



**SDMX-ML:
SCHEMA AND DOCUMENTATION
(VERSION 1.0)**



- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33

Initial Release September 2004

© SDMX 2004

<http://www.sdmx.org/>



34

35 **I. BACKGROUND.....5**

36 **A. XML in the Case Study and Batch Data Exchange Projects 5**

37 **B. Results: the XML Design 6**

38 **C. Fostering the Use of a Standard SDMX-ML 6**

39 **II. NORMATIVE REFERENCES7**

40 **III. CONFORMANCE.....7**

41 **IV. DESIGN OVERVIEW7**

42 **A. Scope and Requirements 7**

43 **B. Design Approach..... 9**

44 **C. SDMX-ML Packaging: Namespace Modules..... 11**

45 **V. GENERIC (NON-KEY-FAMILY-SPECIFIC) SCHEMAS13**

46 **A. SDMX Message Namespace Module 14**

47 Global Elements..... 14

48 Complex Types 15

49 Simple Types..... 19

50 **B. SDMX Structure Namespace Module 19**

51 Complex Types 19

52 Simple Types..... 27

53 **C. SDMX Generic Data Namespace Module..... 28**

54 Global Elements..... 28

55 Complex Types 28



| | | |
|----|--|-----------|
| 56 | D. SDMX Query Namespace Module..... | 31 |
| 57 | Global Elements..... | 31 |
| 58 | Complex Types | 31 |
| 59 | Simple Types..... | 35 |
| 60 | E. SDMX Common Namespace Module..... | 36 |
| 61 | Complex Types | 36 |
| 62 | Simple Types..... | 36 |
| 63 | F. Data Formatting and Character Encoding | 37 |
| 64 | VI. KEY-FAMILY-SPECIFIC SCHEMAS: CORE STRUCTURES & STANDARD | |
| 65 | MAPPINGS | 37 |
| 66 | A. Compact Data Message Core Structure | 38 |
| 67 | Global Elements..... | 38 |
| 68 | Complex Types | 38 |
| 69 | B. Utility Data Message Core Structure | 39 |
| 70 | Global Elements..... | 39 |
| 71 | Complex Types | 39 |
| 72 | C. Cross-Sectional Data Message Core Structure | 40 |
| 73 | Global Elements..... | 40 |
| 74 | Complex Types | 40 |
| 75 | D. Mappings to Key-Family-Specific Schemas | 41 |
| 76 | General Rules: | 41 |
| 77 | Compact Schemas: | 42 |
| 78 | Cross-Sectional Schemas | 47 |
| 79 | Utility Schemas..... | 52 |
| 80 | VII. APPENDIX: SAMPLE SDMX-ML MESSAGES..... | 57 |
| 81 | A. CompactSample.xml..... | 57 |



82 **B. UtilitySample.xml..... 59**

83 **C. GenericSample.xml 59**

84 **D. CrossSectionalSample.xml..... 60**

85

86

87

88

89

90 **I. BACKGROUND**

91 **A. XML in the Case Study and Batch Data Exchange Projects**

92 During the course of the Batch Data Exchange (BDE) and Case Study Projects, two
93 XML schemas were developed, both based on the information model found in the
94 GESMES/TS specification. As a result, they were similar in many respects. However,
95 there were differences resulting from the differing technical requirements of these two
96 projects.

- 97 ○ The BDE XML was optimized for batch exchange of large data sets. It
98 was designed to support exactly the same type of exchanges for
99 which GESMES/TS was designed, but to leverage the benefits of an
100 XML syntax.
- 101 ○ The Case Study XML was designed and optimized to support web
102 dissemination and to accommodate a registry-based data-sharing
103 architecture.

104 It is clear that a single XML would be preferable to having multiple approaches and
105 this has fostered the development of a standard SDMX-ML at the earliest possible
106 date.

107

108 In looking at the combined requirements for all the processes supported by the
109 earlier work, it was determined that having a single document type was probably not
110 the best approach. All the SDMX technology artefacts (XML and EDIFACT data
111 formats, registry, etc.) share an information model, and thus carry the same

112 information. This fact was leveraged in the resulting XML design, for which there are
113 now five or six anticipated document types.

114

115 **B. Results: the XML Design**

116 All of these document types will share a common "envelope" at the message level
117 ("SDMXMessage.xsd"), as well as a set of common low-level components
118 ("SDMXCommon.xsd") so that header information and basic structure will always be
119 the same.

- 120 o Key family structure description schema ("SDMXStructure.xsd")
- 121 o Generic data schema for data-sharing exchange
122 ("SDMXGenericData.xsd")
- 123 o Generic query schema for invoking web services ("SDMXQuery.xsd")
- 124 o Key-family-specific schema for updates and revisions/bilateral
125 exchange ("SDMXCompactData.xsd")
- 126 o Key-family-specific schema for presentational processing and internal
127 use ("SDMXUtilityData.xsd")
- 128 o Requested: Key-family-specific schema for cross-sectional data –
129 which may be combined with the Compact document type
130 ("SDMXCrossSectionalData.xsd")

131

132 **C. Fostering the Use of a Standard SDMX-ML**

133 In addition to these different formats, standard mappings and corresponding
134 transformation tools are to be developed for the creation of key-family-specific
135 schemas from structure descriptions, to transform XML data instances from one XML
136 data description format to another, and from these formats into the corresponding
137 SDMX-ML messages. This level of free tools support will foster the early use of
138 SDMX and permit the data to be easily used across all processes, which is otherwise
139 a difficult requirement to meet. Ultimately, it is the fact that all formats share a
140 common information model that enables this approach to meet the wide set of SDMX
141 requirements.

142

143

144 **II. NORMATIVE REFERENCES**

145 W3C XML Schema Definition Language, version 1.0 (URL:
146 <http://www.w3c.org/XML/Schema#dev>), World Wide Web Consortium

147 W3C Extensible Markup Language, version 1.0, Third Edition (URL:
148 <http://www.w3c.org/TR/2004/REC-xml-20040204/>), World Wide Web Consortium

149 **III. CONFORMANCE**

150 Sections V and VI of this document are normative, providing rules for the creation of
151 conformant SDMX-ML XML instances and W3C XML Schemas.

152 **IV. DESIGN OVERVIEW**153 **A. Scope and Requirements**

154 To understand the relationships between the several document types, it is important
155 to have some familiarity with the requirements they are designed to fulfill.
156 Traditionally, GESMES/TS (and before that, GESMES/CB) were created for the
157 exchange of large amounts of data between counter-parties. This use of the data
158 format presents several requirements, which SDMX-ML adopts as its own, this being
159 one of the use cases it is required to support:

- 160 • Large amounts of data must be captured in a reasonably compact format,
161 because of the potential size of databases being exchanged.
- 162 • It must be possible to send incremental updates, rather than entire,
163 complete databases. The validation of such exchanges demands not
164 that an entire data set be exchanged, but only that enough information
165 be sent to ensure accurate updating and revision processes.
- 166 • Structural information as well as data will need to be transmitted.
- 167 • There must be a reliable transformation to and from the GESMES/TS
168 EDIFACT syntax.

- 169 • It should be possible to present natural-language information in multiple,
170 equivalent languages.

171 This was the set of requirements which the Batch Data Exchange XML format was
172 designed to meet. These types of exchanges tend to be bilateral in nature (or
173 “gateway” exchanges, in which a degree of standardization is imposed on a set of
174 bilateral exchanges). In these types of exchanges, both counter-parties have agreed
175 to the exchange process and the key families to be used, so that there is no difficulty
176 in these areas.

177

178 SDMX-ML faces a larger set of requirements, however. The biggest one of these is
179 the requirement to support web dissemination, in which there are not counter-parties,
180 per se, but rather a data provider and a data consumer. These roles have no
181 necessary relationship outside of a single exchange of data, and thus there may be
182 difficulties involved in understanding the dissemination process, the key families
183 used, etc. Additionally, SDMX-ML is designed to support the use of XML within a
184 registry-centric architecture, potentially using web services technology. These use
185 cases come with requirements additional to those of the bilateral exchange and
186 updating of databases:

- 187 • To support web services and similar technological approaches, there is a
188 requirement to send queries to information sources as well as data and
189 structure.
- 190 • Users (and registry services) may not know about a specific key family,
191 and will need to be able to handle data across key families, and even
192 (for, say, a comparison service) to put data structured according to
193 multiple key families in a single XML instance.
- 194 • The XML must be as simple as possible (but no simpler) to allow use by
195 web-masters and developers who are not familiar with statistics as a
196 domain.
- 197 • The XML should behave as “normally” as possible within standard XML
198 tools such as web development environments, parsers, guided editing
199 tools, etc.

- 200 • Validation of data sets should provide validation that the data set is
201 complete – the validation profile for incremental updates is not
202 sufficient.
- 203 • Data should be structured not only as time series data, but potentially
204 also as cross-sectional data, to meet the demands of different users. It
205 must be possible to take data structured according to a single key
206 family and transform it into a standard format enabling either of these
207 structural optimizations.
- 208 • XML formats should promote re-use of common semantics, concepts,
209 and codelists to the greatest possible extent, while still recognizing the
210 agency which maintains a specific resource (a codelist, a key family, a
211 data set, etc.)

212

213 This is a very broad set of requirements, and in examining these it becomes evident
214 that some of the requirements are very much at cross-purposes. It is almost
215 impossible to design a single XML document type which will satisfy all of these
216 requirements. At the same time, it was very much felt that whatever design was
217 adopted should have a clear relationship with the information model, so that it was
218 easily comprehensible to users who understood the idea of a key family and its
219 relationship to statistical data.

220

221 **B. Design Approach**

222 One of the most powerful aspects of the GESMES/TS implementation guide is its
223 data model, which allows the EDIFACT message to be used for many different types
224 of data. The XML design built on this approach by extending the use of the model to
225 span not only types of statistical data – expressed as key families – but also
226 syntaxes. A key family is a metadata construct – it can be expressed in many
227 syntaxes, but relies on none. In looking at the idea of using the SDMX Information
228 Model (a superset of the GESMES/TS data model) to span syntaxes, it became
229 apparent that a similar approach could be used to span use-case-specific XML
230 formats. Because they would all be based on the same model, their equivalence
231 would be guaranteed. With a simple transformation, anyone's data, expressed in

232 EDIFACT or a process-specific XML, could be transformed into the flavor preferred
233 by the receiver of the data. Further, from a processable description of a key family
234 (the XML description), it would be possible to generate format descriptions, tools, and
235 configurations specific to that key family.

236

237 The main argument against this approach is its apparent complexity, which is a
238 negative factor when launching international standards. In looking at requirements,
239 moreover, it was realized that not only were key-family-specific XML formats needed,
240 but also formats which could accommodate more than one key family without
241 changing – that is, to be non-key-family-specific.

242

243 The result of this analysis was the idea of a compromise position. It was immediately
244 agreed that there could be only one XML format for describing a key family – more
245 than one is unnecessary. A requirement existed for services which could use data
246 structured according to any key family, and sometimes in combination. This
247 presented the need for a “generic” data format. The querying requirement insisted
248 that a Query message be created (which had, at one time, been discussed within the
249 GESMES/TS community, although never finalized.) Additionally, it was seen that
250 there were at least two, and possibly three other scenarios which had significantly
251 conflicting requirements in terms of XML design:

252

- 253 • Database exchange, update, and revision
- 254 • “Normal” XML use and processing for webmasters, developers, and other
255 users of typical XML tools
- 256 • Exchange of cross-sectional data (which could potentially be the same as
257 the Database Exchange scenario)

258

259 To support the broad set of requirements, it was felt that a small number of standard
260 document types should be articulated, to meet specific processing requirements. This
261 included the three scenarios described above, and the use of the query document
262 type, which would only be needed for those developing web services or similar
263 applications involving run-time creation of SDMX-ML data from databases.

264

265 The idea of reuse has not been lost in this design, however – wherever possible,
266 common structures have been reused. This has resulted in a common “message”
267 structure, in which there is a single header shared by all document types, and a
268 single “envelope” (not to be confused with a web-services SOAP envelope, which
269 contains entire SDMX-ML messages of any type). Additionally, the core structure of
270 any key-family-specific XML document type should be common with that of any other,
271 to the greatest extent reasonably possible. A shared set of XML constructs was also
272 developed, to be used throughout all the XML formats, to increase consistency.

273

274 The end result is a primary division between “generic” XML formats, which are not
275 specific to particular key families, and a set of formats which are specific to key
276 families and to particular scenarios for use.

