

Semantic:

If the runtime is configured to evaluate assertions, it evaluates the given boolean expression.

If the expression evaluates to true, nothing happens.

If the expression evaluates to false, the execution of OTX procedure should be stopped (the exact runtime behaviour is unspecified), and the failed assertion is reported to the runtime system itself (out of OTX scope).

The `<errorMessage>` term, if present, shall only be evaluated when the assertion failed. In particular, the `<errorMessage>` term cannot throw any exceptions unless the assertion failed.

If the runtime is configured not to evaluate assertions, the `assert:Assert` action does nothing. In particular, the `<condition>` and `<errorMessage>` terms will not be evaluated, and thus cannot throw any exceptions.

Parameters:

- `<condition>` : `otx:BooleanTerm` [1..1]

The expression to evaluate.

- `<errorMessage>` : `otx:StringTerm` [0..1]

An error message which may be included in the report to the runtime system, if the assertion fails.

Exceptions:

- None

7 OTX BlackBox extension

7.1 General

The **OTX BlackBox extension** data type can be used as a transport or container data type. An **OTX BlackBox extension** can contain a value of a user-specific data type which is unknown to OTX.

For example, the main procedure may have an in parameter of type **OTX BlackBox extension**. When the main procedure is called, the runtime system uses this parameter to pass in a value of a proprietary (non-OTX) data type. The OTX procedures do not interact with the contents of this **OTX BlackBox extension**. They pass it on to an **otx:ActionRealisation** of a proprietary OTX extension. This proprietary extension is able to handle the value contained in the **OTX BlackBox extension**. In other words, the user can transport data through code layers (code is embedded in between test system specific runtime code) even if its type is unknown to OTX (not defined in core or standardized extensions). This extension provides no way of initializing **OTX BlackBox extension** values.

Terms and actions can be added in new, proprietary extensions to interact with the **OTX BlackBox extension** data type.

7.2 Datatypes

7.2.1 Datatypes overview

All data types introduced by this extension are derived from **otx:ComplexType**.

7.2.2 Datatypes syntax

The syntax of the data-type declarations of the **OTX BlackBox extension** is shown in [Figure 6](#).

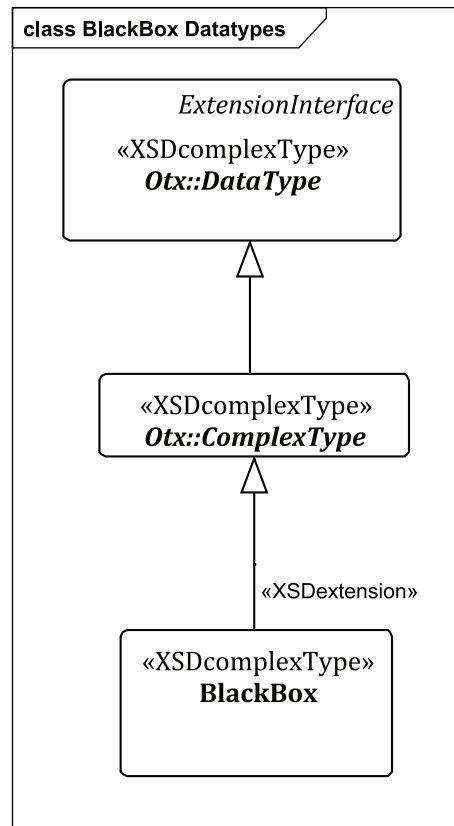


Figure 6 — BlackBox datatypes

7.2.3 Datatypes semantics

7.2.3.1 BlackBox

BlackBox is an **otx:ComplexType**.

This type can be used as a transport or container datatype. It can contain a value of a user-specific data type which is unknown to OTX. If OTX code is embedded in between test system specific runtime code, OTX unknown types should be able to be passed through the code layers. This extension provides no way of initializing **BlackBox** values.

7.3 Variable access

7.3.1 Variable-access overview

All variable-access types introduced by this extension are derived from the OTX core variable extension interface.

7.3.2 Variable-access syntax

The syntax of the **OTX BlackBox extension's** variable-access types is shown in [Figure 7](#).

