SDMX-ML:

SCHEMA AND DOCUMENTATION

(VERSION 1.0)

Initial Release September 2004

http://www.sdmx.org/

I. BACKGROUND

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

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

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

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

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

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

B. Results: the XML Design

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

- Key family structure description schema ("`SDMXStructure.xsd`")
- Generic data schema for data-sharing exchange ("`SDMXGenericData.xsd`")
- Generic query schema for invoking web services ("`SDMXQuery.xsd`")
- Key-family-specific schema for updates and revisions/bilateral exchange ("`SDMXCompactData.xsd`")
- Key-family-specific schema for presentational processing and internal use ("`SDMXUtilityData.xsd`")
- Requested: Key-family-specific schema for cross-sectional data – which may be combined with the Compact document type ("`SDMXCrossSectionalData.xsd`")

C. Fostering the Use of a Standard SDMX-ML

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

II. NORMATIVE REFERENCES

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

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

III. CONFORMANCE

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

IV. DESIGN OVERVIEW

A. Scope and Requirements

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

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

- It should be possible to present natural-language information in mutliple, equivalent languages.

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

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

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

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

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

B. Design Approach

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

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

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

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

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

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

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

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

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

C. SDMX-ML Packaging: Namespace Modules

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

The other major benefit of namespaces – especially in light of the requirement that 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

- An SDMX Namespace Module providing the common framework to be used for all key-family-specific schemas for webmasters and developers using standard XML tools ("SDMXUtilityData.xsd") –for processing and publication production use
- A set of namespaced modules created and maintained by those who create key-family-specific "Utility" schemas – not maintained by SDMX
- An SDMX Namespace Module providing the common framework to be used for all key-family-specific schemas for cross-sectional data ("SDMXCrossSectionalData.xsd") – for bilateral use and cross-sectional processing of data
- A set of namespaced modules created and maintained by those who create key-family-specific "Cross-sectional" schemas – not maintained by SDMX

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

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

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

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

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

A. SDMX Message Namespace Module

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

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

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

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

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

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

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

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

Global Elements

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

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

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

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

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

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

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

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

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

Complex Types

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

Element Content (Type):

Header (HeaderType)

StructureType: StructureType defines the contents of a structure message.

Extends: MessageType

Element Content (Type):

Agencies (structure:AgenciesType) - min. 0

CodeLists (structure:CodeListsType) - min. 0

Concepts (structure:ConceptsType) - min. 0

KeyFamilies (structure:KeyFamiliesType) - min. 0

GenericDataType: GenericDataType defines the contents of a GenericData message.

Extends: MessageType

Element Content (Type):

DataSet (generic:DataSetType)

UtilityDataType: UtilityDataType defines the contents of a UtilityData message.

Extends: MessageType

Element Content (Type):

[Reference] (utility:DataSet)

CompactDataType: CompactDataType defines the contents of a CompactData message.

Extends: MessageType

Element Content (Type):

[Reference] (compact:DataSet)

CrossSectionalDataType: CrossSectionalDataType defines the contents of a CrossSectionalData message.

Extends: MessageType

Element Content (Type):

[Reference] (cross:DataSet)

QueryMessageType: QueryMessageType defines the contents of a QueryMessage.

Extends: MessageType

Element Content (Type):

Query (query:QueryType)

MessageGroupType: MessageGroupType defines the contents of a MessageGroup message.

Extends: MessageType

Choice:

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

Choice:

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

Choice:

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

Choice:

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

Attribute: id(xs:NMTOKEN) - optional

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

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

Element Content (Type):

ID (xs:NCName)

Test (xs:boolean)

Truncated (xs:boolean)

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

Prepared (HeaderTimeType)

Sender (PartyType)

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

KeyFamilyRef (xs:NMTOKEN) - min. 0

KeyFamilyAgency (xs:NMTOKEN) – min. 0

DataSetAgency (xs:NMTOKEN) - min. 0

DataSetID (xs:NMTOKEN) - min. 0

DataSetAction (common:ActionType) - min. 0

Extracted (xs:dateTime) - min. 0

ReportingBegin (HeaderTimeType) - min. 0

ReportingEnd (HeaderTimeType) - min. 0

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

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

Element Content (Type):

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

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

Attribute: id (xs:NMTOKEN) - required

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

Element Content (Type):

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

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

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

Choice: min. 0 - max. unbounded

Telephone (xs:string)

Fax (xs:string)

X400 (xs:string)

URI (xs:string)

Email (xs:string)

Simple Types

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

B. SDMX Structure Namespace Module

Namespace: http://www.SDMX.org/resources/SDMXML/schemas/v1_0/structure

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

Complex Types

AgenciesType: AgenciesType contains one or more Agencies.