277

278 Such design decisions as whether something is to be expressed as an XML element
279 or attribute have been made based on the specific requirements for each XML
280 format. For those formats where compactness of data is paramount, almost
281 everything is expressed as attributes, because this results in a more compact
282 expression of the data. In other cases – in UtilityData messages, for example – other
283 types of structures are used which are more verbose, but which capture more of the
284 metadata expressed in the key family (eg, ordering of the key). This type of
285 difference in design stems always from the requirements for the specific XML format
286 being designed.

287

288 **C. SDMX-ML Packaging: Namespace Modules**

289 In the proposed XML Schema design, there is a packaging scheme based on the
290 idea that XML namespaces can be used as “modules”, so that any given user or
291 application need only be familiar with a subset of the entire library in order to use it.
292 This approach fit very well with the design described above, and is often used in
293 major XML standards for other domains.

294

295 The other major benefit of namespaces – especially in light of the requirement that
296 maintenance agencies be tracked across the potential reuse of the structures and

297 data they maintained – is that it allows SDMX to own certain namespace modules,
298 and allows other maintenance agencies to own namespaces specific to the key-
299 families they also maintain.

300

301 The result is a set of namespace packages which agree with the design approach
302 described above. Each module is a single instance of the W3C XML Schema
303 Language's `schema` element, associated with its own XML namespace. Where
304 these modules have dependencies on one another, they use the XML Schema
305 importing mechanism to draw on constructs described in another module.

306

307 • An SDMX Namespace Module containing the common message
308 constructs, including the common header information
309 (“SDMXMessage.xsd”) - used with all other SDMX-ML namespace
310 modules

311 • An SDMX Namespace Module containing the descriptions of structural
312 metadata such as key families, concepts, and codelists
313 (“SDMXStructure.xsd”)

314 • An SDMX Namespace Module containing constructs shared in common
315 across all of the SDMX message types (“SDMXCommon.xsd”) – needed
316 for all other SDMX-ML namespace modules (also included for
317 convenience is the XML namespace [“xml.xsd”] provided by the W3C
318 for including the `xml:lang` attribute in schemas).

319 • An SDMX Namespace Module describing the generic (non-key-family-
320 specific) format for formatting data (“SDMXGenericData.xsd”)

321 • An SDMX Namespace Module for describing the structure of the generic
322 query message (“SDMXQuery.xsd”) – for web services developers
323 and users, etc.

324 • An SDMX Namespace Module providing the common framework to be
325 used for all key-family-specific schemas for Database Exchange,
326 Update, and Revisions (“SDMXCompactData.xsd”) – for bilateral use

327 • A set of namespaced modules created and maintained by those who
328 create key-family-specific “Compact” schemas – not maintained by
329 SDMX

- 330 • An SDMX Namespace Module providing the common framework to be
331 used for all key-family-specific schemas for webmasters and
332 developers using standard XML tools (“SDMXUtilityData.xsd”) –for
333 processing and publication production use
- 334 • A set of namespaced modules created and maintained by those who
335 create key-family-specific “Utility” schemas – not maintained by SDMX
- 336 • An SDMX Namespace Module providing the common framework to be
337 used for all key-family-specific schemas for cross-sectional data
338 (“SDMXCrossSectionalData.xsd”) – for bilateral use and cross-
339 sectional processing of data
- 340 • A set of namespaced modules created and maintained by those who
341 create key-family-specific “Cross-sectional” schemas – not maintained
342 by SDMX

343 The following sections describe in detail the proposed XML formats, which should be
344 examined alongside the documentation provided. These proposed schemas are
345 divided into the generic schemas, for which a complete set of schema definitions can
346 be provided, and key-family-specific schemas, for which a core structure is provided
347 (with schema code), plus a guide to how a specific key-family can be mapped onto
348 the core structure.

349

350 **V. GENERIC (NON-KEY-FAMILY-SPECIFIC) SCHEMAS**

351 Some SDMX-ML schemas are the same for all key families. These include:

- 352 • `SDMXMessage.xsd`, for generically describing the basic message structure
353 common to all SDMX-ML messages
- 354 • `SDMXStructure.xsd`, for describing key families, code lists, and concepts
- 355 • `SDMXGenericData.xsd`, for describing data across key-families for generic
356 processing
- 357 • `SDMXQuery.xsd`, for marking-up queries against SDMX-conformat
358 databases and web services
- 359 • `SDMXCommon.xsd`, describing the common constructs used in other schemas



360 Of these, only the `SDMXStructure` message and the `SDMXGenericData`
361 message are required for general exchange of data. The documentation for each of
362 these schemas are provided below. (The schemas themselves are appended
363 separately.)

364 A. SDMX Message Namespace Module

365 **Namespace:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/message

366 *Imports:* http://www.SDMX.org/resources/SDMXML/schemas/v1_0/structure
367 (`SDMXStructure.xsd`)

368 *Imports:* http://www.SDMX.org/resources/SDMXML/schemas/v1_0/generic
369 (`SDMXGenericData.xsd`)

370 *Imports:* http://www.SDMX.org/resources/SDMXML/schemas/v1_0/utility
371 (`SDMXUtilityData.xsd`)

372 *Imports:* http://www.SDMX.org/resources/SDMXML/schemas/v1_0/compact
373 (`SDMXCompactData.xsd`)

374 *Imports:* http://www.SDMX.org/resources/SDMXML/schemas/v1_0/cross
375 (`SDMXCrossSectionalData.xsd`)

376 *Imports:* http://www.SDMX.org/resources/SDMXML/schemas/v1_0/query
377 (`SDMXQuery.xsd`)

378 *Imports:* http://www.SDMX.org/resources/SDMXML/schemas/v1_0/common
379 (`SDMXCommon.xsd`)

380

381 Global Elements

382 **Structure(`StructureType`):** The `Structure` is a message that contains all the
383 structural metadata about a data set. This can be key families, concepts, or codelists.

384 **GenericData(`GenericDataType`):** The `GenericDataType` is used to convey data
385 in a cross-key-family form.

386 **UtilityData(`UtilityDataType`):** The `UtilityData` contains data in an XML form which
387 is specific to each key family, according to standard mappings, and which is
388 optimized to support guided editing tools and other applications which expect a
389 "typical" XML schema. This format can be used to validate data in a key-family-
390 specific fashion as is typically expected of XML schemas, and requires the entire
391 data set. It cannot be used for incremental updates.

392 **CompactData(`CompactDataType`):** `CompactData` contains data in an XML
393 format which is optimized for incremental updating, and the transfer of large data sets

394 bilaterally. It is specific to each key family, according to standard mappings. It allows
395 for key values to be expressed at a Group level.

396 **CrossSectionalData(CrossSectionalDataType):** CrossSectionalData contains
397 data in an XML format which is optimized for describing many observations at a
398 single point in time, and for the transfer of large data sets bilaterally. It is specific to
399 each key family, according to standard mappings. It allows for key values to be
400 expressed from the Group level down to the Observation level, and permits multiple
401 observation values with different "measures". Time is attached at the DataSet level.

402 **QueryMessage(QueryMessageType):** The QueryMessageType is used to query
403 databases published on the web, and to invoke web services. It allows for queries to
404 be made regarding both data and structural metadata.

405 **MessageGroup(MessageGroupType):** The MessageGroupType is used to
406 allow for more than one data message of a single type to be included in a single
407 transmission. This element arises from the requirement for some services to be able
408 to exchange data which may come from more than one source, and be structured
409 according to more than one key family.

410 **Header(HeaderType):** Header type is declared globally so that it can function as
411 the head of a substitution group for schemas which are used internally. While this is
412 an exception to the overall design of SDMX-ML, many users feel this construct is
413 useful. Note that when SDMX-ML messages are exchanged outside an organization,
414 the standard header should be used - no assumptions about additional fields in
415 substituted types should be made unless explicitly agreed-to by counterparties.

416

417 **Complex Types**

418 **MessageType:** The Message is an abstract type which is used by all of the
419 messages, to allow inheritance of common features. It also provides uniqueness
420 constraints for the header fields.

421 *Element Content (Type):*

422

423 Header (HeaderType)

424 **StructureType:** StructureType defines the contents of a structure message.

425

426 *Extends:* MessageType

427

428 *Element Content (Type):*

429

430 Agencies (structure:AgenciesType) - min. 0

431 CodeLists (structure:CodeListsType) - min. 0

432 Concepts (structure:ConceptsType) - min. 0

433 KeyFamilies (structure:KeyFamiliesType) - min. 0

434 **GenericDataType:** GenericDataType defines the contents of a GenericData
435 message.

436

437 *Extends:* MessageType

438

439 *Element Content (Type):*

440

441 DataSet (generic:DataSetType)

442 **UtilityDataType:** UtilityDataType defines the contents of a UtilityData message.

443

444 *Extends:* MessageType

445

446 *Element Content (Type):*

447

448 [Reference] (utility:DataSet)

449 **CompactDataType:** CompactDataType defines the contents of a CompactData
450 message.

451

452 *Extends:* MessageType

453

454 *Element Content (Type):*

455

456 [Reference] (compact:DataSet)

457 **CrossSectionalDataType:** CrossSectionalDataType defines the contents of a
458 CrossSectionalData message.

459

460 *Extends:* MessageType

461

462 *Element Content (Type):*

463

464 [Reference] (cross:DataSet)

465 **QueryMessageType:** QueryMessageType defines the contents of a
466 QueryMessage.

467

468 *Extends:* MessageType

469

470 *Element Content (Type):*

471

472 Query (query:QueryType)

473 **MessageGroupType:** MessageGroupType defines the contents of a
474 MessageGroup message.

475

476 *Extends:* MessageType

477

Choice:

478

[Reference] (generic:DataSet) - max. unbounded

479

Choice:

480

[Reference] (utility:DataSet) - max. unbounded

481

Choice:

482

[Reference] (compact:DataSet) - max. unbounded

483

Choice:

484

[Reference] (cross:DataSet) - max. unbounded

485

Attribute: id(xs:NMTOKEN) - optional

486 **HeaderType:** HeaderType defines the header fields used for all messages. ID
487 identifies a data flow definition, which, when combined with time, uniquely identifies
488 the data set. Test indicates whether the message is for test purposes or not.
489 Truncated is used in data messages which are responding to Query messages, and
490 is set to true only if the response has been truncated to meet size limits suggested by
491 the defaultLimit attribute in the Query message. Name provides a name for the
492 transmission. Prepared is the date prepared. Sender is information about the sender,
493 and Receiver is information about the receiver. Agency provides the code
494 identifier/abbreviation for the maintenance agency of a data set. Data set id provides
495 an identifier for a contained data set. Action code provides a code for determining
496 whether the enclosed message is an Update or Delete message (not to be used with
497 the UtilityData message). KeyFamilyRef is used to reference a key family for a
498 contained data set, using its id. (This information is required at the DataSet level for
499 some messages, but is provided here as a convenience for those messages which
500 do not require it.) KeyFamilyAgency specifies the agency of the key family using its
501 coded id. Fields which refer to a contained data set need not be used if the message
502 contains a query or structural information - these messages provide specific fields for
503 holding this information. The ones here are not to be used as defaults. Extracted is a

504 time-stamp from the system rendering the data; ReportingBegin and ReportingEnd
505 provide the time period covered by the message (in the case of data). Source
506 provides human-readable information about the source of the data.

507 *Element Content (Type):*

508

509 ID (xs:NCName)

510 Test (xs:boolean)

511 Truncated (xs:boolean)

512 Name (common:TextType) - min. 0 - max. unbounded

513 Prepared (HeaderTimeType)

514 Sender (PartyType)

515 Receiver (PartyType) - min. 0 - max. unbounded

516 KeyFamilyRef (xs:NMTOKEN) - min. 0

517 KeyFamilyAgency (xs:NMTOKEN) – min. 0

518 DataSetAgency (xs:NMTOKEN) - min. 0

519 DataSetID (xs:NMTOKEN) - min. 0

520 DataSetAction (common:ActionType) - min. 0

521 Extracted (xs:dateTime) - min. 0

522 ReportingBegin (HeaderTimeType) - min. 0

523 ReportingEnd (HeaderTimeType) - min. 0

524 Source (common:TextType) - min. 0 - max. unbounded

525 **PartyType:** PartyType defines the information which is sent about various parties
526 such as senders and receivers of messages. The Name is the ID of the party, and
527 Contact provides contact details.

528 *Element Content (Type):*

529

530 Name (common:TextType) - min. 0 - max. unbounded

531 Contact (ContactType) - min. 0 - max. unbounded

532 *Attribute:* id (xs:NMTOKEN) - required

533 **ContactType:** ContactType provides defines the contact information about a party.
534 The Name provides a human-readable name.