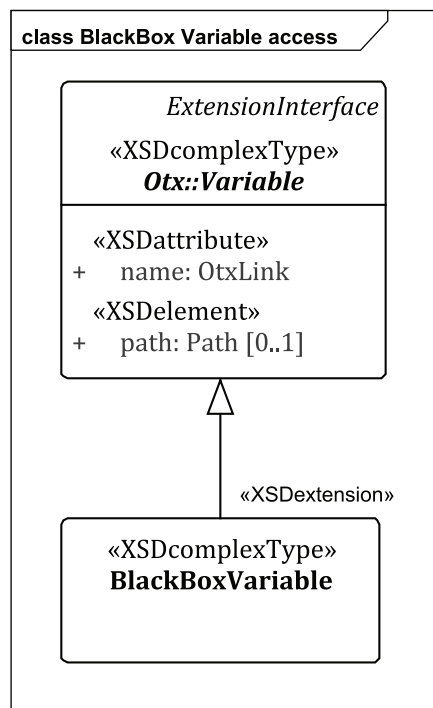


Figure 7 — BlackBox variable access

7.3.3 Variable-access semantics

The general semantics for all variable-access types shall apply. Please refer to [ISO 13209-2](#) for details.

7.4 Terms

7.4.1 Terms overview

All terms introduced by this extension extend the **otx:Term** extension interface. Information about the specific super class of a term is provided in the individual term description clauses.

7.4.2 Terms syntax

[Figure 8](#) shows the syntax of the **OTX BlackBox extension** related terms.

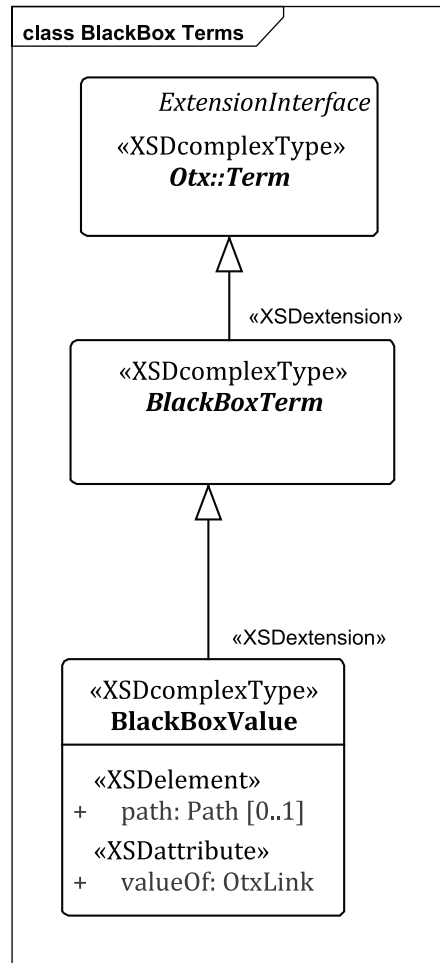


Figure 8 — BlackBox terms

7.4.3 Terms general semantics

7.4.3.1 BlackBoxTerm

Semantic:

The abstract type BlackBoxTerm is an otx:SimpleTerm. It serves as a base for all concrete terms, which return a BlackBox value. It has no special members.

7.4.3.2 BlackBoxValue

Semantic:

Returns the BlackBoxValue stored in a Blackbox variable.

Return type:

blackBox:BlackBoxTerm

Parameters:

— <path> : otx:Path [0..1]

This element addresses parts of complex structures. For more information please refer to [ISO 13209-2](#).

Exceptions:

otx:OutOfBoundsException

— Occurs only, if a <path> is set and the <path> points to a location, which does not exist (like a list index exceeding list length, or a map key which is not part of the map).

8 OTX BusMonitoring extension

8.1 General

The **OTX BusMonitoring extension** is introduced to enable asynchronous collection of communication data (monitoring) at runtime and analysis of monitored data (frames).

IMPORTANT — The bus monitoring of the underlying system (e.g MVCI server) is not specified. The start of bus monitoring can throw an exception because the system does not support it. Also it is not guaranteed that the terms of the extension return any data because the underlying runtime system cannot get it. In such cases an exception is thrown as well. The extension is created to facilitate using bus monitoring.

NOTE The extension is closely related to an ODX/MVCI based system, as defined in the ISO 22901 and ISO 22900 series.

In case of an ODX/MVCI based system it is recommended that the result of `MCDMonitorLink::fetchMonitoringFrames` is formatted as given in [ISO 22900-3:2012](#), Annex F. This is related to the terms `GetBusFrameAsString`, `GetBusFrameTimestamp`, `GetBusFrameAddress`, `GetBusFrameData` and `GetBusFrameType`.

8.2 Datatypes

8.2.1 Datatypes overview

All data types introduced by this extension are derived from `otx:ComplexType` or `otx:SimpleType`.

8.2.2 Datatypes syntax

The syntax of the data-type declarations of the **OTX BusMonitoring extension** is shown in [Figure 9](#).