Element Content (Type):

Agency (AgencyType) - max. unbounded

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

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

Element Content (Type):

Name (common:TextType) - max. unbounded

MaintenanceContact (ContactType) - min. 0

CollectorContact (ContactType) - min. 0

DisseminatorContact (ContactType) - min. 0

ReporterContact (ContactType) - min. 0

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

Attribute: id (xs:NCName) - required

Attribute: version (xs:string) - optional

Attribute: uri (xs:anyURI) – optional

Attribute: isExternalReference (xs:Boolean) - optional

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

Element Content (Type):

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

id (xs:NMTOKEN) - min. 0

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

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

Choice: min. 0 - max. unbounded

Telephone (xs:string)

Fax (xs:string)

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

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

Element Content (Type):

Concept (ConceptType) - max. unbounded

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

Element Content (Type):

Name (common:TextType) - max. unbounded

Annotations (common:AnnotationsType) - min. 0

Attribute: id (xs:NCName) - required

Attribute: agency (xs:NMTOKEN) - optional

Attribute: version (xs:string) - optional

Attribute: uri (xs:anyURI) – optional

Attribute: isExternalReference (xs:Boolean) - optional

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

Element Content (Type):

KeyFamily (KeyFamilyType) - max. unbounded

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

Element Content (Type):

Name (common:TextType) - max. unbounded

Components (ComponentsType)

Annotations (common:AnnotationsType) - min. 0

Attribute: id (xs:NCName) - required

Attribute: agency (xs:NMTOKEN) - optional

Attribute: version (xs:string) - optional

Attribute: uri (xs:anyURI) - optional

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

Element Content (Type):

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

TimeDimension (TimeDimensionType) - min. 0

PrimaryMeasure (PrimaryMeasureType)

CrossSectionalMeasure (CrossSectionalMeasureType) – min. 0 – max unbounded

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

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

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

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

Element Content (Type):

Annotations (common:AnnotationsType) - min. 0

Attribute: concept (xs:NMTOKEN) - required

Attribute: codelist (xs:NMTOKEN) - required

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

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

Attribute: crossSectionalAttachDataSet (xs:boolean) - optional

Attribute: crossSectionalAttachGroup (xs:boolean) - optional

Attribute: crossSectionalAttachSection (xs:boolean) - optional

Attribute: crossSectionalAttachObservation (xs:boolean) - optional

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

Element Content (Type):

TextFormat (TextFormatType) - min. 0

Annotations (common:AnnotationsType) - min. 0

Attribute: concept (xs:NMTOKEN) - required

Attribute: codelist (xs:NMTOKEN) - optional

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

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

Element Content (Type):

DimensionRef (xs:NMTOKEN) - max. unbounded

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

Annotations (common:AnnotationsType) - min. 0

Attribute: name (xs:NMTOKEN) - required

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

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

Element Content (Type):

TextFormat (TextFormatType) - min. 0

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

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

Annotations (common:AnnotationsType) - min. 0

Attribute: concept (xs:NMTOKEN) - required

Attribute: codelist (xs:NMTOKEN) - optional

Attribute: attachmentLevel (structure:AttachmentLevelType) - required

Attribute: assignmentStatus (structure:AssignmentStatusType) - required

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

Attribute: crossSectionalAttachDataSet (xs:boolean) - optional

Attribute: crossSectionalAttachGroup (xs:boolean) - optional

Attribute: crossSectionalAttachSection (xs:boolean) - optional

Attribute: crossSectionalAttachObservation (xs:boolean) - optional

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

Attribute: length (xs:integer) - optional

Attribute: decimals (xs:integer) - optional

Attribute: TextType (TextTypeType) - optional

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

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

Element Content (Type):

Annotations (common:AnnotationsType) - min. 0

Attribute: concept (xs:NMTOKEN) - required

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

Element Content (Type):

Annotations (common:AnnotationsType) - min. 0

Attribute: concept (xs:NMTOKEN) - required

Attribute: measureDimension (xs:NMTOKEN) - required

Attribute: code (xs:NMTOKEN) - required

Simple Types

AttachmentLevelType:

Restricts xs:NMTOKEN

Code: DataSet - Data set level

Code: Group - Group level

Code: Series - Series level

Code: Observation - Observation level

AssignmentStatusType:

Restricts xs:NMTOKEN

Code: Mandatory - Providing attribute value is mandatory

Code: Conditional - Providing attribute value is optional

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

Restricts xs:NMTOKEN

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

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

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

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

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

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

C. SDMX Generic Data Namespace Module

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

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

Global Elements

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

Complex Types

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

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

Attribute: keyFamilyURI (xs:anyURI) – optional

Element Content (Type):

KeyFamilyRef (xs:NCName)

Attributes (ValuesType) - min. 0

Choice: - min. 0 – max. unbounded

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

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

Annotations (common:AnnotationsType) - min. 0

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

Attribute: type (xs:NMTOKEN) – required

Element Content (Type):

GroupKey (ValuesType) – min. 0

Attributes(ValuesType) – min. 0

Series (SeriesType) - max. unbounded

Annotations (AnnotationsType) – min. 0

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

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

Element Content (Type):

SeriesKey (SeriesKeyType)

Attributes (ValuesType) - min. 0

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

Annotations (common:AnnotationsType) - min. 0

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

Element Content (Type):

Value (ValueType) - max. unbounded

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

Element Content (Type):

Time (common:TimePeriodType)

ObsValue (ObsValueType) - min. 0

Attributes (ValuesType) - min. 0

Annotations (common:AnnotationsType) - min. 0

ValuesType:

Element Content (Type):

Value (ValueType) - max. unbounded

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

Attribute: concept (xs:NCName)

Attribute: value (xs:string)

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

Attribute: value (xs:double)

D. SDMX Query Namespace Module

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

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

Global Elements

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

Complex Types

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

Element Content (Type):

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

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

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

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

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

Attribute: defaultLimit (xs:integer) - optional

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

Element Content (Type):

(Choice)

DataSet (xs:string)

KeyFamily (xs:string)

Dimension (DimensionType)

Attribute (AttributeType)

Codelist (CodelistType)

Time (TimeType)

Category (CategoryType)

Concept (xs:string)

Agency (xs:string)

Or (OrType)

And (AndType)

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

Element Content (Type):

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

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

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

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

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

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

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

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

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

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

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

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

Element Content (Type):

DataSet (xs:string) - min. 0 - max. unbounded
KeyFamily (xs:string) - min. 0 - max. unbounded
Dimension (DimensionType) - min. 0 - max. unbounded
Attribute (AttributeType) - min. 0 - max. unbounded
Codelist (CodelistType) - min. 0 - max. unbounded
Time (TimeType) - min. 0 - max. unbounded
Category (CategoryType) - min. 0 - max. unbounded
Concept (xs:string) - min. 0 - max. unbounded
Agency (xs:string) - min. 0 - max. unbounded
Or (OrType) - min. 0 - max. unbounded
And (AndType) - min. 0 - max. unbounded

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

[data] (xs:string)

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

[data] (xs:string)

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

[data] (xs:string)

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

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

[data] (xs:string)

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

Element Content (Type):

(Choice)

KeyFamily (xs:string)

Dimension (DimensionType)

Attribute (AttributeType)

Codelist (CodelistType)

Category (CategoryType)

Concept (xs:string)

Agency (xs:string)

Or (OrType)

And (AndType)

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

Element Content (Type):

(Choice)

Codelist (CodelistType)

Agency (xs:string)

Or (OrType)

And (AndType)

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

Element Content (Type):

(Choice)

Concept (xs:string)

Agency (xs:string)

Or (OrType)

And (AndType)

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

Element Content (Type):

(Choice)

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

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

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

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

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

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

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

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

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

Element Content (Type):

(Choice)

StartTime (common:TimePeriodType)

EndTime (common:TimePeriodType) – min. 0

Or:

Time (common:TimePeriodType)

Simple Types

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

Restricts xs:NMTOKEN

Code: DataSet - Attached at the Data Set level

Code: Group - Attached at the Group level

Code: Series - Attached at the Series level

Code: Observation - Attached at the Observation level

Code: Any - Attached at any attachment level

E. SDMX Common Namespace Module

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

Imports: http://www.w3.org/XML/1998/namespace (xml.xsd)

Complex Types

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

[data] (xs:string)

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

Element Content (Type):

AnnotationTitle (xs:string) - min. 0

AnnotationType (xs:string) - min. 0

AnnotationURL (xs:anyURI) - min. 0

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

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

Element Content (Type):

Annotation (AnnotationType) - max. unbounded

Simple Types

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

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

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

Restricts xs:NMTOKEN

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

Code: Delete - Data is to be deleted.