535 *Element Content (Type):*

536

537 Name (common:TextType) - min. 0 - max. unbounded



538 Department (common:TextType) - min. 0 - max. unbounded
539 Role (common:TextType) - min. 0 - max. unbounded
540 *Choice*: min. 0 - max. unbounded
541 Telephone (xs:string)
542 Fax (xs:string)
543 X400 (xs:string)
544 URI (xs:string)
545 Email (xs:string)

546
547
548
549

550

551 **Simple Types**

552 **HeaderTimeType**: Provides a union type of xs:date and xs:dateTime for the
553 header fields in the message.

554
555

556 **B. SDMX Structure Namespace Module**

557 **Namespace**: http://www.SDMX.org/resources/SDMXML/schemas/v1_0/structure
558 *Imports*: http://www.SDMX.org/resources/SDMXML/schemas/v1_0/common
559 (SDMXCommon.xsd)

560

561 **Complex Types**

562 **AgenciesType**: AgenciesType contains one or more Agencies.

563 *Element Content (Type)*:

564

565 Agency (AgencyType) - max. unbounded

566 **AgencyType**: AgencyType provides a structure for describing agencies and their
567 contact information. The id attribute carries a code identifying the agency. The
568 version attribute indicates the version of the agency description. The uri attribute
569 provides a uri for an alternate way of identifying the agency information (typically a
570 URL resolving to an agency described in SDMX-ML). Name is an element which

571 provides for a human-readable name for the organization. MaintenanceContact
572 provides contact information for the agency when acting as a MaintenanceAgency;
573 CollectorContact does the same when the agency is acting as a statistics collector;
574 DisseminatorContact for when the agency functions as a statistics disseminator; and
575 ReporterContact for when the Agency is functioning as a statistics reporter.
576 OtherContact is used to describe any other role. Note that the Role field in the
577 contact information structure should only be specified for OtherContact. It is
578 allowable to reference full Agency information by using (at a minimum) only the id,
579 name, and uri fields, with the uri pointing to an external description in a valid SDMX-
580 ML Structure message which provides more complete information. (This is termed an
581 "external reference".) If an external reference is being made, the isExternalReference
582 attribute must be set to "true".

583 *Element Content (Type):*

584

585 Name (common:TextType) - max. unbounded

586 MaintenanceContact (ContactType) - min. 0

587 CollectorContact (ContactType) - min. 0

588 DisseminatorContact (ContactType) - min. 0

589 ReporterContact (ContactType) - min. 0

590 OtherContact (ContactType) - min. 0 - max. unbounded

591 *Attribute:* id (xs:NCName) - required

592 *Attribute:* version (xs:string) - optional

593 *Attribute:* uri (xs:anyURI) – optional

594 *Attribute:* isExternalReference (xs:Boolean) - optional

595 **ContactType:** ContactType provides defines the contact information about a party.
596 The id element is used to carry user id information for the contact, whereas Name
597 provides a human-readable name.

598 *Element Content (Type):*

599

600 Name (common:TextType) - min. 0 - max. unbounded

601 id (xs:NMTOKEN) - min. 0

602 Department (common:TextType) - min. 0 - max. unbounded

603 Role (common:TextType) - min. 0 - max. unbounded

604 *Choice:* min. 0 - max. unbounded

605 Telephone (xs:string)

606 Fax (xs:string)

607 X400 (xs:string)

608 URI (xs:string)

609 Email (xs:string)

610

611 **CodeListsType:** CodelistsType contains one or more codelists. It also defines
612 uniqueness constraints for codelist IDs.

613 *Element Content (Type):*

614

615 CodeList (CodeListType) - min. 0 - max. unbounded

616 **CodeListType:** CodeListType defines the contents of a codelist. This includes an
617 ID, the agency which maintains the codelist, its version, and a URL where it is
618 located. Elements are provided for supplying a name and the codes. It is acceptable
619 to provide only the id, name, and uri fields at a minimum, with the uri pointing to an
620 SDMX Structure message containing complete details on the codelist. (This is
621 termed an "external reference".) If an external reference is made, the
622 isExternalReference attribute must be set to "true".

623 *Element Content (Type):*

624

625 Name (common:TextType) - max. unbounded

626 Code (CodeType) – min. 0 - max. unbounded

627 Annotations (common:AnnotationsType) - min. 0

628 *Attribute:* id (xs:NCName) - required

629 *Attribute:* agency (xs:NMTOKEN) - optional

630 *Attribute:* version (xs:string) - optional

631 *Attribute:* uri (xs:anyURI) – optional

632 *Attribute:* isExternalReference (xs:Boolean) - optional

633 **CodeType:** CodeType defines the structure of a code. This allows for plain-text
634 descriptions as element content, and the coded value as the value attribute. (Short
635 descriptions or other presentational information may be added using Annotations with
636 an indicative type field [eg, "ShortDescription"]).

637 *Element Content (Type):*

638

639 Description (common:TextType) - max. unbounded

640 Annotations (common:AnnotationsType) - min. 0

641 *Attribute:* value (xs:NMTOKEN) - required

642 **ConceptsType:** ConceptsType defines the structure of a set of Concepts.

643 *Element Content (Type):*

644

645 Concept (ConceptType) - max. unbounded

646 **ConceptType:** ConceptType specifies the information provided for a single
647 concept. This includes a name, as element content, and an ID. It is possible to use
648 the uri field to point to the location of an SDMX-ML Structure message which
649 contains a more detailed version of the concept. (This is termed an "external
650 reference".) If an external reference is being made, the isExternalReference attribute
651 must be set to "true".

652 *Element Content (Type):*

653

654 Name (common:TextType) - max. unbounded

655 Annotations (common:AnnotationsType) - min. 0

656 *Attribute:* id (xs:NCName) - required

657 *Attribute:* agency (xs:NMTOKEN) - optional

658 *Attribute:* version (xs:string) - optional

659 *Attribute:* uri (xs:anyURI) – optional

660 *Attribute:* isExternalReference (xs:Boolean) - optional

661 **KeyFamiliesType:** KeyFamiliesType defines the structure for describing one or
662 more key families. It also provides uniqueness constraints for each of the key family
663 IDs.

664 *Element Content (Type):*

665

666 KeyFamily (KeyFamilyType) - max. unbounded

667 **KeyFamilyType:** KeyFamilyType defines the structure of a key-family description.
668 This includes the name and a set of components (attributes and dimensions) as
669 element content, and an ID, agency, version, and the URL where located as
670 attributes.

671 *Element Content (Type):*

672

673 Name (common:TextType) - max. unbounded

674 Components (ComponentsType)

675 Annotations (common:AnnotationsType) - min. 0

676 *Attribute:* id (xs:NCName) - required

677 *Attribute:* agency (xs:NMTOKEN) - optional

678 *Attribute:* version (xs:string) - optional

679 *Attribute:* uri (xs:anyURI) - optional

680 **ComponentsType:** ComponentsType describes the dimensions, groups,
681 attributes, and measures of the key family. If TimeDimension is included in the key
682 family - which it must be if time series formats for the data (GenericData,
683 CompactData, and UtilityData formats) are to be used - then there must also be a
684 frequency dimension.

685 *Element Content (Type):*

686

687 Dimension (DimensionType) - min. 0 - max. unbounded

688 TimeDimension (TimeDimensionType) - min. 0

689 PrimaryMeasure (PrimaryMeasureType)

690 CrossSectionalMeasure (CrossSectionalMeasureType) – min. 0 – max
691 unbounded

692 Group (GroupType) - min. 0 - max. unbounded

693 Attribute (AttributeType) - min. 0 - max. unbounded

694

695 **DimensionType:** DimensionType describes the structure of non-Time dimensions.
696 The order of their declaration is significant: it is used to describe the order in which
697 they will appear in data formats for which key values are supplied in an ordered
698 fashion (exclusive of the Time dimension, which is not represented as a member of
699 the ordered key). In the case of key families which are used for cross-sectional data
700 as well as time-series data, any "measure" dimension must have the value of the
701 "isMeasureDimension" attribute set to "true". If a dimension is declared to be a
702 measure dimension, it must have a measure declared elsewhere in the key family
703 which corresponds to each value in the codelist which represents it. Any dimension
704 which corresponds to the frequency concept must have its isFrequencyDimension
705 attribute set to "true". There may only be one such dimension in any key family.
706 (Conventionally, it is the first dimension in the ordered set of dimensions - the key.) If
707 a key family describes cross-sectional data, then for each non-time dimension, the
708 crossSectionalAttachDataSet, crossSectionalAttachGroup,
709 crossSectionalAttachSection, and crossSectionalAttachObservation attributes must
710 be given values. A value of "true" for any of these attributes indicates that the
711 dimension may be provided a value at the indicated level within the cross-sectional
712 structure. Note that these attributes do not need to be provided for any dimension
713 with the isFrequencyDimension set to "true", as these dimensions are always
714 attached only at the group level, as is time. A key family designed for cross-sectional
715 use must be structured such that any observation's complete key can be

716 unambiguously described by taking each dimension value from its observation level,
717 section level, group level, and data set level, and ordered according to the sequence
718 given in the key family.

719 *Element Content (Type):*

720

721 Annotations (common:AnnotationsType) - min. 0

722 *Attribute:* concept (xs:NMTOKEN) - required

723 *Attribute:* codelist (xs:NMTOKEN) - required

724 *Attribute:* isMeasureDimension (xs:boolean) - default: false

725 *Attribute:* isFrequencyDimension (xs:boolean) - default: false

726 *Attribute:* crossSectionalAttachDataSet (xs:boolean) - optional

727 *Attribute:* crossSectionalAttachGroup (xs:boolean) - optional

728 *Attribute:* crossSectionalAttachSection (xs:boolean) - optional

729 *Attribute:* crossSectionalAttachObservation (xs:boolean) - optional

730 **TimeDimensionType:** TimeDimensionType describes the special Time dimension.
731 Any key family which will be used for time-series formats (GenericData,
732 CompactData, and UtilityData) must include the time dimension. Any key family
733 which uses the time dimension must also declare a frequency dimension,
734 conventionally the first dimension in the key (the set of ordered non-time
735 dimensions). A TextFormat element may be included for indicating the representation
736 of time in some non-XML data formats. The concept attribute must contain the
737 concept name of the time concept. The codelist attribute may provide the value of the
738 concept name of a codelist if needed.

739 *Element Content (Type):*

740

741 TextFormat (TextFormatType) - min. 0

742 Annotations (common:AnnotationsType) - min. 0

743 *Attribute:* concept (xs:NMTOKEN) - required

744 *Attribute:* codelist (xs:NMTOKEN) - optional

745 **GroupType:** GroupType declares any useful groupings of data, based on a
746 selection of the declared (non-Time) dimensions (indicated with the DimensionRef
747 element) which form partial keys to which attributes may be attached. The value of
748 the DimensionRef element is the concept of the dimension - that is, the value of the
749 dimension's concept attribute. Thus, if data is to be presented as a set of time series
750 which vary only according to their differing frequencies, a "sibling group" would be
751 declared, with all dimensions except the frequency dimension in it. If data is

752 commonly grouped as a set of all countries, then a "Country Group" could be
753 declared, with all dimensions except the country dimension forming part of the partial
754 key. Any dimension which is not part of a group has a value which varies at the
755 series level (for time series formats). There is no requirement to have only a single
756 dimension omitted from a partial key - it can be any subset of the set of ordered
757 dimensions (that is, all dimensions except the time dimension, which may never be
758 declared as belonging to a group partial key). All groups declared in the key family
759 must be unique - that is, you may not have duplicate partial keys. All groups must
760 also be given unique names (id attributes). Although it is conventional to declare
761 dimensions in the same order as they are declared in the ordered key, there is no
762 requirement to do so - the ordering of the values of the key are taken from the order
763 in which the dimensions are declared. The Description element provides a human-
764 readable description (potentially in multiple, parallel languages) of the group. Note
765 that for cross-sectional formats, the named group mechanism is not used, but is
766 instead replaced by a generic group which carries time and frequency values with it,
767 and allows for any available group-level attributes to be specified if desired.