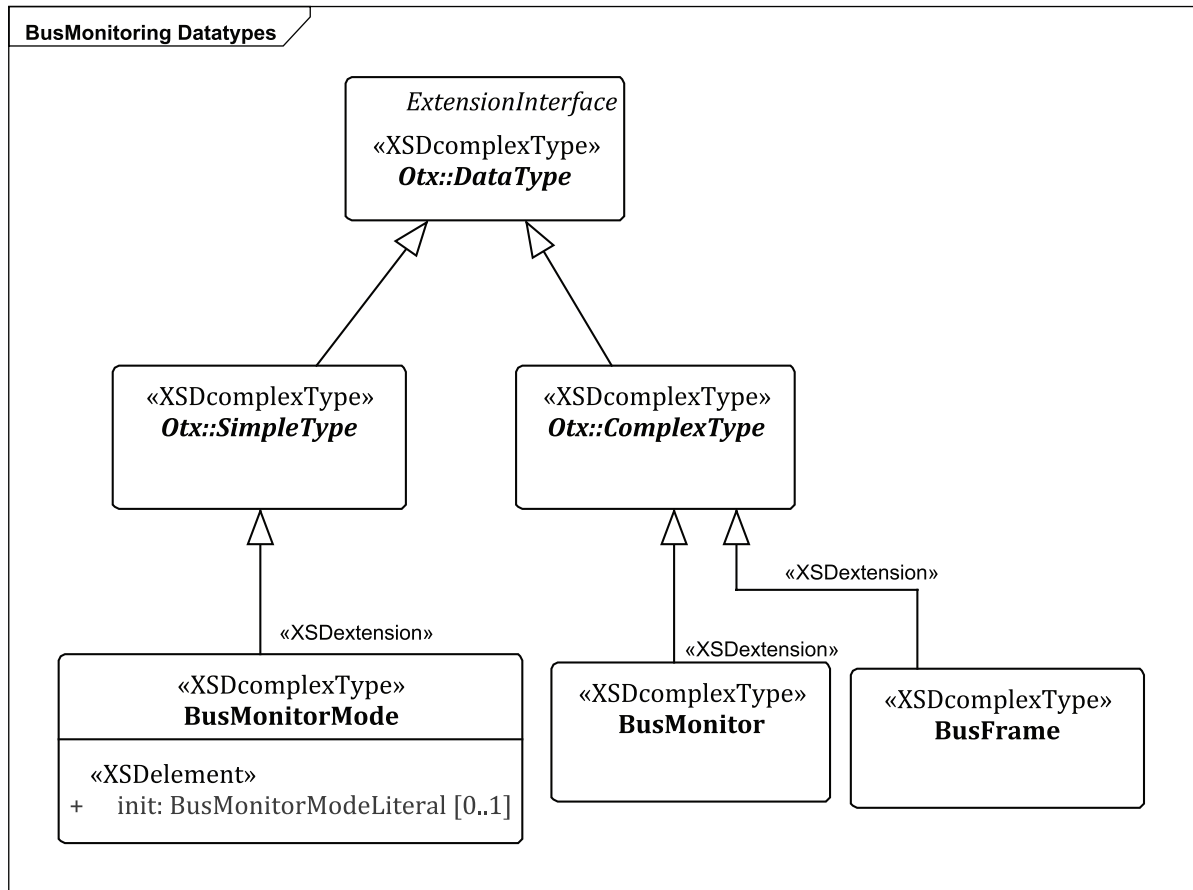


Figure 9 — BusMonitoring datatypes

8.2.3 Datatypes semantics

8.2.3.1 BusFrame

BusFrame is an `otx:ComplexType`.

Represents a handle of a bus frame that contains information about one frame, e.g. a message and a timestamp.

IMPORTANT — The `otx:ToString` conversion term shall return an `otx:String` that represents the whole frame delivered by the runtime system.

8.2.3.2 BusMonitor

BusMonitor is an `otx:ComplexType`.

A **BusMonitor** can be used to monitor a communications bus resource available at an interface (VCI module).

NOTE The related MVCI system class is `MCDMonitoringLink`.

IMPORTANT — In case of an ODX/MVCI based system the `otx:ToString` conversion shall return the **SHORT-NAME** of the related `MCDMonitoringLink`.

8.2.3.3 BusMonitorMode

BusMonitorMode is an `otx:SimpleType`.

BusMonitorMode is an enumeration type describing the modes of the monitor.