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

Restricts xs:string

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

Restricts xs:string

F. Data Formatting and Character Encoding

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

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

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

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

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

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

A. Compact Data Message Core Structure

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

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

(SDMXCommon.xsd)

Global Elements

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

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

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

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

Complex Types

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

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

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

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

B. Utility Data Message Core Structure

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

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

(SDMXCommon.xsd)

Global Elements

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

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

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

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

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

Complex Types

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

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

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

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

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

C. Cross-Sectional Data Message Core Structure

Namespace: http://www.SDMX.org/resources/SDMXML/schemas/v1_0/cross

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

Global Elements

DataSet(DataSetType): DataSet contains the data set.

Group(GroupType): Group contains the group.

Section(SectionType): Section contains the data section.

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

Complex Types

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

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

Attribute: time (common:TimePeriodType) - optional

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

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

D. Mappings to Key-Family-Specific Schemas

General Rules:

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

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

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

- DataSet level
- Group level
- Series level
- Observation level

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

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

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

Compact Schemas:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Cross-Sectional Schemas

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Utility Schemas

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

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

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

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

For each attribute declared in the key family with an attachmentLevel of “DataSet”, an XML attribute will be declared in the extension. It will have the same name as the attribute’s concept in the key family. It will have a use attribute with a value of “required” if the attribute declared in the key family has an assgnmentStatus of “Mandatory:, and a use attribute with a value of optional if its assignmentStatus in the key family is “Conditional”. For coded attributes, the XML attribute will be given a type value which is the id of the codelist which represents it. In the key-family-specific namespace, this codelist will be represented by a simpleType declaration which contains a list of enumerations, equivalent to the values of the codelist, as described in the key family. These will be extension of the XSD "string" datatype. The enumerated values will be the values of the codes. The descriptions of the codes will be placed inside XSD "documentation" elements, contained in XSD "annotation" elements, which are themselves contained in the XSD "enumeration" elements as the 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

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

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

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

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

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

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

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

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

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

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

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

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

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

VII. APPENDIX: SAMPLE SDMX-ML MESSAGES

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

A. CompactSample.xml

ID: Message JD014 (Untruncated Test Message)
Name: Trans46305
Prepared: 2001-03-11T09:30:47-05:00
Sent by: GB Smith from the BIS, +000.000.0000
To: B.S. Featherstone, Statistics Division, ECB, +000.000.0001

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

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

Time	Data
2000-01	3.14
2001-02	2.29
2000-03	3.14
2000-04	5.24
2000-05	3.14
2000-06	3.78
2000-07	3.65
2000-08	2.37
2000-09	3.14
2000-10	3.17
2000-11	3.34
2000-12	1.21

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

Time	Data
2000-01	3.14

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

Time	Data
2000-01	5.14
2001-02	3.29
2000-03	6.14
2000-04	2.24
2000-05	3.14
2000-06	7.78
2000-07	3.65
2000-08	5.37
2000-09	3.14
2000-10	1.17
2000-11	4.34
2000-12	1.21

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

Time	Data
2000–1	4.14

B. UtilitySample.xml

ID: Message JD01678594 (Untruncated Test Message)

Name: Trans46304

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

Sent by: GB Smith from the BIS, +000.000.0000

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

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

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

Time	Data
2000-01	- 3.14
2001-02	- 3.19
2000-03	- 5.26
2000-04	- 5.12
2000-05	- 4.13
2000-06	- 3.12
2000-07	- 3.14
2000-08	– 3.79
2000-09	– 9.79
2000-10	– 3.14
2000-11	– 3.19
2000-12	– 3.14

C. GenericSample.xml

ID: Message JD014 (Untruncated Test Message)

Name: Trans46302

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

Sent by: GB Smith from the BIS, +000.000.0000

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

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

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

Time	Data
2000-01	3.14
2001-02	3.14
2000-03	4.29
2000-04	6.04
2000-05	5.18
2000-06	5.07
2000-07	3.13
2000-08	1.17
2000-09	1.14
2000-10	3.04
2000-11	1.14
2000-12	3.24

D. CrossSectionalSample.xml

ID: Message BIS947586 (Untruncated Test Message)

Name: Trans46305

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

Sent by: GB Smith from the BIS, +000.000.0000

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

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

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

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			