768 *Element Content (Type):*

769

770 DimensionRef (xs:NMTOKEN) - max. unbounded

771 Description (common:TextType) - min. 0 - max. unbounded

772 Annotations (common:AnnotationsType) - min. 0

773 *Attribute:* name (xs:NMTOKEN) - required

774 **AttributeType:** AttributeType describes the structure of attributes declared in the
775 key family. If the codelist attribute is not used, then the attribute is uncoded. You may
776 use the TextFormat element to specify constraints on the value of the uncoded
777 attribute. The concept attribute contains the name of a concept. The codelist attribute
778 supplies the id value of a codelist. The attachmentLevel attribute indicates the level
779 to which the attribute is attached in time-series formats (GenericData, CompactData,
780 and UtilityData formats). The assignmentStatus attribute indicates whether a value
781 must be provided for the attribute when sending documentation along with the data.
782 The AttachmentGroup element is included only when the attribute is attached at the
783 Group level, to indicate which declared group or groups the attribute may be attached
784 to. For each such group, an AttachmentGroup element should appear, with the
785 content of the element being the name of the group. The AttachmentMeasure
786 element is similar, indicating for cross-sectional formats which declared measure or
787 measures the attribute attached at the observation level may be attached to. The
788 isTimeFormat attribute indicates that the attribute represents the concept of time
789 format (typically a mandatory series-level attribute with a codelist representation
790 taken from ISO 8601). For key families not used to structure cross-sectional formats,
791 this element may be omitted. Each such element contains the name of the declared
792 measure. The attributes crossSectionalAttachDataSet, crossSectionalAttachGroup,
793 crossSectionalAttachSection, and crossSectionalAttachObservation indicate what the
794 attachment level or levels are for cross-sectional data formats, and may be omitted
795 if the key family will not be used to structure them. A value of "true" indicates that it
796 is permissible to provide a value for the attribute at the specified level within the
797 structure. Note that all groups in cross-sectional formats are replaced by a generic

798 group which has any values for time and frequency, and allows any group-level
 799 attributes to be attached to it.

800 *Element Content (Type):*

801

802 TextFormat (TextFormatType) - min. 0

803 AttachmentGroup (xs:NMTOKEN) - min. 0 - max. unbounded

804 AttachmentMeasure (xs:NMTOKEN) - min. 0 - max. unbounded

805 Annotations (common:AnnotationsType) - min. 0

806 *Attribute:* concept (xs:NMTOKEN) - required

807 *Attribute:* codelist (xs:NMTOKEN) - optional

808 *Attribute:* attachmentLevel (structure:AttachmentLevelType) -
 809 required

810 *Attribute:* assignmentStatus (structure:AssignmentStatusType) -
 811 required

812 *Attribute:* isTimeFormat (xs:boolean) – *default:* false

813 *Attribute:* crossSectionalAttachDataSet (xs:boolean) - optional

814 *Attribute:* crossSectionalAttachGroup (xs:boolean) - optional

815 *Attribute:* crossSectionalAttachSection (xs:boolean) - optional

816 *Attribute:* crossSectionalAttachObservation (xs:boolean) - optional

817 **TextFormatType:** TextFormatType defines the information for describing a text
 818 format. If the TextType attribute is not specified, any valid characters may be
 819 included in the text field. (It corresponds to the xs:string datatype of W3C XML
 820 Schema.) In this case, the Length attribute is interpreted as a maximum length.
 821 Otherwise, length provides either maximum or set string lengths as per the TextType
 822 attribute value. The decimals attribute provides the precision (the number of decimal
 823 places) that numeric data must use. This is an integer indicating the number of digits
 824 to occur after the decimal separator ("."). If used, a missing digit in numeric data is to
 825 be interpreted as a 0. If not used, no restrictions on the number of digits provided in
 826 data exist for the purposes of exchange.

827 *Attribute:* length (xs:integer) - optional

828 *Attribute:* decimals (xs:integer) - optional

829 *Attribute:* TextType (TextTypeType) - optional

830 **PrimaryMeasureType:** PrimaryMeasureType describes the observation
 831 values for all presentations of the data, except those cross-sectional formats

832 which have multiple measures (for which a set of cross-sectional measures
833 are used instead). The concept attribute points to the unique concept
834 represented by the measure. The PrimaryMeasure is conventionally
835 associated with the OBS-VALUE concept.

836 *Element Content (Type):*

837

838 Annotations (common:AnnotationsType) - min. 0

839 *Attribute:* concept (xs:NMTOKEN) - required

840 **CrossSectionalMeasureType:** CrossSectionalMeasureType describes the
841 observation values for multiple-measure cross-sectional data formats. For
842 non-cross sectional key families, it is not necessary to specify any cross-
843 sectional measures. The concept attribute points to the unique concept
844 represented by the measure. The measureDimension attribute contains the
845 concept name of the measure dimension. The code attribute contains the
846 value of its corresponding code in the codelist used to represent the measure
847 dimension. A CrossSectionalMeasure must be declared for each code in the
848 codelist used to represent the measure dimension - these will replace the
849 primary measure for multiple-measure cross-sectional data formats.

850 *Element Content (Type):*

851

852 Annotations (common:AnnotationsType) - min. 0

853 *Attribute:* concept (xs:NMTOKEN) - required

854 *Attribute:* measureDimension (xs:NMTOKEN) - required

855 *Attribute:* code (xs:NMTOKEN) - required

856

857 Simple Types

858 AttachmentLevelType:

859 *Restricts* xs:NMTOKEN

860 Code: DataSet - Data set level

861 Code: Group - Group level

862 Code: Series - Series level

863 Code: Observation - Observation level

864 AssignmentStatusType:

865 *Restricts* xs:NMTOKEN

866 Code: Mandatory - Providing attribute value is mandatory

867 Code: Conditional - Providing attribute value is optional

868 **TextTypeType:** TextTypeType provides an enumerated list of the types of
869 characters allowed in a TextFormat field.

870 *Restricts* xs:NMTOKEN

871 Code: Alpha - Allows any non-numeric characters to be used in the string,
872 with a maximum as specified in the length attribute.

873 Code: AlphaFixed - Allows any non-numeric characters to be used in the
874 string, with a set length as specified in the length attribute.

875 Code: Num - Allows any numeric character (0 - 9) to be used in the string,
876 with a maximum as specified in the length attribute.

877 Code: NumFixed - Allows any numeric character (0 - 9) to be used in the
878 string, with a set length as specified in the length attribute.

879 Code: AlphaNum - Allows any numeric or non-numeric characters to be
880 used in the string, with a maximum as specified in the length attribute.

881 Code: AlphaNumFixed - Allows any numeric or non-numeric characters to be
882 used in the string, with a set length as specified in the length attribute.

883

884

885

886 C. SDMX Generic Data Namespace Module

887 **Namespace:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/generic

888 *Imports:* http://www.SDMX.org/resources/SDMXML/schemas/v1_0/common

889 (SDMXCommon.xsd)

890

891 Global Elements

892 **DataSet(DataSetType):** The DataSet element contains one or more groups that
893 comprise the data set.

894

895 Complex Types

896 **DataSetType:** DataSetType defines the structure of a data set. This consists of a
897 key family reference which contains the ID of the key family, and the attribute values

898 attached at the data set level. A DataSet may be used to transmit documentation
899 (that is, only attribute values), data, or a combination of both. If providing only
900 documentation, you need not send the complete set of attributes. If transmitting only
901 data, the Group may be omitted if desired. Uniqueness constraints are defined for the
902 attributes of the data set. If dataset-level attributes are sent in a delete message,
903 then any valid attribute value will indicate that the current attribute value should be
904 deleted. The keyFamilyURI attribute is provided to allow a URI (typically a URL) to be
905 provided, pointing to an SDMX-ML Structure message describing the key family.

906 *Attribute:* keyFamilyURI (xs:anyURI) – optional

907 *Element Content (Type):*

908 KeyFamilyRef (xs:NCName)

909 Attributes (ValuesType) - min. 0

910 *Choice:* - min. 0 – max. unbounded

911 Group (GroupType) - min. 0 – max. unbounded

912 Series (SeriesType) – min. 0 – max. unbounded

913

914 Annotations (common:AnnotationsType) - min. 0

915 **GroupType:** The key values at the group level may be stated explicitly, and all
916 which are not wildcarded listed in GroupKey - they must also all be given a value at
917 the series level. It is not necessary to specify the group key, however, as this may be
918 inferred from the values repeated at the series level. If only documentation (group-
919 level attributes) are being transmitted, however, the GroupKey cannot be omitted.
920 The type attribute contains the name of the declared group in the key family. If any
921 group-level attributes are specified in a delete message, then any valid value
922 supplied for the attribute indicates that the current attribute value should be deleted
923 for the specified attribute.

924 *Attribute:* type (xs:NMTOKEN) – required

925 *Element Content (Type):*

926 GroupKey (ValuesType) – min. 0

927 Attributes(ValuesType) – min. 0

928 Series (SeriesType) - max. unbounded

929 Annotations (AnnotationsType) – min. 0

930

931 **SeriesType:** SeriesType specifies the structure of a series. This includes all of the
932 key values, values for all the attributes, and the set of observations making up the
933 series content. Messages may transmit only attributes, only data, or both.
934 Regardless, the series key is always required. Key values appear at the Series level
935 in an ordered sequence which corresponds to the key sequence in the key family. A

936 series in a delete message need not supply more than the key, indicating that the
937 entire series identified by that key should be deleted. If series attributes are sent in a
938 delete message, and valid value specified for an attribute indicates that the attribute
939 should be deleted.

940 *Element Content (Type):*

941

942 SeriesKey (SeriesKeyType)

943 Attributes (ValueType) - min. 0

944 Obs (ObsType) - min. 0 - max. unbounded

945 Annotations (common:AnnotationsType) - min. 0

946 **SeriesKeyType:** SeriesKeyType defines the contents of a series key. Each non-
947 time dimension must have a value supplied for it, in the order in which the
948 dimensions are specified in the key family.

949 *Element Content (Type):*

950

951 Value (ValueType) - max. unbounded

952 **ObsType:** ObsType defines the structure of an observation. This includes a time
953 and observation value, as well as values for each of the attributes assigned at the
954 observation level by the key family. In a delete message, only the time need be
955 given, indicating that the observation identified by the key and time should be
956 deleted. For an update message, both time and observation value are required. If
957 any attributes appear in a delete message, any valid value supplied for an attribute
958 indicates that the current value should be deleted.

959 *Element Content (Type):*

960

961 Time (common:TimePeriodType)

962 ObsValue (ObsValueType) - min. 0

963 Attributes (ValueType) - min. 0

964 Annotations (common:AnnotationsType) - min. 0

965 **ValueType:**

966 *Element Content (Type):*

967

968 Value (ValueType) - max. unbounded

969 **ValueType:** ValueType is used to assign a single value to a concept, as for attribute
970 values and key values. It has no element content.

971 *Attribute:* concept (xs:NCName)

972 *Attribute:* value (xs:string)

973 **ObsValueType:** ObsValueType describes the information set for an observation
974 value. This is associated with the primary measure concept declared in the key
975 family.

976 *Attribute:* value (xs:double)

977

978

979

980 **D. SDMX Query Namespace Module**

981 **Namespace:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/query

982 *Imports:* http://www.SDMX.org/resources/SDMXML/schemas/v1_0/common

983 (SDMXCommon.xsd)

984

985 **Global Elements**

986 **Query(QueryType):** The Query message allows standard querying of SDMX-
987 compliant databases and web services. It allows queries to retrieve data, key
988 families, codelists, and concepts.

989

990 **Complex Types**

991 **QueryType:** The Query element is a top-level element for this namespace, which is
992 referenced by the SDMX message envelope, or could be put inside another
993 envelope, such as SOAP. It contains a query. The defaultLimit attribute is the
994 suggested maximum response size in kilobytes.