IMPORTANT — The `otx:ToString` conversion shall return the name of the enumeration value and the `otx:ToInteger` conversion shall return the index of the value in the enumeration (smallest index is 0).

Its members have the following semantics:

— `<init> : busMon:BusMonitorModeLiteral[0..1]`

This optional element stands for the hard-coded initialisation value of the identifier at declaration time.

IMPORTANT — If **BusMonitorMode** declaration is not explicitly initialised, the default value shall be **MONITORING**.

8.3 Enumerations

8.3.1 Enumerations syntax

The syntax of the enumeration declarations of the **OTX BusMonitoring extension** is shown in [Figure 10](#).

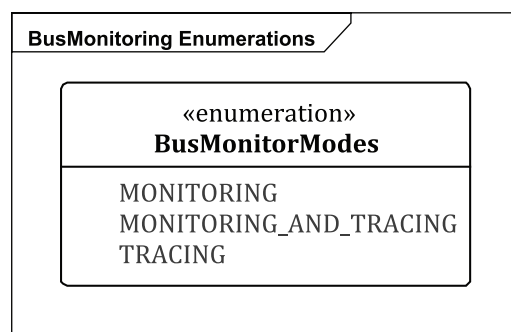


Figure 10 — BusMonitoring enumerations

8.3.2 Enumerations semantics

8.3.2.1 BusMonitorModes

BusMonitorModes is an enumeration type describing the mode of monitoring.

BusMonitorModes has the following enumeration values.

- **MONITORING**
Received frames will be stored into an internal buffer and NOT written into a trace resource.
- **MONITORING_AND_TRACING**
Received frames will be stored into an internal buffer and written into a trace resource.
- **TRACING**
Received frames will be written into a trace resource and NOT stored into an internal buffer.

8.4 Exceptions

8.4.1 Exceptions overview

All exceptions introduced by this extension are derived from the **otx:Exception** type. They represent the full set of exceptions added by the **OTX BusMonitoring extension**.

8.4.2 Exceptions syntax

The syntax of all **OTX BusMonitoring extension** exception-type declarations is shown in [Figure 11](#).

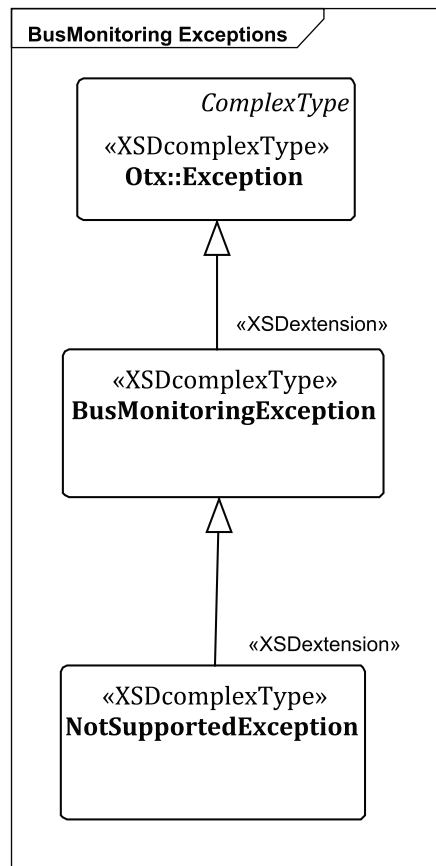


Figure 11 — BusMonitoring exceptions

8.4.3 Exceptions semantics

8.4.3.1 General

Since all **OTX BusMonitoring extension** exception types are implicit exceptions without initialisation parts, they cannot be declared constant.

8.4.3.2 BusMonitoringException

The **BusMonitoringException** is the super class for all exceptions in the **OTX BusMonitoring extension**. A **BusMonitoringException** shall be used in case the more specific exception types described in the remainder of this section do not apply to the problem at hand.

IMPORTANT — All terms and action realisations in this extension may potentially throw this exception.

8.4.3.3 NotSupportedException

This is thrown if the underlying runtime system (MVCI-Server) does not support this operation.

8.5 Variable access

8.5.1 Variable-access overview

All variable-access types introduced by this extension are derived from the OTX core variable extension interface.

8.5.2 Variable-access syntax

The syntax of the **OTX BusMonitoring extension** variable-access types is shown in [Figure 12](#).