995 *Element Content (Type):*

996

997 DataWhere (DataWhereType) - min. 0 - max. unbounded

998 KeyFamilyWhere (KeyFamilyWhereType) - min. 0 - max. unbounded

999 CodelistWhere (CodelistWhereType) - min. 0 - max. unbounded

1000 ConceptWhere (ConceptWhereType) - min. 0 - max. unbounded

1001 AgencyWhere (AgencyWhereType) - min. 0 - max. unbounded

1002 *Attribute:* defaultLimit (xs:integer) - optional

1003 **DataWhereType:** The DataWhere element represents a query for data. It contains
1004 all of the clauses in that query, represented by its child elements.

| | |
|------|--------------------------------|
| 1005 | <i>Element Content (Type):</i> |
| 1006 | (Choice) |
| 1007 | DataSet (xs:string) |
| 1008 | KeyFamily (xs:string) |
| 1009 | Dimension (DimensionType) |
| 1010 | Attribute (AttributeType) |
| 1011 | Codelist (CodelistType) |
| 1012 | Time (TimeType) |
| 1013 | Category (CategoryType) |
| 1014 | Concept (xs:string) |
| 1015 | Agency (xs:string) |
| 1016 | Or (OrType) |
| 1017 | And (AndType) |

1018 **AndType:** For the And element, each of its immediate child elements represent
1019 clauses all of which represent conditions which must be satisfied. If children are A, B,
1020 and C, then any legitimate response will meet conditions A, B, and C.

| | |
|------|---|
| 1021 | <i>Element Content (Type):</i> |
| 1022 | |
| 1023 | DataSet (xs:string) - min. 0 - max. unbounded |
| 1024 | KeyFamily (xs:string) - min. 0 - max. unbounded |
| 1025 | Dimension (DimensionType) - min. 0 - max. unbounded |
| 1026 | Attribute (AttributeType) - min. 0 - max. unbounded |
| 1027 | Codelist (CodelistType) - min. 0 - max. unbounded |
| 1028 | Time (TimeType) - min. 0 - max. unbounded |
| 1029 | Category (CategoryType) - min. 0 - max. unbounded |
| 1030 | Concept (xs:string) - min. 0 - max. unbounded |
| 1031 | Agency (xs:string) - min. 0 - max. unbounded |
| 1032 | Or (OrType) - min. 0 - max. unbounded |
| 1033 | And (AndType) - min. 0 - max. unbounded |

1034 **OrType:** The Or element's immediate children represent clauses in the query any
1035 one of which is sufficient to satisfy the query. If these children are A, B, and C, then
1036 any result which meets condition A, or condition B, or condition C is a match for that
1037 query.

| | |
|------|--------------------------------|
| 1038 | <i>Element Content (Type):</i> |
|------|--------------------------------|

1039

1040 DataSet (xs:string) - min. 0 - max. unbounded

1041 KeyFamily (xs:string) - min. 0 - max. unbounded

1042 Dimension (DimensionType) - min. 0 - max. unbounded

1043 Attribute (AttributeType) - min. 0 - max. unbounded

1044 Codelist (CodelistType) - min. 0 - max. unbounded

1045 Time (TimeType) - min. 0 - max. unbounded

1046 Category (CategoryType) - min. 0 - max. unbounded

1047 Concept (xs:string) - min. 0 - max. unbounded

1048 Agency (xs:string) - min. 0 - max. unbounded

1049 Or (OrType) - min. 0 - max. unbounded

1050 And (AndType) - min. 0 - max. unbounded

1051 **DimensionType:** Dimension elements contain the (single) value being searched on
1052 within the key of data set. The name attribute holds the agency-qualified ID of the
1053 dimension. If the content is empty, then the query is for any dimension with the given
1054 name. If the name attribute is not supplied, then the query is for the given key value
1055 within any dimension.

1056

1057 [data] (xs:string)

1058 **AttributeType:** Attribute elements contain the (single) value of an attribute being
1059 queried for. The name attribute contains the agency-qualified name of the attribute.
1060 The attachmentLevel attribute specifies the attachment level of the attribute. If the
1061 content of Attribute is empty, then the search is for the specified attribute (and
1062 attachment level). If the name attribute is not specified, then the search is on any
1063 attribute. If the attachmentLevel attribute is not specified, then the query is for an
1064 attribute at any attachment level, as the value defaults to "Any".

1065

1066 [data] (xs:string)

1067 **CodelistType:** The Codelist element allows queries to specify a (single) value
1068 found within a codelist as the element content, and the agency-qualified name of the
1069 codelist being queried for in the name attribute. If no content is supplied, then the
1070 query is for the named codelist. If the name attribute is left empty, then the value is
1071 searched for in any codelist.

1072

1073 [data] (xs:string)

1074 **CategoryType:** The Category element allows for a search to be made on the
1075 values within a specific category, which is specified (in agency-qualified form) with
1076 the name attribute. If there is no element content, then the search is for the named

1077 Category; if the name is not supplied, then the category value supplied as content
1078 should be sought-for in all available categories.

1079

1080 [data] (xs:string)

1081 **KeyFamilyWhereType:** The KeyFamilyWhere element represents a query for a
1082 key family or key families. It contains all of the clauses in that query, represented by
1083 its child elements.

1084 *Element Content (Type):*

1085 (Choice)

1086 KeyFamily (xs:string)

1087 Dimension (DimensionType)

1088 Attribute (AttributeType)

1089 Codelist (CodelistType)

1090 Category (CategoryType)

1091 Concept (xs:string)

1092 Agency (xs:string)

1093 Or (OrType)

1094 And (AndType)

1095 **CodelistWhereType:** The CodelistWhere element represents a query for a
1096 codelist or codelists. It contains all of the clauses in that query, represented by its
1097 child elements.

1098 *Element Content (Type):*

1099 (Choice)

1100 Codelist (CodelistType)

1101 Agency (xs:string)

1102 Or (OrType)

1103 And (AndType)

1104 **ConceptWhereType:** The ConceptWhere element represents a query for a
1105 concept or concepts. It contains all of the clauses in that query, represented by its
1106 child elements.

1107 *Element Content (Type):*

1108 (Choice)

1109 Concept (xs:string)

1110 Agency (xs:string)

1111 Or (OrType)

1112 And (AndType)

1113 **AgencyWhereType:** The AgencyWhere element represents a query for details
1114 for an Agency. It contains all of the clauses in that query, represented by its child
1115 elements.

1116 *Element Content (Type):*

1117 (Choice)

1118 DataSet (xs:string) - min. 0 - max. unbounded

1119 KeyFamily (xs:string) - min. 0 - max. unbounded

1120 Codelist (CodelistType) - min. 0 - max. unbounded

1121 Category (CategoryType) - min. 0 - max. unbounded

1122 Concept (xs:string) - min. 0 - max. unbounded

1123 Agency (xs:string) - min. 0 - max. unbounded

1124 Or (OrType) - min. 0 - max. unbounded

1125 And (AndType) - min. 0 - max. unbounded

1126 **TimeType:** TimeType contains the time point or period for which results
1127 should be supplied. When StartTime and EndTime are used, these must be
1128 understood as inclusive.

1129 *Element Content (Type):*

1130 (Choice)

1131 StartTime (common:TimePeriodType)

1132 EndTime (common:TimePeriodType) – min. 0

1133 Or:

1134 Time (common:TimePeriodType)

1135

1136 Simple Types

1137 **AttachmentLevelType:** This type supplies an enumeration of attachment levels
1138 corresponding to those in the SDMX Information Model, plus a value of "Any" where
1139 the search is wildcarded.

1140 *Restricts* xs:NMTOKEN

1141 Code: DataSet - Attached at the Data Set level

1142 Code: Group - Attached at the Group level

1143 Code: Series - Attached at the Series level

1144 Code: Observation - Attached at the Observation level

1145 Code: Any - Attached at any attachment level

1146

1147

1148

1149 E. SDMX Common Namespace Module

1150 **Namespace:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/common

1151 **Imports:** <http://www.w3.org/XML/1998/namespace> (xml.xsd)

1152

1153 Complex Types

1154 **TextType:** TextType provides for a set of language-specific alternates to be
1155 provided for any human-readable construct in the instance.

1156

1157 [data] (xs:string)

1158 **AnnotationType:** AnnotationType provides for non-documentation notes and
1159 annotations to be embedded in data and structure messages. It provides optional
1160 fields for providing a title, a type description, a URI, and the text of the annotation.

1161 *Element Content (Type):*

1162

1163 AnnotationTitle (xs:string) - min. 0

1164 AnnotationType (xs:string) - min. 0

1165 AnnotationURL (xs:anyURI) - min. 0

1166 AnnotationText (TextType) - min. 0 - max. unbounded

1167 **AnnotationsType:** AnnotationsType provides for a list of annotations to be
1168 attached to data and structure messages.

1169 *Element Content (Type):*

1170

1171 Annotation (AnnotationType) - max. unbounded

1172

1173 Simple Types

1174 **TimePeriodType:** TIME_PERIOD is not completely expressible in XML Schema's
1175 date type: instead we use the union of dateTime, date, gYearMonth, and gYear. The
1176 default name for the concept is TIME_PERIOD. Semi-annual and quarterly periods

1177 would be described in terms of their beginning month, weekly periods in terms of
1178 their Monday: e.g. the second quarter of 2002 as 2002-04, since it starts with April.

1179 **ActionType:** ActionType provides a list of actions, describing the intention of the
1180 data transmission from the sender's side. Each action applies to the entire dataset for
1181 which it is given.

1182 *Restricts* xs:NMTOKEN

1183 Code: Update - Data is an incremental update for an existing data set or the
1184 provision of new data or documentation (attribute values) formerly absent.

1185 Code: Delete - Data is to be deleted.

1186 **AlphaType:** This type is used for datatyping the contents of uncoded attributesIt
1187 places no restrictions on characters used, but carries the semantic of the key-family
1188 designer in a fashion similar to that of the corresponding SDMX_EDI message.

1189 *Restricts* xs:string

1190 **AlphaNumericType:** This type is used for datatyping the contents of uncoded
1191 attributes. It places no restrictions on characters used, but carries the semantic of the
1192 key-family designer in a fashion similar to that of the corresponding SDMX_EDI
1193 message.

1194 *Restricts* xs:string

1195

1196

1197

1198 F. Data Formatting and Character Encoding

1199 In all SDMX-ML documents – whether key-family-specific or not - the character
1200 encoding must be UTF-8. To simplify the exchange of statistical data and metadata
1201 globally, restrictions also apply to the expression of numeric formats: the decimal
1202 separator is always a period (“.”). There is no character used to separate thousands
1203 in data.

1204

1205 VI. KEY-FAMILY-SPECIFIC SCHEMAS: CORE STRUCTURES & STANDARD MAPPINGS

1206 Some schemas are specific to key families, and therefore there is no single schema
1207 for all users. In these cases, standard mappings are provided so that even though
1208 one schema cannot be published, the schemas can be predicted from an
1209 examination of SDMXStructure messages that describe the key families on which

1210 they are based. Automatic creation of key-family-specific schemas according to these
1211 mappings is a natural consequence of this correspondence, and free tools to enable
1212 this creation of key-family-specific schemas is envisioned.

1213 It is important to note that all key-family-specific schemas are based on a core of
1214 identical constructs, allowing the smallest possible number of tags to differ from key-
1215 family to key-family. This section first documents these “core” structures, each in their
1216 own SDMX-maintained namespace module, and then discusses the mappings from a
1217 key family to the key-family-specific schema.

1218 These schemas are all as similar as possible. They vary according to where in the
1219 common structure key values and attributes may be specified, and also – in the case
1220 of cross-sectional data – allow for time to be specified only once, at the data set
1221 level, along with the incidence of multiple observations. A less obvious difference is
1222 seen in the Utility schema, which is designed to carry as much structural metadata as
1223 possible in order to allow “typical” XML tools (such as schema-guided editors and
1224 parsers) to benefit from the availability of this data - such tools are generally
1225 incapable of consulting the key family for structural metadata.

1226 **A. Compact Data Message Core Structure**

1227 **Namespace:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/compact

1228 **Imports:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/common

1229 (SDMXCommon.xsd)

1230

1231 **Global Elements**

1232 **DataSet(DataSetType):** The DataSet element contains the data set.

1233 **Group(GroupType):** The Group element contains the group.

1234 **Series(SeriesType):** The Series element contains the series.

1235 **Obs(ObsType):** The Obs element contains the observation.

1236

1237 **Complex Types**

1238 **DataSetType:** DataSetType acts as a structural base, which is extended through
1239 the addition of attributes to reflect the particular needs of a specific key family using
1240 the xs:extends element.

1241 **GroupType:** GroupType acts as a structural base, which is renamed and extended
1242 through the addition of attributes to reflect the particular needs of a specific key
1243 family using the xs:extends element.

1244 **SeriesType:** SeriesType acts as a structural base, which is extended through the
1245 addition of attributes to reflect the particular needs of a specific key family using the
1246 xs:extends element.

1247 **ObsType:** ObsType acts as a structural base, which is extended through the
1248 addition of attributes to reflect the particular needs of a specific key family using the
1249 xs:extends element.

1250

1251

1252

1253 B. Utility Data Message Core Structure

1254 **Namespace:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/utility

1255 **Imports:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/common

1256 (SDMXCommon.xsd)

1257

1258 Global Elements

1259 **DataSet(DataSetType):** DataSet exists to act as the head of a substitution group
1260 to which key-family-specific attributes and elements are bound.

1261 **Group(GroupType):** Group exists to act as the head of a substitution group to
1262 which key-family-specific attributes and elements are bound.

1263 **Series(SeriesType):** Series exists to act as the head of a substitution group to
1264 which key-family-specific attributes and elements are bound.

1265 **Key(KeyType):** Key is an element which serves as the head of a substitution group
1266 containing the key-family-specific key values.

1267 **Obs(ObsType):** Obs exists to act as the head of a substitution group to which key-
1268 family-specific attributes and elements are bound.

1269

1270 Complex Types

1271 **DataSetType:** DataSetType acts as a structural base, which is extended through
1272 the addition of attributes and elements to reflect the particular needs of a specific key
1273 family using the xs:extends element.

1274 **GroupType:** GroupType acts as a structural base, which is renamed and extended
1275 through the addition of attributes to reflect the particular needs of a specific key
1276 family using the xs:extends element.

1277 **SeriesType:** SeriesType acts as a structural base, which is extended through the
1278 addition of attributes to reflect the particular needs of a specific key family using the
1279 xs:extends element.

1280 **KeyType:** KeyType describes the abstract type which defines the Key element.

1281 **ObsType:** ObsType acts as a structural base, which is extended through the
1282 addition of attributes to reflect the particular needs of a specific key family using the
1283 xs:extends element.

1284

1285 C. Cross-Sectional Data Message Core Structure

1286 **Namespace:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/cross
1287 **Imports:** http://www.SDMX.org/resources/SDMXML/schemas/v1_0/common
1288 (SDMXCommon.xsd)

1289

1290 Global Elements

1291 **DataSet(DataSetType):** DataSet contains the data set.

1292 **Group(GroupType):** Group contains the group.

1293 **Section(SectionType):** Section contains the data section.

1294 **Obs(ObsType):** Obs contains the observation, with one or more measures.

1295

1296 Complex Types

1297 **DataSetType:** DataSetType acts as a structural base, which is extended through
1298 the addition of attributes to reflect the particular needs of a specific key family using
1299 the xs:extends element.

1300 **GroupType:** GroupType acts as a structural base, which is extended through the
1301 addition of attributes to reflect the particular needs of a specific key family using the
1302 xs:extends element. The time attribute holds the value for the time dimension
1303 concept as specified in the key family. If time is not used as a concept in the key
1304 family, then no value need be provided.

1305 *Attribute:* time (common:TimePeriodType) - optional

1306 **SectionType:** SectionType acts as a structural base, which is extended through the
1307 addition of attributes to reflect the particular needs of a specific key family using the
1308 xs:extends element.

1309 **ObsType:** ObsType acts as a structural base, which is extended through the
1310 addition of attributes to reflect the particular needs of a specific key family using the
1311 xs:extends element. It is capable of expressing the value and attributes of any single
1312 available cross-sectional measure (when extended).

1313

1314

1315 **D. Mappings to Key-Family-Specific Schemas**

1316 **General Rules:**

1317

1318 For all key-family-specific schemas (Compact, Utility, and Cross-Sectional) SDMX
1319 provides a namespace to be used as the extension base for key-family-specific
1320 schemas of that type. The key-family-specific schema will be created in its own target
1321 name space, owned and maintained by the creating agency. It will use the
1322 targetNamespace attribute of the schema element to identify the namespace which
1323 contains the key-family-specific schema. The namespace module provided by SDMX
1324 for that class of key-family-specific schema will be incorporated using the import
1325 element in the key-family-specific schema. The SDMX Common namespace module
1326 must also be imported into the schema. Other xml:namespace attributes may be
1327 added to the schema element as needed.

1328

1329 The elementFormDefault attribute on the schema element will be given a value of
1330 "qualified", and the attributeFormDefault attribute on the schema element will be
1331 given a value of "unqualified".

1332

1333 All additions to the SDMX module will be made using the extends element from W3C
1334 XML Schema. The term "levels of structure," when referring to the imported SDMX
1335 modules, include the following:

1336

- 1337 • DataSet level
- 1338 • Group level
- 1339 • Series level
- 1340 • Observation level

1341

1342 These levels normally refer to the element provided by the SDMX module to which
1343 attributes and elements may be assigned. In some cases, specific named constructs
1344 in the key family will become members of a set of elements corresponding to one of
1345 the levels named above.

1346

1347 For all of the key-family-specific mappings provided below, SDMX-ML namespace
1348 modules are identified with the abbreviations used in the standard schemas
1349 (“compact:” refers to the CompactData module; “common:” to the Common
1350 namespace module, “utility:” to the UtilityData namespace module; and “cross:” to the
1351 CrossSectionalData module).

1352

1353 Note that for all of the following mappings the term “concept name” is the value of the
1354 id attribute of the concept as found in the SDMX-ML message describing the key
1355 family.

1356

1357 **Compact Schemas:**

1358

1359 Compact schemas express all attribute and dimension values as XML attributes.
1360 These may be placed at various levels within the imported SDMX “compact”
1361 structure. The key-family-specific schema uses XSD substitution groups to attach
1362 key-family-specific elements and attributes to the structures provided in the
1363 “compact:” namespace.

1364

1365 A global element named “DataSet” will be declared, with an XSD substitutionGroup
1366 attribute which has a value referencing the DataSet element in the “compact:”
1367 namespace. Its type attribute will reference DataSetType in the key-family-specific
1368 namespace.

1369

1370 An XSD complexType will be declared named “DataSetType”. It will have XSD
1371 complexContent containing an XSD extension element, with a base attribute of
1372 DataSetType in the “compact:” namespace. The extension will consist of an XSD
1373 choice element, with a minOccurs attribute with a value of “0” and a maxOccurs
1374 value of “unbounded”. The choice will contain an XSD element reference for each

1375 named group declared in the key family. They will each have an XSD ref attribute
1376 with a value of the group name provided in the key family. (These elements will take
1377 the names of the groups declared in the key family.) Additionally, an XSD element
1378 will be declared in the choice with a ref attribute with a value of Series. Further, an
1379 element named Annotations will be added to the choice, with a type of
1380 AnnotationsType from the “common:” namespace.

1381

1382 For each attribute declared in the key family with an attachmentLevel of “DataSet”,
1383 an XML attribute will also be declared in the extension. It will have the same name as
1384 the attribute’s concept in the key family. It will have a “use” attribute value of
1385 “optional”. For coded attributes, the XML attribute will be given a type value which is
1386 the name of the codelist which represents it. In the key-family-specific namespace,
1387 this codelist will be represented by a simpleType declaration which contains a list of
1388 enumerations, equivalent to the values of the codelist, as described in the key family.
1389 These will be extensions of the XSD "string" datatype. The enumerated values will be
1390 the values of the codes. The descriptions of the codes will be placed inside XSD
1391 "documentation" elements, contained in XSD "annotation" elements, which are
1392 themselves contained in the XSD "enumeration" elements as the first instance of the
1393 XSD documentation element. No other text shall occur within this particular instance
1394 of the XSD documentation element, although other XSD documentation elements
1395 may occur within any given XSD enumeration element.

1396

1397 Uncoded attributes will also be represented with XSD simpleType elements declared
1398 in the key-family-specific namespace, with names formed by taking the name of the
1399 attribute in the key family and appending “Type” to them. If unrestricted, these will be
1400 of the W3C XML Schema primitive type “string”; if restrictions are specified in the key
1401 family, these will be restrictions of the XSD "string" datatype, unless they have a
1402 maximum length specified in the key family. If a maximum length is provided in the
1403 key family description, this will be handled as follows:

1404

- 1405 • If numeric, then the restriction base will be of the XSD datatype "decimal".
- 1406 • If alphabetic, then the restriction base will be of the common:AlphaType
1407 datatype.

- 1408 • If alphanumeric, then the restriction base will be of the
1409 common:AlphaNumericType datatype.

1410

1411 If maximum length is specified, but the attribute's value is not fixed length, then the
1412 maxLength facet in the XSD simpleType should be set to equal the maximum length
1413 of the attribute as specified in the key family. If the attribute's value is fixed length,
1414 then the XSD minLength attribute should additionally be set to the same value. If an
1415 uncoded attribute is a numeric type, and a number of decimals has been specified in
1416 the key family, then the simple type's fracDig facet should take the value specified in
1417 the key family.

1418

1419 For each named Group in the key family, a global XSD element will be declared,
1420 taking the name of the group. Its XSD type attribute will have a value formed by
1421 taking the name of the element and adding "Type" to the end of it. It will have a
1422 substitutionGroup attribute which references the Group element declared in the
1423 "compact:" namespace.

1424

1425 An XSD complexType will be declared for each named group declared in the key
1426 family, with a name formed by taking the name of the group in the key family and
1427 appending "Type" to it. It will have an XSD complexContent element which contains
1428 an XSD extends with a base attribute value of compact:GroupType. The extends will
1429 contain an XSD sequence element. An element named Annotations will be added to
1430 the end of the sequence, with a type of AnnotationsType from the "common:"
1431 namespace. It will also have a minOccurs value of "0".

1432

1433 For each attribute in the key family with an attachmentLevel of "Group", an XSD
1434 attribute element will be added to the extends element, with a use attribute set to
1435 "optional" and a type attribute defined as for the DataSet level, above. The name will
1436 be the concept name of the attribute in the key family.

1437

1438 For each dimension referenced by DimensionRef element in the named Group
1439 declaration in the key family XML , an XSD attribute element will also be added to the
1440 extends element, with a use attribute set to "required" and a type defined as for



1441 coded attributes for the dataset level, above. The name will be the concept name of
1442 the dimension in the key family.

1443

1444 A XSD global element named Series will be declared in the key-family-specific
1445 namespace, with a type of SeriesType and a substitutionGroup attribute referencing
1446 compact:Series.

1447

1448 An XSD complexType will then be declared with a name of SeriesType. It will have
1449 XSD complexContent, with an XSD extension element that has a base attribute value
1450 of compact:SeriesType. The extends element will contain an XSD sequence
1451 element, which will contain an XSD element with a ref attribute whose value is “Obs”.
1452 Its minOccurs attribute will have a value of “0” and a maxOccurs value of
1453 “unbounded”. An element named Annotations will be added to the end of the
1454 sequence, with a type of AnnotationsType from the “common:” namespace. It will
1455 also have a minOccurs value of “0”.

1456

1457 For each attribute in the key family with an attachmentLevel of “Series”, an XSD
1458 attribute element will be added to the extends element, with a use attribute set to
1459 “optional” and a type attribute defined as for the DataSet level, above. The name will
1460 be the name of the attribute’s concept in the key family. The exception is where an
1461 attribute has an isTimeFormat attribute value of “true” – in this case, it is treated the
1462 same as other series-level attributes except that its use attribute has a value of
1463 “required”.

1464

1465 An XSD global element will be declared named “Obs”. It will have a
1466 substitutionGroup attribute with a value “compact:Obs”. It will have a type of
1467 “ObsType”.

1468

1469 An XSD complexType element will be declared with a name “ObsType” and an XSD
1470 complexContent. This will contain an XSD extends element with a base attribute of
1471 “compact:ObsType”. It will contain an XSD sequence element. The sequence
1472 element will contain an element named Annotations, with a type of AnnotationsType
1473 from the “common:” namespace. It will have a minOccurs value of “0”.

1474

1475 The extension element will also have an XSD attribute element in it, which will have a
1476 name attribute whose value is the name of the TimeDimension concept from the key
1477 family. It will have a use attribute of “optional” and a type of
1478 “common:TimePeriodType”.

1479

1480 The extension element will also have an XSD attribute element in it, which will have a
1481 name attribute whose value is the concept name of the primary measure from the
1482 key family. It will have a use attribute of “optional” and a type of XSD “double”.

1483

1484 For each attribute declared in the key family with an attachmentLevel of
1485 “Observation”, an XSD attribute will be added to the extends. Each XSD attribute will
1486 take the name of the attribute’s concept declared in the key family, and will have a
1487 use attribute of “optional”. Its type will be defined as for the DataSet-level attributes
1488 described above.

1489

1490 No other declarations or constructs will be added to the schemas created using this
1491 mapping.

1492

1493 **Time Ranges in CompactData:** Unlike any other SDMX-ML data format, the key-
1494 family-specific CompactData format can express a set of observation values without
1495 having to provide a time for each of them. If a Series has a time provided for the first
1496 observation, subsequent observations in the series may omit the time, and only
1497 provide an observation value (a value for the attribute named after the primary
1498 measure), and whatever attributes are needed (see below). The times of the
1499 subsequent observations can be calculated according to the frequency specified by
1500 the relevant time format attribute value (or, failing that, the frequency dimension
1501 value), which can be calculated by the application. Note that support for this
1502 functionality is not mandatory for applications which do not claim this support in their
1503 conformance statements. It is also permissible to supply a time value for the last
1504 observation in the series, to permit double-checking of the calculation, although this
1505 is not mandatory.