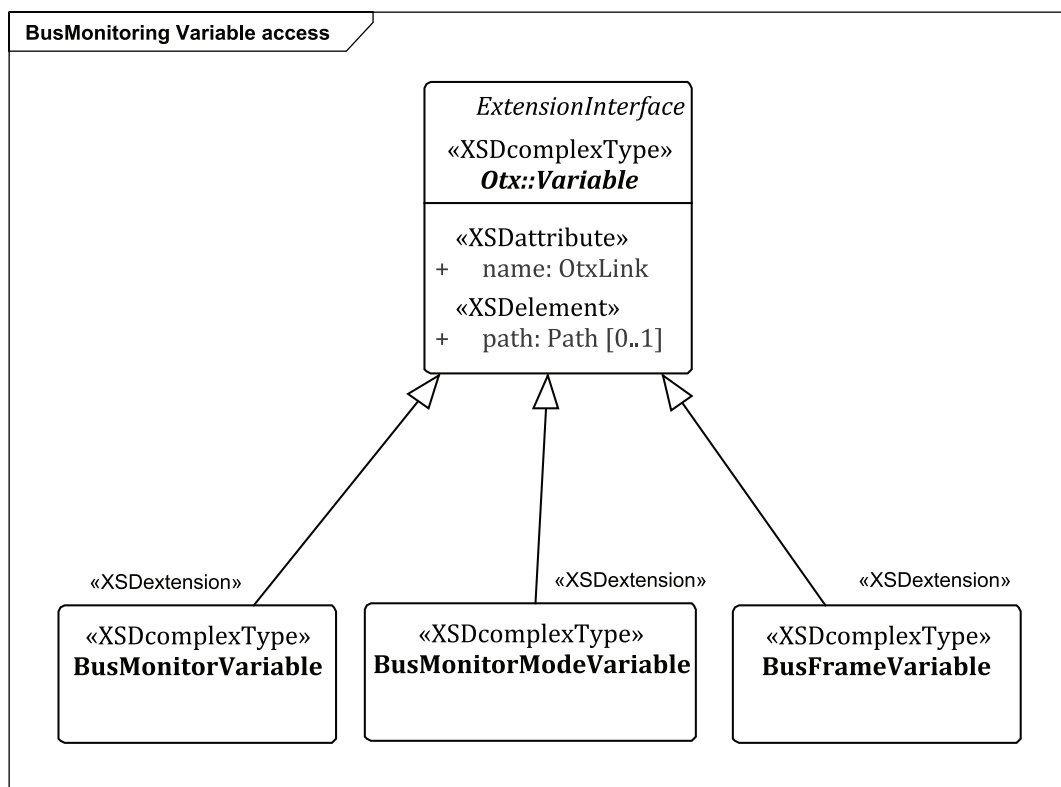


Figure 12 — BusMonitoring variable access

8.5.3 Variable-access semantics

The general semantics for all variable-access types shall apply. Please refer to [ISO 13209-2](#) for details.

8.6 Actions

8.6.1 Actions overview

All actions introduced by this extension extend the **otx:ActionRealisation** extension interface.

8.6.2 Actions syntax

[Figure 13](#) shows the syntax of all actions in the **OTX BusMonitoring extension**.