1506

1507 **Delete and Update Messages in CompactData:** In the Header element, the action
1508 field specifies whether a message is an update message or a delete message. If it is
1509 an update message, it is used to send new information or updated information, which
1510 may include only data, only documentation (that is, attribute values as described in
1511 the key family), or both. (Agreements regarding the use of update messages should
1512 be specified between counterparties.) For a delete message, the requirements are
1513 that a complete series key always be sent for the deletion of data, which is identified
1514 either as an entire series by the absence of any specified time periods, or for a
1515 specific set of time periods, by the inclusion of those time periods. Attribute values
1516 may be deleted by sending a complete or partial set of attributes, with any valid value
1517 for the attribute (according to the XSD schema) being taken as an indication that the
1518 current attribute value should be deleted.

1519

1520

1521

1522 **Cross-Sectional Schemas**

1523

1524 Key-family-specific cross-sectional schemas express all non-time-series-based
1525 presentations of the data which are made possible in the key family. They also are
1526 capable of expressing statistical data for which time is not a concept – that is, they
1527 can provide the only SDMX-ML format for data which is inherently only cross-
1528 sectional. As with the CompactData format, key values and attribute values are
1529 attached to a four-level structure as XML attributes. For cross-sectional data,
1530 however, the term “Series” – an abbreviation of “time series” – is replaced by the
1531 equivalent “Section” construct.

1532

1533 Please note that named groups declared in the key family are ignored for the
1534 purposes of the cross-sectional data format. They are replaced by a generic Group
1535 element, leaving it up to the writing or processing application to enforce the validity of
1536 attribute values for groups of Sections. This is true also because a single SDMX-ML
1537 cross-sectional schema may be described in the key family such that it allows for
1538 more than one dimension to be expressed at the observation level, replacing the role



1539 of time in time-series-oriented formats, and therefore allows key values and attribute
1540 values to be attached at more than one level.

1541

1542 A global element named "DataSet" will be declared, with an XSD substitutionGroup
1543 attribute which has a value referencing the DataSet element in the "cross:"
1544 namespace. Its type attribute will reference DataSetType in the key-family-specific
1545 namespace.

1546

1547 An XSD complexType will be declared named "DataSetType". It will have XSD
1548 complexContent containing an XSD extension element, with a base attribute of
1549 DataSetType in the "cross:" namespace. The extension will consist of an XSD
1550 choice element, with a minOccurs of "0" and a maxOccurs of "unbounded". The choice
1551 element will contain an XSD element reference with a value of "Group". Additionally,
1552 an XSD element will be declared in the choice with a ref attribute, whose value is
1553 Section. Further, an element named Annotations will be added to the choice, with a
1554 type of AnnotationsType from the "common:" namespace. It will have a minOccurs
1555 attribute of "0".

1556

1557 For each attribute or dimension declared in the key family with a
1558 crossSectionalAttachDataSet of "true", an XML attribute will also be declared in the
1559 extension. It will have the same name as the attribute concept or dimension concept
1560 in the key family. It will have a "use" attribute value of "optional". For coded attributes,
1561 the XML attribute will be given a type value which is the name of the codelist which
1562 represents it. In the key-family-specific namespace, this codelist will be represented
1563 by a simpleType declaration which contains a list of enumerations, equivalent to the
1564 values of the codelist, as described in the key family. These will be extension of the
1565 XSD "string" datatype. The enumerated values will be the values of the codes. The
1566 descriptions of the codes will be placed inside XSD "documentation" elements,
1567 contained in XSD "annotation" elements, which are themselves contained in the XSD
1568 "enumeration" elements as the first instance of the XSD documentation element. No
1569 other text shall occur within this particular instance of the XSD documentation
1570 element, although other XSD documentation elements may occur within any given
1571 XSD enumeration element.

1572

1573 Uncoded attributes will also be represented with XSD simpleType elements declared
1574 in the key-family-specific namespace, with names formed by taking the name of the
1575 attribute concept in the key family and appending "Type" to them. If unrestricted,
1576 these will be of the W3C XML Schema primitive type "string"; if restrictions are
1577 specified in the key family, these will be restrictions of the XSD "string" datatype,
1578 unless they have a maximum length specified in the key family. If a maximum length
1579 is provided in the key family description, this will be handled as follows:

1580

- 1581 • If numeric, then the restriction base will be of the XSD datatype "decimal".
- 1582 • If alphabetic, then the restriction base will be of the common:AlphaType
1583 datatype.
- 1584 • If alphanumeric, then the restriction base will be of the
1585 common:AlphaNumericType datatype (where "common:" denotes the
1586 SDMX Common namespace module).

1587

1588 If maximum length is specified, but the attribute's value is not fixed length, then the
1589 maxLength facet in the XSD simpleType should be set to equal the maximum length
1590 of the attribute as specified in the key family. If the attribute's value is fixed length,
1591 then the XSD minLength attribute should additionally be set to the same value. If an
1592 uncoded attribute is a numeric type, and a number of decimals has been specified in
1593 the key family, then the simple type's fracDig facet should take the value specified in
1594 the key family.

1595

1596 A Global XSD element will be declared named Group. Its XSD type attribute will have
1597 a value of GroupType. It will have a substitutionGroup attribute which references the
1598 Group element declared in the "cross:" namespace.

1599

1600 An XSD complexType named GroupType will be declared. It will have an XSD
1601 complexContent element which contains an XSD extends with a base attribute value
1602 of compact:GroupType. The extends will contain an XSD sequence element, which
1603 will contain an XSD element with a reference to the element Section. Its minOccurs
1604 attribute will have a value of "0" and a maxOccurs value of "unbounded". An element

1605 named Annotations will be added to the end of the sequence, with a type of
1606 AnnotationsType from the “common:” namespace. It will also have a minOccurs
1607 value of “0”.

1608

1609 For each attribute or dimension in the key family with a crossSectionalAttachGroup
1610 value of “true” or an isFrequencyDimension value of “true”, an XSD attribute element
1611 will be added to the extends element, with a use attribute set to “optional” and a type
1612 attribute defined as for the DataSet level, above. The name will be the name of the
1613 attribute concept or dimension concept in the key family.

1614

1615

1616 A XSD global element named Section will be declared in the key-family-specific
1617 namespace, with a type of SectionType and a substitutionGroup attribute referencing
1618 compact:Section.

1619

1620 An XSD complexType will then be declared with a name of SectionType. It will have
1621 XSD complexContent, with an XSD extension element that has a base attribute value
1622 of cross:SectionType. The extends element will contain an XSD choice element with
1623 a minOccurs of “0” and a maxOccurs of “unbounded”, which will contain an XSD
1624 element for each CrossSectionalMeasure declared in the key family, with a ref
1625 attribute whose value is the name of the measure’s concept. An element named
1626 Annotations will be added to the end of the choice, with a type of AnnotationsType
1627 from the “common:” namespace.

1628

1629 For each attribute or dimension in the key family with a crossSectionalAttachSection
1630 value of “true”, an XSD attribute element will be added to the extends element, with a
1631 use attribute set to “optional” and a type attribute defined as for the DataSet level,
1632 above. The name will be the name of the attribute concept or dimension concept in
1633 the key family.

1634

1635 An XSD global element will be declared for each CrossSectionalMeasure declared in
1636 the key family, with the name of the measure’s concept. It will have a

1637 substitutionGroup attribute with a value “cross:Obs”. It will have a type of “ObsType”.
1638 If no CrossSectionalMeasures have been declared, use the PrimaryMeasure instead.
1639

1640 An XSD complexType element will be declared for each CrossSectionalMeasure
1641 declared in the key family with a name created by appending “Type” to the concept of
1642 the measure. These declarations will contain an XSD complexContent. This will
1643 contain an XSD extends element with a base attribute of “cross:ObsType”. It will
1644 contain an XSD sequence element. The sequence element will contain an element
1645 named Annotations, with a type of AnnotationsType from the “common:” namespace.
1646 It will have a minOccurs value of “0”.

1647

1648 The extension element will also have an XSD attribute element in it for each attribute
1649 or dimension which has a crossSectionalAttachObservation value of “true” and lists
1650 the name of the measure’s concept in an AttachmentMeasure element in its
1651 declaration. The XSD attribute will take its name value from the name of the
1652 attribute’s concept. It will have a use attribute of optional, and a type as described for
1653 the DataSet level, above. Additionally, an attribute will be declared with a name of
1654 “value” and a type of XSD “double”. Its use attribute will be “optional”. (Note that the
1655 dimension whose coded representation corresponds to the CrossSectionalMeasures
1656 should never have its crossSectionalAttachObservation attribute set to “true”.)

1657

1658 If no CrossSectionalMeasures were declared in the key family, there will be an XSD
1659 attribute element added to the extension, which will have a name attribute whose
1660 value is the concept name of the PrimaryMeasure concept from the key family. It will
1661 have a use attribute of “optional” and a type of XSD “double”.

1662

1663 In this case, for each attribute declared in the key family with an attachmentLevel of
1664 “Observation”, an XSD attribute will be added to the extends. Each XSD attribute will
1665 take the name of the attribute’s concept declared in the key family, and will have a
1666 use attribute of “optional”. Its type will be defined as for the DataSet-level attributes
1667 described above. Additionally, an attribute will be declared with a name of value and
1668 a type of “xs:double”. Its use attribute is “optional”.

1669

1670 No other declarations or constructs will be added to the schemas created using this
1671 mapping.

1672

1673 **Delete and Update Messages in CrossSectionalData:** In the Header element, the
1674 action field specifies whether a message is an update message or a delete message.
1675 If it is an update message, it is used to send new information or updated information,
1676 which may include only data, only documentation (that is, attribute values as
1677 described in the key family), or both. (Agreements regarding the use of update
1678 messages should be specified between counterparties.) For a delete message, the
1679 requirements are that a complete key always be sent for the deletion of data, which is
1680 identified either as an entire series by the absence of any specified time periods, or
1681 for a specific set of time periods, by the inclusion of those time periods. Attribute
1682 values may be deleted by sending a complete or partial set of attributes, with any
1683 valid value for the attribute (according to the XSD schema) being taken as an
1684 indication that the current attribute value should be deleted.

1685

1686

1687 **Utility Schemas**

1688

1689 Utility schemas are different from the Compact and Cross-Sectional schemas
1690 because they differentiate between the expression of the attributes and dimensions
1691 established in the key family. This design serves to preserve the ordering of the keys
1692 - the design provides much of the key-family structural metadata without requiring the
1693 processor to access the XML structure message describing the key family. This
1694 makes the rules inherent in the structure of the key family available to such tools as
1695 schema-guided XML editors, which are part of the primary reason for the Utility
1696 schema format.

1697

1698 The Utility schema employs a technique similar to the Compact and Cross-Sectional
1699 schemas by creating substitution groups which are headed by elements at the
1700 DataSet, Group, Series, and Observation levels. This is done in such a way that the
1701 messages can be more completely validated with a generic XML parser but are
1702 considerably larger in size than the CompactData or CrossSectionalData formats.

1703

1704 A global element named "DataSet" will be declared, with an XSD substitutionGroup
1705 attribute which has a value referencing the DataSet element in the "utility:"
1706 namespace. Its type attribute will reference DataSetType in the key-family-specific
1707 namespace.

1708

1709 An XSD complexType will be declared named "DataSetType". It will have XSD
1710 complexContent containing an XSD extension element, with a base attribute of
1711 DataSetType in the "utility:" namespace. The extension will consist of an XSD
1712 sequence element containing first an XSD choice element, with a maxOccurs value
1713 of "unbounded". The choice will contain an XSD element reference for each named
1714 group declared in the key family. They will each have an XSD ref attribute with a
1715 value of the group name provided in the key family. (These elements will take the
1716 names of the groups declared in the key family.) If there are no named groups
1717 declared in the key family, an XSD element will be declared in the choice with a ref
1718 attribute with a value of Series. An element named Annotations will be added to the
1719 end of the sequence, with a type of AnnotationsType from the "common:" namespace
1720 and a minOccurs attribute of "0".

1721

1722 For each attribute declared in the key family with an attachmentLevel of "DataSet",
1723 an XML attribute will be declared in the extension. It will have the same name as the
1724 attribute's concept in the key family. It will have a use attribute with a value of
1725 "required" if the attribute declared in the key family has an assignmentStatus of
1726 "Mandatory:", and a use attribute with a value of optional if its assignmentStatus in the
1727 key family is "Conditional". For coded attributes, the XML attribute will be given a type
1728 value which is the id of the codelist which represents it. In the key-family-specific
1729 namespace, this codelist will be represented by a simpleType declaration which
1730 contains a list of enumerations, equivalent to the values of the codelist, as described
1731 in the key family. These will be extension of the XSD "string" datatype. The
1732 enumerated values will be the values of the codes. The descriptions of the codes will
1733 be placed inside XSD "documentation" elements, contained in XSD "annotation"
1734 elements, which are themselves contained in the XSD "enumeration" elements as the
1735 first instance of the XSD documentation element. No other text shall occur within this

1736 particular instance of the XSD documentation element, although other XSD
1737 documentation elements may occur within any given XSD enumeration element.