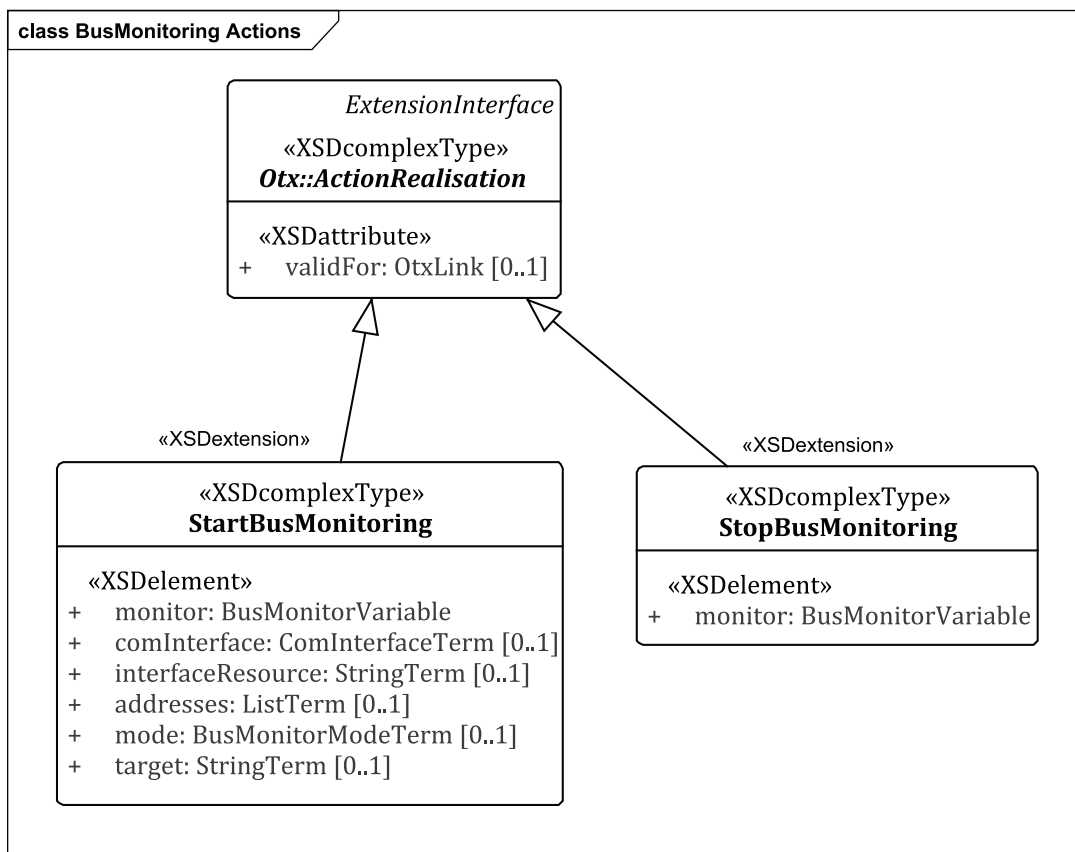


Figure 13 — BusMonitoring actions

8.6.3 Actions general semantics

8.6.3.1 StartBusMonitoring

Semantic:

Starts the monitoring and optional tracing on a physical link. Use FetchBusMonitorBusFrames to retrieve frames on a BusMonitor.

NOTE The related MSCI system methods are MCDProject::createMonitoringLink(MCDInterfaceResource mCDInterfaceResource); MCDMonitorLink::getMessageFilters(); MCDMessageFilters::addByFilterType(int messageFilterType); MCDMessageFilter::enableMessageFilter(true); MCDMonitoringLink::start().

Associated checker rules:

- BusMonitoring_Chk001 – correct element types for addresses list in StartBusMonitoring.

Parameters:

- <monitor> : busMon:BusMonitorVariable [1..1]

This element represents the variable that refers to the handle for the monitor started by this action. This can be used later to fetch the frames of the monitor (FetchBusMonitorBusFrames) or stop the monitor explicitly (StopBusMonitoring).

- <comInterface> : comIf:ComInterfaceTerm [0..1]

Represents a handle for the communication interface on which the monitoring should be started. If no **comIf:ComInterface** is given, the default communication interface shall be used (the interface that is needed for **diag:GetComChannel**, see ComInterface extension).

- <interfaceResource> : otx:StringTerm [0..1]

Represents the name of the physical link, e.g. "CAN1". If no interface resource is given, the default interface resource shall be used (interface that is needed for **diag:GetComChannel**, see ComInterface extension).

- <addresses> : otx:ListTerm [0..1]

Represents an optional list of integer numbers, defining the addresses to be monitored (e.g. on a CAN bus the CAN IDs of signals or ECUs). If no list or an empty list is given, all data shall be monitored.

NOTE In case an MVCI system is used pass filters are applied to CAN ID filtering.

- <mode> : busMon:BusMonitorModeTerm [0..1]

Represents an optional mode to specify how the monitored frames shall be processed, see **BusMonitorModes**.

- <target> : otx:StringTerm [0..1]

This optional element shall be used for locating the resource to which the trace shall be written. The target should be defined by a URI (see ISO 13209-2:-²⁾, Annex G). Other resource-location mechanisms may also be used.

Exceptions:

NotSupportedException

- In case a system without any addresses (e.g. DoIP or K-Line) is used and addresses are supplied by the addresses property, a **NotSupportedException** shall be thrown.

8.6.3.2 StopBusMonitoring

Semantic:

Stops the monitoring and the optional tracing on the BusMonitor that was started before by the StartBusMonitoring action.

Closing an uninitialised or already closed **BusMonitor** shall perform no operation and report no errors. It shall be for all effects a NOP.

NOTE The related MVCI system method is MCDMonitoringLink::stop().

Parameters:

- <monitor> : busMon:BusMonitorVariable [1..1]

This element represents the **BusMonitor** that shall be stopped. After stopping a bus monitor the variable is no longer initialised.

Exceptions:

- None

2) Second edition under preparation. Stage at the time of publication: ISO/DIS 13209-2:2021.

8.7 Terms

8.7.1 Terms overview

All terms introduced by this extension extend the **otx:Term** extension interface. Information about the specific super class of a term is provided in the individual term description clauses.

8.7.2 Terms syntax

[Figure 14](#) shows the syntax of the **OTX BusMonitoring extension** related terms.

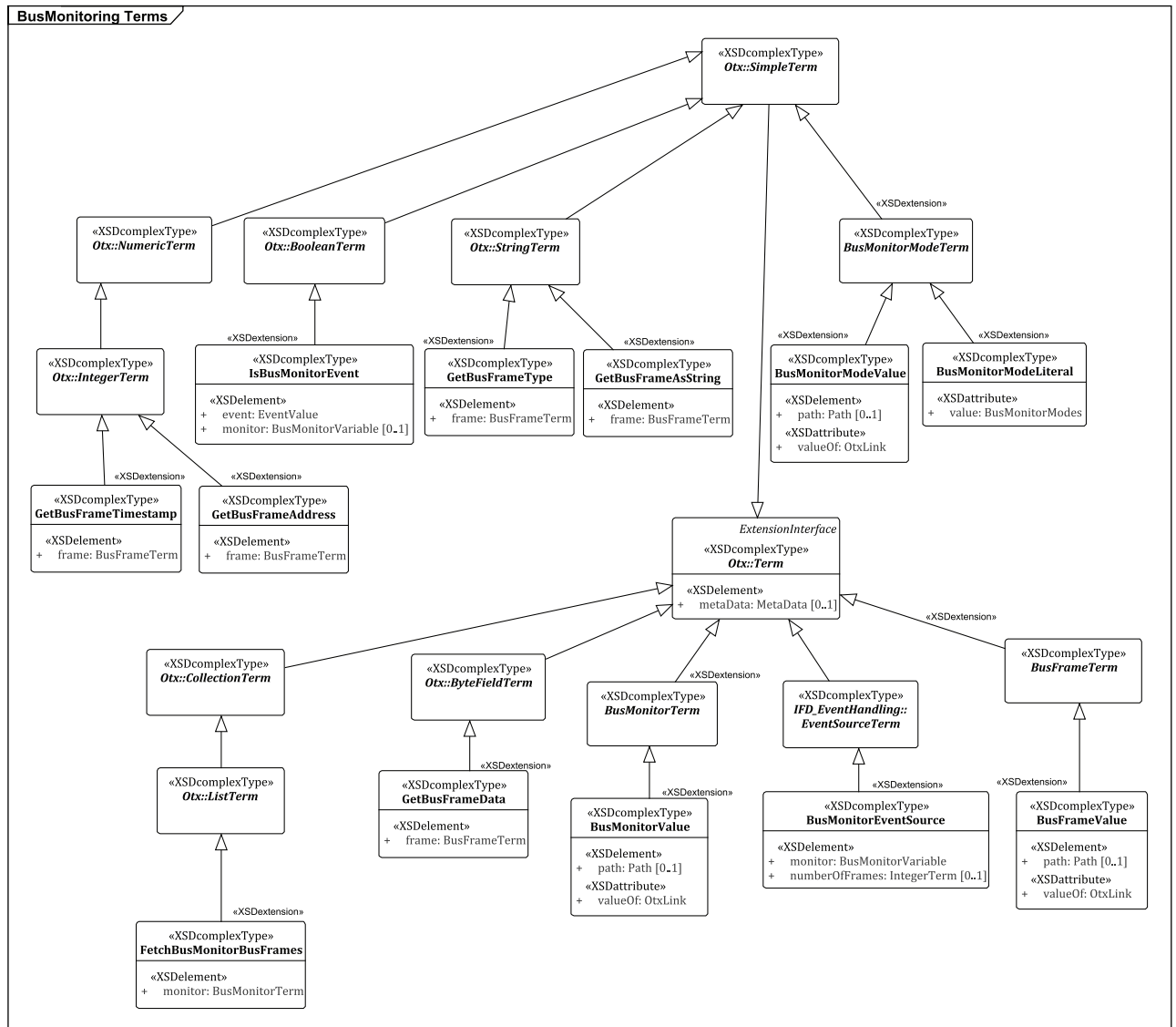


Figure 14 — BusMonitoring terms

8.7.3 Terms general semantics

8.7.3.1 BusFrameTerm

Semantic:

The abstract type `BusFrameTerm` serves as a base for all concrete terms that return a `BusFrame`. It has no special members.

8.7.3.2 BusFrameValue

Semantic:

Represents a frame value.