1738

1739 Uncoded attributes will also be represented with XSD simpleType elements declared
1740 in the key-family-specific namespace, with names formed by taking the name of the
1741 attribute's concept in the key family and appending "Type" to them. If unrestricted,
1742 these will be of the W3C XML Schema primitive type "string"; if restrictions are
1743 specified in the key family, these will be restrictions of the XSD "string" datatype,
1744 unless they have a maximum length specified in the key family. If a maximum length
1745 is provided in the key family description, this will be handled as follows:

1746

- 1747 • If numeric, then the restriction base will be of the XSD datatype "decimal".
- 1748 • If alphabetic, then the restriction base will be of the common:AlphaType
1749 datatype.
- 1750 • If alphanumeric, then the restriction base will be of the
1751 common:AlphaNumericType datatype.

1752

1753 If maximum length is specified, but the attribute's value is not fixed length, then the
1754 maxLength facet in the XSD simpleType should be set to equal the maximum length
1755 of the attribute as specified in the key family. If the attribute's value is fixed length,
1756 then the XSD minLength attribute should additionally be set to the same value. If an
1757 uncoded attribute is a numeric type, and a number of decimals has been specified in
1758 the key family, then the simple type's fracDig facet should take the value specified in
1759 the key family.

1760

1761 For each named Group in the key family, a global XSD element will be declared,
1762 taking the name of the group. Its XSD type attribute will have a value formed by
1763 taking the name of the element and adding "Type" to the end of it. It will have a
1764 substitutionGroup attribute which references the Group element declared in the
1765 "utility:" namespace.

1766

1767 An XSD complexType will be declared for each named group declared in the key
1768 family, with a name formed by taking the name of the group in the key family and

1769 appending “Type” to it. It will have an XSD complexContent element which contains
1770 an XSD extends with a base attribute value of utility:GroupType. The extends will
1771 contain an XSD sequence element, which will contain an XSD element with a
1772 reference to the element Series. Its maxOccurs attribute will have a value of
1773 “unbounded”. An element named Annotations will be added to the end of the
1774 sequence, with a type of AnnotationsType from the “common:” namespace. It will
1775 also have a minOccurs value of “0”.

1776

1777 For each attribute in the key family with an attachmentLevel of “Group”, an XSD
1778 attribute element may be added to the extends element for any given group. To
1779 determine if a declared Group-level attribute in the key family is to be added to a
1780 particular named group XSD type, look at the AttachmentGroup elements in the XML
1781 of the key family. If the group element in the key-family-specific schema that is being
1782 declared appears in an AttachmentGroup element in the key family XML, then the
1783 attribute should be included in the utility schema being created. If added, this
1784 attribute should be declared as defined for the DataSet level, above. The name will
1785 be the name of the attribute’s concept in the key family.

1786

1787 A XSD global element named Series will be declared in the key-family-specific
1788 namespace, with a type of SeriesType and a substitutionGroup attribute referencing
1789 utility:Series.

1790

1791 An XSD complexType will then be declared with a name of SeriesType. It will have
1792 XSD complexContent, with an XSD extension element that has a base attribute value
1793 of utility:SeriesType. The extends element will contain an XSD sequence element,
1794 which will contain first an XSD element whose ref value is “Key”. This is followed by
1795 an XSD element with a ref attribute whose value is “Obs”. Its maxOccurs attribute
1796 will have a value of “unbounded”. An element named Annotations will be added to
1797 the end of the sequence, with a type of AnnotationsType from the “common:”
1798 namespace. It will also have a minOccurs value of “0”.

1799

1800 For each attribute in the key family with an attachmentLevel of “Series”, an XSD
1801 attribute element will be added to the extends element, with name, use, and type
1802 attributes defined as for the DataSet level, above.

1803

1804 A global XSD element named Key will be declared. It will have a type of KeyType,
1805 and a substitutionGroup attribute with a value of utility:Key.

1806

1807 An XSD complexType will be declared, with a name of KeyType. It will have an XSD
1808 complexContent element with an XSD extends element inside it, whose base
1809 attribute will have a value of “utility:KeyType”. The extends element will contain a
1810 XSD sequence of elements, one for each non-time dimension declared in the key
1811 family, in the order in which they appear in the XML for the key family. These
1812 elements will have names that are the same as the dimension’s concepts in the key
1813 family which they represent. Their type attributes will be the names of simpleTypes
1814 created exactly as for coded attributes at the DataSet level, above.

1815

1816 An XSD global element will be declared named “Obs”. It will have a
1817 substitutionGroup attribute with a value “utility:Obs”. It will have a type of “ObsType”.

1818

1819 An XSD complexType element will be declared with a name “ObsType” and an XSD
1820 complexContent. This will contain an XSD extends element with a base attribute of
1821 “compact:ObsType”. It will contain an XSD sequence element. The sequence
1822 element will contain an element whose name is the name of the TimeDimension
1823 concept from the key family, with a type of common:TimePeriodType. It will be
1824 followed by an element whose name is the name of the PrimaryMeasure declared in
1825 the key family, with a type of XSD “double”. Last is an element named Annotations,
1826 with a type of AnnotationsType from the “common:” namespace. It will have a
1827 minOccurs value of “0”.

1828

1829 For each attribute declared in the key family with an attachmentLevel of
1830 “Observation”, an XSD attribute will be added to the extends. Each XSD attribute will
1831 take the name of the attribute’s concept declared in the key family, and will have a

1832 use attribute, name, and type created as defined as for the DataSet-level attributes
1833 described above.

1834

1835 No other declarations or constructs will be added to the schemas created using this
1836 mapping.

1837

1838 **Note:** The UtilityData key-family-specific schema does not have any mechanism for
1839 expressing time ranges across a set of observation values. The only permissible
1840 message for this schema type is an “update” message containing a complete set of
1841 attributes and observation values for the transmitted series. There is no concept of a
1842 “delete” message, and the action field in the message Header element is ignored if
1843 specified.

1844

1845 **VII. APPENDIX: SAMPLE SDMX-ML MESSAGES**

1846 This appendix is presented to provide example layouts for the SDMX-ML sample
1847 data files, allowing them to be more easily understood. For each sample data file,
1848 one or more tables are offered, to show how the data itself might be formatted.
1849 Please note that all data is fictitious, and used for demonstration purposes only.
1850 (Numbers are not consistent across samples, but are randomly generated.)

1851

1852 **A. CompactSample.xml**

1853 **ID:** Message JD014 (Untruncated Test Message)

1854 **Name:** Trans46305

1855 **Prepared:** 2001-03-11T09:30:47-05:00

1856 **Sent by:** GB Smith from the BIS, +000.000.0000

1857 **To:** B.S. Featherstone, Statistics Division, ECB, +000.000.0001

1858

1859 This message contains new or updated data, and was created at 2001-03-
1860 11T09:30:47-05:00.

1861

1862 **External Debt, All Maturities, Bank Loans for Mexico, expressed as Stocks**
 1863 **in Millions of US Dollars, Monthly at the beginning of period. (Free data)**

| 1864 | <u>Time</u> | <u>Data</u> |
|------|-------------|-------------|
| 1865 | 2000-01 | - 3.14 |
| 1866 | 2001-02 | - 2.29 |
| 1867 | 2000-03 | - 3.14 |
| 1868 | 2000-04 | - 5.24 |
| 1869 | 2000-05 | - 3.14 |
| 1870 | 2000-06 | - 3.78 |
| 1871 | 2000-07 | - 3.65 |
| 1872 | 2000-08 | - 2.37 |
| 1873 | 2000-09 | - 3.14 |
| 1874 | 2000-10 | - 3.17 |
| 1875 | 2000-11 | - 3.34 |
| 1876 | 2000-12 | - 1.21 |

1877

1878

1879

1880

1881 **External Debt, All Maturities, Bank Loans for Mexico, expressed as Stocks**
 1882 **in Millions of US Dollars, Annually at the beginning of period. (Free data)**

| 1883 | <u>Time</u> | <u>Data</u> |
|------|-------------|-------------|
| 1884 | 2000-01 | 3.14 |

1886

1887 **External Debt, All Maturities, Debt Securities Issued Abroad for Mexico,**
 1888 **expressed as Stocks in Millions of US Dollars, Monthly at the beginning of**
 1889 **period. (Free data)**

| 1890 | <u>Time</u> | <u>Data</u> |
|------|-------------|-------------|
| 1891 | 2000-01 | 5.14 |
| 1892 | 2001-02 | 3.29 |
| 1893 | 2000-03 | 6.14 |
| 1894 | 2000-04 | 2.24 |
| 1895 | 2000-05 | 3.14 |
| 1896 | 2000-06 | 7.78 |
| 1897 | 2000-07 | 3.65 |
| 1898 | 2000-08 | 5.37 |
| 1899 | 2000-09 | 3.14 |
| 1900 | 2000-10 | 1.17 |
| 1901 | 2000-11 | 4.34 |
| 1902 | 2000-12 | 1.21 |

1904



1905 **External Debt, All Maturities, Debt Securities Issued Abroad for Mexico,**
1906 **expressed as Stocks in Millions of US Dollars, Annually at the beginning**
1907 **of period. (Free data)**

1908

1909 Time Data

1910 2000-1 4.14

1911

1912 **B. UtilitySample.xml**

1913 **ID:** Message JD01678594 (Untruncated Test Message)

1914 **Name:** Trans46304

1915 **Prepared:** 2001-03-11T09:30:47-05:00

1916 **Sent by:** GB Smith from the BIS, +000.000.0000

1917 **To:** B.S. Featherstone, Statistics Division, ECB, +000.000.0001

1918

1919 This message contains new or updated data, and was created at 2001-03-
1920 11T09:30:47-05:00.

1921

1922 **External Debt, All Maturities, Bank Loans for Mexico, expressed as Stocks**
1923 **in Millions of US Dollars, Monthly at the beginning of period. (Free data)**

1924

1925 Time Data

1926 2000-01 - 3.14

1927 2001-02 - 3.19

1928 2000-03 - 5.26

1929 2000-04 - 5.12

1930 2000-05 - 4.13

1931 2000-06 - 3.12

1932 2000-07 - 3.14

1933 2000-08 - 3.79

1934 2000-09 - 9.79

1935 2000-10 - 3.14

1936 2000-11 - 3.19

1937 2000-12 - 3.14

1938

1939

1940 **C. GenericSample.xml**

1941 **ID:** Message JD014 (Untruncated Test Message)



1942 **Name:** Trans46302

1943 **Prepared:** 2001-03-11T09:30:47-05:00

1944 **Sent by:** GB Smith from the BIS, +000.000.0000

1945 **To:** B.S. Featherstone, Statistics Division, ECB, +000.000.0001

1946

1947 This message contains new or updated data, and was created at 2001-03-
1948 11T09:30:47-05:00.

1949

1950 **External Debt, All Maturities, Bank Loans for Mexico, expressed as Stocks**
1951 **in Millions of US Dollars, Monthly at the beginning of period. (Free data)**

1952 Time Data

1953 2000-01 - 3.14

1954 2001-02 - 3.14

1955 2000-03 - 4.29

1956 2000-04 - 6.04

1957 2000-05 - 5.18

1958 2000-06 - 5.07

1959 2000-07 - 3.13

1960 2000-08 - 1.17

1961 2000-09 - 1.14

1962 2000-10 - 3.04

1963 2000-11 - 1.14

1964 2000-12 - 3.24

1965

1966 **D. CrossSectionalSample.xml**

1967 **ID:** Message BIS947586 (Untruncated Test Message)

1968 **Name:** Trans46305

1969 **Prepared:** 2001-03-11T09:30:47-05:00

1970 **Sent by:** GB Smith from the BIS, +000.000.0000

1971 **To:** B.S. Featherstone, Statistics Division, ECB, +000.000.0001

1972

1973 This message contains new or updated data, and was created at 2001-03-
1974 11T09:30:47-05:00.

1975

1976 **External Debt for Mexico, in Millions of US Dollars, at the beginning of**
1977 **period for 2000. (Free data)**

1978 Topic

Stocks Flows



| | | | |
|------|---|------|--------|
| 1979 | All Maturities, Bank Loans | 3.14 | 1.00 |
| 1980 | All Maturities, Debt Securities Issued Abroad | 6.39 | 2.27 |
| 1981 | All Maturities, Brady Bonds | 2.34 | -1.00 |
| 1982 | All Maturities, Non-Bank Trade Credits | 3.19 | - 1.06 |
| 1983 | | | |
| 1984 | | | |