Return type:

busMon:BusFrameTerm

Parameters:

— <path> : otx:Path [0..1]

This element addresses parts of complex structures. For more information please refer to [ISO 13209-2](#).

Exceptions:

otx:InvalidReferenceException

— This exception is thrown if the **BusFrame** variable is not valid (no value was assigned to the variable before).

otx:OutOfBoundsException

— Occurs only, if a <path> is set and the <path> points to a location that does not exist (like a list index exceeding list length, or a map key which is not part of the map).

8.7.3.3 BusMonitorEventSource

Semantic:

The **BusMonitorEventSource** term accepts a **BusMonitor** object that is to be made an event source. This term enables an OTX sequence to use a **BusMonitor** as a source for events in the context of the OTX EventHandling extension. A **BusMonitor** shall trigger an event every time the number of frames in the buffer reaches the number in the numberOfFrames property.

NOTE 1 The related MVCI system operations are MCDMonitoringLink::setNoOfSampleToFireEvent and MCDEventHandler::onMonitoringFramesReady.

NOTE 2 In case of BusMonitoringMode == TRACING, BusMonitorEventSource will never fire an event.

Return type:

event:EventSourceTerm

Parameters:

— <monitor> : busMon:BusMonitorVariable [1..1]

Represents the **BusMonitorVariable** which can cause the event.

— <numberOfFrames> : otx:IntegerTerm [0..1]

Represents a number of frames which have to be collected in the buffer to fire an event. If the internal buffer is less than the number of frames, the event is fired, when the buffer is filled. Default value is 1.

Exceptions:

otx:TypeMismatchException

— This exception is thrown if the specified event has not been raised by a BusMonitorEventSource.

otx:InvalidReferenceException

- This exception is thrown if the variable is uninitialised.

8.7.3.4 BusMonitorModeLiteral

Semantic:

This term shall return a **BusMonitorMode** value from a hard-coded literal.

Return type:

busMon:BusMonitorModeTerm

Parameters:

- value : busMon:BusMonitorModes [1..1]

This attribute shall contain one of the values defined in the **BusMonitorModes** enumeration.

Exceptions:

- None

8.7.3.5 BusMonitorModeTerm

Semantic:

The abstract type **BusMonitorModeTerm** is an **otx:Term**. It serves as a base for all concrete terms that return a **BusMonitorMode** value. It has no special members.

8.7.3.6 BusMonitorModeValue

Semantic:

Returns the **BusMonitorMode** stored in a **BusMonitorMode** variable.

Return type:

busMon:BusMonitorModeTerm

Parameters:

- <path> : otx:Path [0..1]

This element addresses parts of complex structures. For more information please refer to [ISO 13209-2](#).

Exceptions:

otx:OutOfBoundsException

- Occurs only, if a <path> is set and the <path> points to a location, which does not exist (like a list index exceeding list length, or a map key which is not part of the map).

8.7.3.7 BusMonitorTerm

Semantic:

The abstract type **BusMonitorTerm** serves as a base for all concrete terms which returns a **BusMonitor** (e.g. CAN for monitoring). It has no special members.

8.7.3.8 BusMonitorValue

Semantic:

Represents a **BusMonitor** value.

Return type:

busMon:BusMonitorTerm

Parameters:

— <path> : otx:Path [0..1]

This element addresses parts of complex structures. For more information please refer to [ISO 13209-2](#).

Exceptions:

otx:InvalidReferenceException

— This exception is thrown if the **BusFrame** variable is not valid (no value was assigned to the variable before).

otx:OutOfBoundsException

— Occurs only, if a <path> is set and the <path> points to a location which does not exist (like a list index exceeding list length, or a map key which is not part of the map).

8.7.3.9 FetchBusMonitorBusFrames

Semantic:

The **FetchBusMonitorBusFrames** term accepts a **BusMonitor** and returns a list of **BusFrame**. All available frames will be returned and removed from buffer.

NOTE 1 The related MVCI system method is MCDMonitorLink::fetchMonitoringFrames(0).

NOTE 2 In case of BusMonitoringMode == TRACING, FetchBusMonitorBusFrames returns an empty list.

Return type:

otx:ListTerm

Parameters:

— <monitor> : busMon:BusMonitorTerm [1..1]

Represents a **BusMonitor** to get the frames from.

Exceptions:

— None

8.7.3.10 GetBusFrameAddress

Semantic:

The **GetBusFrameAddress** term accepts an **BusFrame** and returns an **otx:Integer** that represents the frame address [e.g. in case of CAN bus the 8 address bytes are converted (two's complement and little endian)].

Return type: